

LabVIEW 기본 매뉴얼

LabVIEW 기본 매뉴얼

목차

1. LabVIEW란?	35
2. LabVIEW 요구사항	35
3. LabVIEW의 전체 구성	36
4. LabVIEW 개념	38
4.1 LabVIEW 개념 개요	38
4.1.1 LabVIEW 환경	39
4.1.1.1 LabVIEW 환경 개요	39
4.1.1.2 LabVIEW 환경 보안하기	39
4.1.1.3 작업 환경 사용자 정의하기	39
4.1.2 LabVIEW 기능	43
4.1.2.1 LabVIEW의 검색 옵션	43
4.1.2.2 컨트롤과 함수 팔레트 탐색하기	45
4.1.2.3 메뉴와 도구 모음	46
4.1.2.4 탐색 윈도우	47
4.1.2.5 불리언 컨트롤과 인디케이터	47
4.1.2.6 컨테이너 컨트롤과 인디케이터	50
4.1.2.7 LabVIEW에서 끌어서 놓기	51
4.1.2.8 블록다이어그램 만들기	52
4.1.2.9 블록다이어그램 객체	52
4.1.2.10 텍스트 특성	59
4.1.2.11 숫자형 데이터	61
4.1.2.12 다형성 단위 사용하기	69
4.1.2.13 컴파일된 객체 캐시	73
4.1.2.14 배포 타입	74
4.1.3 LabVIEW 파일 시스템의 구조	74
4.1.3.1 LabVIEW의 구성	74
4.1.3.2 문자열, 클러스터, 배열, 집합을 이용한 데이터의 그룹화	77
4.1.4 LabVIEW 서브팔레트	78
4.1.4.1 그래프 서브팔레트	78
4.1.4.2 컨트롤 서브팔레트	79
4.1.4.3 데이터타입 서브팔레트	79
4.1.4.4 3D 그래프 서브팔레트	80
4.1.5 LabVIEW 컨트롤과 인디케이터	81
4.1.5.1 I/O 이름 컨트롤과 인디케이터	81

4.1.5.2	리스트, 테이블, 트리 컨트롤과 인디케이터	84
4.1.5.3	.NET 및 ActiveX 컨트롤과 인디케이터	85
4.1.5.4	숫자형 컨트롤과 인디케이터	86
4.1.5.5	참조 번호 컨트롤과 인디케이터	87
4.1.5.6	링 및 열거형 타입 컨트롤과 인디케이터	88
4.1.5.7	문자열 및 경로 컨트롤과 인디케이터	89
4.1.5.8	배리언트 및 클래스 컨트롤과 인디케이터	92
4.1.5.9	사용자 컨트롤, 인디케이터, 타입 정의 생성하기	92
4.1.5.10	LabVIEW의 X컨트롤	101
4.1.6	LabVIEW의 함수	104
4.1.6.1	함수 개요	104
4.1.6.2	다형성 함수	106
4.1.6.3	비교 함수 사용하기	111
4.1.7	LabVIEW의 VI	114
4.1.7.1	익스프레스 VI	114
4.1.7.2	배리언트 데이터 처리하기	125
4.1.7.3	VI와 SubVI 생성하기	127
4.1.7.4	내장된 VI와 함수 사용하기	127
4.1.7.5	LabVIEW의 SubVI	127
4.1.7.6	VI의 계층구조 보기	132
4.1.7.7	재귀 VI	133
4.1.7.8	다형성 VI	134
4.1.7.9	가변 VI	136
4.1.7.10	가변 VI와 다형성 VI 중 선택하기	138
4.1.7.11	VI 아이콘	139
4.1.7.12	VI 저장하기	145
4.1.7.13	VI 사용자 정의하기	148
4.1.7.14	VI 실행하기 및 디버깅하기	151
4.1.7.15	VI 실행하기	151
4.1.7.16	VI 개정 히스토리 설정하기	152
4.1.7.17	정의되지 않은 데이터 방지하기	153
4.1.7.18	잘못된 의존성 방지하기	153
4.1.7.19	빌드 전 또는 후에 VI 실행하기	154
4.1.7.20	VI의 원형을 리셋하는 커넥터 팬 변경 내역 (공유 라이브러리)	154
4.1.7.21	그림 플롯 VI	155
4.1.7.22	그림 함수 VI	156
4.1.7.23	그래픽 포맷 VI	158
4.1.7.24	사운드 VI	158
4.1.7.25	그래픽 & 사운드 VI	163

4.1.7.26	VI 문서화 및 인쇄하기	163
4.1.7.27	3D 그림 컨트롤 VI	164
4.1.7.28	VI 문서화하기	164
4.1.7.29	VI 인쇄하기	165
4.1.7.30	프로그램식으로 VI 제어하기	169
4.1.7.31	다이나믹하게 VI 로딩 및 호출하기	170
4.1.7.32	비동기적으로 VI 호출하기	171
4.1.7.33	콜백 VI	174
4.1.7.34	VI에서 데이터를 E-메일하기	175
4.1.7.35	VI와 일반적인 VI 객체의 조직	176
4.1.7.36	VI 스크립팅	184
4.1.7.37	VI 서버의 기능	184
4.1.7.38	VI 서버 어플리케이션 만들기	185
4.1.7.39	어플리케이션과 VI 셋팅 변경하기	186
4.1.7.40	원격 컴퓨터에서 VI 편집하기와 실행하기	189
4.1.7.41	기본 도움말 윈도우의 VI 스크립팅 정보	190
4.1.7.42	VI 실행 속도	191
4.1.7.43	VI 메모리 사용	199
4.1.7.44	VI 코드 복잡도	233
4.1.7.45	재호출: 같은 SubVI에 동시에 여러 호출을 허용	234
4.1.7.46	행렬 사용하기	238
4.1.7.47	VI 포팅과 현지화하기	242
4.1.7.48	플랫폼 사이에서 VI 포팅하기	242
4.1.7.49	VI 현지화하기	247
4.1.7.50	디버깅 기법	254
4.1.7.51	미리 정의된 명령 라인 작업	267
4.1.7.52	Windows Embedded Standard 타겟	272
4.1.8	LabVIEW의 블록다이어그램	277
4.1.8.1	블록다이어그램 디자인하기	277
4.1.8.2	와이어를 사용하여 블록다이어그램 객체에 연결하기	278
4.1.8.3	블록다이어그램의 데이터 흐름	283
4.1.8.4	강제 변환점	285
4.1.9	LabVIEW에서 프로젝트 생성하기	290
4.1.9.1	LabVIEW에서 프로젝트 관리하기	290
4.1.9.2	프로젝트 및 타겟 사용하기	293
4.1.9.3	예제, VI 템플릿, 프로젝트 템플릿, 샘플 프로젝트	293
4.1.9.4	LabVIEW 프로젝트의 폴더 사용하기	295
4.1.9.5	LabVIEW 프로젝트에서 라이브러리 사용하기	296
4.1.9.6	LabVIEW 프로젝트에서 묶음 프로젝트 라이브러리 사용하기	300

4.1.9.7	LabVIEW 프로젝트에서 의존성 관리하기	302
4.1.9.8	프로젝트 아이템 대체하기	304
4.1.9.9	프로젝트 충돌 해결하기	304
4.1.9.10	LabVIEW 프로젝트에 타겟 추가하기(Windows)	307
4.1.9.11	어플리케이션 인스턴스로 작업하기	308
4.1.9.12	프로젝트 계획하고 설계하기	310
4.1.9.13	LabVIEW Style Checklist	311
4.1.10	LabVIEW의 소스 컨트롤	336
4.1.10.1	LabVIEW에서 소스 컨트롤 사용하기	336
4.1.10.2	소스 컨트롤 제공자 선택하기	337
4.1.10.3	LabVIEW에서 소스 컨트롤 작업	337
4.1.10.4	VI 및 다른 파일 타입에서 컴파일된 코드 분리하여 소스 컨트롤하기	339
4.1.11	빌드 스펙	341
4.1.11.1	어플리케이션 만들기 및 배포하기	341
4.1.11.2	빌드 스펙 사용하기	342
4.1.11.3	빌드 스펙 생성하기	344
4.1.11.4	빌드 스펙을 위해 파일 준비하기	345
4.1.11.5	빌드 스펙 설정하기	346
4.1.11.6	독립 어플리케이션과 공유 라이브러리에서 소스 파일 참조하기	346
4.1.11.7	빌드 스펙 관리하기	347
4.1.12	LabVIEW의 어플리케이션	348
4.1.12.1	어플리케이션 관리하기	348
4.1.12.2	빌드 어플리케이션의 기능	348
4.1.12.3	독립 어플리케이션 및 공유 라이브러리 설치하기	349
4.1.12.4	LabVIEW 런타임 엔진 사용하기	350
4.1.12.5	LabVIEW에서 ActiveX 사용하기	351
4.1.12.6	Windows Vista 및 이후 버전 OS에서 LabVIEW 어플리케이션 배포하기	359
4.1.12.7	Windows Vista 및 이후 버전 OS에 빌드 배포하기	361
4.1.12.8	Windows Embedded Standard 타겟에 설치 프로그램 배포하기	362
4.1.12.9	Windows 연결	363
4.1.13	LabVIEW의 루프와 구조	364
4.1.13.1	루프와 기타 구조	364
4.1.13.2	루프를 사용하여 반복적인 작업 컨트롤하기	366
4.1.13.3	For 루프: 지정된 횟수만큼 작업 반복하기	367
4.1.13.4	While 루프: 조건이 발생할 때까지 작업 반복하기	368
4.1.13.5	루프 반복 사이에서 값 전달하기	369
4.1.13.6	에러 레지스터: 병렬 반복에서 에러 핸들링하기	369

4.1.13.7	시프트 레지스터: 루프 반복 사이에 값 전달하기	370
4.1.13.8	피드백 노드: 루프 반복 사이에 데이터 저장하기	371
4.1.13.9	구조를 사용하여 하나 이상의 코드 섹션 실행하기	373
4.1.13.10	케이스 구조: 입력값에 따라 코드 섹션 실행하기	374
4.1.13.11	시퀀스 구조: 순차적으로 코드 섹션 실행하기	375
4.1.13.12	비활성화 구조: 코드 실행 막기	376
4.1.13.13	In Place 원소 구조: 메모리 효율성 높이기	379
4.1.13.14	LabVIEW 어플리케이션의 타이밍 컨트롤하기 (Real-Time, Windows)	380
4.1.13.15	Timed 구조의 특성(Real-Time, Windows)	381
4.1.13.16	Timed 구조의 절대 시간 및 상대 시간 이해하기	383
4.1.13.17	Timed 루프 및 프레임이 있는 Timed 루프 설정하 기 (Real-Time, Windows)	385
4.1.13.18	Timed 시퀀스 구조 설정하기(Real-Time, Windows)	389
4.1.13.19	Timed 구조의 셋팅 설정하기 (Real-Time, Windows)	390
4.1.13.20	Timed 구조의 우선순위 설정하기 (Real-Time, Windows)	396
4.1.13.21	Timed 구조의 타이밍 소스 선택하기 (Real-Time, Windows)	398
4.1.13.22	다이나믹하게 Timed 구조의 입력 옵션 설정하기 (Real-Time, Windows)	400
4.1.13.23	Timed 구조 이름 붙이기 (Real-Time, Windows)	401
4.1.13.24	Timed 구조의 시작 동기화하기 (Real-Time, Windows)	402
4.1.13.25	Timed 구조의 실행 강제 종료하기 (Real-Time, Windows)	404
4.1.14	LabVIEW의 이벤트	405
4.1.14.1	이벤트 타입	405
4.1.14.2	이벤트 드리븐 프로그래밍	406
4.1.14.3	어떤 타입의 사용자 인터페이스 이벤트를 사용할 지 결정하기	407
4.1.14.4	이벤트 구조에서 이벤트를 모니터하는 방법 선택하기	408
4.1.15	LabVIEW의 데이터 그룹화	410
4.1.15.1	문자열, 클러스터, 배열, 집합을 이용한 데이터의 그룹화	410
4.1.15.2	문자열을 이용한 데이터의 그룹화	410
4.1.15.3	배열과 클러스터를 이용한 데이터의 그룹화	417
4.1.15.4	집합을 이용한 데이터의 그룹화	423
4.1.15.5	포맷 지정자 구문	425
4.1.16	LabVIEW의 그래프와 차트	432
4.1.16.1	그래프와 차트	432
4.1.16.2	그래프와 차트의 종류	433
4.1.16.3	그래프와 차트 사용자 정의	449
4.1.16.4	엘리어스 제거 라인 플롯 사용하기	464
4.1.16.5	스케일 범례 사용하기	465
4.1.16.6	다이나믹하게 그래프 포맷하기	465

4.1.16.7	자연스러운 업데이트 사용하기	466
4.1.16.8	2D 그림 컨트롤 사용하기	466
4.1.16.9	3D 그림 컨트롤 사용하기	467
4.1.16.10	메시를 사용하여 3D 객체 그리기	471
4.1.17	LabVIEW의 파일 I/O 작업	477
4.1.17.1	파일 I/O	477
4.1.17.2	파일 I/O의 기초	478
4.1.17.3	DataPlugin을 사용하여 데이터 파일 지원 확장하기(Windows)	478
4.1.17.4	디스크 스트리밍을 사용하여 메모리 절약하기	479
4.1.17.5	진 파일 I/O 함수에 대한 파일 버퍼링을 비활성화하여 효율성 향상시키기	480
4.1.17.6	어떤 파일 포맷을 사용할지 결정하기	481
4.1.17.7	파일 경로	482
4.1.17.8	진 파일	483
4.1.17.9	설정 파일	484
4.1.17.10	데이터로그 파일	485
4.1.17.11	스프레드시트 파일	487
4.1.17.12	텍스트 파일	490
4.1.17.13	XML 파일	492
4.1.17.14	Zip 파일	498
4.1.17.15	Windows 설정 셋팅 파일 포맷	498
4.1.17.16	재사용을 위해 데이터를 기록하기	502
4.1.17.17	TDM 데이터 모델	506
4.1.17.18	웨이브폼	515
4.1.18	LabVIEW의 클래스와 객체	517
4.1.18.1	LabVIEW 객체 지향 프로그래밍	517
4.1.18.2	클래스와 객체	518
4.1.18.3	LabVIEW 클래스 생성하기	520
4.1.18.4	LabVIEW 클래스 개발하기	533
4.1.18.5	LabVIEW 클래스 객체 비교하기	539
4.1.18.6	LabVIEW 인터페이스	540
4.1.18.7	어플리케이션에서 LabVIEW 클래스 및 인터페이스 사용하기	542
4.1.19	LabVIEW 프런트패널	544
4.1.19.1	프런트패널 컨트롤과 인디케이터	544
4.1.19.2	프런트패널 만들기	545
4.1.19.3	프런트패널 객체 설정하기	546
4.1.19.4	사용자 인터페이스 디자인하기	553
4.1.19.5	프런트패널 객체 컨트롤하기	557
4.1.19.6	프런트패널 데이터 연결의 문제 해결	558

4.1.20 LabVIEW의 변수	559
4.1.20.1 로컬 변수, 글로벌 변수, 피드백 노드	559
4.1.20.2 로컬 변수	560
4.1.20.3 글로벌 변수	560
4.1.20.4 변수 읽기와 쓰기	561
4.1.20.5 로컬과 글로벌 변수를 사용시 유의 사항	562
4.1.20.6 공유 변수 기술 이해하기	564
4.1.20.7 공유 변수로 최근값 공개하기	566
4.1.20.8 실시간 데이터를 프로그램적으로 공유하기	593
4.1.20.9 DataSocket 함수와 배리언트 데이터	596
4.1.20.10 DataSocket 함수를 사용하여 실시간 데이터 읽고 쓰기	597
4.1.20.11 데이터 버퍼링하기	598
4.1.20.12 즉시 네트워크를 통해 공유 변수 데이터 전송하기	601
4.1.20.13 네트워크에서 공유 변수의 이용 가능하게 하기	601
4.1.21 LabVIEW에서 데이터 전송	603
4.1.21.1 데이터 전송하기	603
4.1.21.2 LabVIEW에서의 데이터 통신 방법	603
4.1.21.3 네트워크를 통해 데이터 전송하기	610
4.1.21.4 네트워크 클라이언트 및 서버로서 LabVIEW	612
4.1.21.5 어플리케이션 사이에서 데이터 스트리밍하기 및 명령 전송하기	612
4.1.21.6 네트워크 스트림을 사용해서 어플리케이션 디자인하기	615
4.1.21.7 어플리케이션 사이에서 데이터 스트리밍하기	615
4.1.21.8 어플리케이션 사이에서 명령 전송하기	617
4.1.21.9 끝포인트에 대한 정보 디스플레이하기	621
4.1.21.10 네트워크 스트림 끝포인트	622
4.1.21.11 네트워크 스트림 끝포인트 URL 지정하기	622
4.1.21.12 네트워크 스트림 끝포인트 이름 정리하기	626
4.1.21.13 네트워크 스트림 끝포인트끼리 연결하기	627
4.1.21.14 네트워크 스트림 끝포인트 버퍼에 메모리 할당	631
4.1.21.15 채널 복제하기	632
4.1.21.16 타입 전달	632
4.1.21.17 타입 설명자	633
4.1.22 LabVIEW의 수식과 방정식	646
4.1.22.1 수식과 방정식	646
4.1.23 LabVIEW 어플리케이션과의 통신	652
4.1.23.1 웹 클라이언트에서 LabVIEW 어플리케이션과 통신하기 (Real-Time, Windows)	652
4.1.23.2 개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows)	655

4.1.23.3 웹 서비스 개발하기 (Real-Time, Windows)	657
4.1.23.4 어플리케이션 웹 서버 (Real-Time, Windows)	681
4.1.23.5 어플리케이션 웹 서버 사용 시 HTTP 세션 동안 값 저장하기 (Real-Time, Windows)	682
4.1.23.6 웹 서버에서 프런트패널을 원격으로 보고 컨트롤하기	683
4.1.23.7 TLS/SSL로 웹 서버 보안하기 (Windows)	686
4.1.23.8 LabVIEW를 TCP, UDP 및 기타 하위 레벨 통신 어플 리케이션과 함께 사용하기	689
4.1.23.9 TCP/IP 및 UDP와 함께 LabVIEW 사용하기	690
4.1.23.10 LabVIEW와 OPC 시스템	694
4.1.23.1 채널 와이어를 사용하여 병렬 코드 섹션 사이에서 데이터 통신하기	697
4.1.23.11 참조에 의한 호출 노드와 동적으로 연결하기	703
4.1.23.12 노드 간 사이클 생성하기	704
4.1.23.13 MII Service Locator	705
4.1.23.14 멀티태스킹, 멀티스레딩, 멀티프로세싱	705
4.1.24 LabVIEW의 메모리 관리	724
4.1.24.1 성능과 메모리 관리하기	724
4.1.24.2 성능과 메모리 프로파일 윈도우 사용하기	725
4.1.24.3 코드 복잡도가 컴파일 시간에 미치는 영향	727
4.1.24.4 LabVIEW가 언제 VI를 컴파일하는지 알아내기	727
4.1.24.5 큰 데이터 세트에 대한 메모리 관리	728
4.1.24.6 값 참조를 사용하여 데이터를 저장하고 데이터 복사 줄이기	732
4.1.24.7 LabVIEW가 메모리에 데이터를 저장하는 방법	732
4.1.24.8 패턴화된 데이터	740
4.1.25 LabVIEW의 .NET Framework	743
4.1.25.1 LabVIEW에서 .NET 사용하기	743
4.1.25.2 .NET 환경	744
4.1.25.3 LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건	745
4.1.25.4 .NET 객체 생성하기	746
4.1.25.5 .NET 프로퍼티와 메소드	747
4.1.25.6 .NET 이벤트	747
4.1.25.7 .NET 데이터 타입	748
4.1.25.8 .NET 클라이언트 어플리케이션 설정하기	748
4.1.25.9 LabVIEW에서 .NET 어셈블리 사용하기	749
4.1.25.10 .NET 어셈블리 로드하기	750
4.1.25.11 .NET 클라이언트 어플리케이션으로 .NET 어셈블 리 배포하기	751
4.1.26 LabVIEW의 터치 패널	752
4.1.26.1 Creating a Project Using the Touch Panel Project Template	752
4.1.26.2 Navigation Controls	753

4.1.26.3	Protecting Data Using Write Filters	754
4.1.26.4	Supported Touch Panel Targets	754
4.1.26.5	Using Touch Panel VI Templates	755
5.	LabVIEW 사용법	755
5.1	LabVIEW 사용법	755
5.1.1	LabVIEW 설정하기	755
5.1.1.1	LabVIEW 평가판 및 정품인증 정보 (Windows)	755
5.1.1.2	LabVIEW 사용자 이름 설정하기	757
5.1.1.3	LabVIEW에서 로그인하기 및 로그아웃하기 (Windows)	757
5.1.1.4	키보드 바로 가기 키	758
5.1.1.5	빠른 탐색 바로 가기 키	765
5.1.2	프로젝트 생성하기	766
5.1.2.1	LabVIEW 프로젝트 생성하기	766
5.1.2.2	컨트롤 선택하기	767
5.1.2.3	프로젝트에 폴더 추가하기	767
5.1.2.4	프로젝트에 하이퍼링크 추가하기	769
5.1.2.5	프로젝트에 아이템 추가하기	770
5.1.2.6	프로젝트에 LLB 추가하기	771
5.1.2.7	프로젝트에 타겟 추가하기(Windows)	772
5.1.2.8	프로젝트에서 아이템 찾기	773
5.1.2.9	프로젝트에서 아이템 정렬하기	774
5.1.2.10	프로젝트에서 아이템 삭제하기	775
5.1.2.11	프로젝트에서 의존성 제거하기	775
5.1.2.12	프로젝트 저장하기	776
5.1.2.13	어플리케이션 인스턴스 선택하기	776
5.1.2.14	프로젝트 라이브러리에서 기본 팔레트 파일 설정하기	777
5.1.2.15	프로젝트에서 아이템 보이기와 숨기기	777
5.1.2.16	프로젝트에서 파일 보기	777
5.1.2.17	프로젝트 라이브러리 작업하기	778
5.1.2.18	추가 프로젝트 옵션	785
5.1.3	LabVIEW 사용자 정의하기	787
5.1.3.1	폰트 스타일 설정하기	787
5.1.3.2	사용자 색 선택하기	787
5.1.3.3	사용자 정의 색 설정하기	788
5.1.3.4	색 상자에서 색 설정하기	789
5.1.3.5	LabVIEW 옵션 설정하기	789
5.1.3.6	VI 검색 경로에 디렉토리 추가하기	790
5.1.3.7	기본 메뉴 바로 가기 복구하기	790
5.1.3.8	메뉴 바로 가기 설정하기	791

5.1.3.9	경로 옵션 설정하기	791
5.1.3.10	암호 변경하기(Windows)	792
5.1.4	팔레트 사용자 정의하기	792
5.1.4.1	팔레트 항목 추가하기	792
5.1.4.2	즐거찾기 항목에 팔레트 아이템 추가하기	793
5.1.4.3	사용자와 인스트루먼트 드라이버 서브팔레트에 VI와 컨트롤 추가하기	793
5.1.4.4	함수 또는 컨트롤 팔레트에서 VI나 객체 추가하기	794
5.1.4.5	함수 또는 컨트롤 팔레트에 VI나 객체 추가하기	794
5.1.4.6	팔레트 세트 자동 업데이트하기	796
5.1.4.7	팔레트 포맷 변경하기	796
5.1.4.8	서브팔레트	797
5.1.4.9	팔레트에서 빈 슬롯 삭제하기	799
5.1.4.10	팔레트 세트 편집하기	800
5.1.4.11	팔레트 세트에서 아이콘 짧은 이름 편집하기	800
5.1.4.12	팔레트 세트에서 행이나 열을 삽입하기	801
5.1.4.13	HTML 파일 또는 컴파일된 도움말 파일에 서브팔 레트 링크하기	801
5.1.4.14	팔레트 세트에서 서브팔레트 이동하기	802
5.1.4.15	팔레트 항목 구성하기	802
5.1.4.16	팔레트 세트 공유하기	803
5.1.4.17	팔레트 항목 보이기와 숨기기	803
5.1.4.18	파일 복구에 대한 텍스트 비교 설정하기	803
5.1.5	LabVIEW를 사용한 소스 컨트롤	804
5.1.5.1	소스 컨트롤을 사용하기 위해 LabVIEW 설정하기	804
5.1.5.2	LabVIEW 프로젝트의 소스 컨트롤 설정하기	805
5.1.5.3	소스 컨트롤에서 파일 체크아웃하기	806
5.1.5.4	소스 컨트롤에서 파일 체크아웃 취소하기	807
5.1.5.5	소스 컨트롤에 파일 체크인하기	807
5.1.5.6	소스 컨트롤에서 최신 버전의 파일 얻기	808
5.1.5.7	소스 컨트롤에서 VI 비교하기	809
5.1.5.8	소스 컨트롤에서 개정 히스토리 보기	810
5.1.5.9	소스 컨트롤에 파일 추가하기	810
5.1.5.10	소스 컨트롤에서 파일 삭제하기	811
5.1.5.11	소스 컨트롤에서 파일 이름 바꾸기	811
5.1.5.12	소스 컨트롤 VI를 사용하여 독립 어플리케이션 만들기	812
5.1.6	LabVIEW 클래스	813
5.1.6.1	LabVIEW 클래스 또는 인터페이스 생성하기	813
5.1.6.2	컨트롤의 내용을 LabVIEW 클래스로 변환하기	814
5.1.6.3	LabVIEW 클래스의 프라이빗 데이터 정의하기	814

5.1.6.4	LabVIEW 클래스 또는 인터페이스에서 멤버 VI 생성하기 . . .	815
5.1.6.5	LabVIEW 클래스 또는 인터페이스에서 상속 계층구조 보기 .	821
5.1.6.6	LabVIEW 클래스 또는 인터페이스에서 상속 변경하기	822
5.1.6.7	LabVIEW 클래스 또는 인터페이스에 아이템 추가하기	823
5.1.6.8	LabVIEW 클래스 또는 인터페이스의 와이어 모양 변경하기 .	824
5.1.6.9	다이나믹 디스패치 터미널 생성하기	824
5.1.6.10	다이나믹 디스패치 멤버 VI의 구현 대상 보기 및 열기	825
5.1.6.11	멤버 VI의 접근 영역 설정하기	826
5.1.6.12	LabVIEW 클래스 또는 인터페이스 잠그기 및 잠금 해제하기 .	827
5.1.6.13	LabVIEW 클래스 또는 인터페이스에 기본 프로브 설정하기 .	828
5.1.7	블록다이어그램	829
5.1.7.1	2진 파일로부터 읽기	829
5.1.7.2	2진 파일에 쓰기	829
5.1.7.3	설정 파일 VI 사용하기	830
5.1.7.4	데이터 로깅	831
5.1.7.5	LabVIEW 측정 파일	836
5.1.7.6	LabVIEW 측정 파일로부터 데이터 읽기	837
5.1.7.7	LabVIEW 측정 파일에 데이터 쓰기	837
5.1.7.8	스프레드시트 파일로부터 읽기	838
5.1.7.9	스프레드시트 파일에 쓰기	838
5.1.7.10	스프레드시트 파일에 데이터 스트리밍하기	839
5.1.7.11	TDM/TDMS 파일	841
5.1.7.12	외부 데이터를 TDMS 파일에 스트리밍하기 (Windows)	841
5.1.7.13	텍스트 파일에서 읽기	842
5.1.7.14	텍스트 파일에 쓰기	842
5.1.7.15	LabVIEW 데이터를 XML로 변환하기	843
5.1.7.16	XML을 LabVIEW 데이터로 변환하기	843
5.1.7.17	자동으로 블록다이어그램 정리하기	844
5.1.7.18	블록다이어그램에 컨트롤과 인디케이터 생성하기	845
5.1.7.19	사용자 정의 상수 생성 및 편집하기	846
5.1.7.20	아이콘 또는 데이터 타입에서 터미널 디스플레이하기	847
5.1.7.21	블록다이어그램 객체 제거하고 다시 연결하기	847
5.1.7.22	블록다이어그램 객체 삽입하기	848
5.1.7.23	블록다이어그램 객체 대체하기	848
5.1.7.24	노드 크기 조정하기	849
5.1.7.25	코드의 일부 재사용하기	849
5.1.7.26	다형성 VI의 인스턴스를 수동으로 선택하기	850
5.1.7.27	와이어 사용하기	851
5.1.8	LabVIEW의 검색 옵션	861

5.1.8.1	프론트패널 객체에서 블록다이어그램 터미널 찾기	861
5.1.8.2	블록다이어그램 터미널에서 프론트패널 객체 찾기	861
5.1.8.3	VI의 모든 인스턴스 또는 타입 정의 찾기	861
5.1.8.4	팔레트에서 객체 찾기	862
5.1.8.5	객체 또는 로컬 변수 터미널 찾기	862
5.1.8.6	객체 또는 터미널에 링크된 프로퍼티 노드 찾기	863
5.1.8.7	디렉토리에서 VI 찾기	863
5.1.8.8	VI, 객체, 텍스트 찾고 대체하기	864
5.1.8.9	VI에서 텍스트 찾고 대체하기	864
5.1.8.10	블록다이어그램 객체 찾고 대체하기	865
5.1.9	LabVIEW 기타	866
5.1.9.1	도구 선택하기	866
5.1.9.2	큰 프론트패널과 블록다이어그램 탐색하기	867
5.1.9.3	컨트롤을 인디케이터로 또는 인디케이터를 컨트롤로 바꾸기	867
5.1.9.4	프론트패널 인디케이터 지우기	868
5.1.9.5	바로 가기 메뉴 플러그인	868
5.1.9.6	프론트패널 또는 블록다이어그램에서 공간 추가 또는 축소하기	875
5.1.9.7	한 프론트패널 객체에서 읽고 두 블록다이어그램 위치에 쓰기	876
5.1.9.8	두 블록다이어그램 위치에서 한 프론트패널 객체에 쓰기	876
5.1.10	VI	877
5.1.10.1	VI와 그림, 텍스트 끌어서 놓기	877
5.1.10.2	현재 VI의 스타일 사용자 정의하기	878
5.1.10.3	재귀 VI 생성하기	880
5.1.10.4	VI 계층구조에서 검색하기	881
5.1.10.5	VI의 계층구조 보기	881
5.1.10.6	다형성 VI	883
5.1.10.7	가변 VI 만들기	886
5.1.10.8	가변 VI 디버그하기	887
5.1.10.9	VI 저장하기	887
5.1.10.10	암호로 보호된 VI 생성하기	888
5.1.10.11	VI 잠그기와 잠금해제하기	888
5.1.10.12	일반적인 매스 컴파일 상태 메시지	889
5.1.10.13	LB	890
5.1.10.14	VI 매스 컴파일하기	893
5.1.10.15	VI에서 블록다이어그램 제거하기	894
5.1.10.16	LB에서 파일 제거하기	895
5.1.10.17	VI의 마지막 저장된 버전으로 되돌리기	895

5.1.10.1	LabVIEW에 포함되어 출시되는 VI의 변화 저장하기	896
5.1.10.1	이전 버전으로 저장하기	896
5.1.10.2	개별적인 VI 파일 저장하기	897
5.1.10.2	기존의 LLB에 VI 저장하기	897
5.1.10.2	VI 사용자 정의하기	898
5.1.10.2	VI 윈도우 제목 변경하기	898
5.1.10.2	탐색 경로와 키워드 복사하기	899
5.1.10.2	예제 탐색기에 나타내기 위한 예제 VI 준비하기	899
5.1.10.2	실행 프로퍼티 설정하기	902
5.1.10.2	윈도우 모양과 크기 설정하기	902
5.1.10.2	VI 실행하기	903
5.1.10.2	명령 라인에서 VI 실행하기	903
5.1.10.3	LabVIEW에서 정의된 명령 라인 인수	904
5.1.10.3	개정 히스토리에 주석 추가하기	905
5.1.10.3	VI 개정 히스토리 및 개정 번호 리셋하기	906
5.1.10.3	VI 개정 히스토리 보기	906
5.1.10.3	VI 디버그하기	907
5.1.10.3	메모리 사용	911
5.1.10.3	VI 문자열	916
5.1.10.3	VI 비교 및 병합하기	917
5.1.10.3	VI 매트릭스	927
5.1.10.3	취소 설정하기	929
5.1.10.4	빠른 탐색 대화 상자의 바로 가기 설정하기	930
5.1.10.4	완료되지 않은 태스크 관리하기	931
5.1.10.4	웹플릿과 샘플 프로젝트에서 VI 생성하기	931
5.1.10.4	웹플릿 VI 생성하기	932
5.1.10.4	VI 및 다른 파일 타입에서 컴파일된 코드 분리하기	933
5.1.10.4	비동기적으로 VI 호출하기	934
5.1.10.4	비동기적으로 VI 호출하기 및 결과 수집하기	935
5.1.10.4	결과 수집 없이 비동기적으로 VI 호출하기	937
5.1.10.4	병렬 실행을 위해 VI의 여러 인스턴스를 비동기적으로 호출하기	938
5.1.10.4	옵션 플래그를 단일 옵션값으로 결합하기	940
5.1.10.5	VI 서버 설정하기	942
5.1.10.5	VI 호출 설정하기	942
5.1.10.5	SubVI로부터 프런트패널 객체를 프로그램적으로 컨트롤하기	943
5.1.10.5	다이나믹하게 VI를 호출하는 엄격한 타입의 VI 참조 생성하기	944
5.1.10.5	VI 서버 어플리케이션 생성하기	945

5.1.10.5	프로퍼티 노드와 인보크 노드에서 컨트롤 참조 SubVI 생성하기	946
5.1.10.5	다이나믹하게 VI 호출하기	946
5.1.10.5	어떤 VI를 원격으로 조정할 수 있는지 지정하기	947
5.1.10.5	스크립팅	948
5.1.10.5	새 VI 보이기 엔트리 추가하기	987
5.1.10.6	기존의 VI 보이기 엔트리 편집하기	987
5.1.10.6	보이는 VI 엔트리 삭제하기	988
5.1.10.6	프로그램적으로 모달이 아닌 윈도우 호출하기	988
5.1.10.6	웹 서비스를 VI의 라이브러리로 반입하기	988
5.1.11	객체	989
5.1.11.1	객체 정렬하기	989
5.1.11.2	객체와 배경 색칠하기	990
5.1.11.3	객체 복사하기 및 붙여넣기	991
5.1.11.4	한 객체에서 다른 객체로 색 복사하기	992
5.1.11.5	투명한 객체 생성하기	992
5.1.11.6	숨겨진 프론트패널 객체 디스플레이하기	993
5.1.11.7	객체 간격 조절하기	993
5.1.11.8	객체 그룹화하기 및 잠그기	994
5.1.11.9	객체 순서 재설정하기	995
5.1.11.1	콜립보드로부터 프론트패널 객체 대체하기	996
5.1.11.1	프론트패널 객체 대체하기	996
5.1.11.1	프론트패널 객체 스케일하기	997
5.1.11.1	프론트패널 객체의 탭 순서 설정하기	997
5.1.11.1	스케일을 가지는 숫자형 객체의 값 변경하기	998
5.1.11.1	숫자형 객체에서 기수 디스플레이하기	999
5.1.12	컨트롤과 인디케이터	999
5.1.12.1	컨트롤과 인디케이터 설정하기	999
5.1.12.2	I/O 컨트롤과 인디케이터	1001
5.1.12.3	배리언트 컨트롤과 인디케이터	1002
5.1.12.4	볼리언 컨트롤	1002
5.1.12.5	I/O 컨트롤 설정하기	1002
5.1.12.6	모든 숨겨진 컨트롤과 인디케이터 디스플레이하기	1003
5.1.12.7	컨트롤과 인디케이터의 스크롤 막대 디스플레이하기	1003
5.1.12.8	프론트패널 컨트롤과 인디케이터 숨기기	1004
5.1.12.9	컨트롤과 인디케이터의 부분으로 컨트롤과 인디 케이터 사용자 정의하기	1004
5.1.12.1	프론트패널과 블록 다이어그램에 사용자 컨트롤 과 인디케이터 추가하기	1006
5.1.12.1	숫자형 컨트롤과 인디케이터의 데이터 범위 변경하기	1006

5.1.12.1	슬라이드 컨트롤과 인디케이터의 채우기 옵션 변경하기	1007
5.1.12.1	숫자형 컨트롤과 인디케이터의 포맷과 정밀도 변경하기	1008
5.1.12.1	볼리언 컨트롤과 인디케이터에 그래픽 반입하기	1009
5.1.12.1	프론트패널에서 사용자 컨트롤과 인디케이터 생성하기	1010
5.1.12.1	저장된 사용자 컨트롤과 인디케이터 편집하기	1011
5.1.12.1	사용자 컨트롤과 인디케이터 만들 때의 유의점과 권장사항	1012
5.1.12.1	컨트롤	1013
5.1.12.1	새로 만들기 대화 상자를 이용하여 사용자 컨트롤과 인디케이터 생성하기	1048
5.1.13	배열과 클러스터	1049
5.1.13.1	배열	1049
5.1.13.2	클러스터	1061
5.1.13.3	배열을 클러스터로 변환하기	1066
5.1.13.4	클러스터를 배열로 변환하기	1067
5.1.13.5	배열과 클러스터 움직이기	1067
5.1.13.6	배열이나 클러스터의 원소 사이의 탭 이동하기	1068
5.1.14	행렬	1068
5.1.14.1	행렬	1068
5.1.14.2	행렬 컨트롤, 인디케이터, 상수 생성하기	1069
5.1.14.3	배열을 행렬로 변환하기	1069
5.1.14.4	행렬을 배열로 변환하기	1070
5.1.15	수식 노드, 식 노드 및 스크립트 노드	1070
5.1.15.1	수식 노드와 식 노드 함수	1070
5.1.15.2	수식 노드와 식 노드에서 연산자의 우선 순위	1072
5.1.15.3	수식 노드에서 배열 조작	1073
5.1.15.4	수식 노드 터미널 타입 변경하기	1074
5.1.15.5	수식 노드 생성하기	1074
5.1.15.6	수식 노드 구문	1075
5.1.15.7	수식 노드에서 터미널 삭제하기	1080
5.1.15.8	수식 노드에서 변수를 선언하기 위한 스코프 규칙	1080
5.1.15.9	식 노드 생성하기	1081
5.1.15.10	식 노드 구문	1081
5.1.15.11	스크립트 노드	1082
5.1.15.12	스크립트 막대를 스크립트 노드에 추가하기	1082
5.1.15.13	LabVIEW에서 MATLAB® 소프트웨어 호출하기	1082
5.1.15.14	스크립트 노드에서 스크립트 서버 변경하기	1083
5.1.15.15	스크립트 지우기	1084
5.1.15.16	스크립트 노드 터미널의 데이터 타입 설정하기	1084
5.1.15.17	디버깅 스크립트	1084
5.1.15.18	스크립트 반입 또는 반출하기	1085

5.1.15.1	스크립트 노드의 라인 번호 보이기 및 숨기기	1085
5.1.16	변수	1086
5.1.16.1	변수 읽기와 쓰기	1086
5.1.16.2	로컬 변수	1086
5.1.16.3	글로벌 변수	1088
5.1.16.4	공유 변수	1090
5.1.17	숫자형 데이터	1103
5.1.17.1	숫자형 변환	1103
5.1.17.2	숫자형 변경하기	1104
5.1.17.3	컨트롤 참조 번호의 숫자형 변경하기	1105
5.1.17.4	숫자형 함수의 출력 셋팅 설정하기	1105
5.1.17.5	숫자형 데이터 타입	1106
5.1.17.6	복소수	1107
5.1.17.7	부동소수	1107
5.1.17.8	정수	1108
5.1.17.9	숫자형 데이터 타입 테이블	1108
5.1.17.10	부호없는 정수	1110
5.1.17.11	교정 소수점 데이터	1110
5.1.17.12	배리언트 데이터	1118
5.1.17.13	단위	1125
5.1.18	LabVIEW에서 함수 비교하기	1134
5.1.18.1	배열 비교하기	1134
5.1.18.2	불리언 값 비교하기	1134
5.1.18.3	문자 비교하기	1134
5.1.18.4	클러스터 비교하기	1134
5.1.18.5	숫자값 비교하기	1135
5.1.18.6	문자열 비교하기	1135
5.1.18.7	범위 값과 데이터 강제 변환 결정하기	1135
5.1.18.8	배열에서 범위 값과 데이터 강제 변환 결정하기	1136
5.1.18.9	원소나 집합을 비교하기 위한 비교 함수 설정하기	1137
5.1.19	라벨과 디지털 디스플레이 보이기와 숨기기	1137
5.1.20	구획에서 배경 이미지 반입하기	1137
5.1.21	Linux에서 그래픽 반입하기	1138
5.1.22	macOS에서 그래픽 반입하기	1139
5.1.23	Windows에서 그래픽 반입하기	1140
5.1.24	버튼의 라벨 편집하기	1140
5.1.25	그래프와 차트	1142
5.1.25.1	그래프와 차트 시작하기	1142
5.1.25.2	플롯 범례에 플롯 추가하기	1143

5.1.25.3	플롯 범례에 스크롤 막대 추가하기	1143
5.1.25.4	그래프 또는 차트의 X 스케일과 Y 스케일에 마커 추가하기	1144
5.1.25.5	그래프와 차트에 여러 X 스케일과 Y 스케일 추가하기	1144
5.1.25.6	그래프와 차트 축의 포맷과 정밀도 변경하기	1145
5.1.25.7	그래프와 차트의 데이터 지우기	1145
5.1.25.8	그래프 또는 차트 설정하기	1146
5.1.25.9	그래프 또는 차트에서 X축과 Y축 설정하기	1146
5.1.25.10	플롯 범례를 사용하여 아날로그 플롯 사용자 정의하기	1147
5.1.25.11	그래프나 차트에서 스크롤하기	1148
5.1.25.12	플롯 범례에서 플롯 개수 설정하기	1149
5.1.25.13	그래프 또는 차트의 보기를 줌 또는 이동하기	1149
5.1.25.14	그래프에 주석 추가하기	1150
5.1.25.15	그래프에 커서 추가	1151
5.1.25.16	플롯 영역을 혼합된 신호 그래프에 추가하기	1153
5.1.25.17	디지털 웨이브폼 그래프에서 플롯 범례 보기 변경하기	1153
5.1.25.18	그래프 생성하기	1154
5.1.25.19	플롯 범례를 사용하여 디지털 웨이브폼 그래프에 서 플롯 사용자 정의하기	1155
5.1.25.20	플롯 범례를 사용하여 혼합된 신호 그래프에서 플 롯 사용자 정의하기	1156
5.1.25.21	그래프에서 커서 삭제하기	1158
5.1.25.22	디지털 웨이브폼 그래프에 디지털 라인과 버스 디 스플레이하기	1158
5.1.25.23	XY 그래프에 특별한 평면 디스플레이하기	1159
5.1.25.24	디지털 데이터를 그래프로 나타내기	1163
5.1.25.25	여러 데이터 타입을 혼합된 신호 그래프에서 그래 프로 나타내기	1163
5.1.25.26	웨이브폼 데이터를 그래프로 나타내기	1164
5.1.25.27	XY 데이터를 그래프로 나타내기	1165
5.1.25.28	그래프에서 타임스탬프와 플롯 라벨을 무시하기	1166
5.1.25.29	그래프 좌표를 플롯 이미지 좌표에 맵핑하기	1166
5.1.25.30	플롯 및 버스를 혼합된 신호 그래프의 다른 플롯 영역에 이동하기	1167
5.1.25.31	2D 그래프에 데이터 플롯하기	1168
5.1.25.32	2D 그래프에 데이터 플롯하기	1168
5.1.25.33	3D 표면 그래프에 데이터 플롯하기	1169
5.1.25.34	ActiveX 3D 표면 그래프에 X, Y, Z 데이터 플롯하기	1170
5.1.25.35	3D 헤성형 그래프에서 애니메이션 다시 실행하기	1171
5.1.25.36	웨이브폼 데이터 차트에 그리기	1171
5.1.25.37	차트 생성하기	1172
5.1.26	웨이브폼의 숫자형 변경하기	1173

5.1.27	신호 수집 또는 시뮬레이션	1173
5.1.28	신호 분석하기	1174
5.1.29	신호 디스플레이하기	1174
5.1.30	신호 필터링하기	1175
5.1.31	신호 병합하기	1175
5.1.32	신호 분리하기	1175
5.1.33	인스트루먼트 드라이버 변경 설명	1176
5.1.34	인스트루먼트 드라이버 프로젝트 생성하기	1180
5.1.35	인스트루먼트 드라이버 VI 생성하기	1182
5.1.36	2D 그림 컨트롤에서 보이지 않는 픽셀 디스플레이하기	1184
5.1.37	2D 그림 컨트롤 리셋하기	1184
5.1.38	사운드 입력과 사운드 출력 디바이스 사용하기	1185
5.1.39	3D 그림 컨트롤로 3D 장면 생성하기	1185
5.1.40	3D 장면에서 카메라 설정하기	1187
5.1.41	3D 장면에서 빛 소스 설정하기	1188
5.1.42	3D 객체의 표면 모양 바꾸기	1189
5.1.43	3D 객체간 관계 생성하기	1192
5.1.44	3D 장면에 텍스트 추가하기	1194
5.1.45	3D 장면 윈도우 설정하기	1195
5.1.46	불리언 객체의 기계적 동작 변경하기	1197
5.1.47	혼합 확인란 설정하기	1198
5.1.48	라디오 버튼 컨트롤 생성하기	1198
5.1.49	불리언 컨트롤과 인디케이터 생성하기와 작동하기	1199
5.1.50	디지털 테이블 만들기	1200
5.1.51	리스트박스, 트리 컨트롤, 테이블	1201
5.1.51.1	리스트박스, 트리 컨트롤, 테이블	1201
5.1.51.2	트리 컨트롤 생성하기	1201
5.1.51.3	리스트박스 생성하기	1202
5.1.51.4	여러 열 리스트박스 생성하기	1203
5.1.51.5	리스트박스와 트리 컨트롤에서 행 삭제하기	1204
5.1.51.6	리스트박스와 트리 컨트롤 아이템 활성화 또는 비 활성화하기	1204
5.1.51.7	리스트박스, 테이블, 트리 컨트롤, 디지털 데이터 컨트롤에 구성요소 디스플레이하기	1204
5.1.51.8	테이블의 데이터 편집하기	1205
5.1.51.9	실행 중에 리스트박스와 트리 컨트롤 셀 편집하기	1206
5.1.51.10	테이블에 데이터 입력하기	1206
5.1.51.11	트리 컨트롤 아이템을 한 단계 아래로 보내기	1207
5.1.51.12	리스트박스와 트리 컨트롤에 아이템 삽입하기	1207

5.1.51.1	리스트박스과 여러 열 리스트박스에 행 또는 열을 삽입 및 삭제하기	1208
5.1.51.1	테이블에서 데이터의 행이나 열을 삽입하거나 삭제하기	1208
5.1.51.1	트리 컨트롤 아이템 선택을 등위로 제한하기	1208
5.1.51.1	트리 컨트롤 아이템 열기 및 닫기	1209
5.1.51.1	리스트박스, 테이블, 트리 컨트롤의 행과 열의 크기 조정하기	1209
5.1.51.1	원타임으로 리스트박스와 트리 컨트롤 아이템 선택하기	1210
5.1.51.1	트리 컨트롤에서 직상위 아이템 선택하기	1210
5.1.51.2	테이블의 셀, 행, 열 선택하기	1211
5.1.51.2	리스트박스와 트리 컨트롤에서 대소문자 구분 설정하기	1211
5.1.51.2	원타임에 열 폭 설정하기	1212
5.1.51.2	트리 컨트롤에서 확장/축소 기호 설정하기	1212
5.1.51.2	원타임에서 리스트박스와 트리 컨트롤 안에서 선택할 수 있는 아이템의 개수 설정하기	1212
5.1.51.2	테이블 사용하기	1213
5.1.51.2	리스트박스, 테이블, 트리 컨트롤에 헤더 텍스트 추가하기	1213
5.1.51.2	리스트박스와 트리 컨트롤에 기호 추가하기	1214
5.1.51.2	리스트박스, 여러 열 리스트박스, 트리 컨트롤, 테이블 색칠하기	1214
5.1.51.2	여러 라인 입력을 위해서 셀 설정하기	1215
5.1.52	구조	1215
5.1.52.1	블록다이어그램에 구조 추가하기	1215
5.1.52.2	구조 크기 조정하기	1216
5.1.52.3	구조 안에 객체 추가하기	1216
5.1.52.4	구조를 다른 구조로 대체하기	1217
5.1.52.5	구조에서 객체를 삭제하지 않고 구조를 삭제하기	1217
5.1.53	루프	1218
5.1.53.1	루프에서 나가는 배열을 자동으로 연결하기	1218
5.1.53.2	루프를 사용하여 배열이나 집합의 개별 원소 처리하기	1218
5.1.53.3	루프 출력 터널에 조건적으로 값 연결하기	1223
5.1.53.4	각 반복에서 불리언 값 확인하기	1223
5.1.53.5	루프의 연속 동작 변경하기	1224
5.1.53.6	루프의 타이밍 컨트롤하기	1225
5.1.53.7	루프의 조건 터미널로 여러 핸들링하기	1225
5.1.53.8	For 루프의 작업 실행 횟수 지정하기	1225
5.1.53.9	조건이 충족될 때 For 루프 정지하기	1226
5.1.53.1	병렬 반복: For 루프 실행 속도 향상	1227
5.1.53.1	병렬 For 루프 반복 활성화하기	1228
5.1.53.1	For 루프가 병렬 반복으로 실행될 수 있는지 확인하기	1228
5.1.53.1	병렬 For 루프 반복을 설정하여 성능 변경하기	1229

5.1.53.1	병렬 For 루프 반복의 성능 고려사항	1230
5.1.53.1	While 루프의 조건 지정하기	1231
5.1.53.1	처음 While 루프 반복에서 코드 실행 막기	1231
5.1.53.1	예제: While 루프에서 알림 동작 함수 처리하기	1232
5.1.53.1	다음 루프 반복으로 값 전달하기	1233
5.1.53.1	다음 루프 반복으로 여러 값 전달하기	1234
5.1.53.2	시프트 레지스터의 초기값 리셋하기	1235
5.1.53.2	시프트 레지스터를 터널로 대체하기	1236
5.1.53.2	최종 값을 이전 실행에서 피드백 노드로 전달하기	1237
5.1.53.2	루프 실행 시마다 피드백 노드의 입력값 리셋하기	1238
5.1.53.2	시프트 레지스터를 피드백 노드로 대체하기	1239
5.1.53.2	피드백 노드 화살표 방향 변경하기	1240
5.1.54	서브다이어그램과 케이스 구조	1241
5.1.54.1	서브다이어그램 추가 및 복제하기	1241
5.1.54.2	서브다이어그램 사이에서 움직이기	1242
5.1.54.3	서브다이어그램 삭제하기	1243
5.1.54.4	서브다이어그램 재배치하기	1243
5.1.54.5	조건 코드 실행을 위한 케이스 구조 생성하기	1244
5.1.54.6	각 케이스 구조 서브다이어그램의 값 입력하기	1245
5.1.54.7	케이스 구조에서 연결되지 않은 출력 터널의 연결 피하기	1246
5.1.54.8	케이스 구조로 에러 핸들링하기	1246
5.1.54.9	케이스 구조 서브다이어그램 사이에 케이스 추가하기	1247
5.1.54.10	케이스 구조의 기본 서브다이어그램 지정하기	1248
5.1.54.1	케이스 구조 서브다이어그램 교환하기	1248
5.1.54.1	케이스 구조 서브다이어그램 이동하기	1249
5.1.54.1	플랫 시퀀스 구조의 프레임을 통해 데이터 전달하기	1250
5.1.54.1	다층 시퀀스 구조의 프레임을 통해 데이터 전달하기	1250
5.1.54.1	시퀀스 구조를 언제 사용할지 결정하기	1251
5.1.54.1	조건적 비활성화 구조의 조건 설정하기	1252
5.1.54.1	조건적 비활성화 구조의 조건 사용자 정의하기	1254
5.1.54.1	On Place 원소 구조로 데이터 복사본 개수 줄이기	1255
5.1.54.1	On Place 원소 구조 사용의 유의점과 권장사항	1255
5.1.54.2	Timed 구조 프레임 추가하고 삭제하기 (Real-Time, Windows)	1256
5.1.54.2	Timed 구조 프레임 병합하기 (Real-Time, Windows)	1256
5.1.54.2	Timed 구조 노드의 터미널 디스플레이하고 숨기기 (Real-Time, Windows)	1257
5.1.55	이벤트	1257
5.1.55.1	이벤트 케이스 추가 및 복제하기	1257
5.1.55.2	이벤트 구조에서 처리되는 이벤트 설정하기	1258
5.1.55.3	사용자 이벤트 생성하기	1259

5.1.55.4	다이나믹하게 이벤트 등록하기	1262
5.1.55.5	다이나믹하게 이벤트 등록 수정하기	1263
5.1.55.6	단계 1: 특정 상황에서만 발생하기를 원하는 이벤 트에 자리 표시자를 생성합니다	1264
5.1.55.7	단계 2: LabVIEW가 언제 해당 이벤트를 등록하 기를 원하는지 지정하는 이벤트 구조를 설계합니다	1266
5.1.55.8	단계 3: 해당 이벤트의 등록을 먼저 해제한 후, 모 든 이벤트의 등록을 해제합니다	1267
5.1.55.9	이벤트 케이스와 관련된 컨트롤 찾기	1268
5.1.55.10	프론트패널 잠그기	1269
5.1.55.11	실행 중 큐에 추가된 이벤트 보기	1270
5.1.55.12	LabVIEW에서 이벤트를 사용할 때의 유의점과 권장사항	1270
5.1.55.13	하나의 루프에 두 개의 이벤트 구조를 사용하는 것을 피하기	1272
5.1.55.14	이벤트가 발생할 때마다 이벤트 구조가 이벤트를 처리할 수 있도록 하기	1272
5.1.55.15	이벤트 구조에서 래치된 정지 불리언 컨트롤 핸들링	1274
5.1.55.16	한 케이스가 여러 필터 이벤트 처리하기	1275
5.1.55.17	한 케이스가 여러 알림 이벤트 처리하기	1276
5.1.55.18	마우스 다운? 이벤트 케이스에서 대화 상자를 가 지고 사용자에게 입력요청하기	1276
5.1.55.19	사용자가 직접 상호 작용할 때에만 이벤트 사용하기	1276
5.1.55.20	가능하면 하나의 이벤트 구조만 사용하기	1276
5.1.55.21	이벤트 사용 또는 프론트패널 작업 기다림 함수 사용	1277
5.1.55.22	래치된 불리언 컨트롤을 이용한 이벤트 사용하기	1277
5.1.55.23	서브패널 컨트롤과 함께 이벤트 사용하기	1277
5.1.56	문자열	1278
5.1.56.1	문자열	1278
5.1.56.2	숫자를 문자열로 변환하기	1278
5.1.56.3	문자열 포맷하기	1279
5.1.56.4	문자열에 디스플레이할 수 없는 문자를 삽입하기	1279
5.1.56.5	문자열의 지정된 포인트에서 텍스트 대체하기	1280
5.1.56.6	문자열에서 텍스트를 검색하고 대체하기	1280
5.1.56.7	패턴 일치를 위한 특수 문자	1281
5.1.56.8	정규식 일치 및 문자열 검색과 대체를 위한 특수 문자	1283
5.1.56.9	문자열 분리하기	1296
5.1.56.10	숫자형 문자열에서 포맷 지정자 사용하기	1297
5.1.56.11	문자열에 캐리지 리턴 입력하기	1297
5.1.56.12	문자열에 라인 피드 입력하기	1298
5.1.56.13	문자열에 탭 문자 입력하기	1298
5.1.56.14	문자열 컨트롤에 텍스트를 입력하거나 변경하기	1298

5.1.56.1	원타임으로 콤보 박스 컨트롤에 문자열을 선택하고 입력하기	1299
5.1.56.1	콤보 박스 문자열의 사용자 값을 지정하기	1299
5.1.57	색 램프를 다이얼, 노브, 게이지에 추가하기	1300
5.1.58	슬라이더 또는 지침 추가하기	1300
5.1.59	숫자형 객체에 스케일을 갖는 텍스트 라벨 추가하기	1301
5.1.60	스케일 리미트 변경하기	1302
5.1.61	스크롤 막대 컨트롤의 스크롤 범위 변경하기	1302
5.1.62	숫자형 컨트롤과 인디케이터가 허용하는 문자	1303
5.1.63	타임스탬프 컨트롤에서 시간과 날짜를 설정하기	1304
5.1.64	타임스탬프 컨트롤에서 높은 소수점 자릿수 디스플레이하기	1305
5.1.65	스케일에 임의의 마커 놓기	1306
5.1.66	슬라이드와 회전식 컨트롤 및 인디케이터를 스케일하기	1306
5.1.67	스크롤 막대 컨트롤 사용하기	1307
5.1.68	증가와 감소 버튼 사용하기	1308
5.1.69	숫자형 데이터 타입에 대한 유효한 포맷 지정자	1308
5.1.70	빈 경로	1309
5.1.71	기호형 경로	1310
5.1.72	옵션 디스플레이	1311
5.1.72.1	백슬래시("\") 코드 디스플레이	1311
5.1.72.2	문자열 컨트롤과 인디케이터 설정하기	1313
5.1.72.3	자동 줄 바꿈 활성화	1314
5.1.72.4	16진수 디스플레이	1314
5.1.72.5	한 줄로 제한	1314
5.1.72.6	일반 디스플레이	1315
5.1.72.7	암호 디스플레이	1315
5.1.72.8	입력할 때 값 업데이트	1315
5.1.72.9	라벨 붙이기	1315
5.1.72.10	미리 지정된 위치로 라벨과 캡션 정렬하기	1316
5.1.72.11	프로그램적으로 캡션 변경하기	1317
5.1.72.12	텍스트 특성 변경하기	1317
5.1.72.13	캡션 생성하기	1318
5.1.72.14	독립 라벨 생성하기	1319
5.1.72.15	라벨 편집하기	1320
5.1.72.16	객체에 라벨 붙이기	1321
5.1.72.17	라벨 및 캡션 고정하기	1321
5.1.72.18	모든 라벨과 캡션 이동하기	1322
5.1.72.19	텍스트 라벨 회전하기	1322
5.1.72.20	대화 상자 생성하기	1323
5.1.72.21	정렬 눈금을 사용하여 객체 정렬하기	1324

5.1.72.2	분리자 막대와 구획 생성하기	1324
5.1.72.2	상태 막대 만들기	1325
5.1.72.2	도구 모음 만들기	1326
5.1.72.2	분리자 막대 설정하기	1326
5.1.72.2	구획 설정하기	1326
5.1.72.2	편집 모드로 런타임 프런트패널 경계 보기	1327
5.1.72.2	웹플릿 컨트롤 생성하기	1327
5.1.72.2	최소 윈도우 크기 정의하기	1328
5.1.72.3	객체 삭제하기	1328
5.1.72.3	화면 해상도에 따라 윈도우 비율 유지하기	1329
5.1.72.3	객체 움직이기	1329
5.1.72.3	화벨 크기 조정하기	1330
5.1.72.3	여러 객체를 같은 크기로 크기 조정하기	1330
5.1.72.3	객체 크기 조정하기	1331
5.1.72.3	객체 선택하기	1332
5.1.72.3	하나의 프런트패널 객체를 스케일에 맞게 설정하기	1333
5.1.72.3	프런트패널의 모든 객체를 스케일에 맞게 설정하기	1333
5.1.72.3	단일 프런트패널 객체를 구획에 맞게 피팅하기	1334
5.1.72.4	프런트패널에서 장식 사용하기	1334
5.1.73	타입 정의와 엄격한 타입 정의 생성하기	1334
5.1.74	저장된 타입 정의와 저장된 엄격한 타입 정의 편집하기	1335
5.1.75	수동으로 타입 정의 인스턴스 업데이트하기	1335
5.1.76	타입 정의 또는 엄격한 타입 정의의 링크 재설정하기	1336
5.1.77	SubVI와 익스프레스 VI	1337
5.1.77.1	SubVI와 익스프레스 VI를 아이콘 또는 확장 가능 노드로 디스플레이하기	1337
5.1.77.2	입력과 출력을 삽입하고 제거하기	1337
5.1.77.3	익스프레스 VI	1338
5.1.77.4	SubVI	1347
5.1.78	터미널	1350
5.1.78.1	컨트롤과 인디케이터에 터미널 지정하기	1350
5.1.78.2	터미널 연결 확인하기	1351
5.1.78.3	커넥터 팬에서 터미널 연결 삭제하기	1351
5.1.78.4	실행 중 SubVI 프런트패널 열기	1351
5.1.78.5	SubVI의 현재 인스턴스 결정하기	1352
5.1.79	커넥터 팬 패턴을 선택하기	1353
5.1.80	필수, 권장, 옵션 입력과 출력 설정하기	1353
5.1.81	아이콘	1354
5.1.81.1	아이콘 생성하기	1354
5.1.81.2	라이브러리 아이콘 생성하기	1354

5.1.81.3	VI 아이콘 생성하기	1357
5.1.81.4	사용자 모양 아이콘 생성하기	1358
5.1.81.5	수동으로 아이콘 라이브러리와 동기화하기	1359
5.1.82	메뉴	1359
5.1.82.1	메뉴 사용자 정의하기	1359
5.1.82.2	기존 메뉴 결합하기	1360
5.1.82.3	메뉴를 프로그램적으로 사용자 정의하기	1360
5.1.82.4	정적으로 메뉴 사용자 정의하기	1362
5.1.82.5	메뉴 선택 처리하기	1363
5.1.82.6	메뉴 모음 보이기와 숨기기	1364
5.1.82.7	어플리케이션 아이템 보기	1364
5.1.82.8	런타임 바로 가기 메뉴 사용자 정의하기	1365
5.1.82.9	기존 런타임 바로 가기 메뉴 결합하기	1365
5.1.82.10	프론트패널의 런타임 바로 가기 메뉴를 프로그램적으로 사용자 정의하기	1366
5.1.82.11	컨트롤의 런타임 바로 가기 메뉴를 프로그램적으로 사용자 정의하기	1366
5.1.82.12	컨트롤의 런타임 바로 가기 메뉴를 정적으로 사용자 정의하기	1367
5.1.82.13	런타임 바로 가기 메뉴 비활성화하기	1368
5.1.82.14	런타임 바로 가기 메뉴 선택 처리하기	1368
5.1.83	문제 해결	1369
5.1.83.1	프로브	1369
5.1.83.2	브레이크포인트 관리하기	1379
5.1.83.3	에러 알림 컨트롤하기	1380
5.1.83.4	사용자 에러 코드 정의하기	1381
5.1.83.5	에러 링에서 사용자 에러 코드 정의하기	1381
5.1.83.6	어플리케이션을 통해 배포할 사용자 에러 코드 정의하기	1382
5.1.83.7	텍스트 파일에서 사용자 에러 코드 배포하기	1383
5.1.83.8	[일반 에러 핸들러] VI를 사용하여 사용자 에러 코드 정의하기	1384
5.1.83.9	에러 코드 설명 찾기	1385
5.1.83.10	에러 클러스터 리포트 생성하기	1386
5.1.83.11	케이스 구조로 에러 핸들링하기	1386
5.1.83.12	류프의 조건 터미널로 에러 핸들링하기	1387
5.1.83.13	에러 링에서 내장 에러 선택하기	1387
5.1.83.14	VI에서 에러 I/O 설정하기	1388
5.1.83.15	에러 클러스터 사용하기	1389
5.1.83.16	어플리케이션과 공유 라이브러리 디버그하기	1389
5.1.83.17	빌드 과정 문제 해결하기	1391
5.1.84	LabVIEW에서 문서화하기	1391

5.1.84.1	VI 문서화하기	1391
5.1.84.2	북마크 관리자 생성하기	1392
5.1.84.3	설명 생성하기와 편집하기	1392
5.1.84.4	객체 설명과 팁 상자 생성하기	1393
5.1.84.5	기본 도움말 윈도우의 텍스트 포맷하기	1394
5.1.84.6	VI, 템플릿, 객체를 위한 HTML 문서 생성하기	1394
5.1.84.7	VI, 템플릿, 객체를 위한 RTF 문서 생성하기	1395
5.1.84.8	VI, 템플릿, 객체를 위한 텍스트 문서 생성하기	1395
5.1.84.9	컴파일된 도움말 파일 생성하기	1396
5.1.84.10	VI를 사용자 문서에 링크하기	1397
5.1.84.11	LabVIEW가 생성한 이미지 파일의 명명 규칙	1398
5.1.84.12	VI, 템플릿, 객체를 위한 문서 인쇄하기	1399
5.1.84.13	여러 어플리케이션 인스턴스에서 VI를 위한 문서 인쇄하기	1400
5.1.84.14	HTML 문서 생성하기 및 프런트패널 이미지 내장하기	1400
5.1.84.15	기존의 HTML 문서에 프런트패널 이미지 내장하기	1402
5.1.84.16	기존의 HTML 문서에 VI 내장하기	1403
5.1.85	LabVIEW에서 인쇄하기	1404
5.1.85.1	VI 인쇄물 향상시키기	1404
5.1.85.2	독립적인 프런트패널 객체의 이미지 생성하기	1405
5.1.85.3	실행 후 VI의 프런트패널 인쇄하기	1406
5.1.85.4	프로그램적으로 VI 인쇄하기	1407
5.1.85.5	프린터 에러 문제 해결하기	1407
5.1.85.6	SubVI를 사용하여 상위 레벨 VI의 데이터 인쇄하기	1408
5.1.86	LabVIEW의 리포트	1409
5.1.86.1	리포트 생성하기	1409
5.1.86.2	리포트 생성을 위한 데이터 포매팅	1409
5.1.86.3	리포트에 토큰 사용하기	1409
5.1.87	새 VI 반출 리스트 엔트리 추가하기	1410
5.1.88	기존 VI 반출 리스트 엔트리 편집하기	1410
5.1.89	VI 반출 리스트에서 VI 삭제하기	1410
5.1.90	새 머신 접근 리스트 엔트리 추가하기	1411
5.1.91	기존의 머신 접근 리스트 엔트리 편집하기	1411
5.1.92	머신 접근 리스트 엔트리 제거하기	1412
5.1.93	VI 서버를 위한 머신 접근 설정하기	1412
5.1.94	브라우저 접근 리스트 엔트리 제거하기	1412
5.1.95	LabVIEW에서 공유 라이브러리 호출하기	1412
5.1.96	함수 원형 만들기	1415
5.1.97	.c 파일 완성하기	1417
5.1.98	공유 라이브러리 파일에서 함수 반입하기	1418

5.1.99	예제: 공유 라이브러리 파일에서 함수 반입하기	1421
5.1.100	공유 라이브러리 반입 마법사의 문제 해결하기	1424
5.1.101	패턴화된 문자열로 함수 사용하기	1425
5.1.102	점으로 표시된 프로퍼티와 메소드 생성하기	1426
5.1.103	프로퍼티와 메소드 생성하기	1427
5.1.104	클래스 탐색기 윈도우로 프로퍼티와 메소드 생성하기	1429
5.1.105	.NET와 ActiveX 이벤트 등록 및 핸들링하기	1430
5.1.106	.NET 컨테이너에서 .NET 컨트롤 생성하기	1431
5.1.107	블록 다이어그램에서 .NET 객체 생성하기	1432
5.1.108	정적 .NET 프로퍼티와 메소드 사용하기	1434
5.1.109	LabVIEW에서 .NET 2.0, 3.0, 3.5 어셈블리 로드하기	1435
5.1.110	메모리의 .NET 어셈블리 변경 사항 로드하기	1436
5.1.111	어떤 .NET 어셈블리 버전을 사용할지 지정하기	1436
5.1.112	.NET 어셈블리 호출 디버그하기	1437
5.1.113	ActiveX	1438
5.1.113.1	ActiveX를 지원하는 어플리케이션에 접근하기	1438
5.1.113.2	ActiveX 프로퍼티 접근하기	1439
5.1.113.3	ActiveX 서버로 LabVIEW 활성화하기	1439
5.1.113.4	ActiveX 컨테이너 개요	1440
5.1.113.5	LabVIEW에서 ActiveX 객체 호출하기	1440
5.1.113.6	COM/오토메이션과 LabVIEW 데이터 타입	1441
5.1.113.7	ActiveX 컨테이너에 새 컨트롤 생성하기	1442
5.1.113.8	ActiveX 컨테이너에 새 문서 생성하기	1442
5.1.113.9	ActiveX 문서 편집하기	1443
5.1.113.10	컨트롤패널에 ActiveX 객체 삽입하기	1443
5.1.113.11	ActiveX 컨테이너에 기존의 컨트롤이나 문서 삽입하기	1443
5.1.113.12	ActiveX 클래스 선택하기	1444
5.1.113.13	ActiveX 프로퍼티 페이지 사용하기	1444
5.1.113.14	ActiveX 컨트롤 프로퍼티 브라우저 사용하기	1445
5.1.114	기존의 프로퍼티와 인보크 노드 편집하기	1445
5.1.115	프로퍼티 노드와 인보크 노드를 터미널에 링크하기	1446
5.1.116	LabVIEW에서의 프로그램 연결	1446
5.1.116.1	웹 브라우저에서 원격 디바이스 모니터 및 설정하기	1446
5.1.116.2	어플리케이션 개발하기 및 배포하기	1448
5.1.116.3	macOS에서 Universal Framework 만들기	1454
5.1.116.4	배포용 패키지 생성하기	1455
5.1.116.5	어플리케이션 및 공유 라이브러리에서 런타임 언어 설정하기	1456
5.1.116.6	LabVIEW 작업 환경 옵션 복사하기	1457
5.1.116.7	독립 어플리케이션의 정보 대화 상자 생성하기	1458

5.1.116. 독립 어플리케이션에서 웹 서버 활성화하기	1459
5.1.116. 프로그램적으로 어플리케이션 및 공유 라이브러리 버전 검색하기(Windows)	1459
5.1.116. 독립 어플리케이션에서 설정 파일 사용하기	1460
5.1.116. LabVIEW로 만든 .NET Interop 어셈블리의 특징	1461
5.1.116. LabVIEW로 만든 공유 라이브러리의 특징	1463
5.1.116. 설치 프로그램 배포 구성요소 캐시하기(Windows)	1464
5.1.116. 설치 프로그램 만들기에 대한 유의점과 권장사항	1464
5.1.116. 설치 프로그램 빌드에 추가적인 National Instrument 설치 프로그램 포함하기	1466
5.1.116. LabVIEW로 만든 설치 프로그램 내부적으로 설치하기	1468
5.1.116. LabVIEW로 만든 설치 프로그램 설치 제거하기	1468
5.1.116. IBM Excel Add-In 사용하기 (Windows)	1469
5.1.116. 어플리케이션 매니페스트 작업하기	1469
5.1.116. LabVIEW의 명령 라인 인터페이스를 사용하여 작업 실행하기	1470
5.1.116. 사용자 정의 명령 라인 작업 생성하기	1472
5.1.116. RTCP 기반 NI-PSP의 기본 포트 변경하기 (NI Linux Real-Time, Windows)	1475
5.1.116. JSONP 지원	1476
5.1.116. DataSocket 함수로 버퍼링 사용하기	1476
5.1.116. 프로그램적으로 데이터 연결 열기와 닫기	1478
5.1.116. 실시간 데이터를 프로그램적으로 쓰기	1478
5.1.116. DataSocket 함수를 사용하여 버퍼 크기 지정하기	1479
5.1.116. 버퍼가 쓴 버퍼된 데이터를 읽기	1479
5.1.116. 데이터 연결의 버퍼 크기 모니터하기	1480
5.1.116. 프로그램적으로 실시간 데이터 읽기	1481
5.1.116. dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 원격으로 프론트패널 컨트롤 컨트롤하기	1481
5.1.116. 컨트롤의 dstp, opc, ftp, http, 또는 file 연결 활성화하기	1483
5.1.116. dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 프론트패널 데이터 읽기	1484
5.1.116. dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 프론트패널 데이터 쓰기	1485
5.1.116. 웹 서비스	1486
5.1.116. 웹 브라우저 접근 리스트 엔트리 추가하기	1518
5.1.116. 웹 브라우저 접근 허용 및 거부하기	1518
5.1.116. 웹 브라우저 접근 리스트 엔트리 편집하기	1519
5.1.116. 프론트패널	1519
5.1.116. 웹에 VI와 독립 어플리케이션 공개하기	1524
5.1.116. VI에서 E-메일 보내기	1525
5.1.116. 클라이언트 컨트롤에서 시간 제한 설정하기	1526

5.1.116.43	어떤 VI가 웹에서 보이는지 지정하기	1527
5.1.116.44	라우저 주소 리스트를 위한 구문	1527
5.1.116.45	의 구문 열거하기	1528
5.1.116.46	라우저를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기	1530
5.1.116.47	LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기	1531
5.1.116.48	웹 서버 구문 예제	1532
5.1.116.49	TCP 클라이언트 생성하기	1533
5.1.116.50	TCP 서버 생성하기	1533
5.1.116.51	표메인	1534
6.	LabVIEW 측정 개념	1541
6.1	Overview of Virtual Instruments and Measurements	1541
6.2	Controlling Instruments	1542
6.3	인스트루먼트 및 DAQ 디바이스	1543
6.3.1	General-Purpose DAQ	1543
6.3.2	Instrument I/O	1544
6.3.3	웨이브폼 컨트롤과 디지털 웨이브폼 컨트롤	1545
6.3.4	Data and String Manipulation Techniques	1547
6.3.5	Comparing DAQ Devices and Computer-Based Instruments for Data Acquisition	1549
6.3.6	Configuring DAQ Devices, Instruments, and Other Devices	1550
6.4	인스트루먼트 타입	1551
6.4.1	Types of Instruments	1551
6.4.2	GPIB Communication	1552
6.4.3	Modular Instruments	1552
6.4.4	Serial Port Communication	1553
6.4.5	Ethernet Communication	1554
6.4.6	USB Communication	1555
6.4.7	기타 통신 타입	1555
6.5	인스트루먼트 드라이버	1556
6.5.1	Using Instrument Drivers	1556
6.5.2	Types of Instrument Drivers	1557
6.5.3	Finding and Installing Instrument Drivers	1557
6.5.4	Accessing Instrument Driver VIs	1558
6.5.5	Getting Started with Instrument Drivers	1559
6.5.6	Interactively Testing Instrument Driver VIs	1560
6.5.7	Creating Instrument Control Applications	1560
6.5.8	Modifying an Existing Instrument Driver	1561
6.5.9	Creating Instrument Drivers	1562

6.5.10	Instrument Control Error Handling	1562
6.6	측정 어플리케이션	1563
6.6.1	Creating a Typical DAQ Application	1563
6.6.2	Creating a Typical FieldPoint Application	1564
6.6.3	Using Properties with Measurement Applications	1565
6.7	LabVIEW의 VISA	1565
6.7.1	Using VISA in LabVIEW	1565
6.7.2	Selecting a Resource Name	1566
6.7.3	Verifying VISA Communication	1567
6.7.4	Creating a Typical VISA Application	1567
7.	LabVIEW Shared Libraries	1568
7.1	Calling Shared Libraries	1568
7.1.1	Configuring the Call Library Function Node	1569
7.1.2	Specifying the Location of Shared Libraries on Disk	1580
7.1.3	Calling External APIs	1582
7.1.4	Example 1: Call a Shared Library That You Built	1584
7.1.5	Example 2: Call a Hardware Driver API	1587
7.1.6	Example 3: Call the Win32 API	1588
7.1.7	Unicode Versions and ANSI Versions of Functions	1592
7.1.8	Additional Windows Examples of LabVIEW Calls to DLLs	1594
7.2	Debugging Shared Libraries and Calls to Shared Libraries	1594
7.2.1	Module Definition Files	1598
7.2.2	Using Arrays and Strings in the Call Library Function Node	1599
7.3	Importing Shared Libraries	1603
7.3.1	Advantages of the Import Shared Library Wizard Compared to the Call Library Function Node	1604
7.3.2	Supported Data Types for the Import Shared Library Wizard	1605
7.3.3	Using the Import Shared Library Wizard	1607
7.4	LabVIEW Manager Functions in Shared Libraries	1608
7.4.1	Using LabVIEW Manager Functions in Shared Libraries	1608
7.4.2	LabVIEW Manager Data Types	1611
7.4.3	LabVIEW Manager Function Errors	1616
7.4.4	Memory Manager	1620
7.4.5	File Manager	1624
7.4.6	Support Manager	1628
8.	Signal Processing and Analysis	1629
8.1	Signal Generation in LabVIEW	1629
8.2	IIR Filter Characteristics in LabVIEW	1630
8.3	Curve Fitting in LabVIEW	1630
8.4	Spline Interpolation	1631

8.5	Basic Linear Algebra Subroutines	1632
8.5.1	Introduction to Basic Linear Algebra Subroutines	1632
8.5.2	Choosing Between BLAS VIs and Linear Algebra VIs	1632
8.6	Solving ODEs in LabVIEW	1634
8.7	Solving PDEs in LabVIEW	1634
8.8	Point-By-Point Analysis	1636
8.8.1	Point-by-Point Analysis and Array-Based Analysis in	1636
8.8.2	Performing Analysis without Buffers of Data	1638
8.8.3	Case Study of Point-by-Point Analysis	1638
8.8.3.1	Case Study of Point-By-Point Analysis Application Built in LabVIEW	1638
8.8.3.2	Point-By-Point Analysis of Train Wheels	1639
8.8.3.3	Overview of the LabVIEW Point-By-Point Solution	1639
8.8.3.4	Characteristics of a Train Wheel Waveform	1640
8.8.3.5	Analysis Stages of the Train Wheel	1640
9.	스캔 I/O 데이터 접근하기	1643
9.1	I/O 변수 사용하기 (Real-Time, Windows)	1643
9.2	스캔 I/O 데이터 접근하기 (Real-Time, Windows)	1653
9.3	NI 스캔 엔진 결합 (Real-Time, Windows)	1656
9.4	I/O 변수 컨테이너를 배열로 읽고 쓰기 (Real-Time, Windows)	1658
10.	PID와 퍼지 로직	1659
10.1	PID and Fuzzy Logic	1659
10.1.1	Tutorial: Using the Fuzzy System Designer	1660
10.1.1.1	Creating Input Linguistic Variables	1660
10.1.1.2	Creating Output Linguistic Variables	1662
10.1.1.3	Creating a Rule Base with Pre-Generated Rules	1663
10.1.1.4	Modifying Pre-Generated Rules	1664
10.1.1.5	Specifying a Defuzzification Method	1665
10.1.1.6	Testing the Fuzzy System	1666
10.1.1.7	Controlling the Fuzzy System	1666
10.1.2	Tutorial: Modifying a Fuzzy System with the Fuzzy Logic VIs	1667
10.1.2.1	Observing the Fuzzy System	1667
10.1.2.2	Loading the Fuzzy System	1668
10.1.2.3	Modifying a Linguistic Variable	1669
10.1.2.4	Modifying Membership Functions	1669
10.1.2.5	Modifying a Rule	1671
10.1.2.6	Creating Antecedents	1671
10.1.2.7	Creating Consequents	1672
10.1.2.8	Combining the Antecedents and Consequents for a Rule	1673
10.1.2.9	Saving the Fuzzy System	1674

10.1.3 퍼지 시스템	1675
10.1.3.1 Multi-Loop Control	1675
10.1.3.2 Non-Overlapping Input Terms	1675
10.1.3.3 Partially Overlapping Input Terms	1677
10.1.4 퍼지 컨트롤러	1679
10.1.4.1 Dual-Input Fuzzy Controllers	1679
10.1.4.2 Fault Protection	1680
10.1.4.3 Filtering Control Inputs	1681
10.1.4.4 Fully Overlapping Input Terms	1681
10.1.4.5 Fuzzy Controllers	1683
10.1.4.6 Fuzzy Controllers in Closed-Loop Control Structures	1684
10.1.4.7 Fuzzy Logic	1685
10.1.4.8 Fuzzy System Design Overview	1686
10.1.4.9 Fuzzy Systems	1686
10.1.4.10 Fuzzy-PI Controllers	1687
10.1.4.11 Gain Scheduling	1688
10.1.4.12 I/O Characteristics of Fuzzy Controllers	1690
10.1.5 PID Algorithms	1691
10.1.6 PID Control	1693
10.1.7 PID Relay Autotuning Technique	1695
10.1.8 PID Topologies	1695
10.1.9 Programmatically Autotuning a Previously Implemented PID Controller	1696
10.1.10 Programmatically Implementing and Autotuning a PID Controller	1697
10.1.11 Setpoint Profile Generation	1698
10.1.12 Singletons as Output Terms with Fully Overlapping Input Terms	1699
10.1.13 Specifying a Consequent Implication Method	1701
10.1.14 Specifying an Antecedent Connective	1702
10.1.15 Undefined Input Term Intervals	1704
10.1.16 Using Fuzzy Controllers with PID Controllers	1706
10.2 PID 자동튜닝	1707
10.2.1 Implementing Autotuning with the Autotuning Wizard	1707
10.2.2 Implementing PID Controllers with the PID VIs	1708
10.2.3 Implementing the PID Algorithm with the PID VIs	1710
10.2.4 Internal Model Control Autotuning Method	1714
10.2.5 Lead/Lag	1714
10.2.6 Logging Parameter Values After Autotuning	1714
10.2.7 Online Autotuning Techniques	1715
10.2.8 Advanced PID Algorithm	1717
10.2.8.1 Autotuning Methods for Testing PID Parameters	1720

10.2.8.2	Autotuning PID Controllers (PID and Fuzzy Logic)	1721
10.2.9	Relay Feedback Autotuning Technique	1722
10.2.10	Reproducing a Given I/O Characteristic	1723
10.2.11	Step Closed Loop Autotuning Technique	1724
10.2.12	Step Open Loop Autotuning Technique	1724
10.2.13	Updating PID Parameters After Autotuning	1725
10.2.14	Ziegler-Nichols Autotuning Method	1726
10.3	역퍼지화	1728
10.3.1	Center of Area (CoA)	1728
10.3.2	Center of Maximum (CoM)	1730
10.3.3	Center of Sums (CoS)	1731
10.3.4	Chien-Hrones-Reswick Autotuning Method	1732
10.3.5	Cohen-Coon Autotuning Method	1733
10.3.6	Complete Rule Bases	1734
10.3.7	Control Output Rate Limiting	1736
10.3.8	Converting Between Percentage of Full Scale and Engineering Units	1736
10.3.9	Creating a Rule Base	1736
10.3.10	Creating Linguistic Variables	1738
10.3.11	Creating Membership Functions	1739
10.3.12	Defuzzification Methods	1745
10.3.13	Designing a Control Strategy	1747
10.3.14	Different Overlapping Degrees of Membership Functions for Output Terms	1750
10.3.15	Mean of Maximum (MoM)	1751
10.3.16	Modified Center of Area (mCoA)	1752
10.3.17	Selecting a Defuzzification Method	1753
10.3.18	Using the Mean of Maximum (MoM) Defuzzification Method	1754
10.3.19	Wide and Small Membership Functions for Output Terms	1757
11.	배포된 시스템 관리자	1759
11.1	알람과 이벤트	1759
11.1.1	Acknowledging Alarms and Events	1759
11.1.2	Actions Menu	1759
11.1.3	Alarm Probe View	1760
11.1.4	Alarms and Events View	1760
11.1.5	Alerts Probe View	1761
11.1.6	CPU/Memory Probe View	1762
11.1.7	Date Time Probe View	1762
11.1.8	I/O Variable Probe View	1763
11.1.9	OpenCSRs Probe View	1763
11.1.10	Process Probe View	1764

11.1.11	Scan Engine Probe View	1764
11.1.12	ServerCertificates Probe View	1766
11.1.13	Variable Probe View	1766
11.1.14	Web Server Probe View	1767
11.1.15	Web Server URL Mapping View	1768
11.1.16	Web Service Probe View	1768
11.1.17	Configure Web Services Security View	1769
11.1.18	Configuring Alarms for Shared Variables (DSC Module)	1769
11.1.19	Monitoring Alarms and Events	1770
11.1.19.1	Monitoring and Editing Data Item Ranges	1770
11.1.19.2	Monitoring Data Items in Data Item Ranges	1771
11.1.19.3	Monitoring Variables with the Watch List View	1771
11.2	인증서 및 프로세스 생성하기	1772
11.2.1	Creating a Certificate from a CSR	1772
11.2.2	Creating a Certificate Signing Request	1772
11.2.3	Creating a Self-Signed Certificate	1773
11.2.4	Creating Processes	1774
11.2.5	Creating, Modifying, or Deleting Shared or Network Variables	1774
11.2.6	Custom VI-based Server - On Input Change Wizard	1775
11.2.7	Custom VI-based Server - Periodic Wizard	1776
11.2.8	Customizing the System Manager Workspace	1777
11.2.9	Processes	1778
11.3	시스템 관리	1779
11.3.1	System Manager Overview	1779
11.3.2	File Menu	1781
11.3.3	Forcing I/O Variable Values (NI Scan Engine)	1782
11.3.4	Help Menu	1783
11.3.5	I/O Servers	1783
11.3.6	Observing Trend View of Variables	1784
11.3.7	Programmatically Accessing Published Values	1784
11.3.8	Saving and Restoring a System Manager Workspace	1784
11.3.9	Shared or Network Variables	1785
11.3.10	Shared Variable Engine	1785
11.3.11	Synchronizing Time for Networked Computers	1786
11.3.12	Systems	1787
11.3.13	VI States	1787
11.3.14	View Menu	1788
11.3.14.1	Viewing and Clearing Faults (NI Scan Engine)	1788
11.3.14.2	Viewing and Setting the Scan Engine Mode (NI Scan Engine)	1789

LabVIEW란?

LabVIEW (Laboratory Virtual Instrument Engineering Workbench)는 G 프로그래밍 언어를 사용하여 어플리케이션을 생성하는 데 사용되는 프로그래밍 환경입니다.

정의

프로그램 실행 순서가 명령에 의해 결정되는 텍스트 기반의 프로그래밍 언어와 달리, LabVIEW는 데이터 흐름 프로그래밍을 사용합니다. 데이터 흐름 프로그래밍에서, 블록 다이어그램에서 노드를 통한 데이터의 흐름은 VI와 함수의 실행 순서를 결정합니다. VI 또는 버추얼 인스트루먼트는 물리적인 인스트루먼트와 유사하게 작동하는 LabVIEW 프로그램입니다.

LabVIEW에서는 도구와 객체 세트를 사용하여 사용자 인터페이스를 생성합니다. 사용자 인터페이스는 프런트패널이라고 합니다. 프런트패널을 만든 후, 그래픽 형태의 함수를 사용하여 코드를 추가하여 프런트패널의 객체를 컨트롤합니다. 블록다이어그램에 G 코드 또는 블록다이어그램 코드라고도 불리는 이러한 그래픽 코드를 추가합니다. 블록다이어그램은 어떤 면에서 흐름도와 비슷합니다. 블록다이어그램, 프런트패널, 코드의 그래픽 표현이 VI를 구성합니다.

주요 기능

- 광범위한 분석 함수
- 프로그래밍 가능한 대화형 디스플레이 요소
- 자동화된 인스트루먼트 및 데이터 수집 드라이버
- 업계 표준 프로토콜 및 기타 언어 지원
- FPGA를 사용한 고속 측정 및 제어 시스템
- 센서 또는 액추에이터로 물리적 시스템 측정
- 전자 설계 검증 또는 확인
- 생산 테스트 시스템 개발
- 스마트 머신 또는 산업 장비 설계



관련 정보

[Installing, Updating, Repairing, and Removing NI Software](#)

LabVIEW 요구사항

LabVIEW를 실행하고 사용하려면 시스템이 최소한 다음 요구사항을 충족해야 합니다.

프로세서

- Pentium 4M (또는 동등한 프로세서) 또는 이후 버전 (32비트) Pentium 4 G1 (또는 동등한 프로세서) 또는 이후 버전 (64비트)
- Linux: Pentium 4 G1 (또는 동등한 프로세서) 또는 이후 버전
- macOS: Intel 기반 프로세서

RAM

- Windows: 1GB
- Linux: 1GB
- macOS: 2GB

화면 해상도

- Windows: 1024 x 768 픽셀
- Linux: 1024 x 768 픽셀
- macOS: 1024 x 768 픽셀

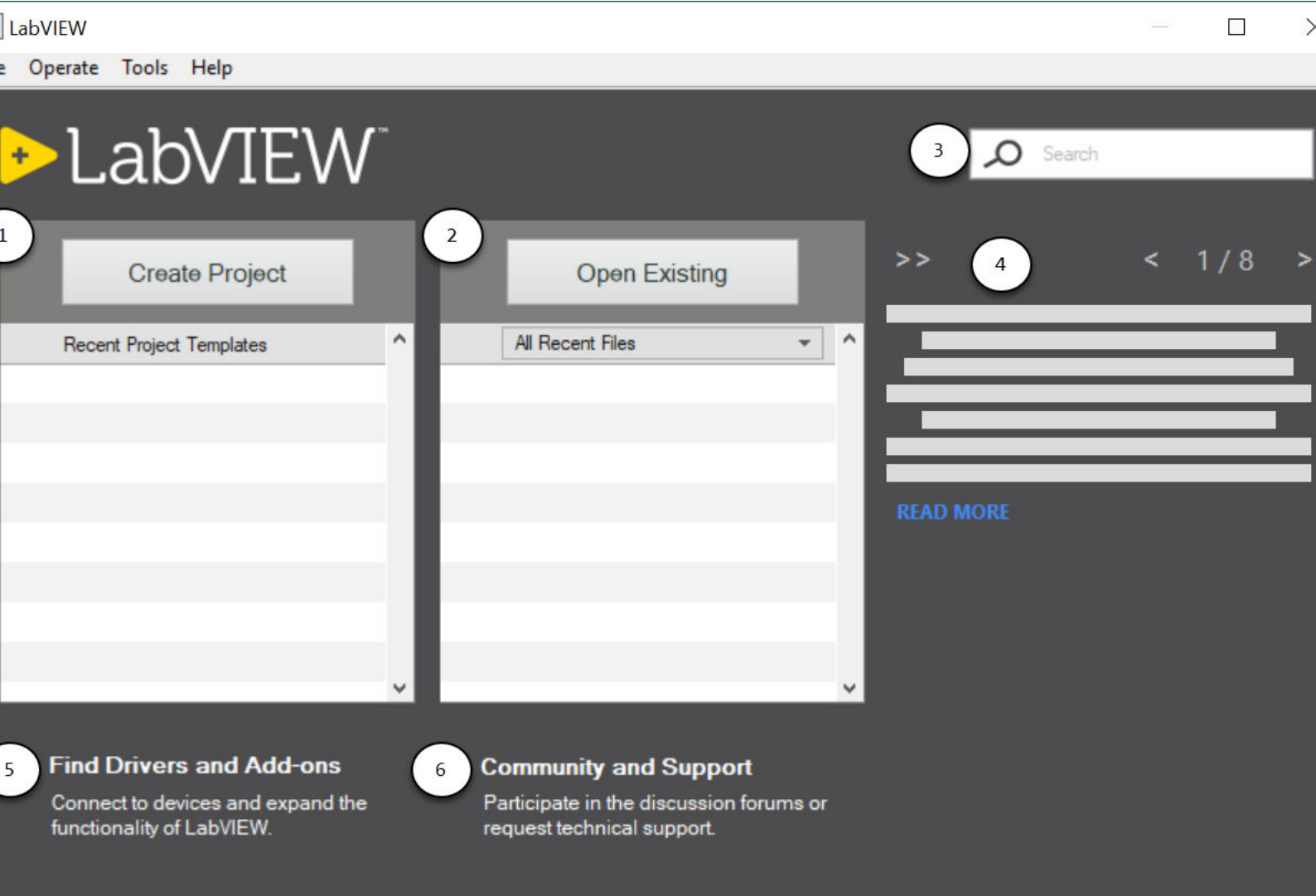
디스크 공간

- Windows: 5 GB
- Linux: 1.1 GB
- macOS: 1.4 GB

LabVIEW의 전체 구성

LabVIEW 환경의 일부와 이를 탐색하는 방법에 대해 알아보십시오.

시작하기 윈도우를 사용하여 템플릿 또는 샘플 프로젝트에서 새 프로젝트를 생성하거나 또는 LabVIEW를 열고 바로 기존의 LabVIEW 파일들을 열 수 있습니다. LabVIEW의 기능을 확장하는 리소스 및 LabVIEW 학습에 도움이 되는 정보에 접근할 수 있습니다.



1 —템플릿 또는 샘플 프로젝트에서 새 프로젝트를 생성합니다.

2 —기존 프로젝트 또는 VI를 엽니다. 파일 목록에서 해당 파일에 항상 액세스할 수 있도록 즐겨 찾는 파일을 고정해 보십시오.

3 —설치된 LabVIEW 문서를 검색하십시오. 인터넷에 연결되어 있다면 ni.com의 검색 결과도 포함됩니다.

4 —NI의 뉴스와 리소스를 찾아보십시오.

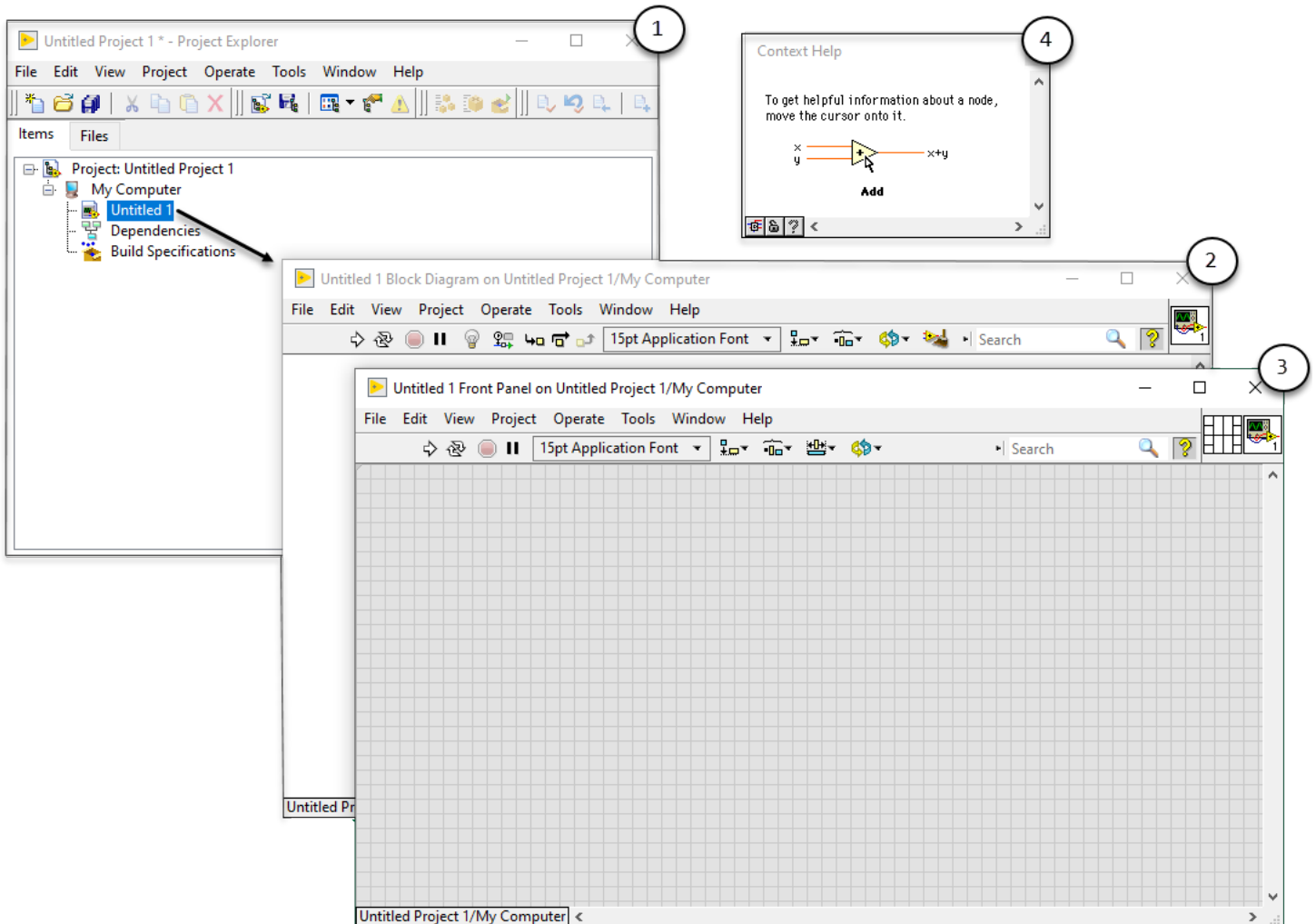
5 —드라이버 및 애드온 찾기

6 —커뮤니티 및 지원

시작하기 윈도우는 사용자가 기존 파일을 열거나 새 파일을 생성할 때 사라지고, 열려 있던 모든 프런트패널 및 블록 다이어그램을 닫을 때 다시 나타납니다. 보기»시작하기 윈도우를 선택하여 윈도우를 디스플레이할 수 있습니다.

LabVIEW 편집기

새 VI로 빈 프로젝트를 생성할 때 LabVIEW 편집기의 구성요소를 살펴보겠습니다.



1 — 프로젝트 탐색기

2 — 프론트패널

3 — 블록다이어그램

4 — 기본 도움말

LabVIEW 개념

LabVIEW 개념 개요

이 모음에서는 LabVIEW의 다양한 기능에 대한 정보를 제공합니다.

LabVIEW 환경

LabVIEW 환경 개요

VI의 프런트패널과 블록다이어그램을 구현할 때 LabVIEW 팔레트, 도구, 메뉴를 사용하십시오. LabVIEW는 컨트롤 팔레트, 함수 팔레트, 도구 팔레트의 세 가지 팔레트를 제공합니다. 또한, LabVIEW는 시작하기 윈도우, 기본 도움말 윈도우, 프로젝트 탐색기 윈도우, 탐색기 윈도우를 포함합니다. 컨트롤과 함수 팔레트를 사용자 정의할 수 있으며, 여러 작업 환경 옵션을 설정할 수 있습니다.



관련 정보

[메뉴와 도구 모음](#) (46 페이지 참조)

[LabVIEW 사용자 이름 설정하기](#) (757 페이지 참조)

[키보드 바로 가기 키](#) (758 페이지 참조)

[컨트롤 선택하기](#) (767 페이지 참조)

[큰 프런트패널과 블록다이어그램 탐색하기](#) (867 페이지 참조)

LabVIEW 환경 보안하기

네트워크에 보안 도메인이 설정되어 있다면, 사용자가 반드시 로그인 및 로그아웃해야 하는 보안된 LabVIEW 환경을 생성할 수 있습니다. 도메인 계정 관리자를 사용하여 도메인을 관리합니다. 사용자 계정을 생성한 이후에, 강제적인 접근 컨트롤 리스트를 갖는 원격 VI 서버와 통신할 때 LabVIEW는 사용자 로그인 정보를 사용하여 LabVIEW 사용자를 인식합니다. 접근 컨트롤 리스트는 특정 사용자와 그룹에게 허가된 접근 권한을 디스플레이합니다.

또한 각각의 LabVIEW 환경에 VI 서버 접근 컨트롤 리스트를 설정함으로써 보안된 LabVIEW 네트워크를 생성할 수 있습니다. 이후 프로그램적으로 로그인, 로그아웃하고, 어플리케이션 프로퍼티 노드, 메소드 노드, 어플리케이션 이벤트를 사용하여 블록다이어그램에서 사용자 계정 변경 내역을 모니터링할 수 있습니다.

작업 환경 사용자 정의하기

컨트롤 및 함수 팔레트를 사용자 정의할 수 있고, 옵션 대화 상자를 사용하여 **팔레트 포맷을 선택**하고 다른 작업 환경 옵션을 설정할 수 있습니다.

컨트롤과 함수 팔레트를 사용자 정의하기

컨트롤과 함수 팔레트를 다음과 같이 사용자 정의할 수 있습니다:

- **팔레트에 VI와 컨트롤을 추가**합니다.

- **팔레트 세트를 편집**하여 내장 팔레트를 다시 배치하고, 서브팔레트를 생성 및 이동하는 등의 작업을 합니다.
- **팔레트에 툴킷을 추가**합니다.
- 함수 팔레트의 아이템을 **즐거찾기 항목**에 추가합니다.



주의

LabVIEW는 설치를 제거하거나 다시 설치할 때, .11b 파일에 저장한 모든 VI와 다른 파일을 포함하는 .11b 파일의 설치를 제거하기 때문에, vi.lib 디렉토리에 사용자 VI나 다른 파일을 LabVIEW와 함께 설치되는 .11b 파일로 저장하지 않습니다. 함수와 컨트롤 팔레트에 추가하려는 사용자 VI와 컨트롤을 user.lib 디렉토리에 저장합니다.

사용자와 인스트루먼트 드라이버 서브팔레트에 VI와 컨트롤 추가하기

함수와 컨트롤 팔레트에 **VI와 컨트롤을 추가**하는 가장 간편한 방법은 **이들을 labview\user.lib 디렉토리에 저장**하는 것입니다. LabVIEW를 다시 시작하면 사용자 라이브러리 및 사용자 컨트롤 팔레트에는 labview\user.lib의 각 디렉토리, LLB, 팔레트 파일 .mnu에 대한 서브팔레트와 labview\user.lib의 각 파일에 대한 아이콘이 포함됩니다. 특정 디렉토리에서 파일을 추가하거나 제거하면, LabVIEW가 재시작될 때마다 자동적으로 팔레트를 업데이트합니다.

인스트루먼트 드라이버 팔레트는 labview\instr.lib 디렉토리에 대응합니다. 이 디렉토리에 인스트루먼트 드라이버를 저장하여 함수 팔레트에 추가합니다.

이 방법으로 함수와 컨트롤 팔레트에 VI나 컨트롤을 추가할 때, 각 서브팔레트의 이름이나 팔레트 상의 VI나 컨트롤의 정확한 위치까지 설정할 수는 없습니다.

팔레트 세트 편집하기

컨트롤과 함수 팔레트 세트 편집 대화 상자를 사용하여 팔레트 세트를 편집합니다. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다. LabVIEW는 컨트롤과 함수 팔레트의 내용을 labview\menus 디렉토리에 저장합니다. 메뉴 디렉토리에는 각 팔레트 항목에 대응하는 디렉토리가 들어있습니다.



노트

menusDir 환경을 각 사용자 환경 파일에 대해서 유일한 다른 경로로 설정하는 환경 파일에 라인을 추가할 수 있습니다.

팔레트를 편집하는 경우, LabVIEW는 편집한 내용을 labview\menus 디렉토리에 저장합니다. LabVIEW는 편집 사항을 원본 팔레트의 복사본에 저장합니다. 원본 팔레트를 보호하면 원본 팔레트를 손상시키지 않고도 팔레트를 실험해 볼 수 있습니다. 원본 팔레트로 돌아가려면 컨트롤과 함수 팔레트 편집 대화 상자에서 기본으로 복구 버튼을 클릭합니다.

또한, 팔레트 편집 VI를 사용하여 팔레트를 프로그램적으로 생성하고 편집할 수 있습니다. 여러 개의 팔레트를 편집하거나, 새 팔레트를 생성하거나, 편집한 팔레트의 모양을 나중에 점검하려는 경우에 팔레트 편집 VI를 사용합니다.

[팔레트 읽기] VI는 팔레트 파일 .mnu로부터 팔레트 데이터를 추출합니다. 팔레트 데이터 출력은 팔레트 파일에 있는 팔레트에 대한 모든 정보를 포함합니다. 컨트롤 또는 함수를 팔레트 타입 입력에서 선택하여 컨트롤 또는 함수 팔레트에 대한 데이터를 팔레트 파일에서 읽어옵니다. 다시 [팔레트 읽기] VI를 실행하여 두번째 팔레트 타입에 대한 데이터를 읽습니다. 기존 팔레트를 템플릿으로 사용하려는 경우, [팔레트 읽기] VI를 사용합니다. 기존 팔레트에서 데이터를 읽고, 해당 데이터를 사용자 정의하여 새 팔레트를 생성할 수 있습니다.

[팔레트 쓰기] VI를 사용하여 팔레트 데이터를 팔레트 파일에 씁니다. 팔레트 데이터 클러스터에 있는 아이템 입력을 사용하여 팔레트에 디스플레이하려는 정보를 지정합니다. 아이템 입력에 있는 아이템의 순서는 팔레트에 있는 아이템의 순서를 정의합니다. 컨트롤 또는 함수를 팔레트 타입 입력에서 선택하여 컨트롤 또는 함수 팔레트에 대한 데이터를 팔레트 파일에 씁니다. 다시 [팔레트 읽기] VI를 실행하여 두번째 팔레트 타입에 대한 데이터를 씁니다.

[팔레트 쓰기] VI의 어플리케이션 참조 입력에서 타겟을 지정하여, 특정 타겟에 대한 팔레트 세트를 편집할 수 있습니다. 참조하려는 타겟을 가진 프로젝트를 생성하십시오. 프로젝트에서 타겟 참조를 얻으려면 프로젝트:프로젝트[] 프로퍼티를 사용하십시오. 타겟 참조를 가질 때 어플리케이션 프로퍼티를 사용하여 어플리케이션 참조를 얻을 수 있습니다.



주의 팔레트 편집 VI를 사용하여 labview\menus 디렉토리에서 팔레트를 편집하는 경우, LabVIEW는 원본 팔레트의 복사본에 편집 내용을 저장하지 않습니다. 원본 팔레트를 편집한 후에는 원본 팔레트로 다시 돌아갈 수 없습니다.

팔레트 세트를 편집한 다음, 팔레트 새로 고침 메소드를 사용하여 팔레트 세트를 업데이트합니다.

labview\examples\Palette API 디렉토리에서 프로그램적으로 컨트롤과 함수 팔레트 세트를 편집하는 예를 참조합니다.

관련 예제 찾기

LabVIEW가 팔레트를 저장하는 방법

.mnu 파일과 .11b 파일은 각각 하나의 컨트롤과 함수 팔레트를 포함합니다. 또한 각 파일은 컨트롤과 함수 팔레트의 아이콘도 가지고 있습니다. 새로 생성한 서브팔레트는 서로 다른 .mnu 파일에 따로 저장해야 합니다.

LabVIEW는 각 VI 또는 컨트롤에 해당하는 아이콘을 팔레트에 생성합니다. 각 서브디렉토리 .mnu 파일 또는 .11b 파일에 대해, LabVIEW는 팔레트에 서브팔레트를 생성합니다.

.NET과 ActiveX 서브팔레트 만들기

.NET이나 ActiveX를 프런트패널에서 사용하려면 도구 >>반입>>.NET 컨트롤을 팔레트로 또는 ActiveX 컨트롤을 팔레트로 선택하여 .NET이나 ActiveX 컨트롤 세트를 각각 사용자 컨트롤로 변환하고 컨트롤 팔레트에 추가합니다. LabVIEW는 이 컨트롤을 기본적으로 labview\menus\Controls\DotNet & ActiveX 디렉토리에 저장합니다. 왜냐하

면 이 디렉토리에 위치한 모든 파일이 자동으로 .NET & ActiveX 팔레트에 나타나기 때문입니다.

팔레트에 툴킷과 모듈을 나타내기

LabVIEW에 모듈이나 툴킷을 설치한 후에 모듈이나 툴킷을 나타내는 팔레트가 함수 팔레트의 관련된 항목에 나타납니다. 모듈과 툴킷 서브팔레트를 사용자 라이브러리 및 즐겨 찾기 팔레트에 추가하여 팔레트 보기를 사용자 정의할 수 있습니다.

작업 환경 옵션 셋팅

도구 >> 옵션을 선택하여 LabVIEW를 사용자 정의합니다. 옵션 대화 상자를 사용하여 프런트패널, 블록 다이어그램, 경로, 성능과 디스크 관련 내용, 정렬 눈금, 팔레트, 실행 취소, 디버깅 도구, 색, 폰트, 인쇄, 히스토리 윈도우, 다른 LabVIEW 기능을 위한 옵션을 설정합니다.

옵션 대화 상자 왼쪽에 있는 항목 리스트를 사용하여 여러 항목의 옵션 중에서 선택합니다.

LabVIEW가 옵션을 저장하는 방법

수동으로 옵션을 편집하거나 옵션의 정확한 포맷을 알 필요는 없습니다. 옵션 대화 상자가 대신 작업을 수행하기 때문입니다. LabVIEW는 각 플랫폼마다 다른 위치에 옵션을 저장합니다. 하지만, 각 플랫폼에 있는 파일 포맷은 다른 설정 파일과 유사합니다. LabVIEW 섹션 마커로 시작하고 그 뒤에 `offscreenUpdates=True`와 같은 옵션 이름과 옵션값이 따라옵니다.

Windows

LabVIEW는 `labview` 디렉토리에서 `labview.ini` 파일에 옵션을 저장합니다.

다른 옵션 파일을 사용하려면, LabVIEW를 시작할 때 사용하는 바로 가기에 이 파일을 지정합니다. 예를 들어, `lvrc`라는 이름의 사용자 컴퓨터에서 `labview.ini` 대신 옵션 파일을 사용하려면 바탕화면의 LabVIEW 아이콘에서 마우스 오른쪽 버튼을 클릭하고 프로퍼티를 선택합니다. 바로 가기 탭을 클릭하고 타겟 텍스트 박스에서 `labview -pref lvrc`를 타입합니다.

macOS

LabVIEW는 홈 디렉토리의 `~/Library/Preferences/LabVIEW x Preferences`에 위치한 LabVIEW 환경 텍스트 파일에 옵션을 저장합니다. 여기서 `x`는 LabVIEW 버전 번호입니다.

다른 옵션 파일을 사용하려면, LabVIEW 환경 파일을 LabVIEW 폴더에 복사하고 옵션 대화 상자에서 옵션을 변경합니다. LabVIEW를 시작하면, 가장 먼저 LabVIEW 폴더에서 옵션 파일을 찾습니다. 그 폴더에서 이 파일을 찾지 못하면, 사용자 환경 폴더에서 찾습

니다. 그 폴더에서 이 파일을 찾지 못하면, 사용자 환경 폴더에 새 파일을 생성합니다. LabVIEW는 옵션 대화 상자의 모든 변경 사항을 가장 먼저 찾은 LabVIEW 환경 파일에 입력합니다.

Linux

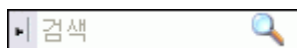
LabVIEW는 옵션을 `/home/<username>/natinst/.config/LabVIEW- x /labview.conf`에 저장합니다. 여기서 `x`는 LabVIEW의 버전 번호입니다. 사용자가 옵션 대화 상자에서 옵션을 바꾸면 LabVIEW는 변경된 내용을 `/home/<username>/natinst/.config/LabVIEW- x /labview.conf` 파일에 저장합니다.

다른 옵션 파일을 사용하려면, LabVIEW를 시작할 때 사용하고자 하는 파일을 명령 라인에 입력해야 합니다. 예를 들어, `/home/<username>/natinst/.config/LabVIEW- x /labview.conf` 대신에 `test` 디렉토리의 `lvrc` 파일을 사용하려면 `labview -pref /test/lvrc`를 입력합니다. LabVIEW는 옵션 대화 상자의 모든 변경 사항을 `lvrc` 옵션 파일에 입력합니다. 명령 라인에서 사용자가 옵션 파일을 지정할 경우, LabVIEW는 여전히 프로그램 디렉토리에서 `/home/<username>/natinst/.config/LabVIEW- x /labview.conf` 파일을 읽지만, 명령 라인에 지정한 옵션 파일은 프로그램 디렉토리에서 충돌하는 엔트리를 덮어쓰게 됩니다.

LabVIEW 기능

LabVIEW의 검색 옵션

LabVIEW에는 어플리케이션 전체를 검색할 수 있는 기능이 있습니다. 검색은 컨트롤과 함수 팔레트, 도움말 시스템과 `ni.com`을 검색한 결과를 반환합니다. 현지화된 LabVIEW 버전에서는 검색 결과가 현지 언어와 영어를 함께 반환할 수도 있습니다. 검색 텍스트 박스는 시작하기 윈도우와 편집 모드인 VI의 프런트패널과 블록 다이어그램 윈도우의 오른쪽 상단에 나타납니다. 다음 그림은 검색 텍스트 박스를 보여줍니다.



프런트패널 윈도우와 블록 다이어그램 윈도우에서는 텍스트 박스 왼쪽의 검색 색화살표를 클릭해서 검색 텍스트 박스를 확장하거나 축소할 수 있습니다. 다음 그림은 축소된 검색 텍스트 박스를 보여줍니다.



검색 텍스트 박스에 검색하려는 단어나 구절을 입력하십시오. 정확한 구절을 검색하려면 구절 앞뒤에 따옴표를 붙입니다.



노트

단 따옴표가 붙은 구절을 검색할 경우, 팔레트 항목은 검색 결과를 반환하지 않습니다.

LabVIEW가 어떤 항목에서 검색을 실행할지 지정할 수 있습니다. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이하고 항목 리스트에서 검색을 선택합니다. 검색을 수행하려는 항목의 확인란에 확인 표시를 합니다. 다음 항목 중에서 선택할 수 있습니다.

- **도움말**—설치한 도움말을 검색합니다. 검색 결과는  아이콘과 함께 나타납니다.

노트 도움말 항목의 검색 결과에는 모든 LabVIEW 모듈과 툴킷 및 LabVIEW 이외의 제품에 대한 도움말이 포함되지 않을 수도 있습니다.

- **팔레트**—컨트롤 팔레트와 함수 팔레트에서 컨트롤, VI, 함수를 검색합니다. 컨트롤 팔레트에서 검색한 결과는  아이콘과 함께 나타납니다. 함수 팔레트에서 검색한 결과는  아이콘과 함께 나타납니다.

노트 팔레트 항목은 시작하기 윈도우에서 디스플레이하지 않습니다.

- **NI 지원**—ni.com에서 도움말, 매뉴얼, 길라잡이를 검색합니다. 검색 결과는  아이콘과 함께 나타납니다.

노트 NI 지원 항목의 검색 결과에는 현재 사용하고 있는 LabVIEW 버전의 도움말이 포함되어 있으며 모든 LabVIEW 모듈과 툴킷 및 LabVIEW 이외의 제품에 대한 도움말은 포함되지 않을 수도 있습니다.

- **커뮤니티 지원**—ni.com에서 Discussion Forum, 기술지원 데이터베이스 문서, 사례 연구를 검색합니다. 검색 결과는  아이콘과 함께 나타납니다.
- **다운로드**—ni.com에서 예제, 드라이버, 업데이트에 대한 정보를 검색합니다. 검색 결과는  아이콘과 함께 나타납니다.
- **제품 정보**—ni.com에서 제품 개요, 서비스, 지원, 데이터 등과 같은 제품 관련 정보를 검색합니다. 검색 결과는  아이콘과 함께 나타납니다.

NI 지원, 커뮤니티 지원, 다운로드 및 제품 정보 항목은 인터넷 연결이 되어 있어야 검색 결과를 반환합니다. 이러한 항목에서 결과를 선택하면 결과는 기본 웹 브라우저에서 디스플레이됩니다.

검색 결과 보기

검색 결과는 항목 별로 결과 리스트에 디스플레이됩니다. 옵션 대화 상자의 검색 페이지를 사용해서 결과 리스트에 나타나는 항목의 순서를 변경할 수 있습니다. 다음 아이콘을 사용해서 결과를 나타내고, 숨기거나, 더 많은 결과를 보여줄 수 있습니다.

▼	항목의 검색 결과 디스플레이하기
▶	항목의 검색 결과 숨기기.
	항목을 확장해서 더 많은 결과를 디스플레이할 수 있습니다. 항목은 페이지 당 최대 15개의 결과를 디스플레이할 수 있도록 확장됩니다. 결과 리스트 맨 아래에 있는 페이지 버튼을 사용해서 더 많은 결과를 나타낼 수 있습니다. 한 항목을 확장하면 다른 항목들은 디스플레이되지 않습니다.
	확장된 항목을 축소하고, 옵션 대화 상자의 검색 페이지에서 선택한 모든 항목을 디스플레이합니다.

검색 결과에서 작업 실행하기

각 검색 결과에서 도움말 토픽 또는 웹 페이지 열기, 팔레트의 객체를 블록 다이어그램이나 프런트패널에 놓기 또는 팔레트의 객체 찾기와 같은 일반적인 작업을 수행할 수 있습니다. 결과를 스크롤해서 한 개의 결과를 하이라이트한 후, 실행 가능한 작업을 봅니다. 또한 위, 아래 화살표를 사용해서 결과를 하이라이트하고, 실행 가능한 작업을 볼 수도 있습니다. 작업 링크를 클릭해서 해당 작업을 실행합니다. 예를 들어 열기 작업 링크나 도움말 항목 검색 결과에서 <Enter> 키를 눌러 도움말 시스템에서 도움말 토픽을 나타낼 수 있습니다. 또한 결과를 클릭하거나, <Enter> 키를 눌러 기본 작업을 수행할 수도 있습니다. 기본 작업 링크는 굵은체로 나타납니다.

컨트롤과 함수 팔레트 탐색하기

팔레트의 객체를 클릭하여 객체를 커서 위에 놓아서, 해당 객체를 프런트패널 또는 블록 다이어그램에 추가할 수 있도록 합니다. 팔레트의 왼쪽 위 가장자리에 있는 압정을 클릭하여 팔레트를 고정하면, 팔레트에 있는 VI 아이콘에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 VI 열기를 클릭하여 VI를 열 수 있습니다.

컨트롤 또는 함수 팔레트의 왼쪽 옆에 있는 검은 화살표를 클릭하여 서브팔레트 항목을 펼치거나 숨깁니다. 이 화살표는 항목(표준)과 항목(아이콘 및 텍스트)로 **팔레트 포맷을 설정**한 경우에만 나타납니다.

다음과 같은 컨트롤과 함수 팔레트 도구 모음의 버튼을 사용하여 팔레트를 탐색하고, 팔레트를 설정하며, 컨트롤, VI, 함수를 검색할 수 있습니다.

	상위 팔레트로—팔레트 계층구조에서 한 레벨 위로 이동합니다. 이 버튼을 클릭하고 마우스 버튼을 계속 누르면 현재 서브팔레트에 이르기 위한 경로의 각 서브팔레트를 나열하는 바로 가기 메뉴가 디스플레이됩니다. 바로 가기 메뉴에서 서브팔레트 이름을 선택하여 서브팔레트를 탐색합니다. 이 버튼은 팔레트 포맷을 아이콘, 아이콘과 텍스트, 또는 텍스트로 설정한 경우에만 나타납니다.
	검색—팔레트를 검색 모드로 변경하면, 텍스트 기반 검색 을 수행하여 팔레트에서 컨트롤, VI, 함수를 찾을 수 있습니다. 팔레트가 검색 모드인 상태에서 돌아가기 버튼을 클릭하면 검색 모드를 끝내고 팔레트로 돌아갑니다.
	사용자 정의—현재 팔레트의 포맷을 선택하고, 모든 팔레트의 항목을 보이거나 숨기고 , 텍스트와 트리 포맷의 아이템을 알파벳 순서로 정렬하는 옵션을 제공합니다. 바로 가기 메뉴에서 옵션을 선택하여 옵션 대화 상자의 컨트롤/함수 팔레트 페이지를 디스플레이하며, 여기서 모든 팔레트에 대한 포맷을 선택할 수 있습니다. 이 버튼은 팔레트의 왼쪽 위 코너에 위치한 압정을 클릭하여 팔레트를 고정한 경우에만 나타납니다.
	팔레트 크기 복구—팔레트 크기를 기본 크기로 다시 조정합니다. 이 버튼은 컨트롤 또는 함수 팔레트의 크기를 재조정하는 하는 경우에만 나타나며, 사용자는 팔레트의 왼쪽 위 코너에 위치한 압정을 클릭하여 팔레트를 고정합니다.

메뉴와 도구 모음

메뉴 및 도구 모음 아이템을 사용하여 LabVIEW에 있는 객체를 사용하고 변경합니다.

메뉴

VI 윈도우 상단의 메뉴에는 열기, 저장, 복사, 붙여넣기와 같이 다른 어플리케이션에도 공통적인 아이템과 LabVIEW에만 사용되는 특수한 아이템이 위치합니다. 또한, 일부 메뉴 아이템은 **키보드 바로 가기**를 나열합니다.

(macOS) 화면 상단에 메뉴가 표시됩니다.



노트

VI 실행 중에는 일부 메뉴 아이템을 사용할 수 없습니다.

바로 가기 메뉴

모든 LabVIEW 객체에는 관련된 바로 가기 메뉴가 있는데, 이는 목차 메뉴, 팝업 메뉴, 마우스 오른쪽 버튼 메뉴라고도 합니다. VI를 생성할 때, 바로 가기 메뉴 아이템을 사용하여 프런트패널과 블록 다이어그램 객체의 모양이나 동작을 변경할 수 있습니다. 바로 가기 메뉴를 사용하려면, 객체에서 마우스 오른쪽 버튼을 클릭합니다.

(macOS) <Control>을 클릭하면 마우스 오른쪽 버튼 클릭과 같은 동작을 수행할 수 있습니다.

VI를 편집하는 동안, 컨트롤이나 인디케이터의 바로 가기 메뉴는 여러가지 **일반적인 옵션**을 가지고 있습니다. 컨트롤이나 인디케이터의 바로 가기 메뉴의 처음 두 라인 위의 메뉴 아이템은 모든 컨트롤과 인디케이터에 공통입니다. 두번째 라인 아래의 메뉴 아이템은 객체에 맞게 사용자 정의되어 있습니다. **바로 가기 메뉴 플러그인을 생성**하여 바로 가기 메뉴에 사용자 아이템을 추가할 수 있습니다.

여러 개의 객체에서 일반적인 작업을 동시에 수행하려면, 객체를 선택한 후 바로 가기 메뉴를 사용합니다. 일부 객체는 바로 가기 메뉴에 있는 모든 작업을 지원하지 않을 수도 있습니다. 예를 들어, 숫자형 컨트롤과 구조를 선택하고 바로 가기 메뉴에서 **생성>참조**를 선택하면, LabVIEW는 이 숫자형 구조에 대해서만 참조를 생성합니다.

또한 런타임에서 VI 안에 나타나는 **바로 가기 메뉴 아이템을 사용자 정의**할 수 있습니다.

키보드 바로 가기 키 사용자 정의하기

모든 기본 LabVIEW 키보드 바로 가기 키를 변경하고 사용자 정의할 수 있습니다. **도구>옵션**을 선택하여 옵션 대화 상자를 디스플레이합니다. 항목 리스트에서 메뉴 바로 가기를 선택하여 VI 메뉴 아이템에 대한 키보드 바로 가기를 설정합니다. 메뉴 바로 가기 페이지에는 메뉴 아이템의 기본 LabVIEW 바로 가기 리스트가 포함됩니다.



노트 LabVIEW는 비활성화된 메뉴 아이템 또는 숨겨진 VI 도구 모음 버튼에 대응하는 키보드의 바로 가기 키에 대해서는 응답하지 않습니다.

VI 도구 모음

VI 도구 모음의 버튼을 사용하여 VI를 실행하고, VI를 일시 정지하고, VI를 강제 종료하고, VI를 **디버그**하고, 폰트를 설정하고, 객체에 대한 정렬, 그룹화, 간격 조절을 수행합니다.

프로젝트 탐색기 윈도우 도구 모음

표준, 프로젝트, 빌드, 소스 컨트롤 도구 모음의 버튼을 사용하여 LabVIEW 프로젝트에서 작업을 수행할 수 있습니다. 이 도구 모음은 **프로젝트 탐색기** 윈도우의 제일 위에서 사용 가능합니다. 모든 도구 모음을 보려면 프로젝트 탐색기 윈도우를 확장해야 할 수도 있습니다.



노트 Unit Test Framework 도구 모음은 LabVIEW Unit Test Framework Toolkit가 설치되어 있는 경우에만 사용 가능합니다.

보기» 도구 모음을 선택하고 보이거나 숨기려는 도구 모음을 선택하여 도구 모음을 보이거나 숨길 수 있습니다. 또한, 도구 모음의 빈 영역에서 마우스 오른쪽 버튼을 클릭한 후 보이거나 숨기려는 도구 모음을 선택할 수도 있습니다.

탐색 윈도우

탐색 윈도우는 편집 모드에서 활성화된 프런트패널 또는 활성화된 블록다이어그램의 개요를 보여줍니다. 탐색 윈도우를 사용하여 큰 프런트패널 또는 블록다이어그램을 탐색합니다. 탐색 윈도우에서 이미지의 영역을 클릭하여 해당 영역을 프런트패널 또는 블록다이어그램에 디스플레이합니다. 탐색 윈도우에서 이미지를 클릭하고 끌기하여 프런트패널 또는 블록다이어그램을 스크롤할 수 있습니다. 프런트패널이나 블록다이어그램에서 보이지 않는 영역은 탐색 윈도우에서 희미하게 나타납니다.

보기» 탐색 윈도우를 선택하여 탐색 윈도우를 디스플레이합니다. (Windows) 또한, <Ctrl-Shift-N> 키를 눌러서 윈도우를 디스플레이할 수 있습니다. (macOS) <Command-Shift-N> 키를 누릅니다. (Linux) <Alt-Shift-N> 키를 누릅니다.

탐색 윈도우를 크기 조정하여 디스플레이하는 이미지의 크기를 조정합니다. LabVIEW는 탐색 윈도우의 위치와 크기를 유지하여 LabVIEW를 다시 시작할 때, 같은 위치에서 같은 크기로 그 윈도우가 나타납니다.



노트

프런트패널이 여러 구획을 포함할 경우, 탐색 윈도우는 프런트패널의 이미지를 디스플레이합니다. 하지만, 프런트패널을 탐색하기 위해서 탐색 윈도우의 이미지를 클릭하거나 끌 수 없습니다.

볼리언 컨트롤과 인디케이터

볼리언 서브팔레트에 있는 볼리언 컨트롤과 인디케이터를 사용하여, 버튼, 스위치, LED와 같은 볼리언(참/거짓) 값을 입력하고 표시합니다.

다음 리스트는 일반, 실버, 시스템 또는 클래식 서브팔레트에서 사용할 수 있는 불리언 컨트롤과 인디케이터의 일반적인 사용례를 설명합니다:



노트

사용할 수 있는 불리언 컨트롤과 인디케이터는 선택한 서브팔레트의 스타일에 따라 다릅니다. 사용할 수 있는 불리언 컨트롤과 인디케이터는 일반, 실버, 시스템 또는 클래식 서브팔레트를 참조하십시오.

- 버튼 — 버튼을 눌러 명령을 선택하는 컨트롤로 사용합니다.
- 로커 스위치, 토글, 스위치 — 객체를 한 위치에서 다른 위치로 로킹(Rocking), 토글, 또는 전환하여 명령을 선택하는 컨트롤로 사용합니다.
- LED — 어떤 조건이 충족되었는지를 나타내는 인디케이터로 사용합니다. LabVIEW는 조건이 충족되었는지 여부에 따라 불을 켜거나 끕니다.
- 라디오 버튼 — 선택 리스트에서 명령을 선택하는 컨트롤로 사용합니다.
- 확인란 — (시스템 팔레트의 경우) 하나의 옵션만 선택할 가능할 때, 참 또는 거짓 상태를 선택하는 컨트롤로 사용합니다.
- 혼합 확인란 — (시스템 팔레트의 경우) 일부는 참이고 일부는 거짓인 여러 객체의 프로퍼티를 동시에 설정할 때와 같이, 참, 거짓, 또는 혼합 상태를 선택하는 컨트롤로 사용합니다.

불리언 컨트롤의 기계적 동작

불리언 컨트롤의 기계적인 동작을 설정하여 오실로스코프나 멀티미터와 같은 물리적 인스트루먼트의 동작과 유사한 프런트패널을 생성할 수 있습니다. 스위치와 래치 동작은 모두 불리언 컨트롤의 값을 변경한다는 점에서 유사합니다. 두 동작은 컨트롤의 값을 이전 값으로 되돌리는 방식에서 차이가 납니다.

- 스위치 동작 — 스위치 동작을 사용하여 오직 사용자만 컨트롤의 값을 이전 값으로 복원시킬 수 있도록 합니다.
- 래치 동작 — 래치 동작을 사용하여 컨트롤을 읽는 VI가 컨트롤의 값을 이전 값으로 되돌릴 수 있도록 합니다. VI가 컨트롤을 읽기 전에는 사용자도 컨트롤의 값을 이전 값으로 되돌릴 수 있습니다.



노트 래치 동작은 로컬 변수와 함께 사용할 수 없습니다. 또한 이벤트 구조와 함께 사용하는 래치 동작의 터미널은 값 변경 이벤트 프레임 내부에 있습니다.

불리언 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 기계적 동작 메뉴에서 동작에 대한 옵션을 선택합니다. 다음 테이블은 스위치와 래치 동작의 3 가지 타입에 대한 더 자세한 정보를 제공합니다:

동작	사용하는 경우	예
누를 때 스위치	클릭하면 현재 값을 변경하고 다시 클릭할 때까지 그 값을 유지하는 컨트롤로 사용합니다.	빛 스위치 — 컨트롤을 전환하면 켜지고 컨트롤을 다시 전환할 때까지 그 상태를 유지합니다.

동작	사용하는 경우	예
놓을 때 스위치	마우스 버튼을 놓을 때 현재 값을 변경하고 다시 클릭할 때까지 새로운 값을 유지하는 컨트롤로 사용합니다.	확인란 — 확인란은 마우스 버튼을 놓은 후에만 값을 변경합니다. 그러므로 마우스 버튼을 놓기 전에는, 마우스를 클릭한 후에도 커서를 확인란 외부로 움직여서 선택할 사항을 다시 고려해 볼 수 있습니다.
놓을 때까지 스위치	마우스 버튼을 클릭한 후 누르고 있는 상태에서 현재 값을 변경하고 마우스 버튼을 놓으면 이전 값으로 되돌리는 컨트롤로 사용합니다.	초인종 — 버튼을 누르면 벨이 즉시 울리고 버튼을 놓을 때까지 계속 울립니다.
누를 때 래치	클릭하면 현재 값을 변경하고 VI가 컨트롤을 읽은 후 이전 값으로 되돌리는 컨트롤로 사용합니다.	긴급 정지 버튼 — 버튼을 누르면 즉시 시스템이 중지되고, 버튼을 다시 누르면 시스템이 값의 변화를 읽는 즉시 리셋됩니다.
놓을 때 래치	마우스 버튼을 놓을 때 현재 값을 변경하고, VI가 컨트롤을 읽은 후에는 이전 값으로 되돌리는 컨트롤로 사용합니다.	달기 버튼 — 이 버튼은 마우스 버튼을 놓은 후, 응용 프로그램이 변경된 값을 읽은 후 프로그램을 닫습니다. 그러므로 마우스 버튼을 놓기 전에는, 마우스를 클릭한 후에도 커서를 버튼의 외부로 움직여서 선택할 사항을 다시 고려해 볼 수 있습니다.
놓을 때까지 래치	마우스 버튼을 클릭한 후 누르고 있는 상태에서 현재 값을 변경합니다. 마우스 버튼을 놓으면, VI가 컨트롤을 읽은 후 이전 값으로 돌아갑니다.	로봇 작동 컨트롤 — 작동 컨트롤이 눌러져 있는 동안 로봇 시스템에 로봇을 작동하도록 명령합니다. 컨트롤을 놓으면, 로봇 시스템은 컨트롤을 읽고 로봇은 작동하지 않는 이전 상태로 돌아갑니다.

동작	사용하는 경우	예
	노트 (FPGA Module) FPGA 타겟에서는 호스트 VI에서 FPGA 타겟으로 새 값이 쓰이면 현재 값을 변경하고, 호스트 VI에서 다른 값이 쓰일 때까지 이 값을 유지하는 컨트롤로 누를 때 스위치, 놓을 때 스위치, 놓을 때까지 스วิต치를 사용합니다. 호스트 VI에서 FPGA 타겟으로 새 값이 쓰일 때 현재 값만 변경하고, FPGA 타겟이 이 새 값 읽은 후에는 다시 이전 값으로 되돌리는 컨트롤로 누를 때 래치, 놓을 때 래치, 놓을 때까지 래치를 사용합니다.	

컨테이너 컨트롤과 인디케이터

컨테이너 및 클래식 컨테이너 팔레트에 위치한 컨테이너 컨트롤을 사용하여 **컨트롤과 인디케이터를 그룹화**하거나, 현재 VI의 프런트패널에 **다른 VI의 프런트패널을 디스플레이**합니다. (Windows) 또한 컨테이너 컨트롤을 사용하여 프런트패널에 **.NET**과 **ActiveX** 객체를 디스플레이할 수 있습니다.

탭 컨트롤

탭 컨트롤을 사용하여 좁은 영역에서 프런트패널 컨트롤과 인디케이터를 겹칩니다. 탭 컨트롤은 페이지와 탭으로 구성됩니다. 탭 컨트롤의 각 페이지에 프런트패널 객체를 놓고 탭을 선택자로 사용하여 다른 페이지를 디스플레이합니다.

탭 컨트롤은 여러 프런트패널 객체를 함께 사용하거나 작업의 특정한 단계에서 사용할 때 유용합니다. 예를 들어 테스트를 시작하기 전 사용자에게 여러 가지 설정을 먼저 설정하도록 요청한 후, 사용자가 진행 중에 테스트의 특성을 수정할 수 있도록 하며, 마지막으로 사용자가 적절한 데이터만을 디스플레이하고 저장할 수 있도록 도와주는 VI를 만들 수 있습니다.

블록다이어그램에서, 탭 컨트롤은 **열거형 타입 컨트롤**입니다. 탭 컨트롤에 놓인 컨트롤과 인디케이터의 터미널은 일반 블록다이어그램 터미널과 동일하게 나타납니다.

서브패널 컨트롤

서브패널 컨트롤을 사용하여 현재 VI의 프런트패널에 다른 VI의 **프런트패널을 디스플레이**합니다. 예를 들어, 서브패널 컨트롤을 사용하여 사용자 인터페이스가 마법사처럼 작동하도록 디자인할 수 있습니다. 이전 또는 다음 버튼을 최상위 레벨 VI의 프런트패널에 놓고 서브패널 컨트롤을 사용하여 마법사의 각 단계마다 다른 프런트패널을 로드할 수 있습니다.

서브패널 컨트롤을 프런트패널에 놓을 때, LabVIEW는 블록다이어그램에 컨트롤의 터미널을 생성하지 않습니다. 대신에, LabVIEW는 VI 삽입 메소드가 선택된 **[인보크 노드]**를 블록다이어그램에 생성합니다. VI를 서브패널 컨트롤에 로드하기 위해서는, VI에 대한 참조를 [인보크 노드]에 연결합니다.



노트

서브패널 컨트롤은 터미널을 가지지 않기 때문에, 서브패널 컨트롤의 배열은 생성할 수 없으며, 서브패널 컨트롤의 타입 정의 또한 생성할 수 없습니다. 서브패널 컨트롤을 클러스터에 놓아서 서브패널 컨트롤을 다른 컨트롤과 함께 그룹화할 수 있지만, 클러스터는 서브패널 컨트롤이나 컨트롤만 포함할 수는 없습니다.

로드하려는 VI의 프런트패널이 열려있거나 같은 프런트패널의 다른 서브패널 컨트롤에서 프런트패널을 로드한 경우, 프런트패널이 **재호출**되지 않는 이상 LabVIEW는 에러를 반환하고, 서브패널 컨트롤에서 프런트패널을 로드할 수 없습니다. 또한, 원격 **어플리케이션 이션 인스턴스**에서 VI의 프런트패널을 로드할 수 없으며, 프런트패널을 반복적으로 로드할 수 없습니다.

실행 중이 아닌 VI를 로드하는 경우, 서브패널 컨트롤의 VI는 편집 모드로 로드됩니다.

LabVIEW는 오직 서브패널 컨트롤에 로드하는 VI의 프런트패널에서 보이는 영역만을 디스플레이합니다. 서브패널 컨트롤을 포함하는 VI 실행을 정지하면, LabVIEW는 서브패널 컨트롤의 프런트패널을 지웁니다. 또한 VI 제거 메소드를 사용하여 서브패널 컨트롤의 VI를 로드 해제할 수 있습니다.

서브패널 컨트롤 사용에 대한 예제는 labview\examples\Controls and Indicators\Containers\Containers.lvproj를 참조하십시오.

LabVIEW에서 끌어서 놓기

VI가 실행하는 중에 컨트롤 사이에서 데이터를 끌어서 놓으려면, 컨트롤이 끌어서 놓기 동작을 허용하도록 설정해야 합니다. 다음 방법을 사용하여 컨트롤 사이에서 끌어서 놓기 동작을 활성화할 수 있습니다:

- 텍스트 기반의 컨트롤 사이: 바로 가기 메뉴를 사용합니다.
- OS에서 경로 컨트롤: 바로 가기 메뉴를 사용합니다.
- 모든 컨트롤 타입 사이: 프로그램적으로 끌어서 놓기 이벤트를 설정합니다.



관련 정보

[텍스트 기반의 컨트롤에서 실행 중 끌어서 놓기 동작 설정하기](#) (1037 페이지 참조)

[텍스트 기반의 컨트롤에서 실행 중 끌어서 놓기 동작 설정하기](#) (1037 페이지 참조)

[사용자 정의된 끌어서 놓기 동작 구현하기](#) (1039 페이지 참조)

블록다이어그램 만들기

프런트패널을 만든 후, 그래픽 형태의 함수를 통해 코드를 추가하여 프런트패널의 객체를 컨트롤합니다. 블록다이어그램은 G 코드 또는 블록다이어그램 코드로 알려진 그래픽 소스 코드를 포함합니다.



관련 정보

[프런트패널 만들기](#) (545 페이지 참조)

[블록다이어그램 객체](#) (52 페이지 참조)

[함수 개요](#) (104 페이지 참조)

[와이어를 사용하여 블록다이어그램 객체에 연결하기](#) (278 페이지 참조)

[블록다이어그램의 데이터 흐름](#) (283 페이지 참조)

[블록다이어그램 디자인하기](#) (277 페이지 참조)

블록다이어그램 객체

블록다이어그램의 객체에는 터미널과 노드가 있습니다. 이 객체들을 와이어로 연결하여 블록다이어그램을 만듭니다. 각 터미널의 색과 기호는 대응하는 컨트롤과 인디케이터의 데이터 타입을 나타냅니다. 상수는 블록다이어그램에 고정 데이터 값을 제공하는 블록다이어그램의 터미널입니다.

블록다이어그램 터미널

프런트패널 객체는 블록다이어그램에서 터미널로 나타납니다. 블록다이어그램의 터미널에서 더블 클릭을 하면, 대응하는 프런트패널의 컨트롤 또는 인디케이터가 하이라이트됩니다.

터미널은 프런트패널과 블록다이어그램 사이에 정보를 교환하는 통로입니다. 프런트패널 컨트롤에 입력한 데이터 값은 컨트롤 터미널을 통해서 블록다이어그램에 입력됩니다. 실행시, 출력 데이터 값은 인디케이터 터미널로 이동하여 블록다이어그램을 빠져나오고, 프런트패널로 다시 들어가서 프런트패널 인디케이터에 나타납니다.

LabVIEW는 구조에 컨트롤과 인디케이터 터미널, 노드 터미널, 상수, 특정한 터미널을 가지고 있습니다. 터미널에 와이어를 연결하고 다른 터미널에 데이터를 전달합니다. 블록 다이어그램 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 아이콘 >> 터미널을 선택하여 터미널을 나타내십시오. 객체에서 마우스 오른쪽 버튼을 클릭한 후 다시 보이는 아이콘 >> 터미널을 선택하여 터미널을 숨깁니다. 바로 가기 메뉴 아이콘은 확장 가능한 VI와 함수에서는 사용할 수 없습니다.

블록 다이어그램에서 프런트패널 컨트롤 또는 인디케이터가 아이콘으로 나타나거나 데이터 타입 터미널로 나타나도록 설정할 수 있습니다. 기본적으로, 프런트패널 객체는 아이콘 터미널로 나타납니다. 예를 들어, 다음에 보이는 노브 아이콘 터미널은 프런트패널의 노브 컨트롤을 나타냅니다.



터미널 아래 부분의 DBL은 배정도 부동소수의 데이터 타입을 나타냅니다. 다음에 보이는 DBL 터미널은 배정도 부동소수 컨트롤을 나타냅니다.



터미널에서 마우스 오른쪽 버튼을 클릭한 후 아이콘으로 보기 바로 가기 메뉴 아이콘 옆에 있는 확인 표시를 제거하여 터미널의 데이터 타입을 디스플레이합니다. 아이콘 터미널을 사용하면 블록 다이어그램에서 프런트패널 객체의 타입을 디스플레이할 수 있고, 동시에 프런트패널 객체의 데이터 타입도 나타낼 수 있습니다. 데이터 타입 터미널을 사용하여 블록 다이어그램의 공간을 절약합니다.



노트

아이콘 터미널은 데이터 타입 터미널보다 크기 때문에, 데이터 타입 터미널을 아이콘 터미널로 변환할 때 의도하지 않게 다른 블록 다이어그램 객체를 가릴 수 있습니다.

컨트롤 터미널은 인디케이터 터미널보다 경계가 더 두껍습니다. 또한 화살표를 통해 터미널이 컨트롤인지 인디케이터인지 알 수 있습니다. 오른쪽에 나가는 화살표가 있는 것이 컨트롤이며, 왼쪽에 들어오는 화살표가 있는 것이 인디케이터입니다.

컨트롤과 인디케이터 데이터 타입

다음 테이블은 컨트롤과 인디케이터 터미널의 서로 다른 데이터 타입 기호와 사용법을 보여줍니다.

각 터미널의 색과 기호는 대응하는 컨트롤과 인디케이터의 데이터 타입을 나타냅니다. 많은 데이터 타입은 데이터를 처리할 수 있는 대응하는 함수의 세트를 가집니다. 예를 들어 문자열 팔레트의 [문자열] 함수는 문자열 데이터 타입에 대응됩니다.

숫자 데이터 타입 사용에 관한 추가적인 정보는 [숫자 데이터 타입](#)을 참조합니다.

컨트롤	인디케이터	데이터 타입	용도	기본값
		단정도 부동소수	메모리를 절약하고 숫자의 범위를 오버플로우하지 않습니다.	0.0
		배정도 부동소수	숫자형 객체의 기본 포맷입니다.	0.0
		확장형 부동소수	플랫폼에 따라 다르게 작동합니다. 필요한 경우에만 사용하십시오.	0.0
		복소수 단정도 부동소수	실수와 허수 부분을 가진 단정도 부동소수와 같습니다.	$0.0 + 0.0i$
		복소수 배정도 부동소수	실수와 허수 부분을 가진 배정도 부동소수와 같습니다.	$0.0 + 0.0i$
		복소수 확장형 부동소수	실수와 허수 부분을 가진 확장된 부동소수와 같습니다.	$0.0 + 0.0i$
		고정 소수점 숫자	사용자가 정의한 범위내의 값을 저장합니다. 이 데이터 타입은 부동소수 형의 다이내믹 범위가 필요하지 않거나, FPGA 타겟과 같이 부동소수 연산이 FPGA 리소스를 많이 사용하는 타겟에서 리소스를 절약하고자 할 때 유용합니다.	0.0
		8비트 부호있는 정수	전체 숫자를 나타내며 양 또는 음의 값이 될 수 있습니다.	0
		16비트 부호있는 정수	위와 같습니다.	0
		32비트 부호있는 정수	위와 같습니다.	0
		64비트 부호있는 정수	위와 같습니다.	0
		8비트 부호없는 정수	음수 아닌 정수만을 나타내며 두 형 모두 비트 수가 같으므로 부호있는 정수보다 더 큰 범위의 양수를 가집니다.	0
		16비트 부호없는 정수	위와 같습니다.	0
		32비트 부호없는 정수	위와 같습니다.	0
		64비트 부호없는 정수	위와 같습니다.	0

컨트롤	인디케이터	데이터 타입	용도	기본값
		<64.64> 비트 타임스탬프	높은 정밀도로 절대 시간을 저장합니다.	1904년 1월 1일 오전 12:00:00.000 (세계시)
		열거형 타입	사용자가 선택할 수 있는 아이템 리스트를 제공합니다.	—
		불리언	불리언(참/거짓) 값을 저장합니다.	거짓
		문자열	정보와 데이터에 플랫폼에 독립적인 포맷을 제공합니다. 이를 사용하여 간단한 텍스트 메시지를 작성하고, 숫자 데이터를 전달 및 저장할 수 있습니다.	빈 문자열
		배열	원소의 데이터 타입을 사각형 괄호 안에 갖게 되고, 그 데이터 타입의 색깔을 가집니다. 배열에 차원을 추가하는 경우, 괄호가 겹쳐집니다.	—
		복소수 원소의 행렬	와이어 패턴은 같은 데이터 타입을 가진 배열의 와이어 패턴과 다릅니다.	—
		실수 원소의 행렬	와이어 패턴은 같은 데이터 타입을 가진 배열의 와이어 패턴과 다릅니다.	—
		클러스터	여러 가지의 데이터 타입을 포함합니다. 클러스터 안의 모든 원소가 숫자형이면 클러스터 데이터 타입이 갈색으로 나타나고, 숫자형 타입이 아닌 원소가 있으면 핑크색으로 나타납니다. 예러 코드 클러스터는 짙은 노란색으로 나타나는 반면, LabVIEW 클래스 클러스터는 기본적으로 빨간색이거나 리포트 생성 VI에 대해 짙은 청록색입니다.	—
		경로	사용하는 플랫폼의 표준 구문을 사용하여 파일 또는 디렉토리의 위치를 저장합니다.	빈 경로
		다이내믹	(익스프레스 VI) 신호와 연관된 데이터와 신호의 이름 또는 데이터가 수집된 날짜와 시간 같은 신호와 연관된 정보를 제공하는 특성을 포함합니다.	—
		웨이브폼	웨이브폼의 데이터, 시작 시간, Δt 를 가집니다.	—
		디지털 웨이브폼	시작 시간, Δx , 디지털 데이터, 디지털 웨이브폼의 모든 특성을 가집니다.	—

컨트롤	인디케이터	데이터 타입	용도	기본값
		디지털	디지털 신호와 연관된 데이터를 포함합니다.	—
		참조 번호(refnum)	파일, 디바이스 또는 네트워크 연결과 같은 객체에 대한 유일한 식별자로 작동합니다.	—
		배리언트	컨트롤 또는 인디케이터 이름, 데이터 타입 정보, 그리고 데이터 자신을 포함합니다.	—
		I/O 이름	인스트루먼트 또는 측정 디바이스와 통신을 할 수 있도록 설정한 리소스를 I/O VI에 전달합니다.	—
		그림	라인, 원, 텍스트, 기타 타입의 그래픽 모양을 가질 수 있는 그림을 디스플레이하는 그리기 해설의 세트를 포함합니다.	—

메모리 사용을 최적화하기 위해서는 일관되고 단순한 데이터 타입을 선택하십시오.

부동소수 데이터의 기호형 숫자값

정의되지 않았거나 예상치 못한 데이터는 모든 이후의 수행을 무효로 만듭니다. 부동소수 연산에서 잘못된 계산 또는 의미 없는 결과가 나오는 경우, 다음의 두 기호 값이 반환됩니다.

- NaN (숫자 아님)은 음수의 제곱근 구하기와 같이 유효하지 않은 연산에서 생성된 부동소수 값을 나타냅니다.
- Inf(무한대)는 데이터 타입 범위 밖의 부동소수 값을 나타냅니다. 예를 들어, 1을 0으로 나누면 Inf가 생성됩니다. LabVIEW는 +Inf 또는 -Inf를 반환할 수 있습니다.

LabVIEW는 정수값의 오버플로우 또는 언더플로우 조건을 확인하지 않습니다.

부동소수의 오버플로우와 언더플로우는 IEEE 754, Standard for Floating-Point Arithmetic을 따릅니다.

부동소수 연산은 NaN과 Inf 데이터를 안정적으로 전달합니다. 그러나 정수와 고정 소수점 수는 기호형 숫자값을 지원하지 않습니다. 명백하게 또는 함축적으로 +Inf를 정수나 고정 소수점 수로 변환할 때, 값은 해당 데이터 타입의 가장 큰 값이 됩니다. 예를 들어, +Inf를 16비트 부호있는 정수로 변환하면 16비트 부호있는 정수의 가장 큰 값인 32,767이 생성됩니다. LabVIEW는 -Inf를 해당 데이터 타입의 가장 작은 값으로 변환합니다.

명백하게 또는 함축적으로 NaN를 정수나 고정 소수점 수로 변환할 때, 값은 해당 데이터 타입의 가장 큰 값이 됩니다.

데이터를 정수 또는 고정 소수점 데이터 타입으로 변환하기 전에, [프로브](#) 도구를 사용하여 중간 부동소수 값이 유효한지 확인합니다. 비교 함수 [숫자/경로/참조 번호 아님?]을 유효하지 않으리라 예상되는 값에 연결하여 NaN을 프로그램적으로 확인합니다. 또한 [범위내 확인과 강제변환] 함수를 사용하여 값을 지정한 범위 내로 유지할 수도 있습니다.



노트(FPGA Module) FPGA의 NaN 출력의 비트 패턴이 개발 컴퓨터의 출력과 다를 수 있습니다.

상수

상수는 블록다이어그램에 고정 데이터 값을 제공하는 블록다이어그램의 터미널입니다. 범용 상수는 원주율(π)과 무한대(∞)와 같이 고정된 값을 가지는 상수입니다. 사용자 정의 상수는 VI를 실행하기 전에 정의하고 편집하는 상수입니다.

상수에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 보이는 항목 $\gg$ 라벨을 선택하여 상수에 라벨을 붙입니다. 범용 상수는 미리 정의된 라벨 값을 가지며, 수행 도구 또는 라벨링 도구를 사용하여 편집할 수 있습니다.

대부분의 상수는 팔레트의 맨 위 또는 맨 아래에 위치합니다.

범용 상수

수학적인 계산과 문자열 또는 경로를 포맷할 때에는 범용 상수를 사용합니다. LabVIEW는 다음과 같은 타입의 범용 상수를 포함합니다:

- 범용 숫자형 상수—자연로그 밑(e)과 광속과 같이 수학과 물리값에 일반적으로 사용되는 정밀한 자릿수의 세트. 범용 숫자형 상수는 수학 & 과학 상수 팔레트에 위치합니다.
- 범용 문자열 상수—라인 피드와 캐리지 리턴과 같이 일반적으로 사용되는 디스플레이할 수 없는 문자열 문자의 세트. 범용 문자열 상수는 문자열 팔레트에 위치합니다.
- 범용 파일 상수—경로 없음, 참조 번호 아님, 기본 디렉토리 등과 같이 일반적으로 사용되는 파일 경로 값의 세트. 범용 파일 상수는 파일 상수 팔레트에 위치합니다.

사용자 정의 상수

함수 팔레트에는 불리언, 숫자, 링, 열거형 타입, 색 상자, 문자열, 배열, 클러스터, 경로 상수와 같은 타입에 의해 구성된 상수가 있습니다.

VI 또는 함수의 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 상수 $\gg$ 생성을 선택하여 사용자 정의 상수를 생성합니다.

VI가 실행 중일 때에는 사용자 정의 상수의 값을 변경할 수 없습니다.

또한 프런트패널 컨트롤을 블록다이어그램으로 끌어서 상수를 생성할 수 있습니다. LabVIEW는 프런트패널 컨트롤을 블록다이어그램으로 끌 때 프런트패널 컨트롤의 값을 가지는 상수를 생성합니다. 프런트패널 컨트롤은 프런트패널에 남습니다. 컨트롤의 값을 변경하는 것은 상수 값에 영향을 주지 않으며, 그 반대도 마찬가지입니다.

수행 도구 또는 라벨링 도구를 사용하여 상수를 클릭하고 그 값을 편집합니다. **자동 도구 선택이 활성화**된 경우, 상수를 더블 클릭하여 라벨링 도구로 바꾼 후 값을 편집합니다.

상수 계산 단순화 및 루프 불변값

LabVIEW는 상수 계산 단순화와 루프 불변값을 사용하여 VI의 성능을 최적화합니다.

- 상수 계산 단순화를 사용하여, LabVIEW는 매 실행 시 상수 값의 처리 결과를 계산하는 대신 첫 번째 VI 실행 시 그 결과를 계산합니다. 모든 입력이 상수인 For 루프와 같은 구조에 상수 계산 단순화를 적용하면, LabVIEW는 VI를 실행할 때마다 구조의 출력을 계산하는 대신, 첫 번째 VI 실행 시 구조를 실행하고 구조의 출력을 계산합니다.

노트 디버깅을 활성화하면 LabVIEW는 구조에 대해 상수 계산 단순화를 사용하지 않습니다.

- 루프 불변값은 For 루프 또는 While 루프가 계산하는 상수 아닌 값으로, 이는 루프의 반복에 따라 변경되지 않습니다. 예를 들어, For 루프 내의 [더하기] 함수가 인덱싱되지 않는 입력 터널에만 연결되어 있는 경우 [더하기] 함수의 출력이 루프의 반복에 따라 변경되지 않기 때문에 이는 루프 불변값이 됩니다. LabVIEW는 루프가 반복될 때마다 루프 불변값을 계산하는 대신 루프가 실행되기 전에 이를 계산합니다.

도구 >> 옵션 >> 블록다이어그램을 선택한 후 블록다이어그램 페이지에서 해쉬 마크 옵션을 설정하면 LabVIEW가 블록다이어그램에 상수 계산 단순화 및 루프 불변값 해쉬 마크를 디스플레이하도록 설정할 수 있습니다.

블록다이어그램 노드

노드는 입력과 출력 중 하나만 있거나 입출력이 모두 있는 블록다이어그램의 객체이며 VI가 실행될 때 동작합니다. 노드는 텍스트 기반 프로그래밍 언어에서의 명령문(statement), 연산자(operator), 함수(function) 및 서브루틴(subroutine)과 유사한 개념입니다. LabVIEW는 다음과 같은 타입의 노드를 가집니다:

- **함수**—연산자, 함수 또는 명령문과 비교 가능한 내장된 실행 원소.
- **SubVI**—서브루틴과 비슷한, 다른 VI의 블록다이어그램에서 사용되는 VI.
- **익스프레스 VI**—일반적인 측정 작업을 수행하도록 디자인된 SubVI. 설정 대화 상자를 사용하여 익스프레스 VI를 설정할 수 있습니다.
- **구조**—For 루프, While 루프, 케이스 구조, 폴렛 시퀀스 구조와 다층 시퀀스 구조, Timed 구조, **이벤트 구조**처럼 실행을 컨트롤하는 원소.
- **수식 노드와 식 노드**—수식 노드는 블록다이어그램에 직접 방정식을 입력할 수 있는 크기 조절이 가능한 구조입니다. 식 노드는 단일 변수를 포함하는 식을 계산하는 구조입니다.

- **프로퍼티 노드와 인보크 노드**—프로퍼티 노드는 클래스의 프로퍼티를 설정하거나 찾는 구조입니다. 인보크 노드는 클래스의 메소드를 실행하는 구조입니다.
- **참조에 의한 호출 노드**—동적으로 로드된 VI를 호출하는데 사용하는 구조입니다.
- **라이브러리 함수 호출 노드**—대부분의 표준 공유 라이브러리 또는 DLL을 호출하는데 사용하는 구조입니다.

다형성 VI와 함수

다형성 VI와 함수는 다른 데이터 타입의 입력 데이터에 따라 조정됩니다. 대부분의 LabVIEW 구조는 일부 VI 및 함수와 같이 다형성입니다.

함수의 **다형성** 정도는 다양하며, 다형성 입력이 없거나, 일부 또는 전체 입력이 다형성일 수 있습니다. 일부 함수의 입력은 숫자값과 불리언 값을 받습니다. 일부는 숫자값 또는 문자열을 받습니다. 또한 몇몇은 스칼라 숫자값 뿐만 아니라 숫자값의 배열, 숫자값의 클러스터, 숫자값의 클러스터 배열 등을 받습니다. 어떤 함수 입력은 오직 1차원 배열만을 받을 수 있으며, 이때 배열 원소는 모든 타입이 될 수 있습니다. 일부 함수는 복소수 값을 포함한 모든 타입의 데이터를 받습니다.

텍스트 특성

LabVIEW는 컴퓨터에 이미 설치된 폰트를 사용합니다. 도구 모음의 텍스트 셋팅 풀다운 메뉴를 사용하여 텍스트의 속성을 변경합니다.

텍스트 셋팅 풀다운 메뉴에는 다음과 같은 내장 폰트가 있습니다.

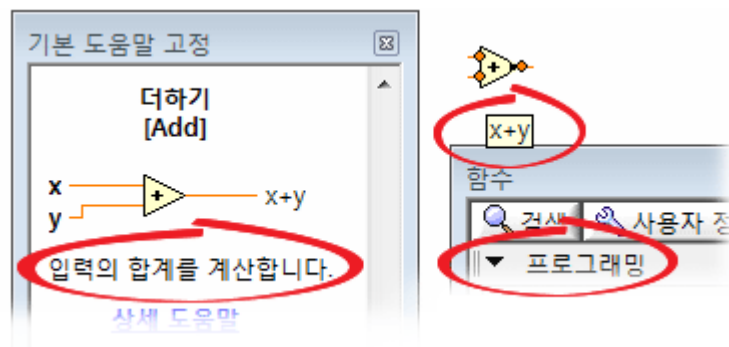


노트

다음 스크린 샷은 기본 폰트를 사용하지 않고, 대신에 다른 폰트를 사용하여 어플리케이션, 시스템, 대화 상자 폰트를 변경함으로써 어디에서 내용이 영향을 받는지 나타내고 있습니다.

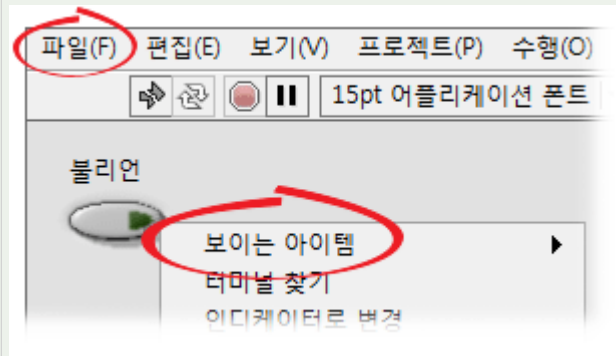
어플리케이션 폰트 — 다음 위치에 있는 텍스트에 대해 사용하는 기본 폰트:

- 컨트롤 팔레트
- 함수 팔레트
- 기본 도움말
- 팁 상자



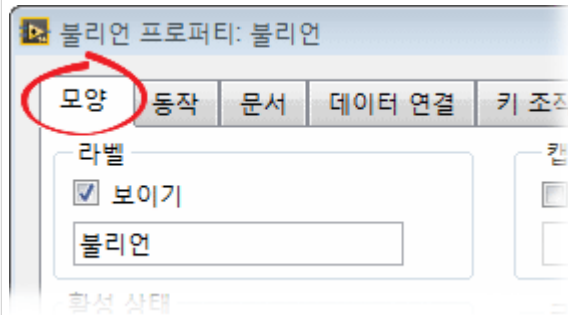
시스템 폰트 — 다음 위치에 있는 텍스트에 대해 사용하는 기본 폰트:

- 메뉴 모음
- 바로 가기 메뉴



대화 상자 폰트 — 다음 위치에 있는 텍스트에 대해 사용하는 기본 폰트:

- 대화 상자
- 시스템 컨트롤



텍스트 셋팅 풀다운 메뉴에서 선택을 하기전에 객체 또는 텍스트를 선택한 경우, 선택한 모든 것들에 변경이 적용됩니다. 선택한 것이 없을 경우, 변경은 기본 폰트에 적용됩니다. 기본 폰트를 변경해도 기존 라벨의 폰트는 변경되지 않습니다. 오직 이후부터 생성하는 라벨에만 영향을 줍니다.

프론트패널에서 텍스트 셋팅 풀다운 메뉴의 폰트 대화 상자를 선택하여 선택한 텍스트에 특정 폰트 스타일을 적용합니다. 텍스트를 선택하지 않은 경우, 기본 패널 옵션에 확인 표시가 있을 것입니다. 블록다이어그램에서 객체를 선택하지 않고 텍스트 셋팅 > 폰트 대화 상자를 선택한 경우, 기본 다이어그램 옵션에 확인 표시가 있을 것입니다. 프론트패널과 블록다이어그램에서 서로 다른 폰트를 설정할 수 있습니다. 예를 들어, 블록다이어그램에는 작은 폰트를 설정하고 프론트패널에는 큰 폰트를 설정할 수 있습니다.

내장 폰트를 포함하는 VI를 **다른 플랫폼**으로 옮기면 폰트는 가장 가까운 폰트로 바뀝니다.



팁 텍스트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 텍스트 크기에 맞춤을 선택하여 플랫폼 간의 폰트 차이에 따라 자동적으로 조정되도록 설정합니다.

텍스트 셋팅 풀다운 메뉴는 또한 크기, 스타일, 자리맞춤, 색 서브메뉴 아이템을 가집니다.

이 서브메뉴에서 선택한 폰트 설정은 선택된 객체에 적용됩니다. 예를 들어, 노브와 그래픽이 선택된 상태에서 새로운 폰트를 선택할 경우, 라벨, 스케일, 디지털 디스플레이가 모두 새 폰트로 바뀝니다.

LabVIEW는 사용자가 변경하려고 할 때 가능한 많은 폰트 속성을 유지합니다. 예를 들어, 굵은체의 Courier 폰트를 사용하는 객체를 선택한 후 텍스트 셋팅 풀다운 메뉴에서 Arial

을 선택하는 경우에는 굵은체 폰트 속성이 유지되어, 굵은체 Arial 폰트가 됩니다. 선택한 객체를 설정하기 위해 폰트 스타일 대화 상자를 사용하는 경우, LabVIEW는 사용자가 객체에 대해 선택한 텍스트의 특성을 적용시킵니다. 텍스트 특성에는 폰트, 크기, 정렬, 색, 폰트 속성이 포함되어 있습니다. 선택한 객체에 대해 어플리케이션 폰트, 시스템 폰트, 대화 상자 폰트 등의 내장된 폰트 중에 하나를 선택하는 경우, LabVIEW는 해당 내장 폰트의 텍스트 특성을 객체에 적용합니다.

슬라이드와 같이 여러 개의 텍스트를 가지는 객체로 작업하는 경우, 폰트 변경은 현재 선택된 객체 또는 텍스트에 적용됩니다. 예를 들어, 전체 슬라이드를 선택하고 텍스트 셋팅 풀다운 메뉴에서 스타일 >> 굵은체를 선택할 경우, 스케일, 디지털 디스플레이, 라벨은 모두 굵은체로 바뀝니다. 오직 라벨만 선택하고 굵은체를 선택한 경우, 라벨만 굵은체로 바뀝니다. 스케일 마커에서 텍스트를 선택하고 굵은체를 선택한 경우, 모든 마커는 굵은체로 바뀝니다.

일반된 표준 폰트를 사용하여 사용하기 쉬운 사용자 인터페이스를 디자인해야 합니다.

숫자형 데이터

LabVIEW는 숫자형 데이터를 **부동소수**, **고정 소수점**, **정수**, **부호없는 정수** 및 **복소수**로 나타냅니다.



노트

이러한 숫자형 데이터 타입 사이의 차이는 **데이터를 저장**하는데 사용하는 비트 수와 나타내는 데이터 값입니다.

또한 일부 데이터 타입은 확장된 설정 옵션을 제공합니다. 예를 들어, 물리적 측정 단위를 복소수를 포함하여 부동소수 데이터와 연계하고, 고정 소수점 데이터에 대한 **인코딩과 범위를 설정**할 수 있습니다.

숫자 단위와 부동소수 연관하기

미터 또는 km/sec와 같은 **물리적 측정 단위**를 부동소수 형을 가지는 모든 숫자형 컨트롤 또는 인디케이터에 결합할 수 있습니다.

컨트롤의 단위는 단위 라벨로 불리는 분리된 고유 라벨에 나타납니다. 컨트롤에 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 단위 라벨을 선택하여 단위 라벨을 디스플레이합니다. 단위 라벨에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 단위 문자열 만들기를 선택하여 단위 라벨을 편집합니다.

LabVIEW가 단위 라벨을 디스플레이할 때, 미터는 m, 피트는 ft, 초는 s 등과 같이 **표준 약자**를 사용하여 단위를 입력할 수 있습니다.

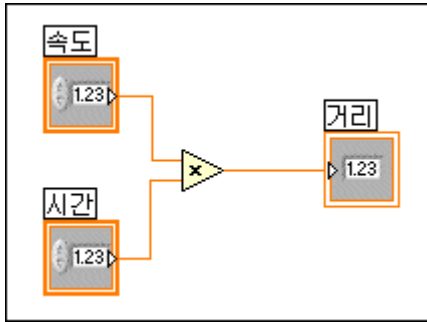


노트 수식 노드에서는 단위를 사용할 수 없습니다.

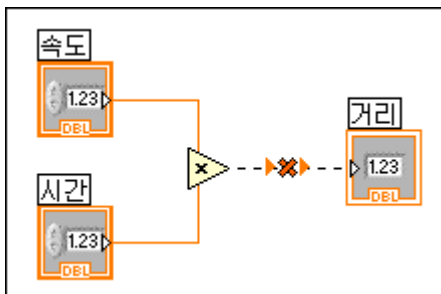
단위와 엄격한 타입 확인

단위를 객체에 연결할 때, 오직 호환 가능한 단위를 가진 객체만 연결할 수 있습니다. LabVIEW는 엄격한 타입 확인을 사용하여 단위가 호환 가능한지를 확인합니다. 호환되지 않는 단위를 가진 두 객체를 연결하면, LabVIEW는 에러를 반환합니다. 예를 들어, LabVIEW는 마일 단위의 객체와 리터 단위의 객체를 연결하면 에러를 반환합니다. 왜냐하면, 마일은 거리의 단위이고 리터는 부피의 단위이기 때문입니다.

다음 VI는 호환 가능한 단위를 가진 객체의 연결을 보여줍니다. 이 VI에서, LabVIEW는 자동으로 거리 인디케이터를 스케일하여 미터 대신에 킬로미터를 디스플레이합니다. 왜냐하면 킬로미터가 인디케이터의 단위이기 때문입니다.

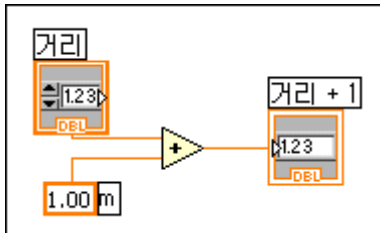


다음 VI에서는 거리가 초의 단위 타입을 가지므로 에러가 발생합니다. 에러를 수정하기 위해서는, 초를 이전 VI에서 보이는 것처럼 킬로미터와 같은 거리의 단위로 변경합니다.



일부 VI와 함수는 단위가 모호한 경우가 있습니다. 이런 VI와 함수는 단위를 가진 다른 터미널에 연결할 수 없습니다. 예를 들어, [증가] 함수는 단위가 모호합니다. 거리 단위를 사용할 때, 증가 함수는 1 미터를 더할지, 1 킬로미터를 더할지, 또는 1 피트를 더할지 명확하지 않습니다. 이 불확정성 때문에, 증가 함수나 다른 함수를 단위를 가진 데이터에 사용할 수 없습니다.

이 예제에서 모호성을 피하기 위해서는, 적절한 단위를 가지는 숫자형 상수와 [더하기] 함수를 사용하여 다음 블록 다이어그램에서와 같이 고유의 단위 증가 함수를 생성합니다.



고정 소수점 데이터 타입

고정 소수점 데이터 타입은 2진수 비트를 사용하여 유리수 세트를 나타내는 숫자형 데이터 타입입니다. 데이터가 다양한 정확도와 범위를 갖도록 하는 **부동소수** 데이터 타입과는 달리, 고정 소수점 수는 항상 특정한 정수와 비트 수를 사용하도록 설정할 수 있습니다.



노트 고정 소수점 데이터 타입을 사용하여 유리수를 나타내려면 2진수는 밑이 2인 시스템이므로 유리수의 분모가 2의 거듭제곱으로 작아질 수 있어야 합니다.

부동소수 형의 다이내믹 범위가 필요하지 않거나 부동소수 연산을 지원하지 않는 타겟에서 작업하려는 경우, 고정 소수점 데이터 타입을 사용하십시오.



노트 (FPGA Module) FPGA 타겟에서 고정 소수점 데이터를 사용하는 것에 대한 더 자세한 정보는 고정 소수점 데이터 타입 사용하기를 참조하십시오. 숫자형 데이터 타입의 용도를 고려하여, 설계에 가장 적합한 데이터 타입을 결정해야 합니다.

고정 소수점 데이터 타입에서 유리수 나타내기

고정 소수점 수는 고정된 정수와 소수 비트를 가집니다. 정수 비트는 2진 포인트의 왼쪽에서, 소수 비트는 2진 포인트의 오른쪽에 나타납니다. 사용하려는 정확한 비트 수를 지정할 수 있으므로, 소수점과 동일한 2진인 2진 포인트에 고정된 위치가 있습니다.

예를 들어, LabVIEW를 설정하여 유리수 0.5를 8개의 총 비트와 4개의 정수 비트를 가진 고정 소수점 수로 나타낼 수 있습니다.

유리수	대응하는 고정 소수점
0.5	0000.1000

또는 LabVIEW를 설정하여 동일한 유리수를 16개의 총 비트와 8개의 정수 비트를 가진 고정 소수점 수로 나타낼 수 있습니다.

유리수	대응하는 고정 소수점
0.5	00000000.10000000

부동소수에서 작업을 수행할 때, 부동소수는 정수와 소수 부분이 변화할 수 있다는 점에서 고정 소수점 수와 차별화됩니다. 즉, 2진 포인트는 부동소수에서 변할 수 있습니다.



노트 유리수 값에 상응하는 고정 소수점 수를 나타내는데 사용할 정확한 비트 수를 지정하지 않으면 LabVIEW는 가능한 데이터 손실을 방지하도록 비트 수를 조절합니다. LabVIEW는 64비트 이상의 수를 처리할 수 없습니다.

고정 소수점 수 설정하기

숫자를 고정 소수점 형으로 설정하려면, 숫자형 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 형을 선택하여 객체의 데이터 타입을 변경합니다. 고정 소수점 수에 대해 **인코딩을 설정**할 수 있습니다. 또한 고정 소수점 수에서 **오버플로우 상태를 포함**할지의 여부를 지정하고, 숫자형 함수가 어떻게 고정 소수점 수에 대해 **오버플로우와 반올림** 조건을 처리할지 지정할 수 있습니다. 고정 소수점 수를 설정하려면, 상수, 컨트롤, 인디케이터 또는 숫자형 함수에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택하여 숫자형 프로퍼티, 숫자형 상수 프로퍼티, 또는 숫자형 노드 프로퍼티 대화 상자를 디스플레이합니다.

인코딩

숫자가 특정한 비트 크기에 부합하게 하려면, 고정 소수점 수의 2진 인코딩을 지정합니다.

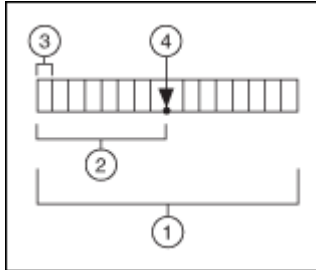
- 부호있음—이 옵션은 고정 소수점 데이터에 부호가 있는지의 여부를 결정합니다. 부호있음을 선택하면 데이터를 나타내는 비트 문자열의 첫 번째 비트가 부호를 나타내는 비트가 됩니다.
- 부호없음—이 옵션은 고정 소수점 데이터에 부호가 없는지의 여부를 결정합니다.
- 워드 길이—LabVIEW가 고정 소수점 데이터의 모든 가능한 값을 나타낼 때 사용하는 비트 문자열의 총 비트 수입니다. LabVIEW가 수용하는 최대 워드 길이는 64비트입니다. 일부 타겟에서는 **데이터가 보다 짧은 워드 길이로 제한**될 수 있습니다. 타겟이 수용할 수 있는 워드 길이보다 더 긴 워드 길이의 고정 소수점 데이터가 있는 VI를 타겟에서 열면, VI의 와이어가 깨지게 됩니다. 타겟이 허용하는 최대 워드 길이에 대한 정보는 각 타겟에 대한 문서를 참조하십시오.
- 정수부분 워드 길이—LabVIEW가 고정 소수점 데이터의 모든 가능한 값을 나타낼 때 사용하는 비트 문자열의 정수 비트의 개수, 또는 최상위 비트의 왼쪽이나 오른쪽에 초기 위치가 주어질 때 2진 포인트를 옮겨서 최상위 비트가 되도록 하는 비트의 개수입니다. 정수부분 워드 길이는 워드 길이보다 클 수 있고, 양수 또는 음수가 될 수 있습니다.

LabVIEW는 고정 소수점 데이터를 전달하는 와이어에서 커서를 이동하면 기본 도움말 원도우에서 다음 포맷으로 이같은 설정을 나타냅니다:

<인코딩,워드 길이,정수부분 워드 길이>

예를 들면:

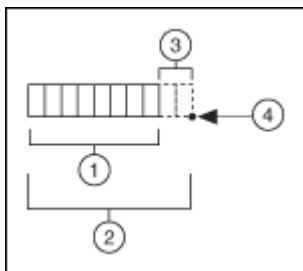
<s, 16, 8>는 8개의 정수 비트를 가진 부호있는, 16비트 수입니다. 다음 이미지는 워드 길이가 16, 정수부분 워드 길이가 8인 부호 있는 수를 나타냅니다.



1	워드 길이 (16비트)
2	정수부분 워드 길이 (8비트)
3	부호 비트
4	2진 포인트

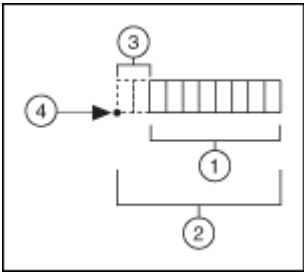
롱 또는 음수 정수부분 워드 길이

정수부분 워드 길이가 워드 길이보다 크면, LabVIEW는 워드 길이를 초과하는 정수 비트를 저장하지 않습니다. 다음 이미지는 워드 길이가 8, 정수부분 워드 길이가 10인 수를 나타냅니다.



1	워드 길이 (8비트)
2	정수부분 워드 길이 (10비트)
3	저장되지 않는 정수 비트
4	2진 포인트

정수부분 워드 길이가 음수이면 LabVIEW는 정수 비트 뿐만 아니라 2진 포인트에서 시작하는 음수에 상응하는 소수 비트 수도 저장하지 않습니다. 다음 이미지는 워드 길이가 8, 정수부분 워드 길이가 -2인 수를 나타냅니다.



1	워드 길이 (8비트)
2	워드 길이 - 정수부분 워드 길이 (8 비트 빼기 -2 비트)
3	저장되지 않는 소수 비트
4	2진 포인트

범위

LabVIEW는 고정 소수점 수의 인코딩에서 지정한 값에 따라 고정 소수 점 수의 범위와 증분을 계산합니다.

- 최소값—고정 소수점 데이터 범위의 최소값입니다.
- 최대값—고정 소수점 데이터 범위의 최대값입니다.
- 증분—지정한 범위 내에 있는 숫자 사이의 증가값입니다.

LabVIEW는 고정 소수점 데이터를 전달하는 와이어에서 커서를 이동하면 기본 도움말 원도우에서 다음 포맷으로 이같은 설정을 나타냅니다:

[최소, 최대]: 델타

예를 들면:

[-4, 4]:1은 최소값 -4, 최대값 4 그리고 델타 1을 가집니다.

고정 소수점 연산

워드 길이를 지정한 고정 소수점 수에 대해 산술 연산을 수행하면, VI를 실행하기 전에 지정한 위치에 비해 2진 포인트의 위치가 변하지 않습니다. 이러한 동작은 산술 연산에 의해 2진 포인트 또는 소수점이 런타임에서 변경될 수 있는 부동소수와는 다릅니다. 산술 연산 결과에서 고정 소수점 설정이 허용하는 워드 길이보다 큰 워드 길이가 필요한 경우, 연산 결과로 얻게되는 데이터를 손실할 수 있습니다.

따라서 LabVIEW는 가능할 경우, 숫자형 함수의 출력값 워드 길이를 조절하여 데이터 손실을 방지합니다. 그러나, LabVIEW는 출력값을 나타내는 워드 길이가 지정한 최대값이나 LabVIEW 최대값 64비트를 초과할 경우, 데이터 손실을 방지할 수 없습니다. 숫자형 함수의 **출력 셋팅을 설정**하려면, 함수에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 숫자형 노드 프로퍼티 대화 상자를 디스플레이합니다.

오버플로우 및 반올림

오버플로우 및 반올림 조건은 고정 소수점 수에서 산술 연산을 수행하거나 [고정 소수점으로] 함수를 사용하여 숫자형 데이터를 고정 소수점 데이터로 변환할 때 발생할 수 있습니다. LabVIEW는 숫자형 함수의 출력을 설정하여, 가능한 경우 고정 소수점 데이터의 오버플로우 또는 반올림을 방지합니다. 그러나 일부 조건에서 LabVIEW는 오버플로우 또는 반올림을 방지할 수 없습니다.

예를 들어, 고정 소수점 값을 더하기, 빼기, 곱하기, 제곱과 같은 숫자형 함수에 연결하는 경우, LabVIEW는 출력값의 워드 길이를 증가시켜 연산의 결과를 수용하는 것이 기본 설정입니다. 이같은 작동은 부동소수 값의 작동방법과 유사합니다. 그러나 LabVIEW가 수용하거나 사용자가 지정하는 최대 워드 길이를 초과하는 출력값이 오버플로우 또는 반올림 방지 시에 필요한 경우, 오버플로우 또는 반올림 조건이 발생할 수 있습니다. LabVIEW가 수용하는 최대 워드 길이는 64 비트입니다.



노트 [나누기] 함수, [역수] 함수 또는 [제곱근] 함수에 고정 소수점 값을 연결하면 그 결과의 정밀도가 무한대가 될 수 있기 때문에 반올림 조건이 항상 발생합니다.

또한 밀이 2인 시스템에서 정확히 **나타낼 수 없는 타입 사이클과 값**에서도 오버플로우 또는 반올림이 발생할 수 있습니다.

오버플로우

연산의 결과가 출력 타입의 값의 범위내에 있지 않으면 오버플로우가 발생합니다. 다음의 오버플로우 모드에서 선택하여 오버플로우를 처리할 수 있습니다:

- 임계값 사용—값이 출력 타입의 원하는 범위 내 최대값보다 큰 경우, LabVIEW는 값을 지정한 최대값으로 강제 변환하거나 반올림합니다. 값이 출력 타입의 원하는 범위내 최소값보다 작은 경우, LabVIEW는 값을 지정한 최소값으로 강제 변환합니다.
- 유효자리 줄이기—값이 출력 타입의 원하는 범위내에 있지 않은 경우, LabVIEW는 값이 출력 타입의 원하는 범위내에 있게 될 때까지 값의 유효자리 비트를 버립니다. 이 옵션을 사용하려면 출력값이 사용자가 지정한 인코딩에 부합해야 합니다. 이 옵션을 선택한 경우, 원하는 범위를 지정할 수 없습니다. LabVIEW가 자동으로 인코딩에 맞도록 범위를 설정하기 때문입니다.



노트 일반적으로, 임계값 사용 모드는 FPGA에서 추가적인 하드웨어 리소스를 요청합니다.

오버플로우의 발생 여부를 구분하기 위해, 오버플로우 상태를 포함하도록 **고정 소수점 수를 설정**할 수 있습니다. 고정 소수점 수에 오버플로우 상태를 포함하면, LabVIEW는 추가적인 저장 공간을 할당하여 고정 소수점 수가 오버플로우된 작업의 결과인지 확인할

수 있도록 합니다. 오버플로우 상태를 포함하도록 고정 소수점 수를 설정한 이후, 고정 소수점 컨트롤, 상수, 인디케이터에 **오버플로우 상태 LED를 디스플레이**할 수 있습니다. 고정 소수점 수의 오버플로우 상태가 참일 때 이 LED가 켜집니다. 또한 고정 소수점 오버플로우? 함수를 사용하여 고정 소수점 수의 오버플로우 상태를 구분할 수 있습니다. 고정 소수점 함수를 사용하여 고정 소수점 수의 오버플로우 상태를 조정합니다.



노트 LabVIEW는 고정 소수점 수에 대해 언더플로우 조건을 확인하지 않습니다.

[비교] 함수를 사용하여 고정 소수점 수를 비교할 때, LabVIEW는 오버플로우 상태를 무시하고 숫자만을 비교합니다. 예를 들어, `같음?` 함수를 사용하여 오버플로우 상태가 참인 고정 소수점 값 1.5를 오버플로우 상태가 아닌 고정 소수점 값 1.5와 비교하는 경우, LabVIEW는 참을 반환합니다.

숫자형 함수의 **출력 설정을 설정**하려면, 함수에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 숫자형 노드 프로퍼티 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 LabVIEW의 오버플로우 및 반올림 조건 처리 방식을 설정할 수 있습니다.

반올림

입력값 또는 연산 결과의 정밀도가 출력 타입의 정밀도보다 큰 경우에 반올림이 발생합니다. LabVIEW는 입력값을 출력 타입과 동일한 정밀도를 가진 값으로 강제 변환합니다. 예를 들어, 출력 타입이 연속적인 값 s 와 t 를 나타낸다고 가정해 봅시다. 연산 결과가 값 x 이며, x 가 $s < x < t$ 에서처럼 s 와 t 사이에 있는 경우, 출력 타입이 x 를 정확히 나타낼 수 없기 때문에 반올림이 발생합니다.

다음 반올림 모드에서 선택하여 반올림을 처리할 수 있습니다:

- **내림(Truncate)**—출력 타입이 표현할 수 있는 가장 가까운 값으로 내림합니다. LabVIEW는 값의 최하위 비트를 버립니다. 이 반올림 모드는 성능면에서는 가장 효율적이지만 출력값의 정확도는 다소 떨어집니다. 예를 들어, 출력 값 x 가 $s < x < t$ 가 성립되도록 연속적인 값 s 와 t 사이에 있는 경우, LabVIEW는 x 를 s 와 동일한 값으로 설정합니다.
- **반올림(Round-half-up)**—출력 타입이 표현할 수 있는 가장 가까운 값으로 반올림합니다. 반올림할 값이 두 값의 정중앙에 위치하는 경우, 이 모드는 두 개의 유효값 중에서 더 높은 값으로 반올림합니다. LabVIEW는 최하위 비트의 반을 출력값에 추가한 다음, 이 값을 내림합니다. 이 반올림 모드는 잘라버림 모드보다는 출력값이 정확하지만 성능에 미치는 영향도 큼니다. 예를 들어, 출력 값 x 가 $s < x < t$ 가 성립되도록 연속적인 값 s 와 t 사이에 있는 경우, LabVIEW는 x 를 s 또는 t 중 더 가까운 값으로 설정합니다. x 가 정확히 s 와 t 사이에 있다면 LabVIEW는 t 를 선택합니다.
- **반올림(Round-half-even)**—출력 타입이 표현할 수 있는 가장 가까운 값으로 반올림합니다. 반올림할 값이 두 값의 정중앙에 위치하는 경우, LabVIEW는 반올림 이후에 최하위 비트가 되는 값의 비트를 확인합니다. 비트가 0인 경우, 이 모드는 값을 출력 타입이 표현할 수 있는 두 값 중에서 더 낮은 값으로 반올림합니다. 비트가 0이 아닌 경우, 이 모드는 값을 둘 중에서 높은 값으로 반올림합니다. 이 반올림 모드는 성능에 가장 큰 영향을 미치지만 출력값은 잘라버림 모드에서 보다 정확합니다. 이 모드는 반올림(Round-half-up) 모드를 사용하는 여러 반올림 작업을 실행할 때 더 높은 값에 치중하게 되는 현상을 상쇄시킵니다. 이 모드가 기본 반올림 모드입니다. 예를 들어, 출력 값 x 가 $s < x < t$ 가 성립되도록 연속적인 값 s 와 t 사이에 있는 경우, LabVIEW는 x 를 s 또는 t 중 더 가까운 값으로 설정합니다. x 가 s 와 t 의 정

중앙에 위치하는 경우, 반올림한 후의 최하위 비트(LSB)가 0이면 s를 선택합니다. 반올림 후의 최하위 비트가 0이 아니면 t를 선택합니다.



노트 반올림 모드에 따른 성능의 영향은 최소입니다. 그러나 FPGA와 같은 타겟에서 오버플로우를 처리하는 VI를 실행하면 성능에 대한 영향이 클 수 있습니다.

FPGA 성능에 미치는 영향에 대한 추가적인 정보는 고정 소수점 데이터 타입 사용하기를 참조합니다.

다형성 단위 사용하기

단위를 사용하여 차원 분석으로도 알려진 식과 수식을 계산할 때 추가적으로 일관성을 확인할 수 있습니다. LabVIEW에서는 데이터를 디스플레이할 때 또는 데이터를 입력할 때 단위 변환을 LabVIEW가 처리하므로 단위 시스템간의 변환에 대해서 염려하지 않아도 됩니다.

[더하기]와 [곱하기]와 같이 LabVIEW에 내장된 함수는 단위와 관련하여 다형성이며 서로 다른 단위를 자동적으로 처리합니다. 하지만, 동일한 다형성 단위 기능을 가지는 subVI를 만들기 위해서는 반드시 **다형성 단위를 사용**해야 합니다.

입력에서 받은 단위와 상관없이 한 VI가 동일한 계산을 수행하기 위해서 다형성 단위를 사용할 수 있습니다. 예를 들어, 웨이브폼의 RMS 값을 계산하는 VI를 생성하려는 경우, 웨이브폼에 연계된 단위를 반드시 정의해야 합니다. 전압 웨이브폼, 전류 웨이브폼, 온도 웨이브폼 등에서는 독립된 VI가 필요합니다. 하지만, 각 케이스에 대해 같은 VI를 다시 만들지 않고, 다형성 단위를 갖는 단일 subVI를 만들어 특정 단위를 가진 VI로부터 호출할 수 있습니다.

LabVIEW는 다형성 단위를 유일한 단위로 처리합니다. 다형성 단위는 다른 단위로 변환할 수 없으며 다른 단위처럼 블록 다이어그램을 통하여 전달합니다. 다형성 단위 \$1을 가진 컨트롤을 역시 다형성 단위 \$1을 가진 인디케이터에 연결할 때, 단위는 일치하며 해당 VI는 컴파일할 수 있습니다.

다른 단위와 마찬가지로 조합에 \$1을 사용할 수 있습니다. 예를 들어, 컨트롤에 3초를 곱하고 인디케이터에 연결하는 경우, 인디케이터는 \$1 s단위여야 합니다. 컨트롤과 다른 단위를 가진 인디케이터를 연결하면 와이어가 깨집니다.

다형성 단위를 포함하는 subVI를 호출하면 입력이 받은 단위를 기초로 출력 단위를 계산합니다. 예를 들어, 다형성 단위 \$1과 \$2의 두 입력을 가진 subVI를 생성하며, 이는 \$1\$2/s 형태의 출력을 생성한다고 가정합니다. m/s 입력을 가진 subVI를 \$1 입력에 연결하고 kg을 \$2 입력에 연결하는 경우, LabVIEW는 출력 단위를 kg m/s^2로 계산합니다.

다른 VI가 \$1과 \$1/s의 두 입력을 가지고 \$1^2의 출력을 계산한다고 가정합니다. m/s 입력을 가진 이 VI를 \$1 입력에 연결하고 m/s^2를 \$1/s 입력에 연결하는 경우, LabVIEW는 출력 단위를 m^2/s^2로 계산합니다. 그러나, m 입력을 가진 이 VI를 \$1 입력에 연결하고 kg을 \$1/s 입력에 연결하는 경우, subVI 호출은 깨집니다. LabVIEW는 입력 중 하나를 단위 충돌로 정의하고 (가능한 경우) 다른 단위에서 출력을 계산합니다. LabVIEW는 각

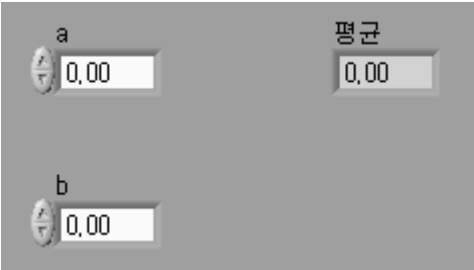
단위를 구별하므로 다형성 단위를 가진 VI는 다형성 단위를 가진 subVI를 가질 수 있습니다.

다형성 단위를 사용하는 경우

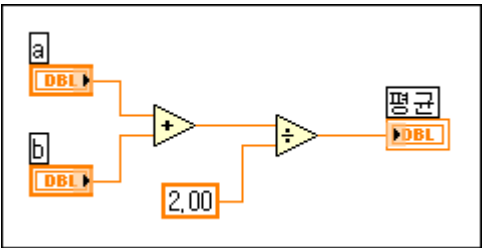
이 섹션은 다형성 단위를 가진 VI와 다형성 단위가 없는 VI의 몇몇 예제를 보여줍니다. 처음에는 단위가 없는 단순한 평균값 연산 예제로 시작합니다. 그 후 기본 단위를 가진 예제를 보여줍니다. 다음에 이 예제를 다형성 단위를 가진 예제로 바꾸고 다른 예제의 subVI로 사용합니다. **여러 평균값 연산** 예제와 **여러 속도** 예제는 다형성 단위 기능을 가진 subVI를 언제 사용해야 하는지 설명합니다.

단위 없이 평균값 연산하기

다음은 두 숫자의 평균을 계산하는 VI의 프런트패널입니다. 컨트롤 또는 인디케이터에는 단위가 없습니다.



다음은 VI의 블록다이어그램입니다.



하나의 단위로 평균값 연산하기

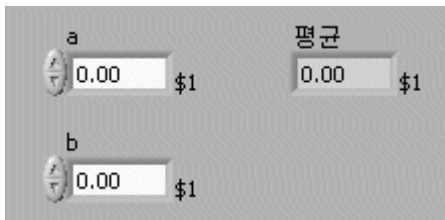
다음은 각각 미터 단위를 가진 두 숫자의 평균값을 계산하는 VI의 프런트패널을 보여줍니다.



이 VI의 블록다이어그램은 이전 블록다이어그램과 똑같습니다. 단위는 오직 프런트패널에서만 나타납니다. 프런트패널의 하나의 컨트롤 또는 인디케이터가 단위를 가지는 경우, 프런트패널의 모든 컨트롤과 인디케이터는 반드시 단위를 가져야 하며, 그렇지 않은 경우 블록다이어그램은 깨진 와이어를 디스플레이하고 VI는 실행되지 않습니다.

다형성 단위로 평균값 연산하기

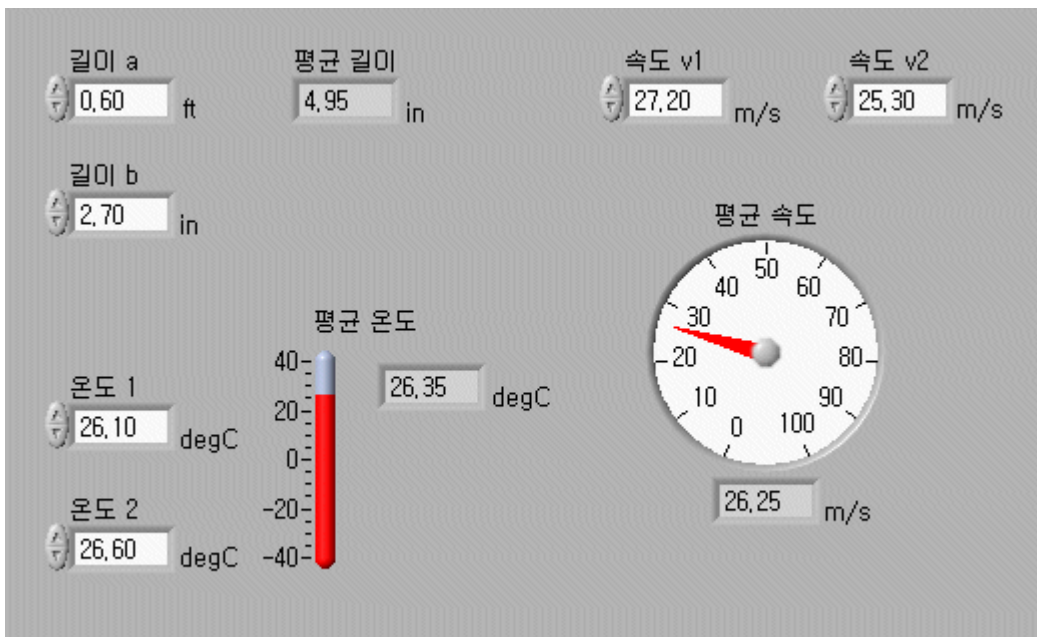
다음은 두 숫자의 평균을 계산하는 VI의 프런트패널입니다. 이전 프런트패널과는 달리, 다음 프런트패널은 다형성 단위를 가지고 있습니다. **여러 평균값 연산** 예제는 이 VI를 subVI로 사용합니다.



이 VI의 블록다이어그램은 이전 블록다이어그램과 똑같습니다. 미터 평균값 계산 예제와 마찬가지로 다형성 단위는 프런트패널에만 나타납니다. 프런트패널의 하나의 컨트롤 또는 인디케이터가 단위를 가지는 경우, 프런트패널의 모든 컨트롤과 인디케이터는 반드시 단위를 가져야 하며, 그렇지 않은 경우 블록다이어그램은 깨진 와이어를 디스플레이하고 VI는 실행되지 않습니다.

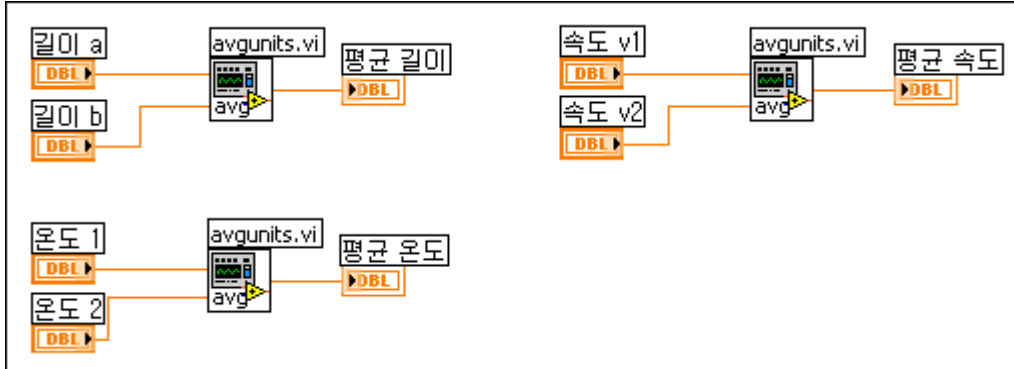
여러 평균값

다음은 각각 다른 단위를 가진 3 개의 평균값을 계산하는 VI의 프런트패널을 보여줍니다. 이는 다형성 단위를 가진 SubVI를 사용하는 경우를 보여주는 예입니다.



길이 a는 피트 단위이며, 길이 b는 인치 단위이고, 평균 길이는 인치 단위입니다. 이 VI가 다형성 단위를 사용하므로 LabVIEW는 자동으로 피트에서 인치로의 변환을 처리합니다.

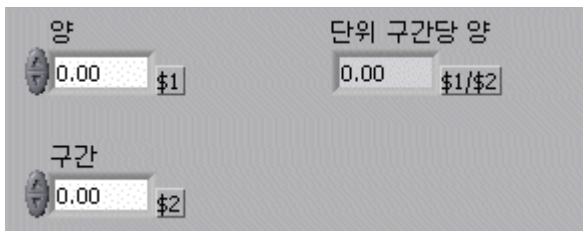
다음 블록 다이어그램은 **다형성 단위로 평균값 연산하기 VI**를 사용하여 **여러 평균값 연산** 예제에 있는 각 예의 평균값을 계산합니다.



여러 평균값 연산 VI에 있는 모든 단위를 가진 다형성 단위로 평균값 연산하기 subVI를 사용할 수 있습니다. 단위를 가진 두 개의 숫자의 평균값을 계산할 때마다 별도의 VI를 생성하는 대신에 다형성 VI를 가진 subVI를 사용할 수 있습니다.

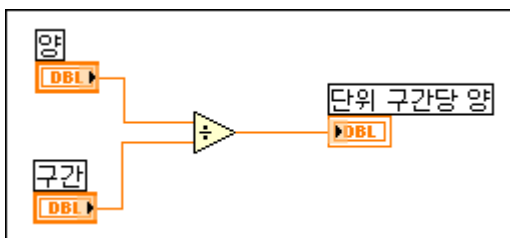
다형성 단위를 가진 비율

또한 비율을 계산할 때와 같이 인디케이터 단위가 입력 단위의 조합일 때 다형성 단위를 사용할 수 있습니다. 다음은 한 간격 당 양을 계산하는 VI의 프런트패널입니다. 일반적으로 이렇게 단순한 subVI는 생성하지 않습니다. 이는 subVI에서 두 개의 다형성 단위를 설명하는 예제일 뿐입니다. **여러 비율** 예제는 이 VI를 subVI로 사용합니다.



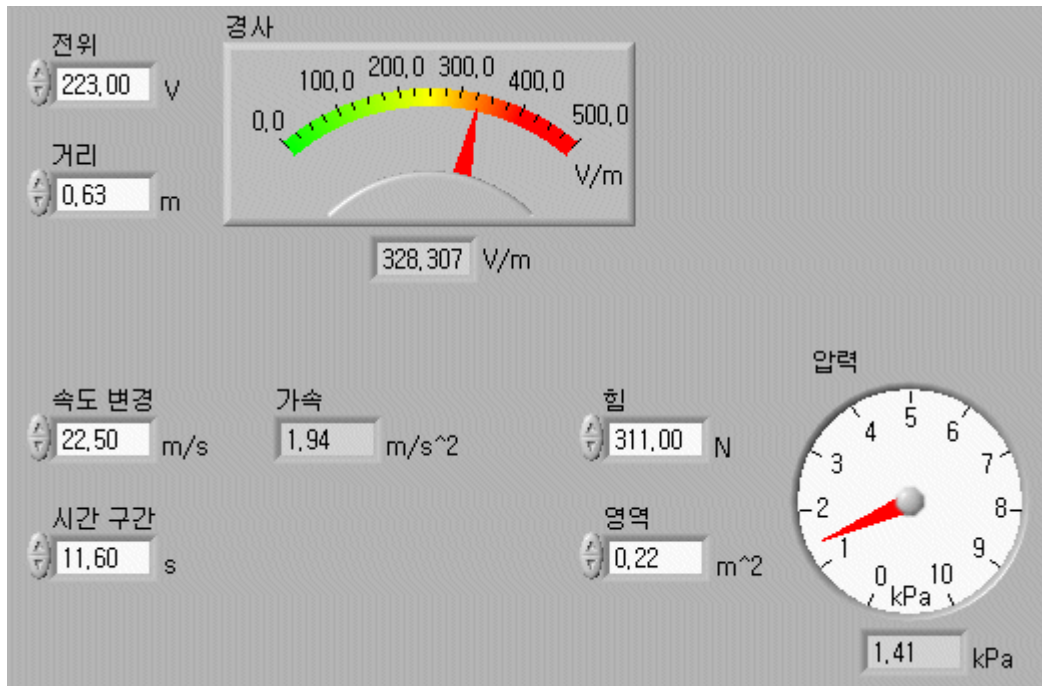
양 단위는 \$1이고 간격 단위는 \$2입니다, 1부터 9까지의 모든 숫자를 다형성 단위에 사용할 수 있습니다.

다음은 VI의 블록 다이어그램입니다.



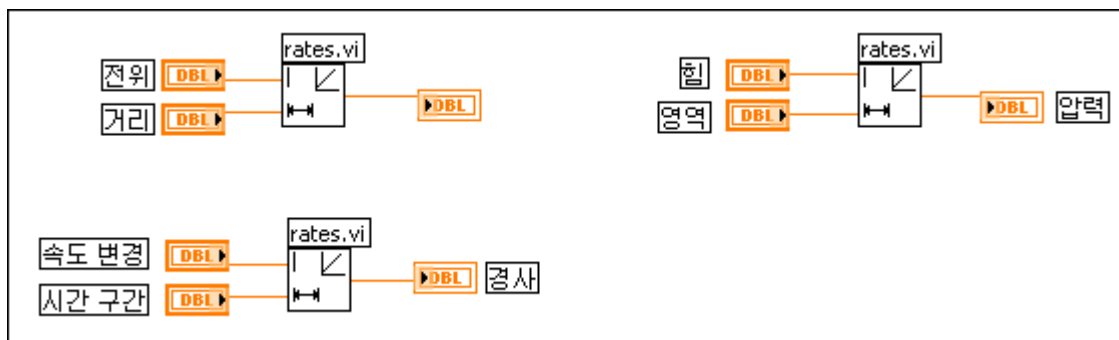
여러 비율

다음은 세 개의 다른 비율, 즉 경사, 가속, 압력을 계산하는 VI의 프런트패널을 보여줍니다.



컨트롤의 단위 중 두 개는 복합 단위입니다. 이전의 다형성 단위를 가진 비율 subVI는 단순하고 복합적인 단위 모두를 받아 들입니다.

다음은 VI의 블록 다이어그램입니다.



여러 평균값과 여러 비율을 위한 블록 다이어그램은 거의 같습니다. 각각은 다형성 단위를 가진 하나의 subVI를 사용하여 세 개의 다른 계산을 수행합니다.

컴파일된 객체 캐시

컴파일된 객체 캐시는 LabVIEW가 다음과 같은 종류의 파일의 컴파일된 코드를 저장하는 위치입니다:

- **분리된 컴파일된 코드**가 있는 VI 및 기타 파일 타입
- **RT 타겟에 배포**하는 VI
- **빌드 스펙 만들기** 중에 컴파일되는 VI

객체 캐시는 LabVIEW 데이터 디렉토리에 위치합니다.

직접 객체 캐시와 상호작용할 필요는 없습니다. 그러나 이전 리스트에서 다수의 VI를 삭제하는 경우, 객체 캐시를 삭제하는 것이 좋습니다. 이러한 종류의 VI를 삭제할 때, LabVIEW는 관련 컴파일된 코드를 자동으로 삭제하지 않습니다. 삭제된 VI의 컴파일된 코드가 차지하는 디스크 공간을 복구하려면, 삭제된 VI의 컴파일된 코드를 포함하는 전체 캐시를 삭제해야 합니다.



주의

객체 캐시를 지운 후, LabVIEW는 객체 캐시에 컴파일된 코드가 저장되어 있던 각 파일을 다시 컴파일할 시간을 필요로 합니다. 각 파일은 LabVIEW가 다음에 파일을 로드할 때 다시 컴파일됩니다. 객체 캐시를 삭제하기 전에 이러한 성능상 결점을 고려하십시오.

배포 타입

배포는 소프트웨어의 클라이언트가 사용할 수 있도록 묶은 소프트웨어 콘텐츠의 모음입니다.

다음 출력 타입 중 하나로 어플리케이션을 클라이언트에 배포할 수 있습니다.

- **패키지**—배포의 내용이 NI 패키지 관리자 패키지 파일 .nipkg로 빌드됩니다. 클라이언트는 패키지 관리자에서 직접 배포를 설치할 수 있습니다. 또한 배포를 피드에 추가하면 클라이언트가 피드를 구독하고 네트워크 접근을 통해 패키지 관리자에서 배포를 설치할 수 있습니다.
- **패키지 설치 프로그램**—배포의 내용이 설치 프로그램 파일 .exe로 빌드됩니다. 기본적으로, 패키지 설치 프로그램은 클라이언트가 네트워크 접근 없이 배포를 설치할 수 있도록 모든 패키지 의존성을 출력에 포함합니다.

피드는 포함된 패키지에 대한 정보가 들어있는 매니페스트를 포함하는 패키지 파일의 모음입니다. NI는 서로 의존성이 있는 여러 패키지를 배포할 때 피드를 사용하도록 권장합니다. 피드를 사용하면, 최종 사용자의 접근이 가능한 모든 패키지를 한 위치에서 호스트하고 유지관리할 수 있습니다.

LabVIEW 파일 시스템의 구조

LabVIEW의 구성

이 토픽은 LabVIEW 파일 시스템의 구조와 권장하는 파일 저장 위치를 설명합니다.

LabVIEW 디렉토리 구조의 구성

이 섹션은 Windows, macOS, Linux에서 LabVIEW 파일 시스템의 구조를 설명합니다. LabVIEW는 해당 플랫폼에서 사용 가능한 GPIB, DAQ, VISA, IVI, Motion Control, IMAQ 하드웨어 등에 대한 드라이버 소프트웨어를 설치합니다.

설치가 완료되면 labview 디렉토리에는 다음과 같은 그룹이 포함됩니다.



노트

사용자가 설치한 LabVIEW 모듈과 툴킷이 labview 디렉토리 내에 추가적으로 디렉토리를 생성할 수도 있습니다.

라이브러리

- `user.lib`—사용자가 생성한 **컨트롤과 VI를 저장**할 수 있는 디렉토리입니다. LabVIEW는 사용자 컨트롤 팔레트에 컨트롤을 디스플레이하고, 사용자 라이브러리 팔레트에 VI를 디스플레이합니다. 이 디렉토리는 LabVIEW를 업그레이드하거나 제거하여도 변하지 않습니다.
- `vi.lib`—내장된 VI의 라이브러리가 포함되어 있고 LabVIEW는 이를 함수 팔레트에서 관련된 그룹으로 묶어 디스플레이합니다. LabVIEW를 업그레이드하거나 다시 설치할 때 이 파일을 덮어쓰기 때문에 `vi.lib` 디렉토리에 파일을 저장하지 마십시오.
- `instr.lib`—PXI, VXI, GPIB, 시리얼, 컴퓨터 기반의 INSTRUMENT를 컨트롤하는데 사용하는 INSTRUMENT 드라이버를 포함합니다. National Instruments의 INSTRUMENT 드라이버를 설치하고 이 디렉토리에 위치시키면, LabVIEW는 이것을 INSTRUMENT 드라이버 팔레트에 추가합니다.

구조와 지원

- `menus`—LabVIEW가 컨트롤 및 함수 팔레트의 구조를 설정하는데 사용하는 파일을 포함합니다.
- `resource`—LabVIEW 어플리케이션의 추가 지원 파일을 포함합니다. LabVIEW를 업그레이드나 새로 설치할 때 이 파일을 덮어쓰므로 이 디렉토리에 파일을 저장하지 마십시오.
- `project`—LabVIEW 도구 메뉴의 아이템이 되는 파일을 포함합니다.
- `ProjectTemplates`—프로젝트 생성 대화 상자를 사용하여 사용자 정의한 일반적인 LabVIEW 디자인 패턴을 여기에서 시작할 수 있습니다.
- `templates`—일반 VI의 템플릿을 포함합니다.
- `wizard`—LabVIEW 파일 메뉴의 아이템이 되는 파일을 포함합니다.
- `www`—웹 서버를 통해 접근할 수 있는 HTML 파일의 위치.

학습 및 설명 자료

- `examples`—예제 VI 포함. 도움말 >> 예제 찾기를 선택하여 예제를 탐색 또는 검색합니다.

문서

- `manuals`—PDF 형식의 문서를 포함합니다.
- `help`—도움말 파일을 포함합니다. 도움말 >> LabVIEW 도움말을 선택해서 LabVIEW 도움말을 엽니다. 이 디렉토리에는 LabVIEW 도움말 메뉴의 아이템이 되는 파일이 있습니다.
- `readme`—LabVIEW `readme` 파일과 사용자가 설치한 임의의 모듈 또는 툴킷의 `readme` 파일이 있습니다. (Windows) `labview\readme` 디렉토리를 탐색하여 `readme` 파일에 접근합니다. (macOS 및 Linux) `labview` 디렉토리를 탐색하여 `readme` 파일에 접근합니다.

권장되는 파일 저장 위치

LabVIEW는 `vi.lib`와 `resource` 디렉토리를 LabVIEW 시스템 전용으로 설치합니다. 이 디렉토리에는 파일을 저장하지 마십시오.

다음의 디렉토리에 파일을 저장할 수 있습니다:

- `user.lib`—**사용자 컨트롤 또는 사용자 라이브러리 팔레트**에 디스플레이하려는 모든 컨트롤 또는 VI.

노트 SubVI를 `user.lib` 디렉토리에 저장하려면 해당 프로젝트가 끝날 때까지 수정하지 않을 것만 저장하십시오. `user.lib`에 있는 VI의 경로는 `labview` 디렉토리의 상대적인 위치에 있습니다. 다른 곳에 저장하는 subVI의 경로는 상위 VI를 기준으로 한 상대적인 경로입니다. 그러므로 특별한 경우 `user.lib`에서 VI를 복사하여 수정하면, `user.lib`에 위치한 이 SubVI의 경로는 변하지 않습니다.

VI와 subVI를 다른 디렉토리나 컴퓨터에 복사하려면, VI를 `user.lib`에 저장하지 마십시오. 왜냐하면 새로운 위치에서 상위 레벨 VI를 실행할 때 LabVIEW는 원래의 subVI를 `user.lib`에서 참조하기 때문입니다.

- `instr.lib`—인스트루먼트 드라이버 팔레트에 디스플레이하려는 모든 인스트루먼트 드라이버 VI.
- `project`—LabVIEW의 기능 확장에 사용하는 VI. 이 디렉토리에 저장한 VI는 도구 메뉴에 나타납니다.
- `wizard`—이 디렉토리에 저장한 VI는 파일 메뉴에 나타납니다.
- `www`—웹 서버를 통해 접근할 수 있는 HTML 파일의 위치.
- `help`—도움말 메뉴에서 확인할 수 있는 모든 VI, `.hlp` 파일과 `.chm` 파일.
- `LabVIEW Data`—`.lvn` 또는 `.txt` 파일과 같이 LabVIEW가 생성한 모든 데이터 파일.

사용자가 생성한 LabVIEW 파일을 저장하기 위해 하드 드라이브의 어느 위치든 **디렉토리를 생성**할 수 있습니다.

labview\help, labview\project, labview\wizard 디렉토리에 놓은 VI를 도움말, 도구, 또는 파일 메뉴에서 열면, LabVIEW는 프라이빗 **어플리케이션 인스턴스** (NI.LV.Dialog)의 VI를 엽니다.

문자열, 클러스터, 배열, 집합을 이용한 데이터의 그룹화

LabVIEW에서는 다음 데이터 타입을 사용하여 데이터를 그룹화합니다.

- 문자열 — ASCII 문자의 시퀀스를 그룹화합니다.
- 클러스터 — 여러 타입의 데이터 원소를 함께 묶어 그룹화합니다.
- 배열 — 같은 타입의 데이터 원소를 정렬되지 않은 순서로 그룹화합니다. 2D 배열인 행렬 데이터 타입을 포함합니다. 이를 사용하면 보다 쉽게 수학 문제를 모델링할 수 있습니다.
 - 행렬 — 선형 대수 연산과 같은 일부 수학 연산에서 실수 또는 복소수 스칼라 데이터의 행이나 열을 그룹화합니다.
- 집합 — 다음 서브 타입을 포함합니다.
 - 세트 — 같은 타입의 데이터 원소를 정렬된 순서로 그룹화합니다.
 - 맵 — 키-값 쌍을 그룹화합니다. 이때 모든 키는 고유합니다. 키와 값은 각각 어떤 데이터 타입이든 될 수 있습니다.

관련 개념

문자열을 이용한 데이터의 그룹화

배열과 클러스터를 이용한 데이터의 그룹화

집합을 이용한 데이터의 그룹화

행렬 사용하기

LabVIEW 서브팔레트

그래프 서브팔레트

그래프 팔레트 객체를 사용하여 숫자형 데이터를 그래프, 차트 또는 플롯으로 플롯합니다.



노트
사용할 수 있는 그래프와 차트 컨트롤 및 인디케이터는 선택한 서브팔레트의 스타일에 따라 다릅니다. 사용할 수 있는 그래프 컨트롤과 인디케이터는 일반, 실버, 시스템 또는 클래식 서브팔레트를 참조하십시오.

팔레트 객체	설명
웨이브폼 차트와 그래프	일반적으로 일정한 속도로 수집한 데이터를 디스플레이합니다.
XY 그래프	일정하지 않은 속도로 수집한 데이터와 여러 값을 갖는 함수를 디스플레이합니다.
익스프레스 XY 그래프	XY 그래프와 동일한 데이터를 디스플레이합니다. 대화 상자에서 대화식으로 익스프레스 XY 그래프의 셋팅을 설정할 수 있습니다.
강도 차트와 그래프	3 차원 값을 디스플레이하기 위해 색을 이용하여 2D 플롯에서 3D 데이터를 디스플레이합니다.
디지털 웨이브폼 그래프	데이터를 디지털 라인의 펄스나 그룹으로 디스플레이합니다.
혼합된 신호 그래프	데이터 타입을 웨이브폼 그래프, XY 그래프, 디지털 웨이브폼 그래프 및 이러한 데이터 타입의 조합을 포함하는 클러스터로 디스플레이합니다.
콤파스 플롯	순환선의 중심에서 발산하는 벡터를 디스플레이합니다.
에러 막대 플롯	라인 그래프의 위와 아래의 각 포인트에 에러 막대를 디스플레이합니다.
화살표 플롯	수평 축을 따라 균일한 간격의 포인트에서 발산하는 벡터를 디스플레이합니다.
XY 플롯 행렬	산포 그래프의 행 및 열을 디스플레이합니다.
3D 그림	3D 객체의 그래픽적 표현을 디스플레이합니다.

서브팔레트	설명
컨트롤	컨트롤 팔레트 객체를 사용하여 2D 객체의 고급 그래프 및 그래픽적 표현을 디스플레이합니다.

서브팔레트	설명
3D 그래프	3D 그래프 팔레트를 사용하여 데이터를 3D로 플롯합니다.

컨트롤 서브팔레트

컨트롤 팔레트 객체를 사용하여 2D 객체의 고급 그래프 및 그래픽적 표현을 디스플레이합니다.

팔레트 객체	설명
ActiveX 3D 그래프	프런트패널에 있는 ActiveX 객체에서 3D 데이터를 3D 플롯으로 디스플레이합니다.
극좌표 플롯	극좌표로 표현되는 데이터를 디스플레이합니다.
스미스 플롯	극좌표 플롯에서 복소수 임피던스에 해당하는 복소수 반사 계수를 디스플레이합니다.
최소-최대 플롯	플롯의 최소값과 최대값을 디스플레이합니다.
분포 플롯	주파수 히스토그램을 디스플레이합니다.
방사형 플롯	다변량(multivariate) 데이터를 각 변수에 하나의 축이 있는 순환 플롯으로 디스플레이합니다.
2D 그림	2D 객체의 그래픽적 표현을 디스플레이합니다.

서브팔레트	설명
데이터 타입	데이터타입 팔레트 객체를 그림 함수 VI와 2D 그림 컨트롤과 함께 사용하여 데이터의 시각적 표현을 생성합니다.

데이터타입 서브팔레트

데이터타입 팔레트 객체를 그림 함수 VI와 2D 그림 컨트롤과 함께 사용하여 데이터의 시각적 표현을 생성합니다.

팔레트 객체	설명
포인트	포인트에 대한 x 및 y 좌표를 정의하기 위해 사용합니다.
직사각형	사각형의 코너를 정의하기 위해 사용합니다.
폰트 열거형	사용자 지정 폰트 또는 어플리케이션 폰트, 시스템 폰트, 대화 상자 폰트라는 3개의 내장 폰트 중 하나를 선택하기 위해 사용합니다.

팔레트 객체	설명
사용자 폰트	폰트, 텍스트 크기, 텍스트 포맷을 지정하기 위해 폰트 열거형이 사용자 지정 폰트로 설정되어 있을 때 사용합니다.
텍스트 정렬	텍스트의 수평 및 수직 정렬을 설정하기 위해 사용합니다.

3D 그래프 서브팔레트

3D 그래프 팔레트를 사용하여 데이터를 3D로 플롯합니다.

팔레트 객체	설명
산포	두 변수 사이의 관계를 이산 포인트의 집합으로 디스플레이합니다.
막대	수직 막대 그래프를 디스플레이합니다.
파이	파이 차트를 디스플레이합니다.
리본	평행 라인 플롯을 디스플레이합니다.
컨투어	두 개의 변수가 있는 함수가 상수값을 갖는 커브를 디스플레이합니다.
화살형	포인트의 집합의 크기와 방향을 나타내는 벡터 화살표를 디스플레이합니다. 화살형 플롯은 또한 벡터 필드라고도 알려져 있습니다.
헤성형	데이터 포인트를 따라서 원을 포함하는 애니메이션 그래프를 디스플레이합니다.
표면	연결하는 표면과 함께 데이터를 디스플레이합니다.
메시	열린 공간의 메시 표면을 디스플레이합니다.
워터폴	데이터의 표면과 데이터 포인트 아래의 y 축에 있는 영역을 디스플레이합니다.
3D 표면 그래프	연결하는 표면과 함께 3D 데이터를 디스플레이합니다.
3D 파라미터형 그래프	x, y, z 변수가 공통 파라미터의 각 함수인 그래프를 디스플레이합니다.
3D 라인 그래프	x,y,z 점에 따라 라인 또는 커브를 그립니다.

LabVIEW 컨트롤과 인디케이터

I/O 이름 컨트롤과 인디케이터

I/O와 클래식 I/O 팔레트의 I/O 이름 컨트롤과 인디케이터를 사용하여 설정한 DAQ 채널 이름, VISA 리소스 이름, IVI 논리 이름을 I/O VI에 전달하여 인스트루먼트 또는 DAQ 디바이스와 통신합니다.

I/O 이름 상수는 함수 팔레트에 있습니다. 상수는 블록다이어그램에 고정 데이터 값을 제공하는 블록다이어그램의 터미널입니다.



노트

모든 I/O 이름 컨트롤 또는 상수는 모든 플랫폼에서 사용 가능하고, 이를 통해 플랫폼 특정 디바이스와 통신할 수 있는 I/O VI를 모든 플랫폼에서 개발할 수 있습니다. 그러나 해당 디바이스를 지원하지 않는 플랫폼에서 플랫폼 특정 I/O 컨트롤과 함께 VI를 실행하려는 경우, 에러가 발생합니다.

(Windows) 도구 메뉴를 통해 Measurement & Automation Explorer(MAX)를 열고 DAQ 채널 이름, VISA 리소스 이름, IVI 논리적 이름을 설정합니다.

(macOS 및 Linux) 인스트루먼트에 대한 설정 유틸리티를 이용하여 VISA 리소스 이름과 IVI 로직 이름을 설정하십시오. 설정 유틸리티에 대한 추가적인 정보는 인스트루먼트의 문서를 참조하십시오.

IMAQ 세션 컨트롤은 하드웨어에 대한 연결을 나타내는 유일한 식별자입니다.

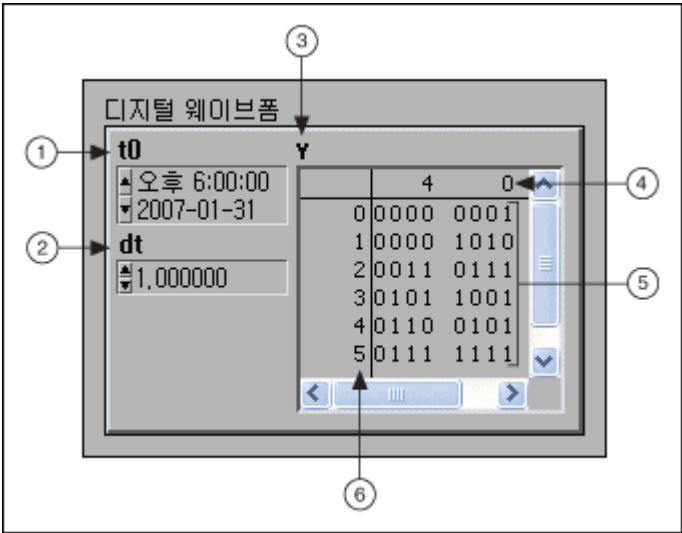
웨이브폼 컨트롤

웨이브폼 컨트롤을 사용하여 **웨이브폼**의 개별 데이터 원소를 다룹니다. 웨이브폼 컨트롤은 웨이브폼의 데이터, 시작 시간, 델타 t를 포함합니다.

디지털 웨이브폼 컨트롤

디지털 웨이브폼 컨트롤을 사용하여 **디지털 웨이브폼**의 개별 원소를 다룹니다. 디지털 웨이브폼 컨트롤을 사용하여 시작 시간, 델타 t, 데이터, 디지털 웨이브폼의 속성을 지정합니다.

다음 그림의 디지털 웨이브폼 컨트롤은 6개의 샘플이 있는 디지털 데이터를 디스플레이 하며, 각 샘플에는 8개의 신호가 포함되어 있고, 시작 시간은 오후 6:00:00 2007-01-31, 델타 t는 1입니다.

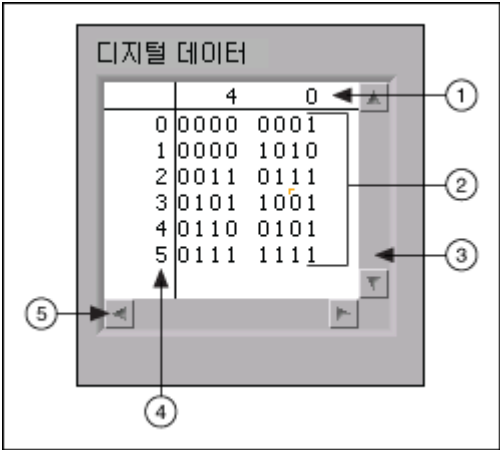


1 시작 시간	3 디지털 데이터	5 데이터
2 델타 t	4 신호	6 샘플

디지털 데이터 컨트롤

디지털 데이터 컨트롤은 디지털 데이터를 행과 열로 정렬된 **디지털 테이블**로 디스플레이합니다. 디지털 데이터 컨트롤을 사용하여 디지털 웨이브폼을 만들거나 디지털 웨이브폼에서 추출한 디지털 데이터를 디스플레이합니다. 디지털 웨이브폼 데이터 컨트롤을 디지털 데이터 인디케이터에 연결하여 디지털 웨이브폼의 샘플과 신호를 봅니다.

다음 그림의 디지털 데이터 컨트롤은 각각 8개의 신호를 가지는 6개의 샘플을 디스플레이합니다.



1 신호	3 수직 스크롤 막대	5 수평 스크롤 막대
2 데이터	4 샘플	

디지털 데이터 컨트롤에서 행과 열을 **삽입하고 삭제**할 수 있습니다. 바로 가기 메뉴를 사용하여 컨트롤 내에서 디지털 데이터를 **잘라내고 복사하고 붙여넣기**합니다.

디지털 데이터 컨트롤의 데이터를 2진수, 16진수, 8진수, 10진수 포맷으로 디스플레이할 수 있습니다. 16진수, 8진수 또는 10진수 포맷으로 값을 디스플레이하는 경우, 일부 측정 디바이스가 사용하는 L, H, Z, X, T, V의 디지털 상태는 컨트롤에서 물음표로 나타납니다. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 포맷을 선택하고 컨트롤에 대한 데이터 포맷을 선택합니다.

디지털 테이블

디지털 데이터 컨트롤과 디지털 웨이브폼 컨트롤은 디지털 테이블의 데이터를 포함합니다. 디지털 테이블은 여덟 개의 데이터 상태로 데이터를 나타낼 수 있습니다. 다음 테이블은 디지털 데이터 상태를 설명합니다.

값	디지털 데이터 상태	설명
0	0 (로우로 유도)	로직 로우 강제. 낮은 전압 레벨(V_{OL})로 유도.
1	1 (하이로 유도)	로직 하이 강제. 높은 전압 레벨(V_{OH})로 유도.
2	Z (강제로 끄기)	로직 하이 임피던스 강제. 드라이버를 끄.
3	L (로우를 비교)	로직 로우(에지) 비교. 낮은 전압 임계점보다 낮은 전압 레벨을 비교.
4	H (하이를 비교)	로직 하이(에지) 비교. 높은 전압 임계점보다 높은 전압 레벨을 비교.
5	X (알 수 없음 비교)	알 수 없는 로직 비교. 비교하지 않음.
6	T (비교 끄기)	로직 하이 임피던스(에지) 비교. 낮은 전압 임계점(V_{OL})과 높은 전압 임계점(V_{OH}) 사이의 전압 레벨을 비교.
7	V (유효함 비교)	로직 유효 레벨(에지) 비교. 낮은 전압 임계점(V_{OL})보다 낮거나 높은 전압 임계점(V_{OH})보다 높은 전압 레벨을 비교.



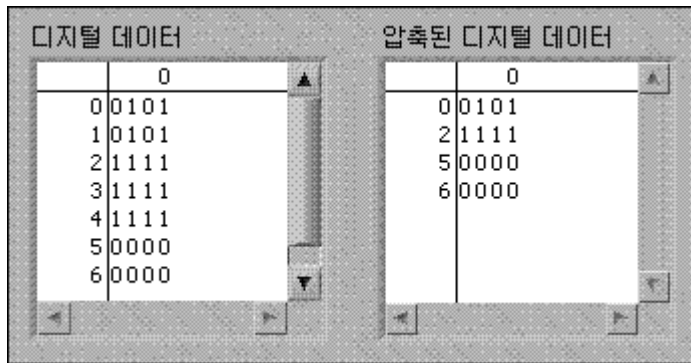
노트

[디지털 데이터 만들기] 함수를 사용하여 **디지털 테이블을 만들 수 있습니다.**

디지털 데이터 압축하기

디지털 테이블의 **데이터를 압축**할 수 있습니다. 데이터를 보다 시각적으로 하기 위해서 두 개 또는 그 이상의 디지털 신호를 같은 비트 세트를 사용하여 같은 항에 디스플레이하려는 경우, 디지털 데이터를 압축합니다. 예를 들어, 10개의 디지털 웨이브폼 샘플을 얻었는데 9 개의 웨이브폼은 똑같으나 10번째 웨이브폼이 다른 경우, 디지털 데이터를 압축하면 어떤 웨이브폼이 다른지 쉽게 찾을 수 있습니다. 또한 디지털 데이터 압축은 메모리 리소스를 절약합니다.

다음 그림은 같은 데이터를 디스플레이하는 두 개의 디지털 데이터 테이블을 보여줍니다. 첫번째 테이블은 압축되지 않은 디지털 데이터를 나타냅니다. 두번째 테이블은 압축 후의 같은 디지털 데이터를 나타냅니다.



디지털 웨이브폼 VI와 함수를 사용하여 디지털 데이터와 디지털 웨이브폼을 압축하거나, 압축을 풀거나, 조작할 수 있습니다.

리스트, 테이블, 트리 컨트롤과 인디케이터

리스트, 테이블 & 트리와 클래식 리스트, 테이블 & 트리 팔레트에 위치한 리스트박스, 트리 컨트롤, 테이블 컨트롤, 인디케이터를 사용하여 사용자에게 선택할 수 있는 아이템의 리스트를 제공합니다.

리스트박스

리스트박스가 하나 또는 여러 개의 선택을 허용하도록 설정할 수 있습니다. 여러 열 리스트박스를 사용하여 아이템의 크기와 생성된 날짜와 같이 각 아이템에 대한 추가적인 정보를 디스플레이합니다.

실행 중에 리스트박스에 문자를 입력하면, LabVIEW는 입력한 문자로 시작하는 리스트박스의 첫번째 아이템을 선택합니다. 왼쪽과 오른쪽 화살표 키를 사용하여 입력하는 문자와 일치하는 이전 아이템과 다음 아이템으로 이동합니다. 리스트박스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 선택 모드 >> 전체 행 하이라이트를 선택하여 아 아이템을 선택했을 때 전체 행을 하이라이트합니다.

디렉토리나 파일이 서로 다른 기호를 지니는 LLB 관리자 윈도우와 같은 곳에서는 리스트 아이템 옆에 **기호를 추가**할 수 있습니다. 또한, 리스트 아이템 사이에 분리 라인을 삽입할 수 있습니다.

프로퍼티 노드를 사용하여 리스트 아이템을 수정하고 리스트 아이템에 대한 정보를 모을 수 있습니다. 현재 선택된 아이템을 감지하거나 또는 실행 중 문자를 리스트박스에 입력할 때 대소문자의 처리 방식을 설정하는 것을 예로 들 수 있습니다.

리스트박스 사용에 대한 예제는 다음 VI를 참조하십시오:

- Listbox VI: labview\examples\Controls and Indicators\Listbox

트리 컨트롤

트리 컨트롤을 사용하여 사용자가 선택할 수 있는 아이템의 계층구조 리스트를 제공합니다. 트리 컨트롤에 입력한 아이템을 아이템이나 노드의 그룹으로 묶습니다. 노드 옆의 확장 기호를 클릭하여 노드를 확장하고 노드의 모든 아이템을 디스플레이합니다.

실행 중에 트리 컨트롤에 문자를 입력하면, LabVIEW는 트리 컨트롤에서 입력한 문자로 시작하는 첫번째 아이템을 선택합니다. 아이템을 선택한 후 마침표 . 키를 눌러서 현재 아이템을 하위 아이템으로 밀거나 또는 쉼표 , 키를 눌러서 현재 아이템을 상위 아이템으로 당겨서 트리 컨트롤의 아이템의 계층구조를 바꿀 수 있습니다. 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 선택 모드 >> 전체 행 하이라이트를 선택하여 아이템을 선택했을 때 전체 행을 하이라이트합니다.

리스트박스과 같은 방법으로 트리 컨트롤의 아이템을 설정합니다. 또한 각 노드 옆에 나타나는 심볼의 타입을 바꿀 수 있고, 사용자가 트리 컨트롤에 아이템을 **끌어서 놓을 수 있는**지를 설정할 수 있습니다.

[인보크 노드]를 사용하여 트리 컨트롤의 아이템을 수정하고 사용자가 더블 클릭한 아이템을 감지하는 것과 같이 아이템에 대한 정보를 얻을 수 있습니다. 트리 컨트롤에 아이템을 추가할 때, LabVIEW는 아이템에 고유한 태그를 생성합니다. 이 태그를 이용하여 프로그램적으로 아이템을 수정하거나 아이템 정보를 수집할 수 있습니다. 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 아이템 편집을 선택하여 LabVIEW가 각 아이템에 대해서 생성하는 태그를 수정합니다.

테이블

테이블 컨트롤을 사용하여 프런트패널에 **테이블**을 생성합니다.

.NET 및 ActiveX 컨트롤과 인디케이터

.NET & ActiveX 팔레트에 있는 **.NET** 및 **ActiveX** 컨트롤을 사용하여 리치 텍스트 박스, 픽처 박스 및 달력과 같은 일반적인 .NET 또는 ActiveX 컨트롤을 다룹니다. 추후 사용을 위해 .NET 또는 ActiveX 컨트롤을 이 팔레트에 추가할 수 있습니다. 도구 >> 반입 >> .NET 컨트롤을 팔레트로 또는 ActiveX 컨트롤을 팔레트로 선택하여 .NET이나 ActiveX 컨트롤 세트를 각각 사용자 컨트롤로 변환하고 .NET & ActiveX 팔레트에 추가합니다.



노트

.NET 객체를 생성하고 이 객체와 통신하려면 .NET Framework 4.0이 필요합니다. National Instruments는 **LabVIEW 프로젝트** 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

숫자형 컨트롤과 인디케이터

프론트패널에 숫자형 컨트롤과 인디케이터를 사용하여 LabVIEW 어플리케이션에 숫자형 데이터를 디스플레이합니다. 다음 리스트는 일반, 실버, 시스템 또는 클래식 서브팔레트에서 사용할 수 있는 숫자형 컨트롤과 인디케이터의 일반적인 사용례를 설명합니다:



노트

사용할 수 있는 숫자형 컨트롤과 인디케이터는 선택한 서브팔레트의 스타일에 따라 다릅니다. 사용할 수 있는 숫자형 컨트롤과 인디케이터는 일반, 실버, 시스템 또는 클래식 서브팔레트를 참조하십시오.

숫자형 컨트롤 또는 인디케이 터	설명
숫자형	숫자형 데이터를 입력하거나 디스플레이할 때 사용합니다.
타임스탬프	시간과 날짜 값을 전송하거나 얻을 때 사용합니다.
채움 슬라이드	숫자형 데이터를 사용자가 지정한 스케일을 사용하여 수평 또는 수직 슬라이드로 디스플레이할 때 사용합니다.
포인터 슬라이드	숫자형 데이터를 정확한 값을 보여주는 포인터 및 사용자가 지정한 스케일을 사용하여 수평 또는 수직 슬라이드로 디스플레이할 때 사용합니다.
진행 막대	진행을 수직 또는 수평 막대의 형태로 보여줄 때 사용합니다.
눈금 막대	동일한 여러 부분으로 나누어 볼 수 있도록 지정한 크기로 구분된 수직 또는 수평 막대의 형태로 진행을 보여줄 때 사용합니다. 예를 들어, 값이 타겟 값의 25%에 이르렀음을 보여줄 수 있습니다.
노브, 다이얼, 미터, 게이지	숫자형 데이터를 회전 동작으로 입력 또는 디스플레이할 때 사용합니다.
탱크 및 온도계	숫자형 데이터를 실제 탱크나 온도 계측기와 유사한 수직 슬라이드의 형태로 디스플레이할 때 사용합니다.
스크롤 막대	숫자형 데이터를 사각 스크롤 박스를 끌거나, 마우스 휠을 스크롤하거나, 증가 또는 감소 버튼을 클릭하거나, 스크롤 막대의 막대의 빈 공간을 클릭하면 데이터 값이 변화하는 수직 슬라이드의 형태로 디스플레이할 때 사용합니다.
색 상자	특정 값에 해당하는 색을 디스플레이할 때 사용합니다. 예를 들면, 색 상자를 사용하여 다른 조건, 즉 범위를 벗어난 값 등을 가리킵니다.
색 램프	숫자형 값을 색으로 디스플레이할 때 사용합니다. 입력값이 변경되는 경우, 해당 값에 대해 미리 지정된 색으로 디스플레이됩니다.

숫자형 컨트롤 또는 인디케이터	설명
웨이브폼	숫자형 데이터를 웨이브폼으로 입력 또는 디스플레이할 때 사용합니다.



팁

<Enter> 키를 누르거나, 디지털 디스플레이 윈도우의 바깥쪽을 클릭하거나, 또는 입력 버튼을 클릭하여 숫자형 컨트롤이나 인디케이터의 기존 값을 새로운 값으로 대체합니다. VI가 실행되는 동안, LabVIEW는 이 중 하나의 동작이 수행될 때까지 대기한 후, 새로운 값을 분석합니다. 예를 들어, 디지털 디스플레이의 값을 135로 변경하면서, 135 이전에 VI가 1과 13의 값을 받도록 할 수는 없습니다.



노트

숫자형 컨트롤 또는 인디케이터를 공학 또는 System International (SI) 표기법으로 포맷할 때 복소수형이 없으면, 부동 소수형을 유지하도록, LabVIEW는 해당 단위 문자열에 필요한 접두어를 추가합니다.

관련 정보

숫자형 변경하기

숫자형 컨트롤과 인디케이터의 데이터 범위 변경하기

숫자형 컨트롤과 인디케이터의 포맷과 정밀도 변경하기

타임스탬프 컨트롤에서 시간과 날짜를 설정하기

스케일을 가지는 숫자형 객체의 값 변경하기

참조 번호 컨트롤과 인디케이터

참조 번호와 클래식 참조 번호 팔레트에 위치한 참조 번호 컨트롤을 사용하여 파일, 디렉토리, 디바이스, 네트워크 연결과 관련된 작업을 수행합니다. **컨트롤 참조 번호**를 사용하여 프런트패널 객체 정보를 SubVI에 전달합니다.

참조 번호, 즉 refnum은 파일, 디바이스 또는 네트워크 연결과 같은 객체에 대한 유일한 식별자입니다. 파일, 디바이스 또는 네트워크 연결을 열 때, LabVIEW는 해당 파일, 디바이스 또는 네트워크 연결과 연계된 참조 번호를 생성합니다. 열린 파일, 디바이스, 또는 네트워크 연결에서 수행하는 모든 작업은 참조 번호를 사용하여 각 객체를 식별합니다. 참조 번호 컨트롤을 사용하여 VI 내부 또는 외부로 참조 번호를 전달할 수 있습니다. 예를 들어, 참조 번호 컨트롤 또는 인디케이터를 사용하여 참조 번호가 참조하는 파일을 닫거나 다시 열지 않고 파일의 내용을 수정합니다.

참조 번호는 열어 놓은 객체의 임시 포인터이므로, 객체가 열려 있는 동안만 유효합니다. 객체를 닫으면, LabVIEW는 참조 번호와 객체의 연결을 끊어서 해당 참조 번호를 사용할

수 없게 됩니다. 객체를 다시 열면, LabVIEW는 처음 참조 번호와는 다른 새 참조 번호를 생성합니다. LabVIEW는 참조 번호와 관련된 객체에 메모리를 할당합니다. 참조 번호를 닫아 객체를 메모리에서 해제합니다.

LabVIEW는 객체에서 읽어 오거나 쓴 현재 위치와 사용자 접근 등급과 같은 각 참조 번호에 관련된 정보를 기억하고 있기 때문에, 한 객체에 대하여 서로 독립적인 작업을 동시에 수행할 수 있습니다. VI가 객체를 여러번 여는 경우, 열 때마다 매번 다른 참조 번호를 반환합니다. LabVIEW는 VI의 실행이 끝나면 자동으로 참조 번호를 닫습니다. 그러나 메모리와 기타 리소스를 가장 효율적으로 사용하려면 사용이 끝난 직후 참조 번호를 닫는 것이 좋은 프로그래밍 방법입니다. 참조 번호는 연 순서와 반대로 닫아야 합니다. 예를 들어 객체 A의 참조 번호를 얻고 객체 A에서 메소드를 실행시켜 객체 B의 참조 번호를 얻은 경우, 객체 B의 참조 번호를 먼저 닫고 객체 A의 참조 번호를 닫습니다.

For 루프 또는 While 루프 안의 참조 번호를 여는 경우, LabVIEW는 반복적으로 참조 번호에 메모리를 할당하고 VI가 실행을 완료할 때까지 메모리를 비우지 않기 때문에 루프의 각 반복에서 해당 참조 번호를 닫으십시오.

링 및 열거형 타입 컨트롤과 인디케이터

링 & 열거형과 클래식 링 & 열거형 팔레트에 있는 링 및 열거형 타입 컨트롤과 인디케이터를 사용하여 선택이 가능한 문자열의 리스트를 생성합니다.

링 컨트롤

링 컨트롤은 숫자값을 문자열 또는 그림에 연관시키는 숫자형 객체입니다. 링 컨트롤은 사용자가 돌아가면서 선택할 수 있는 풀다운 메뉴로 나타납니다.

링 컨트롤은 트리거 모드와 같이 상호 배타적인 아이템을 선택할 때 유용합니다. 예를 들어, 링 컨트롤을 사용하여 사용자가 연속, 단일, 외부 트리거에서 선택하도록 합니다.

링 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 아이템 편집을 선택하여 컨트롤의 링 컨트롤 리스트에 아이템을 추가합니다. 링 프로퍼티 대화 상자의 아이템 편집 페이지에서 아이템의 순서는 컨트롤의 아이템 순서를 결정합니다. 또한, 컨트롤에 대해 정의된 아이템 리스트의 엔트리와 아직 아무런 연관이 없는 숫자값을 사용자가 입력할 수 있도록 **링 컨트롤을 설정**할 수 있습니다.

실행 중에 링 컨트롤에 정의되지 않은 값을 입력하기 위해서는, 컨트롤을 클릭한 후 바로 가기 메뉴에서 <그 외>를 선택하여 나타난 디지털 디스플레이에 숫자값을 입력하고 <Enter> 키를 누릅니다. 정의되지 않은 값이 꺾쇠 괄호(<>)로 링 컨트롤에 나타납니다. LabVIEW는 정의되지 않은 값을 링 컨트롤 리스트에 추가하지 않습니다.

링 컨트롤에 대한 아이템 리스트를 설정할 때, 각 아이템에 **특정 숫자값을 지정**할 수 있습니다. 아이템에 특정 숫자값을 지정하지 않는 경우, LabVIEW는 첫번째 아이템에 대해서 0의 값을 시작으로 리스트의 아이템 순서에 따라서 순차적인 값을 지정합니다.

열거형 타입 컨트롤

열거형 타입 컨트롤을 사용하여 사용자가 선택할 수 있는 아이템 리스트를 제공합니다. 열거형 타입 컨트롤 또는 열거형은 텍스트 또는 메뉴 링 컨트롤과 비슷합니다. 둘 다 사

이끌할 수 있는 문자열 목록을 포함합니다. 그러나, 열거형 타입 컨트롤은 다음에서 링 컨트롤과 다릅니다:

- 열거형 타입 컨트롤의 데이터 타입은 컨트롤과 해당하는 문자열의 숫자값에 대한 정보를 포함합니다. 링 컨트롤의 데이터 타입은 오직 숫자형입니다.
- 열거형 타입 컨트롤의 숫자형은 8, 16 또는 32 비트 부호없는 정수입니다. 그러나 링 컨트롤은 다른 형도 가질 수 있습니다. 컨트롤의 타입에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 형을 선택하여 컨트롤의 형을 변경합니다.
- 사용자가 열거형 타입 컨트롤에 정의되지 않은 값을 입력하도록 허용할 수 없으며, 열거형 타입 컨트롤의 아이템에 특정 숫자값을 할당할 수 없습니다. 이 기능이 필요한 경우, 링 컨트롤을 사용하십시오.
- 열거형 타입 컨트롤의 아이템은 편집 시에만 편집할 수 있습니다. 그러나 링 컨트롤 아이템은 프로퍼티 노드를 사용하여 실행 중에도 편집할 수 있습니다.
- 열거형 타입 컨트롤을 케이스 구조의 선택자 터미널에 연결하면, LabVIEW는 케이스를 숫자값이 아닌 컨트롤에서의 아이템의 문자열 값과 일치시킵니다. 링 컨트롤을 케이스 구조에서 사용하면, LabVIEW는 케이스를 아이템의 숫자값과 일치시킵니다.
- 열거형 타입 컨트롤을 케이스 구조의 선택자 터미널에 연결하면, 구조에서 마우스 오른쪽 버튼을 클릭하고 모든 값에 대한 케이스 추가를 선택하여 컨트롤의 각 아이템 문자열 값에 대한 케이스를 생성할 수 있습니다. 그러나 링 컨트롤을 케이스 구조의 선택자 터미널에 연결하면, 수동으로 각 케이스를 입력해야 합니다.

증가와 감소를 제외한 모든 연산 함수는 열거형 타입 컨트롤을 부호없는 정수와 동일하게 취급합니다. 증가는 마지막 열거형 값을 처음 값으로 변경하고, 감소는 첫번째 열거형 값을 마지막 값으로 변경합니다. 부호있는 정수를 열거형 타입으로 강제 변환할 때, 음수는 첫번째 열거형 값과 같도록 바뀌고, 범위를 벗어난 양수는 마지막 열거형 값과 같도록 바뀝니다. 범위를 벗어난 부호없는 정수는 항상 마지막 열거형 값과 같도록 바뀝니다.

부동소수 값을 열거형 타입 인디케이터에 연결할 경우, LabVIEW는 부동소수 값을 열거형 타입 인디케이터에서 가장 가까운 숫자값으로 강제 변환합니다. LabVIEW는 범위를 벗어난 숫자를 앞에 설명한 것과 같이 다룹니다. 열거형 컨트롤을 임의의 숫자값에 연결할 경우, LabVIEW는 열거형 타입 값을 숫자값으로 강제 변환합니다. 열거형 타입 컨트롤을 열거형 타입 인디케이터에 연결하기 위해서는, 인디케이터의 아이템은 반드시 컨트롤의 아이템과 일치해야 합니다. 하지만, 인디케이터가 컨트롤보다 많은 아이템을 가질 수 있습니다.



팁

열거형 타입 컨트롤을 사용할 경우, 항상 **컨트롤의 타입 정의를 만드십시오**. 타입 정의를 생성해 놓으면, 열거형 타입 컨트롤에 아이템을 추가하거나 제거할 때 마다 코드를 다시 쓸 필요가 없습니다.

문자열 및 경로 컨트롤과 인디케이터

문자열 컨트롤과 인디케이터를 사용하여 텍스트를 입력하거나 보여줍니다. 경로 컨트롤과 인디케이터를 사용하여 파일과 폴더 경로를 입력하거나 디스플레이합니다.

텍스트 필드의 바깥 쪽을 클릭하여 실행 중 문자열 또는 경로 컨트롤에서 블록 다이어그램으로 새 값 또는 변경된 값을 전달합니다. 기본으로, LabVIEW는 사용자가 이 동작을 수행할 때까지 기다린 후 새 값을 해석합니다. 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력할 때 값 업데이트를 선택하여, 문자열 값을 입력하여 업데이트할 수 있습니다. 또한, 입력할 때 업데이트? 프로퍼티를 사용하여 프로그램적으로 값을 업데이트할 수 있습니다.



노트

사용할 수 있는 문자열과 경로 컨트롤과 인디케이터는 선택한 서브팔레트의 스타일에 따라 다릅니다. 사용할 수 있는 문자열과 경로 컨트롤 및 인디케이터는 일반, 실버, 시스템 또는 클래식 서브팔레트를 참조하십시오.

문자열 컨트롤과 인디케이터

문자열 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하여 암호 디스플레이나 16진수 디스플레이와 같은 문자열 디스플레이 타입을 변경할 수 있습니다. 또한 파일 I/O 팔레트의 vi와 함수를 사용하여 문자열을 텍스트 파일이나 스프레드시트와 같은 외부 파일로 전달할 수 있습니다.

관련 정보

문자열 컨트롤과 인디케이터 설정하기

문자열 포맷하기

문자열을 이용한 데이터의 그룹화

콤보 박스 컨트롤

콤보 박스 컨트롤을 사용하여 풀다운 메뉴와 같이 전체를 볼 수 있는 문자열 리스트를 생성할 수 있습니다. 콤보 박스 컨트롤은 데이터를 숫자형이 아닌 문자열로 출력한다는 점에서 링 및 열거형 컨트롤과는 다릅니다.



노트

실행 중에 콤보 박스 컨트롤에 문자열을 입력하면, LabVIEW는 입력한 문자로 시작하는 첫번째 가장 짧은 문자열을 선택합니다. 문자열이 타이핑 문자와 일치하지 않고 컨트롤이 정의되지 않은 문자열 값을 허용하지 않으면, LabVIEW는 컨트롤에서 타이핑 문자를 받지 않거나 디스플레이하지 않습니다.



팁

<Up>과 <Down> 화살표 키를 눌러 콤보 박스 값을 살펴볼 수 있습니다.

관련 정보

콤보 박스 컨트롤에서 아이템 편집하기

콤보 박스 문자열의 사용자 값을 지정하기

런타임으로 콤보 박스 컨트롤에 문자열을 선택하고 입력하기

경로 컨트롤과 인디케이터

또한, 놓기가 활성화되어 있는 경우 (Windows) Windows 탐색기 또는 (macOS) Finder에서 경로, 폴더 또는 파일을 끌어와 경로 컨트롤에 놓을 수 있습니다.



노트

놓기는 기본으로 활성화됩니다. 놓기를 비활성화하려면 경로 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급»놓기 허용을 선택합니다.

경로 컨트롤과 인디케이터는 문자열 컨트롤과 인디케이터와 유사하게 동작하지만, LabVIEW는 사용자의 플랫폼에 대한 표준 구문을 사용하여 경로를 포맷합니다.

유효하지 않은 경로

함수가 경로를 반환하는데 실패할 경우, 함수는 인디케이터에 <경로 없음>을 반환합니다. <경로 없음>을 경로 컨트롤의 기본값으로 사용하면, 사용자가 경로를 제대로 전달하지 못하는 경우 이를 감지하고 경로 선택 옵션이 있는 파일 대화 상자를 디스플레이할 수 있습니다. [파일 대화 상자] 익스프레스 VI를 사용하여 파일 대화 상자를 디스플레이합니다.

빈 경로

빈 경로를 사용하여 사용자가 경로를 지정하도록 입력 요청합니다. 빈 경로는 Windows와 macOS 32비트에서는 빈 문자열로 나타나고, Linux와 macOS 64비트에서는 슬래쉬(/)로 나타납니다. (Windows) 빈 경로를 파일 입력/출력(I/O) 함수에 연결할 때, 빈 경로는 컴퓨터에 매핑된 드라이브의 리스트를 참조합니다. (macOS) 빈 경로가 마운트된 볼륨을 참조합니다. (Linux) 빈 경로가 루트 디렉토리를 참조합니다.

절대 경로와 상대 경로

상대 경로는 파일 시스템의 임의의 위치를 기준으로 파일 또는 디렉토리의 위치를 나타냅니다. 절대 경로는 파일 시스템의 최상위 레벨에서 시작하여 파일 또는 디렉토리의 위치를 나타냅니다. 상대 경로를 사용하면 다른 컴퓨터에서 VI를 실행하거나 어플리케이션을 빌드할 때 경로에 대해 재작업을 하는 것을 피할 수 있습니다.

관련 정보

파일 경로

경로 컨트롤의 탐색 옵션 설정하기

경로 컨트롤의 값 변경하기

배리언트 및 클래스 컨트롤과 인디케이터

배리언트 & 클래스 팔레트에서 배리언트 컨트롤과 인디케이터 및 클래스 컨트롤과 인디케이터를 사용하여 배리언트 및 클래스 데이터와 상호작용합니다.

LabVIEW 객체

LabVIEW 객체는 모든 **LabVIEW 클래스**의 공통 상위 데이터 타입입니다. LabVIEW 객체를 사용하여 모든 LabVIEW 클래스 데이터 타입을 처리할 수 있는 **일반 메소드**를 생성할 수 있습니다.

사용자 컨트롤, 인디케이터, 타입 정의 생성하기

사용자 컨트롤과 인디케이터를 사용하여 프런트패널 객체 세트를 확장합니다. 내장된 LabVIEW 컨트롤과 인디케이터로부터 어플리케이션을 위한 다양한 모양의 사용자 지정 사용자 인터페이스 구성요소를 생성할 수 있습니다. 생성한 사용자 컨트롤 또는 인디케이터를 디렉토리 또는 LLB에 저장한 후 다른 프런트패널에서 사용자 컨트롤 또는 인디케이터를 사용할 수 있습니다. 또한, 사용자 컨트롤 또는 인디케이터를 위한 아이콘을 생성하고 컨트롤 팔레트에 추가할 수 있습니다. 사용자 정의 컨트롤과 인디케이터를 생성하기 전에 **유의점 및 권장사항**을 참조하십시오.

타입 정의와 엄격한 타입 정의를 사용하여 사용자 컨트롤 또는 인디케이터의 모든 인스턴스를 저장된 사용자 컨트롤 또는 인디케이터 파일에 링크합니다. 저장된 사용자 컨트롤 또는 인디케이터 파일만을 편집하여 사용자 컨트롤 또는 인디케이터의 모든 인스턴스를 변경할 수 있으며, 이는 여러 VI에서 같은 사용자 컨트롤 또는 인디케이터를 사용하는 경우에 유용합니다.



노트

또한, **X컨트롤**을 사용하여 LabVIEW에서 복잡한 컨트롤과 인디케이터를 디자인하고 생성할 수 있습니다. 사용자 컨트롤과는 달리, X컨트롤은 배경에서 실행되는 VI로 정의된 다이나믹 런타임과 편집 모드를 가지고 있습니다. LabVIEW Professional Development System에서만 X컨트롤을 생성하고 편집할 수 있습니다. VI에 X컨트롤이 있을 경우, 모든 LabVIEW 패키지에서 그 VI를 실행할 수 있습니다.

컨트롤 편집기 윈도우 사용하기

컨트롤 편집기 윈도우를 사용하여 컨트롤과 인디케이터를 사용자 정의하며 LabVIEW 클래스의 **프라이빗 데이터 컨트롤을 정의**합니다. 예를 들어, 컨트롤 또는 인디케이터의 크기, 색, 원소의 상대적인 위치를 변경하고, 컨트롤이나 인디케이터에 이미지를 반입할 수 있습니다.

다음과 같은 방법으로 컨트롤 편집기 윈도우를 열 수 있습니다:

- 프런트패널 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 사용자 정의를 선택합니다.
- 위치 도구를 사용하여 프런트패널에서 선택한 컨트롤 또는 인디케이터에서 편집 >> 컨트롤 사용자 정의를 선택합니다.

- 새로 만들기 대화 상자를 **사용**합니다.
- 기존 컨트롤이나 컨트롤 템플릿에서 마우스 오른쪽 버튼을 클릭하고 열기를 선택합니다.
- 프로젝트 탐색기 윈도우의 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 새로 만들기»타입 정의를 선택합니다.
- **LabVIEW 클래스를 생성**하고 프라이빗 데이터 컨트롤을 더블 클릭합니다.

노트 클래스 프라이빗 데이터 컨트롤을 정의할 경우, 컨트롤 편집기 윈도우가 자동으로 블록 다이어그램의 클래스 프라이빗 데이터의 클러스터와 함께 열립니다. 또한 LabVIEW는 컨트롤 편집기 윈도우의 링 컨트롤을 클래스 프라이빗 데이터에 기본으로 하며 변경할 수 없습니다.

- 프런트패널 또는 블록 다이어그램의 타입 정의 또는 엄격한 타입 정의의 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의 열기를 선택합니다.

컨트롤 편집기 윈도우는 프런트패널처럼 보이지만, 단일 컨트롤과 인디케이터의 사용자 정의를 위해서만 사용합니다. LabVIEW 클래스의 경우, 단일 컨트롤은 클러스터여야 합니다. 컨트롤 편집기 윈도우는 블록 다이어그램이 없으며 실행되지 않습니다.

컨트롤 편집기 윈도우는 하나의 컨트롤 또는 인디케이터만 포함할 수 있습니다. 다른 서브컨트롤 또는 서브인디케이터를 가진 배열 또는 클러스터의 컨트롤이나 인디케이터를 사용자 정의할 수 있습니다. 배열이나 클러스터 자체는 단일 컨트롤입니다. 컨트롤 또는 인디케이터를 배열 또는 클러스터에 집어넣고 빼는 동안에는 컨트롤 편집기 윈도우의

도구 모음에 유효하지 않은 컨트롤 버튼 이 나타납니다. 유효하지 않은 컨트롤 버튼을 클릭하여 에러 리스트 윈도우에 에러의 설명을 디스플레이합니다.

컨트롤 편집기 윈도우 도구 모음의 모드 인디케이터 버튼은 편집 모드 인지 사용자 정의 모드 인지 보여줍니다. 이 버튼을 클릭하여 두 가지 모드 사이에서 토글합니다. 또한 수행»사용자 정의 모드로 변경 또는 수행»편집 모드로 변경을 선택하여 모드 사이를 전환할 수 있습니다. 컨트롤 편집기 윈도우는 편집 모드로 열립니다.

프런트패널의 편집 모드와 마찬가지로, 편집 모드를 사용하여 컨트롤 또는 인디케이터의 크기나 색을 변경하고 바로 가기 메뉴에서 옵션을 선택합니다

컨트롤 또는 인디케이터의 개별 부분을 변경하여 컨트롤 또는 인디케이터를 포괄적으로 변경할 때 사용자 정의 모드를 사용합니다.

사용자 정의 모드 바로 가기 메뉴

사용자 정의 모드에서 각 부분의 바로 가기 메뉴는 컨트롤 또는 인디케이터 전체의 바로 가기 메뉴를 대체합니다. 다른 부분은 다른 바로 가기 메뉴를 가집니다. 바로 가기 메뉴를 사용하여 다음 작업을 수행합니다:

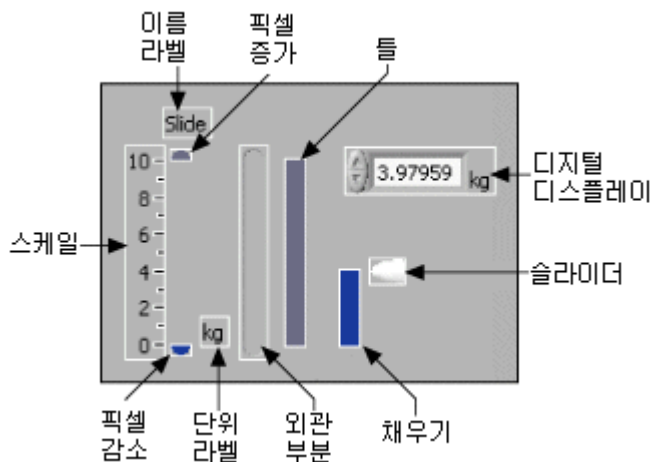
- 사용자 컨트롤 또는 인디케이터에 추가된 장식 팔레트의 객체와 같은 **외관 부분**을 사용자 정의합니다.
- 이름 라벨과 같은 텍스트 부분을 사용자 정의합니다.
- 슬라이드 컨트롤의 디지털 디스플레이를 위해 사용되는 숫자형 컨트롤과 같은 **부분으로 사용되는 컨트롤 또는 인디케이터**를 사용자 정의합니다. 노브, 미터 및 차

트도 디지털 디스플레이를 위해 숫자 컨트롤을 사용합니다. 일부 컨트롤은 좀 더 복잡합니다. 예를 들면, 그래프는 커서 디스플레이 부분에 트리 컨트롤을 사용합니다.

컨트롤 및 인디케이터의 개별 부분 사용자 정의하기

컨트롤 또는 인디케이터의 개별 부분

모든 컨트롤과 인디케이터는 작은 부분으로 구성되어 있습니다. 예를 들어 다음 그림은 슬라이드 컨트롤의 개별 부분을 보여줍니다.



컨트롤 편집기 윈도우에서 사용자 정의 모드로 전환하면 컨트롤 또는 인디케이터의 부분은 개별로 독립됩니다. 다른 부분에 영향을 주지 않고 각 부분을 변경할 수 있습니다. 사용자 정의 모드는 숫자 컨트롤에서 이름 라벨 또는 기수와 같은 편집 모드에서 보이지 않았던 부분들을 포함하여 컨트롤 또는 인디케이터의 모든 부분을 보여줍니다. 컨트롤 또는 인디케이터의 부분이 서로 떨어지게 되므로 사용자 정의 모드에서는 컨트롤 또는 인디케이터의 값을 수행하거나 변경할 수 없습니다.

사용자 정의 모드에서 위치 도구를 사용하여 두 개 또는 그 이상의 컨트롤 또는 인디케이터의 부분을 선택할 수 있으며, 도구 모음에서 **객체 정렬**, **객체 간격 조절**, **순서 재설정** 풀다운 메뉴를 사용하여 각 부분을 다시 정렬할 수 있습니다.

컨트롤 부분 윈도우

윈도우 >> 부분 윈도우 보이기를 선택하여 컨트롤 부분 윈도우를 디스플레이합니다. 컨트롤 부분 윈도우를 사용하여 컨트롤 또는 인디케이터의 개별 부분을 크기 조절하고 위치시킵니다. 컨트롤 부분 윈도우는 컨트롤 또는 인디케이터의 부분을 확인하고 픽셀 단위로 각 부분의 정확한 위치와 크기를 보여줍니다.

컨트롤 부분 윈도우의 현재 부분 섹션은 컨트롤 편집기 윈도우 안에서 선택된 부분의 이미지와 이름을 포함합니다. 현재 부분 디스플레이를 클릭하여 컨트롤 또는 인디케이터의 모든 부분을 디스플레이하고 그 중에서 선택합니다. 현재 부분 증가 또는 감소 화살표를 클릭하여 컨트롤 또는 인디케이터의 부분을 스크롤합니다.

컨트롤 편집기 윈도우에서 부분을 이동하거나 크기를 조정할 때, LabVIEW는 컨트롤 부분 윈도우의 현재 부분과 부분 위치와 차원 섹션에 위치와 크기를 업데이트합니다. 또한 부분 위치와 차원 섹션에 직접 위치와 크기 값을 입력하여 컨트롤 편집기 윈도우에서 부분을 이동하고 크기를 조정할 수 있습니다. 이는 두 부분을 정확히 같은 크기로 만들거나 한 부분을 다른 부분에 맞춰 정렬해야 할 때 유용합니다. 부분 위치와 차원 섹션에 위치와 크기 값을 입력한 후에, <Enter> 키를 누르거나 마우스 왼쪽 버튼을 클릭하면 변경이 적용됩니다.

사용자 컨트롤 및 인디케이터에 외관 부분 추가하기

편집 모드 또는 사용자 정의 모드에서 사용자 컨트롤 또는 인디케이터에 그래픽, 텍스트, 장식을 추가하기 위해 컨트롤 편집기 윈도우를 사용할 수 있습니다. 클립보드에서 그래픽이나 텍스트를 붙여넣고 라벨링 도구를 사용하여 라벨을 생성하거나 장식 팔레트에서 장식을 선택하면, 객체는 컨트롤 또는 인디케이터의 새로운 장식 부분이 되고 프런트 패널에 컨트롤 또는 인디케이터를 놓았을 때 컨트롤이나 인디케이터와 함께 나타나게 됩니다. 컨트롤 편집기 윈도우에서 새로운 장식 부분을 이동, 크기 조정, 계층 순서 변경, 또는 대체할 수 있습니다. 프런트 패널에서 사용자 컨트롤 또는 인디케이터를 사용할 때, 추가한 모든 장식 부분의 크기를 변경할 수 있습니다. 그러나 추가된 장식 부분을 이동할 수는 없습니다.

외관 부분을 위한 사용자 정의 모드 바로 가기 메뉴 아이템

외관 부분은 단일 그래픽, 다른 시간에 디스플레이되는 여러 그래픽 및 각각 독립적인 여러 그래픽을 가질 수 있습니다. 외관 부분을 사용자 정의하려면, 컨트롤 편집기 윈도우를 사용자 정의 모드로 놓고 사용자 정의하려는 외관 부분에서 마우스 오른쪽 버튼을 클릭합니다. 바로 가기 메뉴 아이템은 외관 부분의 타입에 따라 사용 가능 여부가 결정됩니다.

단일 그래픽의 외관 부분

단일 그래픽의 외관 부분을 위한 바로 가기 메뉴는 다음 아이템을 포함합니다.:

- 클립보드에 복사—클립보드에 해당 부분의 그래픽을 복사하여 놓습니다. 클립보드에 형태를 복사한 후, 부분에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 클립보드로부터 그림 반입을 선택하여 부분 안에 형태를 반입합니다. 외관 부분을 사용자 정의하기 위해 단순한 형태가 필요할 때, 그래픽 어플리케이션에서 생성하지 않고 다른 컨트롤 또는 인디케이터의 내장 부분으로부터 형태를 복사하는 것을 고려하십시오. 예를 들면, 단순한 사각 형태가 필요하다면, 틀에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 클립보드에 복사를 선택합니다. 내장 부분이나 장식에서 가져온 그래픽은 그래픽 어플리케이션에서 만든 그래픽보다 크기 조정하기 좋습니다. 예를 들어 그래픽 어플리케이션에서 그린 사각형을 크기 조정하면 사각형은 균일하게 그 영역을 넓혀 증가하고 경계가 두꺼워집니다. 그러나 내장 컨트롤 또는 인디케이터의 부분에서 복사한 사각형을 크기 조정하면 그 사각형은 같은 얇은 경계를 유지합니다. 내장 컨트롤 또는 인디케이터에서 복사한 부분은 컬러 모니터와 흑백 모니터에서 유사한 모양을 가집니다. 추가적으로 색칠 도구를 사용하여 내장 부분 및 장식에서 복사한 그래픽에 색을 추가할 수 있습니다. 다른 소스에서 반입한 그래픽에서는 색이 그래픽 정의의 일부이기 때문에 그래픽의 색을 변경할 수 없습니다.
- 클립보드로부터 그림 반입—외관 부분의 현재 그래픽을 클립보드에 있는 그래픽으로 대체합니다. 예를 들면, 불리언 스위치의 열기와 닫기 밸브의 그래픽을 반입

할 수 있습니다. 클립보드가 그래픽을 포함하고 있지 않으면 클립보드로부터 그림 반입 아이템은 사용할 수 없습니다.

- 클립보드로부터 같은 크기로 반입—현재 그래픽을 클립보드에 있는 그래픽으로 대체하고 클립보드 그래픽을 줄이거나 늘려 맞추어 부분의 원래 크기를 유지합니다. 클립보드가 그래픽을 포함하고 있지 않으면 클립보드로부터 같은 크기로 반입 아이템은 사용할 수 없습니다.
- 파일로부터 반입—외관 부분의 현재 그래픽을 파일 대화 상자에서 선택한 그래픽으로 대체합니다.
- 파일로부터 같은 크기로 반입—현재 그래픽을 파일 대화 상자에서 선택한 그래픽으로 대체하고 파일의 그래픽을 줄이거나 늘려 맞추어 부분의 원래 크기를 유지합니다.
- 되돌리기—부분의 위치를 변경하지 않고 원래의 모양으로 부분을 다시 복원합니다. 프론트패널 컨트롤 또는 인디케이터를 위한 컨트롤 편집기 윈도우를 디스플레이하고 부분을 변경하는 경우, 바로 가기 메뉴에서 되돌리기를 선택하면 부분을 프론트패널에서 보이는 그대로 되돌립니다. 컨트롤 편집기 윈도우에서 사용자 컨트롤 또는 인디케이터를 열면 되돌리기 아이템은 사용할 수 없습니다.
- 원래 크기—부분의 그래픽을 원래의 크기로 복원합니다. 이는 다른 어플리케이션에서 반입한 후 크기를 조정된 그래픽에 유용합니다. 다른 어플리케이션에서 반입된 일부 그래픽은 크기를 조정하면 원본만큼 좋게 보이지 않습니다. 그래픽의 모양을 개선하기 위해 반입된 그래픽을 원래의 크기로 다시 복원하기를 원할 경우도 있습니다. 그래픽을 반입하지 않은 경우, 원래 크기 아이템은 사용할 수 없습니다.



노트 또한, 장식을 선택한 후 편집>>붙여넣기를 선택하여 장식을 클립보드의 이미지로 대체할 수 있습니다. 장식을 파일의 이미지로 대체하려면, 장식에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴의 파일로부터 반입을 선택합니다.

여러 의존 그래픽의 외관 부분

일부 외관 부분은 다른 시간에 디스플레이되는 여러 개의 의존 그래픽을 가집니다. 다른 그래픽은 같은 크기이고 같은 색을 사용합니다. 외관 부분의 현재 그래픽을 이동 또는 크기 조정할 때 다른 그래픽 또한 같은 정도로 이동하거나 변경한 크기에 비례하여 크기가 변경됩니다. 예를 들면 불리언 스위치는 4개의 다른 그래픽을 가집니다. 첫번째 그래픽은 거짓 상태를 보여줍니다. 두번째 그래픽은 참 상태를 보여줍니다. 세번째 그래픽은 **놓을 때 스위치 상태**를 보여주는데, 이것은 참에서 거짓으로 전이 상태입니다. 네번째 그래픽은 참에서 거짓으로 전이 상태인 놓을 때 래치 상태를 보여줍니다. 불리언 컨트롤이 놓을 때 스위치 상태 또는 놓을 때 래치 상태이면 불리언 컨트롤의 값은 마우스 버튼을 놓을 때까지 변경되지 않습니다. 마우스 버튼을 클릭하고 마우스를 놓을 때까지의 시간 사이에 불리언 컨트롤은 전이 상태로 세번째 또는 네번째 그래픽을 보여줍니다.

여러 의존하는 그래픽의 외관 부분을 위한 바로 가기 메뉴는 그림 아이템 아이템과 단일 그래픽의 외관 부분을 위해 사용 가능한 모든 아이템을 포함합니다. 바로 가기 메뉴에서 그림 아이템을 선택하면 외관 부분에 속하는 모든 그래픽을 디스플레이합니다. 현재 그래픽 아이템은 어두운 경계로 둘러싸여집니다. 그래픽을 반입하면 현재 그래픽 아이템만 변경됩니다. 다른 그래픽 아이템 중의 하나를 위해 그래픽을 반입하려면, 부분에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 그림 아이템을 선택하고, 그래픽 아이템을 선택하여 새 그래픽을 반입합니다.

여러 독립되는 그래픽의 외관 부분

하나 이상의 그래픽 외관 부분은 각각 다른 색과 다른 크기의 그래픽을 가질 수 있습니다. 예를 들면 슬라이드는 여러 값의 슬라이드에서 어떤 슬라이드가 활성화되어 있는지 나타내기 위해 서로 다른 크기의 그래픽 두 개를 사용합니다. 여러 독립적인 그래픽의 외관 부분을 위한 바로 가기 메뉴는 독립적인 크기 아이콘과 단일 및 여러 의존 그래픽의 외관 부분을 위해 사용 가능한 모든 아이콘을 포함합니다. 바로 가기 메뉴에서 독립적인 크기 아이콘을 선택하여 외관 부분에서 다른 그래픽의 변경없이 독립적으로 각 그래픽을 이동 및 크기 조정할 수 있도록 옆에 확인 표시를 합니다.

텍스트 부분

텍스트 부분은 라벨과 같은 텍스트를 가진 그래픽입니다. 텍스트 부분을 위한 바로 가기 메뉴는 편집 모드의 프런트패널에서 텍스트 원소를 위해 사용 가능한 아이콘 및 단일 그래픽의 외관 부분을 위해 사용 가능한 아이콘 모두를 포함합니다. 컨트롤 부분 윈도우는 텍스트 자체가 아닌 텍스트 부분을 위한 배경 그래픽만 보여줍니다. 텍스트가 아닌 배경 그래픽만을 사용자 정의할 수 있습니다.

불리언 컨트롤과 인디케이터를 위한 편집 모드 옵션

컨트롤 편집기 윈도우가 사용자 정의 모드일 때만 그래픽을 반입할 수 있는 다른 컨트롤과 인디케이터와는 달리, 컨트롤 편집기 윈도우가 편집 모드일 때에도 그래픽을 불리언 컨트롤과 인디케이터에 반입할 수 있습니다. 편집 모드의 컨트롤 편집기 윈도우에서 불리언 컨트롤 또는 인디케이터를 마우스 오른쪽 버튼으로 클릭하면, 바로 가기 메뉴는 클립보드로부터 그림 반입 아이콘뿐만 아니라 편집 모드의 프런트패널에서 불리언 컨트롤과 인디케이터에 사용 가능한 모든 아이콘을 나타냅니다.

저장된 사용자 컨트롤 또는 인디케이터 파일에 사용자 컨트롤 및 인디케이터 링크하기

사용자 컨트롤 또는 인디케이터를 VI에 놓을 때, 저장한 사용자 컨트롤 또는 인디케이터와 VI의 사용자 컨트롤 또는 인디케이터의 인스턴스는 서로 연결되어 있지 않습니다. 사용자 컨트롤 또는 인디케이터의 각 인스턴스는 개별적이고 독립적인 복사본입니다. 그렇기 때문에, 사용자 컨트롤 또는 인디케이터 파일을 변경해도 이미 사용자 컨트롤 또는 인디케이터를 사용한 VI에는 영향을 주지 않습니다. 사용자 컨트롤 또는 인디케이터의 인스턴스를 사용자 컨트롤 또는 인디케이터 파일에 연결하려는 경우, 사용자 컨트롤 또는 인디케이터를 **타입 정의 또는 엄격한 타입 정의**로 저장합니다. 타입 정의 또는 엄격한 타입 정의의 모든 인스턴스는 해당 인스턴스를 생성한 원래 파일에 링크되어 있습니다.

사용자 컨트롤 또는 인디케이터를 타입 정의 또는 엄격한 타입 정의로 저장할 때, 타입 정의 또는 엄격한 타입 정의에 데이터 타입을 변경하면 그것을 사용하는 모든 VI의 타입 정의 또는 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다. 또한, 엄격한 타입 정의의 모양 변화는 프런트패널의 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다.

타입 정의와 엄격한 타입 정의

타입 정의와 엄격한 타입 정의를 사용하여 사용자 컨트롤 또는 인디케이터의 모든 인스턴스를 저장된 사용자 컨트롤 또는 인디케이터 파일에 링크합니다. 저장된 사용자 컨트롤 또는 인디케이터 파일만을 편집하여 사용자 컨트롤 또는 인디케이터의 모든 인스턴

스를 변경할 수 있으며, 이는 여러 VI에서 같은 사용자 컨트롤 또는 인디케이터를 사용하는 경우에 유용합니다.

타입 정의

타입 정의는 사용자 컨트롤 또는 인디케이터의 각 인스턴스를 위한 정확한 데이터 타입을 확인합니다. 타입 정의의 데이터 타입을 변경하면, 타입 정의의 모든 인스턴스는 자동으로 업데이트됩니다. 즉, 타입 정의의 인스턴스의 데이터 타입은 그 타입 정의가 사용되고 있는 각 VI 안에서 변경됩니다. 그러나 타입 정의는 데이터 타입만 정의하기 때문에, 데이터 타입의 일부인 값만 업데이트됩니다. 예를 들어, 숫자형 컨트롤에서의 데이터 범위는 데이터 타입의 부분이 아닙니다. 그러므로 숫자형 컨트롤의 타입 정의는 그 타입 정의의 인스턴스에 대한 데이터 범위를 정의하지 않습니다. 또한, 링 컨트롤에서 아이템 이름은 데이터 타입을 정의하지 않기 때문에 타입 정의에서 링 컨트롤 아이템 이름을 변경해도 타입 정의의 인스턴스에서 아이템 이름은 변경되지 않습니다. 그러나 열거형 타입 컨트롤을 위한 타입 정의에서 아이템 이름을 변경하면, 아이템 이름이 데이터 타입의 일부이기 때문에 인스턴스는 업데이트됩니다. 타입 정의의 인스턴스는 슬라이드 대신에 노브를 사용하는 것과 같이 컨트롤 또는 인디케이터 고유의 스타일, 캡션, 라벨, 설명, 팁 상자, 기본값, 크기, 색을 가질 수 있습니다.

타입 정의에서 데이터 타입을 변경하면, LabVIEW는 가능한 한 타입 정의의 인스턴스의 이전 기본값을 새 데이터 타입으로 변환합니다. 숫자형 컨트롤 및 인디케이터를 문자열 컨트롤 및 인디케이터로 대체하는 것처럼 데이터 타입이 호환되지 않는 타입으로 변경되면, LabVIEW는 그 인스턴스의 기본값을 보존할 수 없게 됩니다. 타입 정의의 데이터 타입이 이전 타입 정의와 호환되지 않는 데이터 타입으로 변경되면, LabVIEW는 사용자가 .ctl 파일에서 지정한 기본값을 인스턴스의 기본값으로 설정합니다. 기본값을 지정하지 않는 경우, LabVIEW는 그 데이터 타입의 기본값을 사용합니다. 예를 들어 숫자에서 문자열 타입으로 타입 정의를 변경했다면, LabVIEW는 이전 숫자 데이터 타입과 관련된 모든 기본값을 빈 문자열로 대체합니다.

엄격한 타입 정의

엄격한 타입 정의는 캡션, 라벨, 설명, 팁 상자, 기본값을 제외한 인스턴스의 모든 것을 엄격한 타입 정의와 동일하도록 합니다. 엄격한 타입 정의의 데이터 타입은 엄격한 타입 정의를 사용한 모든 곳에 똑같이 남게 됩니다. 엄격한 타입 정의는 또한 링 컨트롤의 아이템 이름, 숫자 컨트롤의 범위 확인과 같이 다른 값도 정의합니다. 엄격한 타입 정의에서는 VI 서버 프로퍼티 중 보이기, 비활성화, 키 포커스, 깜박거림, 위치, 경계와 같이 컨트롤 및 인디케이터의 모양에 영향을 주는 요소만을 사용할 수 있습니다.



노트 상수를 엄격한 타입 정의에 링크해도 상수는 링크가 엄격하지 않은 것처럼 동작합니다. 엄격한 타입 정의에서 타입 정의의 모든 인스턴스는 동일한 외양적 속성을 유지해야 합니다. 그러나 대부분의 외양적 속성은 블록 다이어그램에서 유효하지 않습니다. 상수는 블록 다이어그램에만 존재하기 때문에 외양적 속성을 유지하지 않습니다.

인스턴스와 엄격한 타입 정의 사이의 **링크를 제거**하지 않으면 엄격한 타입 정의 인스턴스의 자동 업데이트를 방지할 수 없습니다.

타입 정의와 엄격한 타입 정의 생성 및 편집하기

프론트패널 또는 블록 다이어그램에서 **타입 정의 또는 엄격한 타입 정의를 생성하고 편집**할 수 있습니다. 컨트롤 편집기 윈도우를 사용하여 그리고 다른 컨트롤과 인디케이터를 사용자 정의하는데 사용하는 것과 동일한 기술을 사용하여 타입 정의 및 엄격한 타입 정의를 사용자 정의합니다.

타입 정의 인스턴스의 자동 업데이트 방지하기

인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 타입 정의로부터 자동 업데이트 바로 가기 메뉴 항목 옆의 확인 표시를 제거해 타입 정의의 데이터 타입이 변경되었을 때 타입 정의의 개별 인스턴스가 자동적으로 업데이트 되는 것을 방지할 수 있습니다. 타입 정의의 데이터 타입은 변경되나 인스턴스와 타입 정의 사이의 링크는 보존하려할 때 타입 정의로부터 자동 업데이트 항목의 확인 표시를 삭제하여 인스턴스의 자동 업데이트를 방지합니다. 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의로부터 연결 끊기를 선택하여 인스턴스와 타입 정의의 링크를 제거할 수 있습니다.



노트 인스턴스와 엄격한 타입 정의 사이의 **링크를 제거**하지 않으면 엄격한 타입 정의 인스턴스의 자동 업데이트를 방지할 수 없습니다.

타입 정의 인스턴스의 자동 업데이트를 비활성화하고 인스턴스의 데이터 타입을 변경하거나 타입 정의의 데이터 타입을 변경하면, 인스턴스는 회색화되어 나타나고 실행 버튼이 깨져 나타납니다. **수동으로 인스턴스를 업데이트**하려면, 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 타입 정의로부터 검토 및 업데이트를 선택하여 타입 정의로부터 검토 및 업데이트 대화 상자를 디스플레이합니다. 인스턴스를 업데이트하지 않고 실행 되도록 어플리케이션을 활성화하려면, 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의로부터 연결 끊기를 선택하여 인스턴스와 타입 정의 사이의 **링크를 제거**합니다.

수동으로 타입 정의 인스턴스 업데이트하기

타입 정의의 인스턴스가 회색화되어 나타나는 경우, 타입 정의로부터 검토 및 업데이트 대화 상자를 사용하여 반드시 **인스턴스를 수동으로 업데이트**해야 합니다.

타입 정의의 데이터 타입을 변경하면, LabVIEW는 자동으로 타입 정의의 모든 인스턴스에 대한 업데이트를 시도합니다. 그러나, 인스턴스를 업데이트할 때 LabVIEW가 이전 기본값을 업데이트된 컨트롤 또는 상수에 맵핑할 수 없는 경우, LabVIEW는 이러한 인스턴스를 결정되지 않은 상태로 둡니다. 결정되지 않은 인스턴스는 비활성화되고, 이러한 인스턴스가 있는 VI의 실행 버튼은 깨져서 나타납니다. 데이터 맵핑 차이를 해결하려면 인스턴스를 수동으로 업데이트해야 합니다. 또는, 결정되지 않은 인스턴스와 타입 정의 사이의 링크를 제거하여 어플리케이션의 실행을 활성화할 수 있습니다.

인스턴스와 타입 정의 또는 엄격한 타입 정의 사이의 링크 제거하기

인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의로부터 연결 끊기를 선택하여 타입 정의 또는 엄격한 타입 정의의 인스턴스와 타입 정의 또는 엄격한 타입 정의 사이의 링크를 영구히 제거할 수 있습니다. 타입 정의 또는 엄격한 타입 정의로부터 연결을 끊으면 인스턴스의 데이터 타입에서 제한을 제거하여 인스턴스를 컨트롤 또는 인디케이터로 만듭니다.

타입 정의와 엄격한 타입 정의의 사용 불가능

타입 정의 또는 엄격한 타입 정의의 각 인스턴스에서 인스턴스를 포함하는 VI는 타입 정의 또는 엄격한 타입 정의를 저장한 파일로의 연결을 유지합니다. 타입 정의 또는 엄격한 타입 정의의 인스턴스를 포함하는 VI가 실행되려면, VI는 반드시 사용 가능한 타입 정의 또는 엄격한 타입 정의를 포함한 파일을 가지고 있어야 합니다. VI를 열고 LabVIEW가 VI가 필요로 하는 타입 정의 또는 엄격한 타입 정의를 찾을 수 없는 경우, LabVIEW는 VI에서 타입 정의 또는 엄격한 타입 정의의 인스턴스를 비활성화시키고 실행 버튼은 깨져서 나타납니다. VI가 실행하도록 하려면 정확한 타입 정의 또는 엄격한 타입 정의를 찾아서 엽니다. 또는 비활성화된 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의로부터 연결 끊기를 선택하여 타입 정의 또는 엄격한 타입 정의와 인스턴스 사이의 링크를 영구히 제거할 수 있습니다.

타입 정의와 엄격한 타입 정의의 사용하기

프론트패널 또는 블록다이어그램에 타입 정의 및 엄격한 타입 정의를 프론트패널 또는 블록다이어그램에 사용자 컨트롤 또는 인디케이터를 놓는 방법과 같은 방법으로 놓습니다. 타입 정의 또는 엄격한 타입 정의의 인스턴스에서 마우스 오른쪽 버튼을 클릭할 때, 바로 가기 메뉴는 타입 정의 옵션을 포함합니다. 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의 열기를 선택하여 프론트패널 또는 블록다이어그램에서 타입 정의 또는 엄격한 타입 정의를 열 수 있습니다.

인스턴스의 왼쪽 위 코너에 표시되는 문양 으로 타입 정의 또는 엄격한 타입 정의의 인스턴스를 식별할 수 있습니다. 문양 위로 커서를 움직이면, 톱 상자가 타입 정의 또는 엄격한 타입 정의의 이름을 디스플레이합니다. 타입 정의를 변경하면 변경이 프론트패널 인스턴스에 영향을 주는 것과 같이 블록다이어그램에 상수로 놓은 인스턴스에 영향을 줍니다.



노트 엄격한 타입 정의를 변경하면, 블록다이어그램에서 상수로 놓은 인스턴스가 링크가 엄격하지 않은 것처럼 동작하며 LabVIEW는 데이터 타입 변경에서만 상수를 업데이트합니다. 예를 들어, 엄격한 타입 정의로 저장한 **링크 컨트롤**의 문자열 값을 업데이트하면 LabVIEW는 문자열 값이 링크 컨트롤의 데이터 타입 일부가 아니므로 블록다이어그램에 있는 엄격한 타입 정의의 인스턴스를 업데이트하지 않습니다.

클러스터 타입 정의와 엄격한 클러스터 타입 정의 사용하기

프론트패널에 클러스터 타입 정의와 엄격한 클러스터 타입 정의의 인스턴스를 놓으려면, 블록다이어그램에서 [이름으로 묶기] 함수와 [이름으로 풀기] 함수를 사용하여 클러스터의 원소에 접근합니다. 클러스터 순서로 클러스터 원소를 참조하는 [묶기] 함수 및 [풀기] 함수와는 다르게 [이름으로 묶기] 함수 및 [이름으로 풀기] 함수는 이름으로 클러스터의 원소를 참조합니다. 그러므로 클러스터 타입 정의 또는 엄격한 클러스터 타입 정의에 원소 순서를 다시 설정하거나 새로운 원소를 추가해도 VI는 깨지지 않습니다.

[이름으로 묶기] 함수 또는 [이름으로 풀기] 함수에 연결된 클러스터 타입 정의 또는 엄격한 클러스터 타입 정의로부터 원소를 삭제하려면 반드시 블록다이어그램을 변경해야 합니다. 없어진 원소의 참조는 유효하지 않게 되기 때문입니다. [이름으로 묶기] 함수와 [이름으로 풀기] 함수 안의 유효하지 않은 이름은 검정색으로 나타납니다. [이름으로 묶기] 함수 또는 [이름으로 풀기] 함수를 클릭하여 선택할 유효한 이름의 리스트를 디스플레이합니다.

클러스터 타입 정의 또는 엄격한 클러스터 타입 정의에서 컨트롤의 참조를 생성할 때, LabVIEW는 클러스터 순서를 사용하여 클러스터에서 참조가 어떤 컨트롤을 가리키는지 결정합니다. 이와 같은 동작 때문에, 클러스터 타입 정의나 엄격한 클러스터 타입 정의에서 컨트롤을 삭제하면, 삭제된 컨트롤을 가리키는 남은 참조가 이제 클러스터에서 다른 컨트롤을 가리킬 수 있습니다. 클러스터 타입 정의 또는 엄격한 클러스터 타입 정의에서 컨트롤을 삭제하는 경우, 클러스터에서 각 참조가 여전히 올바른 컨트롤을 가리키는지 확인합니다.

LabVIEW의 X컨트롤

X컨트롤 사용하기

X컨트롤을 사용하여 LabVIEW에서 복잡한 컨트롤과 인디케이터를 설계하고 생성할 수 있습니다.



노트

LabVIEW Professional Development System에서만 X컨트롤을 생성하고 편집할 수 있습니다. VI에 X컨트롤이 있을 경우, 모든 LabVIEW 패키지에서 그 VI를 실행할 수 있습니다.

X컨트롤을 생성할 때, 내장되어 있는 LabVIEW 컨트롤과 인디케이터를 결합하여 내장된 LabVIEW 컨트롤과 인디케이터와 결합된 기능의 이점을 가질 수 있습니다. 내장된 LabVIEW 컨트롤과 인디케이터로부터 외형적으로 다른 사용자 인터페이스 구성요소를 생성하는데 사용하는 **사용자 컨트롤**과 달리, X컨트롤은 배경에서 실행되는 VI로 정의되는 다이나믹한 실행 시간과 편집 시간을 가지고 있습니다. VI에서 X컨트롤을 사용할 때, 이 VI의 블록 다이어그램은 X컨트롤이 컨트롤의 동작을 포함하기 때문에 단순화됩니다.

X컨트롤을 생성할 때, **프로젝트 탐색기** 윈도우를 사용하여 X컨트롤 라이브러리를 편집할 수 있습니다. X컨트롤 라이브러리에는 X컨트롤의 모든 구성요소가 있습니다. 전형적인 X컨트롤은 기능, 프로퍼티, 메소드로 구성됩니다. X컨트롤 **기능**은 X컨트롤이 올바르게 작동하는데 요구되는 X컨트롤의 기본 요소입니다. 요청된 기능에는 모양, 데이터 타입, 디스플레이 상태, X컨트롤의 초기화가 정의되어 있습니다. X컨트롤은 추가적이고 옵션적인 기능을 가질 수 있습니다. 기능 선택 대화 상자에는 X컨트롤에 적용할 수 있는 필수적이고 옵션적인 모든 기능을 열거하고 설명합니다. X컨트롤의 **프로퍼티**와 **메소드**를 사용하여 X컨트롤의 사용자는 컨트롤을 프로그램적으로 설정할 수 있습니다. X컨트롤에서 프로퍼티와 메소드를 생성하는 것은 옵션입니다.

X컨트롤 사용하기에 대한 예제는 labview\examples\Controls and Indicators\XControls\Dual Mode Thermometer 디렉토리의 Simple Dual Mode Thermometer XControl 프로젝트를 참조하십시오. 관련 예제 찾기

X컨트롤 기능 사용하기

X컨트롤 기능은 X컨트롤이 올바르게 작동하는데 요구되는 X컨트롤의 기본 요소입니다. LabVIEW가 적절한 시기에 호출하는 VI나 컨트롤(.ctl)을 통해 기능을 나타냅니다. 요청된 기능에는 모양, 데이터 타입, 디스플레이 상태, X컨트롤의 초기화가 정의되어 있습니다. X컨트롤은 다음과 같은 필수 기능을 포함합니다:

- 데이터—데이터는 X컨트롤의 데이터 타입을 지정하는 **타입 정의** VI입니다. X컨트롤 라이브러리에서 데이터 기능을 더블 클릭하여 데이터 타입 정의를 열고 편집합니다. 기본적으로, 데이터 타입 정의는 숫자형 컨트롤을 포함합니다. VI의 기존 컨트롤을 X컨트롤의 데이터 타입을 나타내는 새로운 컨트롤로 대체합니다. 데이터 타입 정의를 변경한 후 저장합니다.
- 상태—디스플레이 상태로 알려진 상태는 컨트롤의 모양에 영향을 주는 X컨트롤의 데이터 이외의 모든 정보를 지정하는 **타입 정의** VI입니다. 언제든지, X컨트롤은 데이터와 상태 기능을 사용하여 그 모양을 업데이트할 수 있습니다. X컨트롤 라이브러리에서 상태 기능을 더블 클릭하여 상태 타입 정의를 열고 편집합니다. 기본적으로, 상태 타입 정의는 숫자형 컨트롤을 가진 클러스터를 포함합니다. VI의 기존 컨트롤을 X컨트롤의 디스플레이 상태를 나타내는 새로운 컨트롤로 대체합니다. 상태 타입 정의를 변경한 후 저장합니다.



노트

디스플레이 상태의 타입이 변하면, X컨트롤 프로퍼티 대화 상자를 사용하여 X컨트롤의 버전을 업데이트합니다. 이 버전이 변경되었으면, 초기화를 사용하여 상태 기능을 업데이트할 수 있습니다.

- 외관—외관은 X컨트롤의 모양을 정의합니다. 외관 기능은 외관 VI로 나타냅니다. 외관 VI의 프런트패널은 X컨트롤의 모양을 정의합니다. 외관 VI의 블록다이어그램은 X컨트롤의 동작을 설명합니다. X컨트롤 라이브러리의 외관 기능을 더블 클릭하여 열리는 **외관 VI를 편집**합니다. 외관 VI를 변경한 후 저장합니다.
- 초기화—X컨트롤이 처음으로 프런트패널에 놓여질 때나 X컨트롤을 포함하는 VI가 메모리에 로드될 때 LabVIEW는 초기화 기능을 호출합니다. 이 기능을 사용하여 X컨트롤이 디스플레이 되기 전에 디스플레이 상태를 초기화할 수 있습니다. 또한, 초기화 기능은 X컨트롤의 이전 버전을 가장 최신 버전으로 업데이트하는데 중요한 역할을 합니다. X컨트롤이 있는 VI를 로드할 때, LabVIEW는 초기화를 호출하고 X컨트롤의 버전이 VI가 저장된 이후로 변경되었는지 검사합니다. 이 버전이 변경되었으면, 초기화를 사용하여 상태 기능을 업데이트할 수 있습니다. 초기화는 이전 버전과 이전 상태 컨트롤을 포함하고, 현재 상태 인디케이터를 포함합니다. 버전이 변경될 때, 초기화는 이전 상태의 값을 변환하고 이것은 배리언트로 새로운 상태 포맷에 전달될 것입니다. 새로운 상태가 현재 상태 인디케이터에 전달됩니다.

X컨트롤은 추가적이고 옵션적인 기능을 가질 수 있습니다. 기능 선택 대화 상자에는 X컨트롤에 적용할 수 있는 필수적이고 옵션적인 모든 기능을 열거하고 설명합니다.

X컨트롤 외관 기능 사용하기

외관 **기능**은 **X컨트롤**의 모양을 정의합니다. 외관 기능은 외관 VI로 나타냅니다. X컨트롤이 외관 기능을 요구하므로, 사용자가 X컨트롤을 생성할 때 LabVIEW는 자동으로 외관 VI를 X컨트롤 라이브러리에 추가합니다. 외관 VI는 **재호출** VI여야만 합니다.

외관 VI 편집하기

외관 VI에 LabVIEW 컨트롤과 인디케이터를 추가하여 X컨트롤의 프런트패널을 생성합니다. 기본으로, 외관 VI는 데이터, 디스플레이 상태, 컨테이너 상태 컨트롤과 데이터, 디스플레이 상태, 작업 인디케이터를 포함합니다. 이 컨트롤과 인디케이터는 숨겨져 있으며 X컨트롤 모양의 일부가 아닙니다. 프런트패널 윈도우의 크기를 조정하여 X컨트롤의 모양을 구성하는 컨트롤과 인디케이터에만 맞도록 합니다. 외관 VI의 **윈도우 모양 프로퍼티**를 설정하여 X컨트롤의 **크기**와 **모양**을 조절합니다.

외관 VI의 블록 다이어그램은 프런트패널 컨트롤과 외관 VI 자체의 이벤트를 다룹니다. 이벤트 구조를 가진 While 루프를 포함합니다. 외관 VI는 연속적으로 실행하는 VI가 아닙니다. LabVIEW는 VI가 처리해야 할 지연된 이벤트가 있을 때만 외관 VI를 호출합니다. LabVIEW는 현재 데이터, 디스플레이 상태, 컨테이너 상태와 함께 외관 VI를 호출합니다. VI는 이벤트를 처리하고 업데이트된 데이터, 디스플레이 상태, 수행된 동작의 타입을 반환합니다.

컨테이너 상태 클러스터를 사용하여 X컨트롤의 인스턴스에 대한 정보를 제공합니다. 컨테이너 상태 클러스터는 세 원소 즉, 인디케이터?, 실행 모드?, 참조 번호를 포함합니다. 인디케이터?를 사용하여 X컨트롤 인스턴스가 컨트롤인지 또는 인디케이터인지 결정합니다. 실행 모드?를 사용하여 X컨트롤을 사용하는 VI가 실행중인지 유휴인지 결정합니다. 실행 모드?는 객체소유 VI가 실행 중일 때 참으로 설정됩니다. 참조 번호는 X컨트롤에 대한 정보를 쿼리하는 데에만 사용하는 X컨트롤 인스턴스의 참조입니다. 참조 번호를 사용하여 외관 VI의 X컨트롤 인스턴스를 수정하지 마십시오.

외관 VI가 데이터나 디스플레이 상태를 변경할 때, 외관 VI는 동작 클러스터를 업데이트하고 LabVIEW에 변경을 알립니다. 작업 클러스터는 세 원소 즉, 변경된 데이터, 변경된 상태, 작업 이름을 포함합니다. 외관 VI가 데이터 출력 인디케이터를 변경할 때 변경된 데이터를 참으로 설정합니다. 외관 VI가 디스플레이 상태 출력 인디케이터를 변경할 때 변경된 상태를 참으로 설정합니다. 옵션으로, 작업 이름을 사용하여 수행된 동작의 이름을 붙입니다. 이 이름은 편집>실행 취소 옵션을 선택할 때 나타납니다. 일반적으로, 외관 VI는 프런트패널 컨트롤에 생성된 이벤트에 대한 응답으로 데이터나 디스플레이 상태를 변경합니다.

외관 VI 이벤트

또한, 외관 VI는 특정한 사용자 상호작용에 대응하여 생성된 이벤트에 응답합니다. 이러한 이벤트에는 데이터 변경, 디스플레이 상태 변경, 방향 변경, 실행 상태 변경 이벤트가 포함됩니다. 외관 VI는 데이터 변경과 디스플레이 상태 변경 이벤트를 처리하여 X컨트롤이 적절하게 기능하도록 해야 합니다. 방향 변경과 실행 상태 변경 이벤트 처리는 옵션입니다.

LabVIEW는 외관 VI를 호출하여 이벤트를 처리합니다. 예를 들어, X컨트롤을 복사하면 LabVIEW는 4 개의 이벤트 모두를 생성하고, 외관 VI를 X컨트롤의 새 인스턴스로 호출합니다. LabVIEW는 또한 X컨트롤을 포함하는 VI의 프런트패널이 로드될 때에도 4 개의 이벤트 모두를 생성하고 외관 VI를 호출합니다. X컨트롤을 포함하는 VI가 편집 모드에서 실행 모드로 전환하거나 그 반대로 전환하는 경우, LabVIEW는 실행 상태 변경 이벤트를 생성하고 VI의 실행 상태 변화에 응답할 수 있도록 외관 VI를 호출합니다.

편집>실행 취소 또는 편집>다시 실행을 선택할 때에도 LabVIEW는 외관 VI를 호출합니다. LabVIEW는 외관 VI의 동작이 상태를 변경하기 전에 X컨트롤의 디스플레이 상태를 되돌립니다. 현재 디스플레이 상태가 이전 상태를 대체합니다. LabVIEW는 디스플레이 상태 변경 이벤트를 생성하고, 외관 VI를 호출하여 실행 취소 또는 다시 실행 동작 이후에 X컨트롤을 업데이트합니다.

각 이벤트 케이스를 편집하여 X컨트롤이 각 이벤트에 응답하는 방법을 설정합니다. 빨리 응답하는 X컨트롤을 생성하려면, 이벤트를 처리하고 최대한 빨리 끝내도록 외관 VI를 설

정합니다. [기다림] 함수를 사용하여 이벤트 구조 안의 While 루프의 실행을 지연하지 않습니다.



노트

또한, 외관 VI를 사용하여 X컨트롤의 **바로 가기 메뉴를 프로그램적으로 사용자 정의**할 수 있습니다. 바로 가기 메뉴 편집기를 사용하여 X컨트롤의 바로 가기 메뉴를 사용자 정의할 수 없습니다.

VI에서 X컨트롤 사용하기

X컨트롤을 생성한 후에 내장된 LabVIEW 컨트롤이나 인디케이터를 사용하는 것처럼 VI에서 이 X컨트롤을 사용할 수 있습니다. VI에서 X컨트롤을 사용할 때, 이 VI의 블록 다이어그램은 X컨트롤이 컨트롤의 동작을 포함하기 때문에 단순화됩니다. VI의 프런트패널에 X컨트롤을 놓고, 컨트롤 팔레트에서 컨트롤 아이콘을 선택하고 프런트패널에 놓으려는 X컨트롤 .Xctl을 탐색합니다.



노트

프런트패널에 X컨트롤을 놓을 때, LabVIEW는 X컨트롤 라이브러리를 로드합니다. LabVIEW는 VI가 X컨트롤을 사용하고 있는 동안 X컨트롤 라이브러리를 잠급니다. VI가 X컨트롤을 사용하고 있는 동안 X컨트롤 라이브러리를 편집하려면 **X컨트롤 라이브러리를 고정해제**해야 합니다.

X컨트롤을 VI에 추가한 후에, 내장된 LabVIEW 컨트롤이나 인디케이터를 사용하는 것처럼 이 X컨트롤을 사용합니다. 프런트패널에 잘 맞도록 X컨트롤의 **크기를 조정**할 수 있습니다. 컨트롤에 대해 사용자와 내장된 메뉴 항목을 접근할 수 있습니다. 또한, X컨트롤에 설치된 사용자 **프로퍼티**와 **메소드**를 통해서 X컨트롤을 프로그램적으로 설정할 수 있습니다.

X컨트롤 사용하기에 대한 예제는 labview\examples\Controls and Indicators\XControls\Dual Mode Thermometer 디렉토리에 있는 Run Dual Mode Thermometer XControl VI 프로젝트를 참조하십시오. 관련 예제 찾기

LabVIEW의 함수

함수 개요

함수는 LabVIEW의 필수 실행 원소입니다. 함수 팔레트의 함수 아이콘은 연한 노란색 배경과 검은색 전경을 가집니다. 함수는 프런트패널 또는 블록 다이어그램을 가리지 않지만 커넥터 팬을 가집니다. 함수를 열거나 편집할 수 없습니다.

함수 팔레트에는 또한 LabVIEW와 함께 제공되는 VI들이 있습니다. 어플리케이션에서 SubVI로 **내장된 VI를 사용**하면 개발시간을 단축할 수 있습니다.

함수에 터미널 추가하기

일부 함수의 터미널 개수를 바꿀 수 있습니다. 예를 들어, 10개의 원소를 가지는 배열을 만들려면, 반드시 10개의 터미널을 [배열 만들기] 함수에 추가해야만 합니다.

위치 도구를 사용하여 함수의 상위 또는 하위 경계를 각각 위 아래로 끌어서 함수에 터미널을 추가할 수 있습니다. 또한 위치 도구를 사용하여 함수에서 터미널을 제거할 수 있지만, 이미 연결된 터미널은 제거할 수 없습니다. 터미널을 제거하기 위해서는 반드시 기존의 와이어를 먼저 삭제해야만 합니다.

또한, 함수의 터미널 중 하나에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 추가, 출력 추가, 입력 제거, 또는 출력 제거를 선택하여 터미널을 추가하거나 제거할 수 있습니다. 함수에 따라서, 입력, 출력, 또는 참조 번호 컨트롤에 터미널을 추가할 수 있습니다. 입력 추가 및 출력 추가 바로 가기 메뉴 아이템은 마우스 오른쪽 버튼을 클릭한 터미널 바로 다음에 터미널을 추가합니다. 입력 제거와 출력 제거 바로 가기 메뉴 아이템은 마우스 오른쪽 버튼을 클릭한 터미널을 제거합니다. 바로 가기 메뉴 아이템을 사용하여 연결된 터미널을 제거하는 경우, LabVIEW는 터미널을 제거하고 와이어를 끊습니다.

내장 항목

LabVIEW는 항목을 사용하여 함수 팔레트에 있는 VI와 함수를 정리합니다. LabVIEW를 설치하면, 내장되어 있는 기본 항목이 함수 팔레트에 나타납니다. 일부 내장 항목에는 기본적으로 객체가 포함되어 있지 않습니다. 특정 모듈, 툴킷, 드라이버를 설치한 후에, 기본적으로 비어있는 내장 항목에 객체가 나타나게 됩니다.

LabVIEW에 내장되어 있지 않은 항목도 있습니다. 내장되지 않은 항목은 특정 모듈, 툴킷, 드라이버를 설치한 후에만 나타납니다.

함수 팔레트의 사용자 정의 버튼을 클릭한 후, 바로 가기 메뉴에서 팔레트 보기 변경을 선택하여 함수 팔레트의 모든 내장된 항목을 디스플레이합니다. 또한 고정된 함수 팔레트의 아래에 있는 팔레트 보기 변경 링크를 클릭하여 내장된 항목을 디스플레이할 수 있습니다. 다음은 함수 팔레트에 있는 내장 항목입니다.

- **프로그래밍** — 프로그래밍 VI와 함수를 VI를 만드는 기본 블록으로 사용합니다.
- **측정 I/O** — 측정 I/O VI 및 함수를 사용하여 데이터 수집 디바이스에 인터페이스로 연결합니다. 이 팔레트는 설치한 하드웨어 드라이버에 대한 VI와 함수를 디스플레이합니다. 대부분의 일반 National Instruments 하드웨어 드라이버는 LabVIEW에 함께 제공됩니다.
- **인스트루먼트 I/O** — 인스트루먼트 I/O VI와 함수를 사용하여 GPIB, 시리얼, 모듈러, PXI 및 기타 유형의 인스트루먼트에 인터페이스로 연결합니다. 이 팔레트는 GPIB, 시리얼 등 설치되어 있는 인스트루먼트 드라이버를 디스플레이합니다.
- **비전과 모션** — 비전과 모션 VI 및 함수를 사용하여 NI Vision 및 Motion 제품에 인터페이스로 연결합니다. 이 팔레트는 사용자가 설치하는 NI Vision, NI-Motion 등과 같은 NI 제품에 대한 VI 및 함수를 디스플레이합니다.
- **수학** — 수학 VI를 사용하여 다양한 종류의 수학적 분석을 수행합니다. 또한 실용적인 솔루션을 얻기 위해서 실제 측정을 수학 알고리즘에 결부시킬 수 있습니다.
- **신호 처리** — 신호 처리 VI를 사용하여 신호 생성, 디지털 필터링, 데이터 윈도우, 스펙트럼 분석을 수행합니다.

- 데이터 통신 — 데이터 통신 VI와 함수를 사용하여 어플리케이션 간에 데이터를 교환합니다.
- 연결 — 연결 VI와 함수를 사용하여 .NET 객체, ActiveX 활성화 어플리케이션, 입력 디바이스, 레지스터 주소, 소스 컨트롤, Windows 레지스트리 키와 연계된 작업을 처리합니다.
- 컨트롤 & 시뮬레이션 — 컨트롤 & 시뮬레이션 VI와 함수를 사용하여 다이나믹 시스템 모델을 설계, 분석, 시뮬레이션하고 배포합니다. 이 팔레트에는 Control Design and Simulation Module과 같은 NI Control Design and Simulation 제품을 위한 VI와 함수가 포함되어 있습니다.
- 익스프레스 — 익스프레스 VI와 함수를 사용하여 일반적인 측정 태스크를 만듭니다.
- 애드온 — 애드온 항목을 사용하여 LabVIEW에 설치한 일부 모듈 또는 툴킷의 팔레트를 찾습니다.
- 즐겨찾기 — 즐겨찾기 항목을 사용하여 함수 팔레트에서 자주 사용하는 아이템을 모아서 그룹화합니다. 사용자가 직접 즐겨찾기 항목에 **아이템을 추가**할 수 있습니다.
- 사용자 라이브러리 — 사용자 라이브러리 항목을 사용하여 함수 팔레트에 **VI를 추가**합니다. 기본적으로, 사용자 라이브러리 항목은 어떤 객체도 포함하지 않습니다.

다형성 함수

다형성은 VI와 함수가 다른 데이터 타입의 입력 데이터를 받을 수 있도록 자동으로 적응하는 것입니다. 함수의 다형성 정도는 다양하며, 다형성 입력이 없거나, 일부 또는 전체 입력이 다형성일 수 있습니다. 일부 함수의 입력은 숫자값과 불리언 값을 받습니다. 일부는 숫자값 또는 문자열을 받습니다. 일부는 스칼라 숫자값뿐만 아니라 숫자값의 배열, 숫자값의 클러스터, 숫자값 클러스터의 배열 등을 받습니다. 일부는 1차원 배열만 받을 수 있는데, 배열의 원소는 어떤 타입도 될 수 있습니다. 일부 함수는 복소수 값을 포함한 모든 타입의 데이터를 받습니다. 또한, **다형성 단위**를 생성하고 사용할 수 있습니다.

숫자형 변환

모든 숫자형은 다른 숫자형으로 변환할 수 있습니다. 다른 형의 둘 이상의 숫자 입력을 하나의 함수에 연결할 때, 일반적으로 함수는 더 크고 넓은 형으로 데이터를 반환합니다. 함수는 실행하기 전에 작은 형을 가장 넓은 형으로 강제 변환하며, LabVIEW는 변환이 일어난 터미널에 **강제 변환점**을 놓습니다.

나누기, 사인, 코사인과 같은 일부 함수는 기본적으로 부동소수 출력을 생성합니다. 이 입력에 정수를 연결하는 경우, 해당 함수는 계산을 수행하기 전에 정수를 배정도 부동소수로 변환합니다. 그러나 **대부분의 숫자형 함수를 설정**하여 지정한 데이터 타입으로 반환할 수 있습니다.

부동소수 스칼라 값의 경우, 일반적으로 배정도 부동소수를 사용하는 것이 가장 좋습니다. 단정도 부동소수는 크기가 작아서 런타임을 거의 또는 전혀 저장하지 못하고 쉽게 **오버플로우**가 발생할 수 있습니다. 예를 들어, 분석 라이브러리는 배정도 부동소수를 사용합니다. 꼭 필요한 경우에만 확장형 부동소수를 사용해야 합니다. 확장형 연산의 성능과 정밀도는 플랫폼에 따라 다양합니다.

정수의 경우, 일반적으로 32 비트 부호있는 정수를 사용하는 것이 가장 좋습니다.

출력을 다른 숫자형을 가진 대상에 연결한 경우, LabVIEW는 다음 원칙에 따라서 데이터를 변환합니다:

- 부호있는 또는 부호없는 정수를 부동소수로—LabVIEW는 데이터를 가장 가까운 값으로 변환합니다. 이 변환은 부동소수가 정수보다 더 정밀할 때 정확합니다. 정수가 부동소수보다 더 정밀할 경우, LabVIEW는 변환에서 최대한 정밀도를 유지합니다.
- 부동소수를 부호있는 또는 부호없는 정수로—LabVIEW는 범위를 벗어나는 값을 정수의 최소 또는 최대값으로 보냅니다. For 루프의 반복 터미널과 같은 대부분의 정수 객체는 부동소수를 반올림합니다. LabVIEW는 0.5의 소수 부분을 가장 가까운 짝수로 반올림합니다. 예를 들어, LabVIEW는 6.5를 7이 아니라 6으로 반올림합니다.
- LabVIEW는 열거형을 부호없는 정수로 취급합니다—예를 들어, 부동소수 -1을 열거형으로 변환하는 경우, LabVIEW는 이 값을 열거형 범위 안으로 강제 변환합니다. 열거형의 범위가 0 ~ 25인 경우, LabVIEW는 이 값을 0으로 변환합니다.
- 정수를 정수로—LabVIEW는 범위를 벗어나는 값을 정수의 최소 혹은 최대값으로 보내지 않습니다. 소스가 대상보다 더 작은 경우, LabVIEW는 부호있는 소스의 부호를 확장하고 부호없는 소스의 나머지 비트를 제로로 채웁니다. 소스가 대상보다 더 큰 경우, LabVIEW는 값의 최하위 비트만 복사합니다.
- 정수, 부동소수, 또는 고정 소수점 수를 고정 소수점 수에 연결—LabVIEW는 고정 소수점 수에서 원하는 범위 밖의 값을 최대값과 최소값 범위내로 강제 변환시킵니다.

숫자형 함수의 다형성

연산 함수는 숫자 입력 데이터를 받습니다. 함수 설명에 언급된 일부 예외를 제외하면, 기본 출력은 입력과 같은 숫자 형이거나, 입력이 서로 다른 형일 경우, 기본 출력은 입력의 가장 넓은 형입니다. 예를 들어, 8 비트 정수와 16 비트 정수를 추가하는 경우에 기본 출력은 16 비트 정수입니다. 숫자형 함수의 출력을 설정한 경우, 지정한 셋팅이 기본 동작보다 우선으로 적용됩니다.

연산 함수는 숫자, 숫자 배열, 숫자 클러스터, 숫자 클러스터 배열, 복소수 등을 다룹니다. 사용 가능한 입력 타입의 공식적인 반복된 정의는 다음과 같습니다:

숫자 타입 = 숫자 스칼라 OR 배열 [숫자 타입] OR 클러스터 [숫자 타입]

숫자 스칼라는 부동소수, 정수, 또는 복소수 부동소수가 될 수 있습니다. LabVIEW에서 배열의 배열은 사용할 수 없습니다.

배열은 모든 크기의 모든 차원을 가질 수 있습니다. 클러스터는 어떤 원소 개수도 가질 수 있습니다. 함수의 출력 타입은 입력 타입과 숫자 형이 같습니다. 한 개의 입력을 가진 함수에서, 함수는 배열 또는 클러스터의 각 원소에 대하여 연산을 수행합니다.

두 개의 입력을 가진 함수에서는 다음의 입력 조합을 사용할 수 있습니다:

- 동일한 경우—두 입력이 같은 구조이고, 출력도 입력과 같은 구조입니다.

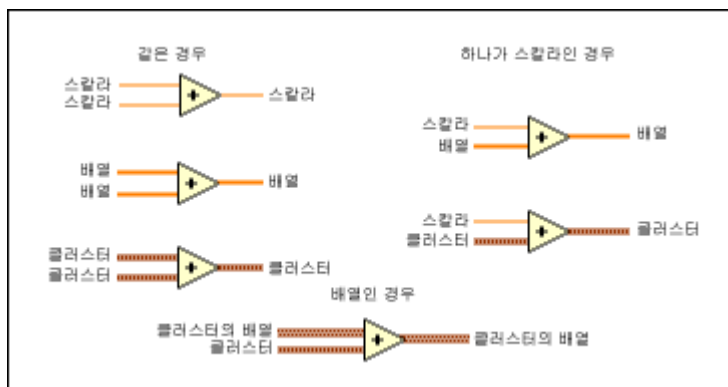
- 하나의 스칼라—한 개의 입력은 숫자 스칼라이고, 다른 입력은 배열이나 클러스터이며, 출력은 배열이나 클러스터입니다.
- 배열—한 개의 입력은 숫자 배열이고, 다른 입력은 숫자 타입 자체이며, 출력은 배열입니다.

동일한 입력의 경우, LabVIEW는 구조의 해당 원소에서 함수를 수행합니다. 예를 들어, LabVIEW는 두 개의 배열을 원소별로 더합니다. 두 배열은 같은 차원이어야 합니다. 원소의 개수가 다른 배열을 더할 수 있습니다. 이 경우, 출력의 원소 개수는 가장 작은 입력의 개수와 같습니다. 클러스터는 원소의 개수가 같아야 하며, 각 원소는 같은 타입이어야 합니다.

하나의 스칼라와 배열 또는 클러스터를 연산하는 경우, LabVIEW는 스칼라와 구조의 각 원소에서 함수를 실행합니다. 예를 들어, LabVIEW는 배열의 차원과 관계없이, 배열의 모든 원소에서 같은 숫자를 뺄 수 있습니다.

숫자 타입과 같은 타입의 배열을 연산하는 경우, LabVIEW는 배열의 각 원소에 함수를 수행합니다. 예를 들어, 그래프는 포인트의 배열입니다. 그리고 포인트는 두 개의 숫자 타입, x와 y의 클러스터입니다. 그래프를 x 방향으로 5 단위, y 방향으로 8 단위 오프셋하려면, 그래프에 포인트(5, 8)을 더하면 됩니다.

다음 설명은 [더하기] 함수의 가능한 다형성 조합을 보여줍니다.



볼리언 함수의 다형성

논리 함수는 볼리언 입력 데이터, 숫자형 입력 데이터 및 에러 클러스터를 받습니다. 입력이 숫자인 경우, LabVIEW는 비트 단위 계산을 수행합니다. 입력이 정수인 경우, 출력은 같은 형을 가집니다. 입력이 부동소수인 경우, LabVIEW는 32비트 정수로 반올림하며, 출력은 32비트 정수입니다. 입력이 에러 클러스터인 경우, LabVIEW는 에러 클러스터의 상태 파라미터에서 참 또는 거짓 값만을 입력 터미널에 전달합니다.

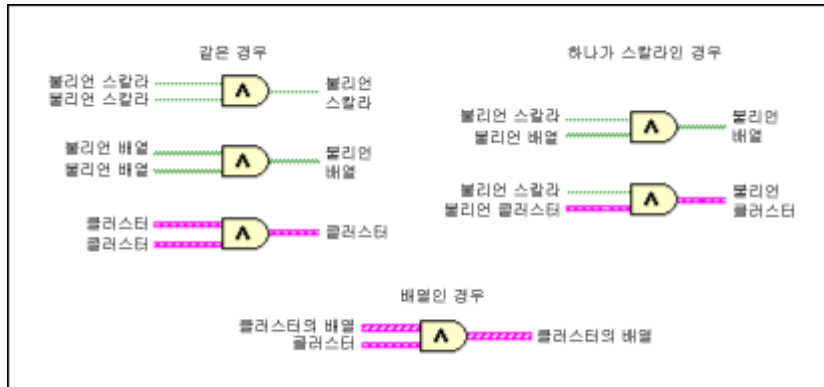
논리적 함수는 숫자나 볼리언 값의 배열, 숫자나 볼리언 값의 클러스터, 숫자나 볼리언 값의 클러스터 배열 등을 다룹니다.

사용 가능한 입력 타입의 공식적인 반복된 정의는 다음과 같습니다:

논리적 타입 = 볼리언 스칼라 OR 숫자 스칼라 OR 배열 [논리 타입] OR 클러스터 [논리 타입]

복소수와 배열의 배열은 허용되지 않습니다.

두 개의 입력을 가진 논리적 함수는 연산 함수와 같은 입력 조합을 가질 수 있습니다. 그러나 논리적 함수는 두 불리언 값 사이 또는 두 숫자 사이에서만 기본 연산이 가능하다는 제한이 있습니다. 예를 들어, 불리언 값과 숫자를 AND 연산할 수 없습니다. 다음 설명은 [AND] 함수를 위한 몇몇 불리언 값의 조합을 보여줍니다.



배열 함수의 다형성

대부분의 배열 함수는 모든 타입의 n차원 배열을 받습니다. 그러나, 함수의 와이어링 다이어그램에는 숫자 배열이 기본 데이터 타입으로 설명되어 있습니다.

문자열 함수의 다형성

[문자열 길이], [대문자로], [소문자로], [문자열 뒤집기], [문자열 회전]은 문자열, 문자열의 클러스터와 배열, 클러스터 배열을 받습니다. [대문자로]와 [소문자로] 함수는 숫자, 숫자의 클러스터, 숫자의 배열을 문자의 ASCII 코드로 해석하고 받습니다. 폭과 정밀도 입력은 반드시 스칼라여야 합니다.

문자열 변환 함수의 다형성

[경로를 문자열로]와 [문자열을 경로로] 함수는 다형성입니다. 즉, 이 함수는 스칼라 값, 스칼라의 배열, 스칼라의 클러스터, 스칼라 클러스터의 배열 등을 다룹니다. 출력은 입력과 같은 조합이지만 타입은 새로운 것입니다.

추가적인 문자열에서 숫자로 함수의 다형성

[숫자를 10진수 문자열로], [숫자를 16진수 문자열로], [숫자를 8진수 문자열로], [숫자를 공학 문자열로], [숫자를 소수점형 문자열로], [숫자를 지수형 문자열로] 함수는 숫자의 배열 및 클러스터를 받아 문자의 배열 및 클러스터를 생성합니다. [10진수 문자열을 숫자로], [16진수 문자열을 숫자로], [8진수 문자열을 숫자로], [소수점/지수 문자열을 숫자로] 함수는 문자열의 배열 및 클러스터를 받아 숫자의 배열 및 클러스터를 생성합니다. 폭과 정밀도 입력은 반드시 스칼라여야 합니다.

클러스터 함수의 다형성

[묶기]와 [풀기] 함수는 터미널에 객체를 연결하기 전까지는 개별 입력이나 출력 터미널의 데이터 타입을 보여주지 않습니다. 연결하면, 이 터미널은 대응되는 프런트패널 컨트롤 또는 인디케이터 터미널의 데이터 타입과 비슷하게 보입니다.

비교 함수의 다형성

비교 함수인 [같음?], [같지 않음?], [선택] 함수는 모든 타입의 입력을 받을 수 있습니다. 그러나 입력은 같은 타입이어야 합니다.

[크거나 같음?], [작거나 같음?], [보다 작음?], [보다 큼?], [최대 & 최소], [범위내 확인과 강제 변환] 함수는 복소수, 경로, 참조 번호를 제외한 모든 타입의 입력을 받습니다. 그러나 입력은 같은 타입이어야 합니다. 숫자, 문자열, 불리언 값, 문자열의 배열, 숫자의 클러스터, 문자열의 클러스터 등을 비교할 수 있습니다. 그러나 숫자를 문자열과 또는 문자열을 불리언 값과 비교할 수는 없습니다.

값과 제로를 비교하는 함수는 숫자 스칼라, 클러스터, 숫자 배열을 받습니다. 이 함수는 입력과 같은 데이터 구조의 불리언 값을 출력합니다.

[숫자/경로/참조 번호 아님?] 함수는 값을 제로와 비교하는 함수와 같은 입력 타입을 받습니다. 또한 이 함수는 경로와 참조 번호도 받습니다. [숫자/경로/참조 번호 아님?]은 입력과 같은 데이터 구조에서 불리언 값을 출력합니다.

[10진수?], [16진수?], [8진수?], [인쇄가능?], [화이트 스페이스?]는 스칼라 문자열 또는 숫자 입력, 문자열이나 복소수가 아닌 숫자의 클러스터, 문자열이나 복소수가 아닌 숫자의 배열 등을 받습니다. 출력은 입력과 같은 데이터 구조의 불리언 값으로 구성됩니다.

[빈 문자열/경로?] 함수는 경로, 스칼라 문자열, 문자열 클러스터, 문자열 배열 등을 받습니다. 출력은 입력과 같은 데이터 구조의 불리언 값으로 구성됩니다.

[같음?], [같지 않음?], [숫자/경로/참조 번호 아님?], [빈 문자열/경로?], [선택] 함수는 경로나 참조 번호를 사용할 수 있습니다. 그러나 다른 비교 함수는 경로나 참조 번호를 입력으로 받지 않습니다.

배열과 클러스터를 받는 비교 함수는 일반적으로 같은 구조의 불리언 배열과 클러스터를 반환합니다. 이 함수가 단일 불리언 값을 반환하도록 하려는 경우, 함수에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 비교 모드 >> 집합 비교를 선택하여 **집합을 비교**합니다.

로그 함수의 다형성

로그 함수는 숫자 입력 데이터를 받습니다. 입력이 정수인 경우, 출력은 배정도 부동소수입니다. 정수가 아닌 경우, 출력은 입력과 같은 숫자형입니다.

로그 함수는 숫자, 숫자 배열, 숫자 클러스터, 숫자 클러스터 배열, 복소수 등을 다룹니다. 사용 가능한 입력 타입의 공식적인 반복된 정의는 다음과 같습니다:

숫자 타입 = 숫자 스칼라 OR 배열 [숫자 타입] OR 클러스터 [숫자 타입]

배열의 배열은 사용할 수 없습니다.

배열은 어떤 크기도 될 수 있으며 어떤 차원 수도 가질 수 있습니다. 클러스터는 어떤 원소 개수도 가질 수 있습니다. 출력 타입은 입력과 같은 숫자형입니다. 그리고 이 함수는 클러스터 또는 배열의 각 원소에 대하여 계산을 수행합니다. **두 개의 입력을 가진 로그 함수**의 가능한 입력 타입 조합은 다음과 같습니다:

- 동일한 경우—두 입력이 같은 구조이고, 출력도 입력과 같은 구조입니다.
- 하나의 스칼라—한 개의 입력은 숫자 스칼라이고, 다른 입력은 숫자 배열이나 클러스터이며, 출력은 배열이나 클러스터입니다.

비교 함수 사용하기

비교 함수를 사용하여 **숫자값**, **문자열**, **불리언 값**, **배열**, **클러스터**를 비교합니다. 대부분의 비교 함수는 하나의 입력을 테스트하거나 두 개의 입력을 비교하고 불리언 값을 반환합니다.

숫자값 비교하기

대부분의 비교 함수는 한 개의 입력을 테스트하거나 두 개 입력을 비교하여 불리언 값을 반환합니다. 함수는 숫자를 비교하기 전에 먼저 같은 형으로 변환합니다. 함수는 각 입력을 정확한 비교를 위해 필요한 가장 큰 형으로 변환합니다. **숫자 아님**(NaN) 값을 갖는 하나 또는 두 개의 입력을 비교하면 비교가 같지 않다는 것을 나타내는 결과가 반환됩니다.

모든 실수가 ANSI/IEEE 표준에서 **부동소수**로 나타나지는 않습니다. 따라서 부동소수를 사용하는 비교는 반올림 에러 때문에 예상밖의 결과를 발생시킬 수 있습니다.

문자열 비교하기

비교 함수는 ASCII 문자 코드의 값에 따라 문자열을 비교합니다. 비교 함수는 문자열을 비교하는데 일치하지 않는 것을 만나거나 하나의 문자열이 끝날 때까지 0번째 원소로 시작해서 한 번에 하나의 원소를 비교합니다. 이 함수는 긴 문자열을 짧은 문자열보다 큰 것으로 취급합니다.

예를 들어, a (십진수 97)는 A (65)보다 크며, 이것은 숫자 0 (48)보다 크고, 이는 스페이스 문자(32)보다 큼니다. LabVIEW는 문자열의 문자를 처음부터 하나씩 다른 값이 나올 때까지 비교하며, 다른 값이 나오면 비교를 끝냅니다. 예를 들어, LabVIEW는 문자열 abcd와 abef를 e의 값보다 작은 c를 찾을 때까지 비교합니다. 문자가 있는 것은 문자가 없는 것보다 큼니다. 따라서, 문자열 abcd는 abc보다 큼니다. 첫번째 문자열이 더 길기 때문입니다.

[10진수?] 및 [인쇄가능?] 함수와 같이 문자열 문자의 범주를 확인하는 함수는 문자열의 첫번째 문자만을 평가합니다.

불리언 값 비교하기

비교 함수는 참의 불리언 값을 거짓의 불리언 값보다 큰 것으로 취급합니다.

배열과 클러스터 비교하기

일부 비교 함수에는 데이터 배열 또는 클러스터를 비교하는데 사용할 수 있는 모드가 두 가지 있습니다. 집합 비교 모드에서 두 배열 또는 클러스터를 비교할 경우, 함수는 단일 불리언 값을 반환합니다. 원소 비교 모드에서, 함수는 개별 원소를 비교한 후 불리언 값의 배열이나 클러스터를 반환합니다.

집합 비교 모드에서, 문자열 비교와 배열 비교 작업은 정확히 같은 방식으로 진행됩니다—비교 함수는 문자열을 ASCII 문자의 배열로 취급합니다.



노트

[최대 & 최소] 함수는 다른 비교 함수와는 다르게 동작합니다. [최대 & 최소] 함수는 사용자가 지정하는 모드와 출력에 따라 배열의 최대 원소나 배열의 최소 원소를 반환합니다.

비교 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 비교 모드 >> 원소 비교 또는 비교 모드 >> 집합 비교를 선택하여 함수의 모드를 변경합니다. 일부 비교 함수는 오직 집합 비교 모드로만 사용할 수 있기 때문에, 바로 가기 메뉴 아이템이 나타나지 않습니다.

배열

다차원 배열을 비교할 때, 함수에 연결된 각 배열은 반드시 같은 차원이어야 합니다. 집합 비교 모드로만 사용할 수 있는 비교 함수는 문자열을 비교하는 것과 같은 방법으로 배열을 비교합니다—첫번째 원소부터 하나씩 차례로 비교하여 서로 다른 원소가 나올 때까지 비교합니다.

원소 비교 모드

원소 비교 모드에서, 비교 함수는 입력 배열과 같은 차원의 불리언 값의 배열을 반환합니다. 출력 배열의 각 차원은 두 입력 배열 중에서 작은 차원의 크기입니다. 행, 열, 또는 페이지와 같은 각 차원을 따라서, 함수는 각 입력 배열의 해당 원소를 비교하여 출력 배열의 해당 불리언 값을 만듭니다.

집합 비교 모드

집합 비교 모드에서, 비교 함수는 배열의 원소를 비교한 후 단일 불리언 결과를 반환합니다. 비교 함수는 결정이 날 때까지 순차적으로 원소를 비교하며, 영어 단어의 알파벳 순서에서 서로 다른 문자가 나올 때까지 각 단어의 철자를 비교하는 경우와 유사합니다. 그렇기 때문에, 함수는 다음 단계를 수행하여 비교의 결과를 찾습니다.

- 함수는 배열의 처음부터 시작하여 각 입력 배열의 대응하는 원소를 비교합니다.
- 대응하는 원소가 같지 않은 경우, 함수는 멈춘 후 비교의 결과를 반환합니다.

- 대응하는 원소가 같은 경우, 비교 함수는 서로 다른 값을 찾거나 입력 배열 중 하나의 끝에 도달할 때까지 다음 비교 값을 처리합니다.
- 입력 배열의 모든 값이 같은 값이지만 한 배열의 끝에 비교되지 않은 원소가 남아있을 경우, 긴 배열이 짧은 배열보다 큰 것으로 간주됩니다. 예를 들어, 비교 함수는 배열 [1, 2, 3, 2]를 배열 [1, 2, 3]보다 큰 것으로 간주합니다.



노트 범위내 확인과 강제변환 함수는 세 개의 입력을 필요로 하기 때문에 집합 비교 모드의 다른 비교 함수와는 다르게 작동합니다. 첫번째 원소 세트를 비교한 후, 이 함수는 상위 리미트, 하위 리미트 및 x 입력이 모두 같은 경우에만 다음 값 세트를 처리합니다. 그렇지 않은 경우, 이 함수는 첫번째 원소 세트를 비교한 결과를 반환합니다.

클러스터

비교할 클러스터는 반드시 같은 수의 원소를 포함해야만 하고, 클러스터의 각 원소는 반드시 호환 가능한 타입이어야 하며, 원소는 반드시 클러스터에서 같은 순서여야 합니다. 예를 들어, DBL과 문자열의 클러스터를 I32와 문자열의 클러스터에 비교할 수 있습니다.

원소 비교 모드

원소 비교 모드에서, 비교 함수는 입력 클러스터의 각 해당 원소에 대응하는 볼리언 원소의 클러스터를 반환합니다.

집합 비교 모드

집합 비교 모드에서, 비교 함수는 클러스터의 원소를 비교한 후 단일 볼리언 결과를 반환합니다. 비교 함수는 결정이 날 때까지 순차적으로 원소를 비교하며, 영어 단어의 알파벳 순서에서 서로 다른 문자가 나올 때까지 각 단어의 철자를 비교하는 경우와 유사합니다. 그렇기 때문에, 함수는 다음 단계를 수행하여 비교의 결과를 찾습니다.

- 함수는 클러스터의 처음부터 시작하여 각 입력 클러스터의 대응하는 원소를 비교합니다.
- 대응하는 원소가 같지 않은 경우, 함수는 멈춘 후 비교의 결과를 반환합니다.
- 대응하는 원소가 같은 경우, 비교 함수는 서로 다른 값을 찾거나 입력 클러스터의 끝에 도달할 때까지 다음 비교 값을 처리합니다.
- 입력 클러스터의 모든 값이 같지만 한 배열의 끝에 비교되지 않은 원소가 남아있을 경우, 큰 클러스터가 작은 클러스터보다 큰 것으로 간주됩니다. 예를 들어, 비교 함수는 클러스터 [1, 2, 3, 2]를 클러스터 [1, 2, 3]보다 큰 것으로 간주합니다.



노트 범위내 확인과 강제변환 함수는 세 개의 입력을 필요로 하기 때문에 집합 비교 모드의 다른 비교 함수와는 다르게 작동합니다. 첫번째 원소 세트를 비교한 후, 이 함수는 상위 리미트, 하위 리미트 및 x 입력이 모두 같은 경우에만 다음 값 세트를 처리합니다. 그렇지 않은 경우, 이 함수는 첫번째 원소 세트를 비교한 결과를 반환합니다.

정렬된 데이터를 포함하는 두 레코드를 비교할 경우, 클러스터에 집합 비교 모드를 사용합니다. 이 때 클러스터에서 앞에 있는 원소가 뒤에 있는 원소보다 먼저 비교됩니다. 예를 들어, 차례로 성과 이름의 두 문자열을 포함하는 클러스터를 비교할 경우, 함수는 성이 일치할 경우에만 이름을 비교할 것입니다.

참조 비교하기

LabVIEW는 [같음?]과 [같지 않음?] 비교 함수를 사용하여 VI 서버 참조를 비교합니다. 이 비교 함수를 사용하여 데이터 타입의 배열 또는 클러스터를 같은 데이터 타입의 스칼라와 비교하고 불리언 값의 배열 또는 클러스터를 생성할 수 있습니다. VI 서버 참조를 비교할 때, 이 함수는 두 개의 참조가 같은 값이나 같은 기능을 하는지에는 관계없이 같은 객체를 참조하는지 여부에 따라 참이나 거짓을 반환합니다. 같은 객체에 대해 여러 참조를 가질 수 있기 때문에 참조의 정수값은 다를 수 있지만 여전히 같은 객체를 참조할 수 있습니다. 또한, 두 개의 참조의 기능은 같은 객체를 참조한다고 할지라도 다를 수 있습니다. 예를 들면, 참조 중의 하나는 인증을 받고 다른 하나는 인증을 받지 않는다면, 이 참조들은 다른 기능을 가지지만 여전히 같은 객체를 참조합니다.

예를 들면, 하나의 VI 서버 참조는 디지털 숫자형 컨트롤에 대한 엄격한 참조가 될 수 있고 다른 참조는 일반 참조가 될 수 있습니다. 그러나 같은 객체를 참조하면 동등합니다. 마찬가지로, 두 개의 참조가 같은 객체를 참조하지만 그 참조 중의 하나가 인증받고 다른 하나는 인증을 받지 못할 때, 이 함수는 참을 반환합니다. 그러나, 같은 LabVIEW 객체에 대해 로컬 참조와 원격 참조를 가질 때, 같음? 함수는 거짓을 반환합니다.

배리언트 비교하기

배리언트 데이터를 비교할 때, 데이터 타입, 값, 속성 이름, 속성 값이 같아야 일치하는 것으로 간주됩니다.

LabVIEW의 VI

익스프레스 VI

익스프레스 VI는 대화 상자를 통해 사용자가 대화식으로 설정을 설정할 수 있는 VI입니다. 익스프레스 VI는 파란색 영역으로 둘러싸인 아이콘을 가지는 **확장 가능 노드**로 블록 다이어그램에 나타납니다.

익스프레스 VI를 블록 다이어그램에 놓을 때 나타나는 설정 대화 상자의 옵션을 사용하여 익스프레스 VI를 설정할 수 있습니다. 익스프레스 VI를 더블 클릭하거나 익스프레스 VI에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택하여 설정 대화 상자를 다시 열 수 있습니다. 또한, 블록 다이어그램에서 익스프레스 VI의 터미널에 값을 연결하여 익스프레스 VI를 설정할 수 있습니다.

설정 대화 상자에서 익스프레스 VI에 대해 설정할 수 있는 파라미터는 **설정 가능 파라미터**라고 부릅니다. 블록 다이어그램에서 익스프레스 VI에 대해 설정할 수 있는 파라미터는 **확장 가능 파라미터**라고 확장 가능 터미널이라고 부릅니다. 설정 가능하면서 확장 가능한 파라미터도 있는데, 이는 두 방법 모두를 통해 설정할 수 있는 파라미터입니다. 설정 가능 파라미터를 사용하여 익스프레스 VI에 대해 지정한 설정에 의해 블록 다이어그램에 나타나는 확장 가능 터미널이 결정됩니다.

익스프레스 VI를 사용하여 일반 측정 태스크를 수행할 수 있습니다. 또한, 익스프레스 VI를 **생성** 및 편집한 후, 어플리케이션을 쉽게 빌드할 수 있도록 여러 사용자에게 배포할 수 있습니다. 익스프레스 VI 생성 및 편집 대화 상자는 도구 » 고급 » 익스프레스 VI 생성 및 편집을 선택하여 불러올 수 있는데, 이 대화 상자를 통해 **기존 VI**, **다른 익스프레스 VI**, 또는 **새 VI**로부터 익스프레스 VI를 생성할 수 있습니다.

익스프레스 VI의 장점

대화식으로 설정 가능하다는 점을 익스프레스 VI의 가장 큰 장점으로 꼽을 수 있습니다. 최소한의 프로그래밍 기술만으로도 손쉽게 어플리케이션을 만들 수 있도록 사용자에게 VI 또는 VI 라이브러리를 제공하고자 할 때 익스프레스 VI는 유용합니다.

익스프레스 VI는 사용자가 완벽하게 이해하지 못하는 작업이더라도 대화식으로 셋팅을 선택할 수 있도록 대화식 설정 기능을 지원하고 있습니다. 예를 들어, 특정 신호를 위한 디지털 필터가 필요한 경우에 수십개의 필터 VI를 포함하고 있는 라이브러리 VI가 있더라도, 사용자가 올바른 필터 응답을 선택하거나 신호에 가장 직접적인 영향을 줄 수 있는 파라미터를 결정하는데 도움이 안될 수 있습니다. 필터 익스프레스 VI는 사용자로 하여금 대화식으로 필터 타입을 선택하고, 필터 파라미터를 조작하고, 여러 방법으로 필터 응답을 볼 수 있도록 지원합니다. 다양한 필터 및 필터 파라미터가 신호에 직접적으로 어떤 결과를 가져오는지 익스프레스 VI의 설정 대화 상자 안에서 미리 확인할 수 있습니다. 필터 익스프레스 VI를 실행하면, 익스프레스 VI는 새 신호를 기본 데이터로 설정 대화 상자 안에 디스플레이합니다.

익스프레스 VI의 다른 정점은 익스프레스 VI의 개별 인스턴스가 각자 독립적으로 기능을 수행한다는 점입니다. 익스프레스 VI를 블록다이어그램에 놓으면, 하나의 익스프레스 VI의 인스턴스가 해당 블록다이어그램에 임베딩 됩니다. 설정 대화 상자에서 사용자가 선택하는 셋팅은 해당 익스프레스 VI 인스턴스에만 적용됩니다. 같은 블록다이어그램 상에서 서로 다른 5개의 위치에 표준 VI를 놓으면, 결과적으로 똑같은 5개의 VI 복제본을 놓는 것입니다. 5개의 복제본의 소스 코드, 기본 값, 프런트패널은 모두 같습니다. 반면, 같은 블록다이어그램 상에서 서로 다른 5개의 위치에 익스프레스 VI를 놓으면, 각자 개별적으로 설정이 가능하고 **이름이 서로 다른** 5개의 독립적인 익스프레스 VI를 놓은 것이 됩니다. 각각의 익스프레스 VI 인스턴스는 서로 다른 개수 및 타입의 확장 가능 터미널을 포함할 수 있습니다. 기본 도움말 윈도우에서도 각 인스턴스의 서로 다른 셋팅 정보가 **업데이트** 됩니다.

익스프레스 VI를 사용할 때의 유의점

익스프레스 VI는 VI에 대해 런타임 대화식 설정을 제공하지 않습니다. 런타임 설정이 필요한 경우, 설정 대화 상자와 비슷한 기능을 포함하는 사용자 인터페이스를 지닌 어플리케이션을 만드십시오.

익스프레스 VI는 최대한 사용하기 쉽도록 디자인되었습니다. 어플리케이션을 실행할 때 엄격하게 메모리에 제한을 두어야 하거나 실행 속도가 높아야 하는 경우, 표준 VI를 사용하십시오.

다이나믹 데이터 타입 사용하기

대부분의 **익스프레스 VI**는 다이나믹 데이터 타입을 받고 반환합니다. 다이나믹 데이터 타입은 다음과 같이 짙은 파란색 터미널로 나타납니다.



다이나믹 데이터 타입은 다음과 같은 데이터 타입과 데이터를 주고 받습니다. 이때 스칼라 데이터 타입은 부동소수 또는 불리언 값입니다:

- 웨이브폼의 1D 배열
- 스칼라의 1D 배열
- 스칼라의 1D 배열 — 가장 최근값
- 스칼라의 1D 배열 — 단일 채널
- 스칼라의 2D 배열 — 열이 채널임
- 스칼라의 2D 배열 — 행이 채널임
- 단일 스칼라
- 단일 웨이브폼

다이나믹 데이터를 보려면 그래프, 차트와 같은 인디케이터 또는 숫자형 인디케이터에 데이터를 직접 연결하십시오. 그러나, LabVIEW가 데이터를 인디케이터의 데이터 타입으로 변환해야 하기 때문에, 다이나믹 데이터를 디스플레이하면 VI의 실행 속도가 느려질 수 있습니다.



노트

익스프레스 VI를 제외한 대부분의 VI 및 함수는 다이나믹 데이터 타입을 자동으로 받아들이지는 않습니다. 그러므로 내장 VI 또는 함수를 사용하여 다이나믹 데이터를 분석 또는 처리하려면 다이나믹 데이터를 VI 또는 함수가 받아들이는 타입으로 **변환**해야 합니다.

다이나믹 데이터 타입은 신호와 연관된 데이터 및 신호의 이름 또는 데이터가 수집된 날짜와 시간 등의 신호와 연관된 정보를 제공하는 속성을 포함합니다. 속성은 신호가 그래프 또는 차트에 나타나는 방식을 지정합니다. 예를 들어, DAQ 어시스턴트 익스프레스 VI를 사용하여 신호를 수집하고 그래프에 신호를 그리는 경우, 신호의 이름은 그래프의 플롯 범례에 나타나고, x스케일은 신호의 속성에 따라서 상대 시간 또는 절대 시간으로 신호에 연계된 타이밍 정보를 디스플레이하기 위해서 조정됩니다. [스펙트럼 측정] 익스프레스 VI를 사용하여 신호에 FFT 분석을 수행하고 결과 값을 그래프에 그리는 경우, x스케일은 신호의 속성에 따라서 주파수 영역에 신호를 그리기 위해서 자동적으로 조정됩니다. 블록 다이어그램에 있는 VI 또는 함수의 다이나믹 데이터 타입 출력 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>> 그래프 인디케이터를 선택하여 데이터를 그래프에 디스플레이하거나, 또는 바로 가기 메뉴에서 생성>> 숫자형 인디케이터를 선택하여 데이터를 숫자형 인디케이터에 디스플레이합니다.

다음 테이블은 다이나믹 데이터 타입을 받는 인디케이터와 다이나믹 데이터 타입이 포함할 수 있는 데이터의 타입을 나열합니다. 이 테이블은 또한 인디케이터가 데이터를 처리하는 방법을 설명합니다.

다이나믹 데이터 타입의 데이터	인디케이터	결과
단일 숫자값	그래프	타임스탬프와 속성을 포함하여 단일값을 플롯합니다.

다이나믹 데이터 타입의 데이터	인디케이터	결과
단일 채널	그래프	타임스탬프와 속성을 포함하여 전체 웨이브폼을 플롯합니다.
여러 채널	그래프	타임스탬프와 속성을 포함하여 모든 데이터를 플롯합니다.
단일 숫자값	숫자형 인디케이터	단일값을 디스플레이합니다
단일 채널	숫자형 인디케이터	채널로부터 데이터의 마지막 값을 디스플레이합니다.
여러 채널	숫자형 인디케이터	첫번째 채널로부터 데이터의 마지막 값을 디스플레이합니다.
단일 숫자값	불리언 인디케이터	숫자값이 0.5보다 크거나 같은 경우 참값을 디스플레이합니다
단일 채널	불리언 인디케이터	채널로부터의 데이터의 최종값이 0.5보다 크거나 같은 경우 참값을 디스플레이합니다.
여러 채널	불리언 인디케이터	첫번째 채널로부터의 데이터의 최종값이 0.5보다 크거나 같은 경우 참값을 디스플레이합니다.

다이나믹 데이터 속성 가져오기 및 설정하기

[다이나믹 데이터 속성 얻기] 익스프레스 VI를 사용하여 다이나믹 데이터 속성을 가져올 수 있습니다. [다이나믹 데이터 속성 얻기] 익스프레스 VI를 블록다이어그램에 놓으면, 설정 대화 상자가 나타납니다. 이 대화 상자를 사용하여 익스프레스 VI에 와이어로 연결한 다이나믹 데이터에 포함된 신호의 속성을 가져옵니다.

[다이나믹 데이터 속성 설정] 익스프레스 VI를 사용하여 신호 이름, 타임스탬프, 시간 모드 등의 다이나믹 데이터 속성을 설정합니다. [다이나믹 데이터 속성 설정] 익스프레스 VI를 블록다이어그램에 놓으면, 설정 대화 상자가 나타납니다. 이 대화 상자를 사용하여 익스프레스 VI에 와이어로 연결한 다이나믹 데이터에 포함된 신호의 속성을 변경하거나 설정합니다.



노트 [다이나믹 데이터 속성 얻기] 익스프레스 VI와 [다이나믹 데이터 속성 설정] 익스프레스 VI는 채널과 신호를 모두 신호로 참조합니다.

다이나믹 데이터 변환하기

다이나믹 데이터로 변환 익스프레스 VI를 사용하여, **익스프레스 VI**와 함께 사용할 수 있도록 숫자형, 불리언, 웨이브폼, 배열 데이터를 다이나믹 데이터로 변환합니다. 다이나믹 데이터로 변환 익스프레스 VI를 블록다이어그램에 놓으면, 설정 대화 상자가 나타납니다. 이 대화 상자를 사용하여 다이나믹 데이터 타입으로 변환하려는 데이터의 종류를 선택합니다.

다이나믹 데이터로부터 변환 익스프레스 VI를 사용하여 다이나믹 데이터를 다른 VI와 함수에서 사용하도록 숫자, 웨이브폼, 배열 데이터 타입으로 변환합니다. 다이나믹 데이터로부터 변환 익스프레스 VI를 블록다이어그램에 놓으면, 설정 대화 상자가 나타납니다. 설정 대화 상자에 디스플레이된 옵션을 사용하여 다이나믹 데이터로부터 변환 익스프레스 VI에서 반환하는 데이터의 포맷을 지정할 수 있습니다.

다이나믹 데이터를 배열 인디케이터로 연결하면, LabVIEW는 다이나믹 데이터로부터 변환 익스프레스 VI를 블록다이어그램에 놓습니다. 다이나믹 데이터로부터 변환 익스프레스 VI를 더블 클릭하여 설정 대화 상자를 열어서 배열에서 데이터를 어떻게 나타낼지 컨트롤합니다.

익스프레스 VI로부터 SubVI 생성하기

익스프레스 VI 설정에서 SubVI를 생성할 수 있습니다. 예를 들어, 매번 익스프레스 VI를 다시 설정하는 것보다 다른 VI에서 subVI로 사용하기 위해서 [측정 파일에 쓰기] 익스프레스 VI의 설정을 저장할 수도 있습니다. 익스프레스 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **프론트패널 열기**를 선택하여 익스프레스 VI로부터 SubVI를 생성합니다.

익스프레스 VI로부터 subVI를 생성할 때, subVI의 프론트패널이 나타납니다. 그 후, VI를 편집하고 저장할 수 있습니다. 컨트롤에 입력된 값을 저장하기 위해서는, **편집 > 현재값을 기본값으로**를 선택하거나 각 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **데이터 처리 > 현재값을 기본값으로**를 선택합니다. **확장 가능한 노드**로 디스플레이되는 새 SubVI는 블록다이어그램에서 익스프레스 VI를 대체합니다.

익스프레스 VI로부터 VI를 생성한 후, subVI를 다시 익스프레스 VI로 변환할 수 없습니다.

익스프레스 VI 생성하기

익스프레스 VI를 구성하는 요소는 다음과 같습니다:

- **설정 대화 상자 VI**—익스프레스 VI의 런타임 동작에 대한 셋팅을 설정할 수 있습니다.
- **소스 VI**—익스프레스 VI의 코드를 포함합니다. 소스 VI는 또한 설정 대화 상자 VI에 대한 링크를 포함합니다.

블록다이어그램에 익스프레스 VI를 놓으면, 설정 대화 상자가 나타납니다. 설정 대화 상자 VI가 이 설정 대화 상자의 외관과 기능을 결정합니다. 익스프레스 VI를 설정하고 설정 대화 상자를 닫으면, 익스프레스 VI는 블록다이어그램 상에서 **확장 가능한 노드**로 나타납니다. 확장 가능 노드의 입력 및 출력을 사용자가 지정할 수 있습니다. 그러나, SubVI처럼 확장 가능 노드를 더블 클릭하여 열거나 관련된 프론트패널 및 블록다이어그램을 볼 수는 없습니다. 관련 코드를 보려면 **익스프레스 VI를 SubVI로 변환**해야 합니다. 익스프레스 VI의 코드는 소스 VI에 의해 결정됩니다. 일단 익스프레스 VI를 SubVI로 변환하면, 더이상 설정 대화 상자를 사용하거나 SubVI를 다시 익스프레스 VI로 변환할 수 없습니다.

사용자가 생성하는 모든 익스프레스 VI에 대해 설정 대화 상자 VI와 소스 VI 모두를 디자인해야 합니다.

익스프레스 VI를 생성하고 배포하려면, 다음의 개발 단계를 완료해야 합니다.

- 익스프레스 VI의 **소스 VI를 디자인**합니다.
- 사용자 인터페이스를 포함하여 **설정 대화 상자를 디자인**합니다.
- 익스프레스 VI를 **도움말 파일과 연계**합니다.
- 익스프레스 VI를 LabVIEW **디렉토리 구조**에 놓습니다.
- 익스프레스 VI를 **새 팔레트 또는 기존에 있는 팔레트에 추가**합니다.

익스프레스 VI를 labview\user.lib_express 디렉토리에 추가할 때, 익스프레스 VI는 익스프레스 사용자 라이브러리 팔레트에 나타납니다. paletteMenu 서브팔레트는 labview\user.lib_express에 위치하며 밑줄(_)로 시작하지 않는 각 LLB 또는 디렉토리에 대응하여 나타납니다. 그 결과, 익스프레스 VI 및 소스 VI의 SubVI가 모두 팔레트에 나타납니다.



노트

paletteMenu 서브팔레트가 익스프레스 사용자 라이브러리 팔레트에 나타나지 않는 경우, 팔레트 보기를 변경하여 팔레트를 새로 고치기합니다. LabVIEW를 다시 시작해야 새 익스프레스 VI를 팔레트 상에서 볼 수 있는 경우도 있습니다.

팔레트를 편집하여 익스프레스 VI만을 디스플레이하고 추가적인 서브팔레트 계층을 제거하여 더 간편하게 익스프레스 VI를 사용할 수도 있습니다. 다른 객체와 관련하여 팔레트를 편집하는 것과 같은 방식으로 익스프레스 VI로 사용자 팔레트 보기를 편집할 수 있습니다.



노트 익스프레스 VI를 팔레트에 놓을 때 아이콘을 둘러싼 파란색 상자가 안 보일 수도 있습니다. 하지만, 팔레트의 변경 내역을 일단 저장한 후에는 익스프레스 VI가 바르게 나타날 것입니다.

익스프레스 VI에 대한 문서 생성하기

설정 대화 상자에서 도움말 버튼을 클릭할 때 **도움말 파일이 실행되는 것과 같이 익스프레스 VI를 설정**할 수 있습니다. 또한, 기본 도움말 윈도우에 **현재 설정을 디스플레이**하도록 익스프레스 VI를 설정할 수 있습니다.

도움말 파일을 익스프레스 VI와 연계하기

온라인 도움말 컨트롤 함수를 사용하여 도움말 파일을 익스프레스 VI와 연계시켜서 설정 대화 상자에서 도움말 버튼을 클릭할 때 도움말 파일이 실행되도록 만듭니다. 이 함수는 사용자가 만드는 각 **설정 대화 상자 VI**의 블록 다이어그램에 나타납니다.

온라인 도움말 컨트롤 함수는 도움말 파일의 경로 입력에서 사용자가 지정한 디렉토리 안에 있는 .chm 또는 .html 파일을 엽니다. 함수는 사용자가 검색할 문자열 입력에서 지정한 텍스트를 포함하는 파일을 찾고, 해당 텍스트를 포함하는 위치에서 파일을 엽니다.

익스프레스 VI에 대한 기본 도움말 디스플레이하기

도움말 생성 VI를 사용하여 기본 도움말 윈도우에 익스프레스 VI의 **현재 설정을 디스플레이**합니다. 도움말 생성 VI는 블록 다이어그램에서 익스프레스 VI 위에 마우스가 지나갈 때 기본 도움말 윈도우에 디스플레이하려는 컨트롤의 값을 지정합니다. 또한 컨트롤 이름과 같은 텍스트를 지정하여, 기본 도움말 윈도우에서 컨트롤 값 앞에 나오도록 만들 수 있습니다. 도움말 생성 VI는 익스프레스 VI 생성 마법사를 통해 생성한 모든 익스프레스 VI의 설정 대화 상자 VI에서 SubVI로 나타납니다. 이 SubVI의 이름은 genHelpExpressVIName입니다.

도움말 생성 VI의 ex_PPGetProp SubVI는 컨트롤 값을 도움말 문자열에 추가합니다. 기본 도움말 윈도우에 디스플레이하려는 값을 포함하는 각각의 컨트롤에 대해 개별적으로 ex_PPGetProp VI 복사본을 사용하십시오. ex_PPGetProp VI는 각 컨트롤에 대한 도움말에 연계되기 때문에, 1 개의 복사본의 출력으로부터 다음에 오는 VI의 입력을 와이어로 연결해야 합니다.

익스프레스 VI에 대해 도움말 생성 VI를 생성한 이후, 기본 도움말 윈도우는 익스프레스 VI의 설정 대화 상자가 닫힐 때마다 업데이트됩니다. 도움말 생성 VI를 바르게 수정하지 않는 경우, 설정 대화 상자를 닫을 때 기본 도움말과 관련된 에러 메시지가 나타납니다. 덧붙여서, 이에 대응하는 익스프레스 VI **검증 테스트**가 실패하게 됩니다.

익스프레스 VI 구성요소의 위치 찾기

익스프레스 VI는 **설정 대화 상자 VI**, **소스 VI**, SubVI로 구성됩니다. LabVIEW는 소스 VI와 설정 대화 상자 VI를 서로 다른 LLB에 저장하는데, 이 LLB는 파일 시스템에 같이 저장합니다. 소스 VI와 설정 대화 상자 VI가 SubVI를 공유하는 경우, SubVI를 제3의 LLB에 저장하거나 가장 논리적인 위치에 저장할 수 있습니다.

LabVIEW는 다음 명명 규칙에 따라 익스프레스 VI 구성요소를 저장합니다.

구성요소	소스 VI와 같은 종류	설정 대화 상자 VI와 같은 종류
파일 라이브러리	ExpressVIName Source.llb	_ExpressVIName Config.llb
VI 파일 이름	ExpressVIName Source.vi	Configure ExpressVIName.vi
SubVI 파일 이름	sub ExpressVIName.vi	genHelp ExpressVIName.vi

다음의 디렉토리에서 익스프레스 VI 및 구성요소를 찾을 수 있습니다:

- labview\vi.lib\express—LabVIEW가 제공하는 익스프레스 VI를 포함합니다. 이 디렉토리는 함수 팔레트의 서브 팔레트에 대응하는 익스프레스 VI의 서브 디렉토리를 포함합니다.
- labview\vi.lib\express\express_shared—익스프레스 VI 및 설정 대화 상자 VI가 호출하는 지원 VI를 포함합니다. 이 디렉토리에 있는 VI를 사용하여 라디오 버튼 핸들링 및 **기본 도움말 생성**과 같은 작업을 수행합니다.
- labview\user.lib_express—사용자가 생성하여 기본 위치에 저장하는 익스프레스 VI를 포함합니다. (macOS 및 Linux) 사용자가 생성하는 익스프레스 VI

의 기본 위치는 LabVIEW 기본 데이터 디렉토리입니다. `user.lib` 디렉토리에 대해 쓰기 권한이 없는 사용자도 있기 때문입니다.

익스프레스 VI에 대한 소스 VI는 관련된 설정 대화 상자 VI에 대한 링크를 포함합니다. 설정 대화 상자 VI를 `vi.lib` 또는 `user.lib`에 대해 상대적으로 다른 위치로 옮기는 경우, 이 링크는 깨지고 익스프레스 VI는 설정 대화 상자 VI를 찾도록 사용자에게 요청합니다.

익스프레스 VI를 포함하는 VI를 저장할 때, LabVIEW는 익스프레스 VI를 VI 안에 저장합니다. VI를 열 때, 안에 있는 익스프레스 VI는 익스프레스 VI의 소스 VI 파일 대신에 해당 VI와 동일한 파일로부터 로드됩니다. 그러므로, 필요할 때마다 업데이트 할 수 있도록 익스프레스 VI의 모든 기능을 SubVI 안에 놓아야 합니다.

다른 컴퓨터에 익스프레스 VI 배포하기

익스프레스 VI를 다른 컴퓨터로 옮기고 싶은 경우, 익스프레스 VI에 대한 2 개의 VI 라이브러리와 관련된 `.mnu` 파일 모두를 같이 옮깁니다. 다른 컴퓨터에 배포할 익스프레스 VI를 생성할 때, 타겟 컴퓨터가 사용하는 것과 동일한 디렉토리 구조를 사용해야 합니다. 익스프레스 VI의 구성요소는 상대적인 경로 및 특정 경로 모두를 통해 서로 연계되어 있습니다. 배포하는 여러 타겟 컴퓨터의 디렉토리 구조가 서로 다른 경우, 링크는 깨지고 익스프레스 VI는 설정 대화 상자 VI를 찾도록 사용자에게 요청합니다.

설정 대화 상자 VI 디자인하기

익스프레스 VI의 설정 대화 상자 VI는 사용자로 하여금 익스프레스 VI의 런타임 동작에 대한 셋팅을 설정할 수 있도록 지원합니다. 설정 대화 상자 VI는 또한 익스프레스 VI에 대한 사용자 인터페이스를 포함합니다. 익스프레스 VI 생성자 마법사를 사용하여 새 익스프레스 VI를 생성하고 **소스 VI를 디자인**한 후, 설정 대화 상자 VI의 **프론트패널을 디자인**하고 **제공되는 블록 다이어그램 템플릿을 변경**하여 익스프레스 VI의 런타임 동작을 설정할 수 있습니다.

설정 대화 상자 VI 프론트패널 디자인하기

설정 대화 상자 VI의 프론트패널을 열어서 사용자 인터페이스를 완성합니다. 기본적으로, LabVIEW는 오직 확인, 취소, 도움말 기본 버튼만을 정의합니다. 주요 선택 프레임은 설정 대화 상자에 대한 사용자 인터페이스 객체를 포함합니다. 익스프레스 VI 설정 대화 상자에 대한 사용자 인터페이스 객체를 주요 선택 프레임 안에 놓습니다.

다음 리스트는 **사용자 인터페이스를 만들** 때 고려해야 할 사항을 설명합니다.

- 되도록이면 시스템 팔레트에 있는 컨트롤과 인디케이터를 사용하거나 다른 설정 대화 상자에 있는 객체를 복사하여 붙여넣기 하십시오. 시스템 컨트롤은 특정 플랫폼에 맞는 그리기 기능을 사용하기 때문에, 각 플랫폼에 대한 표준 대화 상자와 유사하도록 대화 상자의 모양과 동작을 만드는데 유용합니다.
- 시스템 색과 폰트를 사용하십시오.
- 숨은 컨트롤, 탭 컨트롤, 여러 페이지 등과 같이 숨겨진 선택 사항의 개수를 최소화 하십시오. 숨은 객체가 없이는 설정 대화 상자를 생성할 수 없는 경우, 두번째 페이지를 만들거나 탭 컨트롤을 사용합니다.

- 사용자 인터페이스 스타일의 일관성을 유지하십시오. 예를 들어, 확인, 취소, 도움말 버튼은 설정 대화 상자의 오른쪽 아래에 보여야 합니다.

또한, 사용자 인터페이스를 만들 때 [LabVIEW Style Checklist](#) 및 기타 Development Guidelines를 사용하십시오.

사용자 인터페이스를 생성한 후, 프런트패널의 크기를 조정하여 익스프레스 VI 설정에 필요한 아이템만을 볼 수 있도록 만드십시오. 일반적으로, 오직 주요 선택 프레임 및 확인, 취소, 도움말 버튼만이 디스플레이되도록 프런트패널의 크기를 조정합니다.

설정 대화 상자 VI 블록다이어그램 디자인하기

설정 대화 상자의 사용자 인터페이스를 생성한 후, 제공된 블록다이어그램 템플릿을 수정합니다. 설정 대화 상자 VI의 블록다이어그램은 세 부분으로 구분되어 있습니다: 초기화, 주요 While 루프, 재설정.

초기화 섹션

블록다이어그램의 초기화 섹션은 컨트롤 데이터 배열 및 비어있는 다층 시퀀스 구조를 포함합니다. 컨트롤 데이터 배열은 소스 VI의 [설정 가능 파라미터](#)를 나열합니다. 다층 시퀀스 구조를 통해 설정 가능 컨트롤에 대한 기본 값을 설정할 수 있습니다.

컨트롤 데이터 배열에 있는 컨트롤 이름은 소스 VI의 컨트롤 이름과 일치합니다. 배열은 설정 대화 상자과 익스프레스 VI 사이에서 값을 전달하는 모든 컨트롤을 포함해야 합니다. 익스프레스 VI는 컨트롤 값을 읽고, 이에 대한 [기본 도움말 정보를 생성](#)하고, 이를 다시 익스프레스 VI에 씁니다.

컨트롤 데이터 배열에 있는 각 원소는 다음의 객체를 포함하는 클러스터입니다:

- 컨트롤 타입—프런트패널 컨트롤, 상수, 익스프레스 VI 제목, 또는 다이내믹 데이터 타입 컨트롤과 같이 소스 VI에 있는 컨트롤의 타입을 지정합니다.
- 소스 VI 컨트롤 라벨—소스 VI의 프런트패널에 나타나는 것과 같이 아이템의 라벨을 지정합니다.
- 설정 대화 상자 컨트롤 라벨—설정 대화 상자 VI의 프런트패널에 나타나는 것과 같이 아이템의 라벨을 지정합니다. 비어있는 문자열은 해당 라벨이 소스 VI의 라벨과 동일하다는 표시입니다.



팁

컨트롤 데이터 배열에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 라벨을 선택하여, 각 배열 원소에 대한 라벨을 봅니다.

주요 While 루프 섹션

블록다이어그램의 주요 While 루프 섹션은 사용자 인터페이스 이벤트를 처리합니다. 이 섹션은 프런트패널에서 익스프레스 VI의 기본값 및 가장 최근 반복의 값을 디스플레이하는 영역에 대응합니다.

설정 대화 상자 VI가 사용자 이벤트를 기다리기 때문에, While 루프에 있는 주요 객체는 이벤트 구조입니다. 이벤트 구조는 모든 대화 상자에 포함되어 있는 3개의 버튼 중 하나가 클릭되기를 기다립니다: 확인, 취소, 도움말. 확인 또는 취소 버튼을 클릭하면, 루프가 완료됩니다. 확인 버튼을 클릭하면 설정 대화 상자 VI가 새로운 설정을 저장합니다. 취소 버튼을 클릭하면 변경 사항을 저장하지 않고 설정 대화 상자를 닫습니다. 도움말 버튼을 클릭하면 설정 대화 상자를 계속 열어 놓으며, [참조된 도움말 파일](#)도 열립니다.

설정 대화 상자 VI의 주요 While 루프 섹션을 통해 사용자는 VI가 재설정 섹션에 쓰는 데이터를 입력할 수 있습니다. 사용자가 익스프레스 VI 설정 대화 상자에서 원하는 셋팅을 선택할 때, 주요 While 루프 섹션은 이 선택 사항을 처리합니다. 주요 While 루프 섹션은 필요한 경우에 컨트롤을 비활성화하고, 라디오 버튼을 업데이트하고, 샘플 그래프 또는 미리보기 데이터와 같은 피드백 인디케이터를 업데이트 합니다.

재설정 섹션

블록 다이어그램의 재설정 섹션은 확장할 터미널 배열 및 도움말 생성 VI를 포함합니다. 확장할 터미널 배열은 기본적으로 확장된 상태로 나타나는 익스프레스 VI의 터미널을 나열합니다. 배열을 편집한다고 확장된 터미널이 사라지거나 순서가 바뀌지는 않습니다. 결과적으로 사용자가 확장하고 와이어로 연결한 터미널은 설정 대화 상자에서 사용자가 선택한 값을 덮어씁니다. 도움말 생성 VI는 익스프레스 VI의 현재 설정에 맞게 기본 도움말 윈도우를 업데이트합니다.

또한 재설정 섹션을 사용하여 입력 및 출력의 이름을 변경하고 터미널 팬에 컨트롤 및 인디케이터를 와이어로 연결할 수 있습니다.

소스 VI 디자인하기

익스프레스 VI의 소스 VI는 그 익스프레스 VI의 근간이 되는 코드를 포함합니다. 소스 VI는 익스프레스 VI의 동작을 결정하는 SubVI에 대한 래퍼(wrapper)의 역할을 합니다. 소스 VI는 또한 익스프레스 VI의 커넥터 팬과 아이콘을 결정합니다. 익스프레스 VI 생성자 마법사를 사용하여 새 익스프레스 VI를 **생성**한 후에, 소스 VI를 만들어 익스프레스 VI의 동작을 지정하고 커넥터 팬과 아이콘을 디자인합니다.

소스 VI 프런트패널 만들기

소스 VI의 프런트패널을 만들어서 SubVI에 정보를 전달해야 하는 모든 객체를 포함시키도록 합니다. 이 프런트패널은 사용자에게 공개되지 않습니다. **이미 있는 기존 VI로부터** 익스프레스 VI를 생성하는 경우, 소스 VI의 프런트패널은 원본 VI에서 사용자가 **확장 가능**으로 설정한 모든 컨트롤과 인디케이터를 포함하게 됩니다. **다른 익스프레스 VI로부터** 익스프레스 VI를 생성하는 경우, 소스 VI의 프런트패널은 원본 익스프레스 VI와 똑같습니다. **비어있는 새 VI로부터** 익스프레스 VI를 생성하는 경우, 소스 VI의 프런트패널은 비어 있으며 사용자가 처음부터 이를 만들어야 합니다.

다음 리스트는 익스프레스 VI를 위한 소스 VI의 프런트패널을 만들 때 고려해야 할 몇몇 사항을 설명하고 있습니다.

- 컨트롤 팔레트 상에 있는 컨트롤과 인디케이터만을 사용합니다. 익스프레스 VI 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤/인디케이터를 선택하면, LabVIEW는 익스프레스 소스 VI의 프런트패널에 컨트롤/인디케이터의 복사본을 생성합니다.

- 여러 값을 포함하는 입력에 대해서는 웨이브폼의 1D 배열을 사용합니다. 여러 값을 포함하는 출력에 대해서는 웨이브폼 그래프를 사용합니다. 그 외에도, 스칼라 값에 대해서는 숫자형 컨트롤을, 불리언 값에 대해서는 LED 컨트롤을, 문자열에 대해서는 문자열 컨트롤을 사용합니다. 컨트롤 및 인디케이터를 수정하지 마십시오.

소스 VI 블록다이어그램 만들기

소스 VI의 프런트패널을 생성한 후, 블록다이어그램을 변경하여 익스프레스 VI의 동작을 지정할 수 있습니다. 기존 VI로부터 익스프레스 VI를 생성한 경우, 소스 VI의 블록다이어그램은 원본 VI와 똑같은 SubVI를 포함합니다. 다른 익스프레스 VI로부터 익스프레스 VI를 생성한 경우, 소스 VI의 블록다이어그램은 원본 익스프레스 VI와 똑같습니다. 새 VI로부터 익스프레스 VI를 생성한 경우, 소스 VI의 블록다이어그램은 템플릿 SubVI를 포함합니다. 익스프레스 VI에 대해 구상한 기능을 수행할 수 있도록 SubVI를 만들고, SubVI의 **커넥터 팬**과 **고유 아이콘**을 디자인합니다. 익스프레스 VI를 통해 프로그램적으로 설정하려는 모든 입력과 출력이 커넥터 팬에 포함되어 있어야 합니다.

다음 리스트는 익스프레스 VI에 대한 SubVI를 만들 때 고려할 사항을 설명합니다.

- 생성하고 있는 익스프레스 VI가 메모리를 많이 차지하면서 동시에 활성화되지 않는 여러 개의 SubVI를 포함하고 있는 경우, 참조를 통해 SubVI를 호출합니다. 하지만, LabVIEW 어플리케이션 빌더를 통해 생성한 실행 가능한 어플리케이션 안에 익스프레스 VI를 포함시키는 경우, 빌드 참조에 의해 호출된 SubVI를 사용자가 수동으로 추가해야 합니다.
- LabVIEW 모듈 또는 툴킷 VI, 드라이버, 기타 소프트웨어 등 필수 LabVIEW 개발 환경에 속하지 않는 객체를 포함시킬 수 있습니다. 그러나, 이러한 객체를 포함하는 익스프레스 VI는 해당 객체가 존재하고 있는 컴퓨터 상에서만 실행이 가능합니다.

소스 VI의 블록다이어그램은 익스프레스 VI에 있는 각 설정 가능 및 확장 가능 파라미터에 대한 컨트롤 또는 상수를 포함해야 합니다. 파라미터가 설정 가능하면서도 확장 가능한 경우, 또는 파라미터가 확장 가능하지만 설정 가능하지 않은 경우, 소스 VI의 블록다이어그램은 해당 파라미터에 대응하는 컨트롤 또는 인디케이터를 포함해야 합니다. 파라미터가 설정 가능하지만 확장 가능하지 않은 경우, 소스 VI의 블록다이어그램은 해당 파라미터에 대응하는 상수를 포함해야 합니다.

익스프레스 VI 커넥터 팬 및 아이콘 디자인하기

소스 VI의 프런트패널 및 블록다이어그램을 만든 후, 소스 VI에 대한 커넥터 팬 및 고유 아이콘을 디자인합니다. 소스 VI의 커넥터 팬에 모든 확장 가능 터미널을 포함시켜야 합니다. 설정 가능 파라미터는 커넥터 팬에 포함시키지 않아도 됩니다. 소스 VI의 커넥터 팬과 아이콘은 익스프레스 VI의 커넥터 팬 및 아이콘에 대응합니다.

익스프레스 VI 파라미터 옵션 선택하기

설정 대화 상자에서 **익스프레스 VI**에 대해 설정할 수 있는 파라미터는 설정 가능 파라미터라고 부릅니다. 블록다이어그램에서 익스프레스 VI에 대해 설정할 수 있는 파라미터는 확장 가능 파라미터라고 확장 가능 터미널이라고 부릅니다. 설정 가능하면서도 확장 가능한 파라미터도 있는데, 이는 두 방법 모두를 통해 설정할 수 있는 파라미터입니다. 설

설정 가능 파라미터를 사용하여 익스프레스 VI에 대해 지정한 설정에 의해 블록 다이어그램에 나타나는 확장 가능 터미널이 결정됩니다.

설정 가능 또는 확장 가능한 것으로 설정한 파라미터 **소스 VI** 및 **설정 대화 상자 VI** 모두 안에 존재해야 합니다. 소스 VI에서 각각의 설정 가능 또는 확장 가능 파라미터에 대한 컨트롤 및 상수를 포함하도록 블록 다이어그램을 편집합니다. 파라미터가 설정 가능하면 서도 확장 가능한 경우 또는 파라미터가 확장 가능하지만 설정 가능하지 않은 경우, 소스 VI는 해당 파라미터에 대응하는 컨트롤 또는 인디케이터를 포함해야 합니다. 파라미터가 설정 가능하지만 확장 가능하지 않은 경우, 소스 VI는 해당 파라미터에 대응하는 상수를 포함해야 합니다. 소스 VI의 커넥터 팬에 모든 확장 가능 터미널을 포함시켜야 합니다. 설정 가능 파라미터는 커넥터 팬에 포함시키지 않아도 됩니다.

설정 대화 상자 VI의 블록 다이어그램에서는 컨트롤 데이터 배열이 익스프레스 VI에 대한 설정 가능 파라미터를 나열하고, 확장할 터미널 배열은 익스프레스 VI에 대한 확장 가능 터미널을 나열합니다. 이러한 배열의 원소를 추가, 제거, 편집하여 설정 가능 파라미터 및 확장 가능 파라미터의 개수를 설정합니다.

또한 익스프레스 VI 생성 마법사를 사용하여 표준 VI를 익스프레스 VI로 변환할 때, 설정 가능 또는 확장 가능한 파라미터를 설정할 수 있습니다. VI 변환 페이지에 있는 설정 가능 열은 설정 대화 상자 VI의 블록 다이어그램에 있는 컨트롤 데이터 배열에 대응합니다. VI 변환 페이지에 있는 확장 가능 열은 확장할 터미널 배열에 대응합니다.

익스프레스 VI를 생성하기 이전에, 다음 테이블을 사용하여 각 파라미터에 대해 가장 적절한 옵션을 결정합니다.

선택한 파라미터 옵션	결과	사용자에게 미치는 영향
설정 가능 및 확장 가능 (기본)	확장할 터미널 배열에 원소를 추가합니다. 파라미터가 컨트롤인 경우, 컨트롤 데이터 배열에 원소가 추가되고 소스 VI에 컨트롤이 추가됩니다.	사용자는 설정 대화 상자에서 컨트롤을 설정하거나 익스프레스 VI에 값을 와이어로 연결해야 합니다.
설정 가능	컨트롤 데이터 배열에 원소가 추가되고 소스 VI에 상수가 추가됩니다.	사용자는 설정 대화 상자에서 컨트롤을 설정해야 합니다.
확장 가능	확장할 터미널 배열에 원소가 추가되고 소스 VI에 컨트롤/인디케이터가 추가됩니다.	사용자는 익스프레스 VI에 값을 와이어로 연결해야 합니다.
없음	어떤 익스프레스 VI 구성요소를 수정하지 않습니다.	사용자는 기본값을 유지해도 됩니다.

익스프레스 VI를 실행할 때, 확장 가능 터미널을 통해 지정한 값은 설정 가능 파라미터를 통해 지정한 값보다 우선 순위가 높습니다.

배리언트 데이터 처리하기

때로는 VI가 다양한 타입의 데이터를 보편적인 방법으로 처리해야 하는 경우도 있습니다. 특정 데이터 타입에 대해 각각 서로 다른 VI를 작성할 수 있지만, 한 VI를 변경하면 나

머지 VI도 모두 변경할 필요가 있기에 여러 개의 사용자 VI의 복사본을 보유하면 관리하기 어렵습니다. 이러한 상황에 대한 해결책으로 LabVIEW에는 배리언트 데이터 타입이 있습니다. 배리언트 데이터 타입은 LabVIEW의 모든 데이터 타입을 위한 일반적인 컨테이너입니다. 다른 데이터를 배리언트로 변환할 때 배리언트는 데이터 및 원래 데이터 타입을 저장하여 LabVIEW가 차후에 배리언트 데이터를 원래 데이터로 올바르게 변환시킬 수 있습니다. 예를 들어, 문자열 데이터를 배리언트로 변환하는 경우에는 배리언트가 문자열의 텍스트와 함께 해당 데이터가 원래 문자열이었음(경로, 바이트 배열, 기타 LabVIEW 데이터 타입이 아니었음)을 알리는 인디케이터를 저장하게 됩니다.

[배리언트] 함수를 사용하여 배리언트 데이터를 생성하고 조작합니다. 모든 LabVIEW 데이터 타입은 배리언트 데이터 타입으로 변환하여 다른 VI와 함수에서 배리언트 데이터로 사용할 수 있습니다. VI 열기: 커넥터 팬 데이터 타입 메소드 및 커넥터 팬: 데이터 타입 프로퍼티 등의 여러 프로퍼티와 메소드는 데이터 타입을 배리언트로 반환합니다. 데이터 타입 분석 VI를 사용하여 배리언트의 데이터 타입과 데이터 타입에 대한 타입 정보를 가져옵니다. 또한 배리언트의 데이터 타입이 특정 데이터 타입과 일치하는지 확인할 수 있습니다.

데이터를 전송하거나 보관할 때, 알려지지 않은 디바이스에 쓰거나 읽을 때, 또는 서로 다른 컨트롤의 세트에 연산을 수행하는 것과 같이 데이터 타입에 독립적으로 데이터를 처리하는 것이 중요할 때 배리언트 데이터 타입을 씁니다.

예를 들어, 배리언트 데이터 타입을 요구하는 **ActiveX 객체**를 사용할 때 배리언트 데이터 타입을 사용합니다. 또한, 배리언트 데이터 타입을 컨트롤 참조와 함께 사용하여 **프로그래밍적으로 선택된 컨트롤**의 값을 읽거나 수정할 수 있습니다.

또한, [패턴화된 문자열로] 함수를 사용하여 독립적인 타입의 데이터를 나타내기 위해서 데이터 타입을 문자열 데이터 타입으로 변환할 수 있습니다. 문자열로 **데이터를 패턴화**하는 것은 TCP/IP를 사용하여 데이터를 전송할 때 유용합니다. 왜냐하면 프로토콜은 오직 문자열만 인식하기 때문입니다. 하지만, LabVIEW는 원래 데이터 타입이 변환하려는 데이터 타입과 일치하지 않을 때 패턴화된 데이터를 강제 변환할 수 없기 때문에 패턴화된 데이터를 사용하는 데에는 한계가 있습니다. ActiveX 배리언트 데이터는 패턴화할 수 없습니다. 또한, 패턴화된 정수를 확장형 부동소수로 패턴화 해제할 수 없습니다.

배리언트 데이터 타입을 사용하는 또 다른 장점은 데이터의 속성을 저장할 수 있는 기능입니다. 속성은 사용자가 정의하고 배리언트 데이터 타입이 저장하는 데이터에 대한 정보입니다. 예를 들어, 일정 데이터가 언제 생성되었는지를 알리는 경우, 데이터를 배리언트 데이터로 저장하고 시간 문자열을 저장하기 위해서 시간으로 불리는 속성을 추가할 수 있습니다. 속성 데이터는 어떠한 타입도 될 수 있습니다. 특정 속성으로 데이터를 분류하고, 데이터를 생성하는 디바이스 또는 어플리케이션을 식별하거나, 또는 특정 속성을 가지는 배리언트만을 위해 데이터를 필터하려는 때에 배리언트 속성을 사용합니다.



노트

배리언트 데이터에서 [패턴화된 문자열로] 함수를 사용하는 경우, LabVIEW는 배리언트와 그 속성을 포함하는 모든 내용을 패턴화합니다. 그러나, [배리언트를 패턴화된 문자열로] 함수를 사용하는 경우, LabVIEW는 오직 배리언트만을 패턴화하고 모든 속성을 삭제합니다.

VI와 SubVI 생성하기

VI는 사용자 인터페이스 또는 자주 사용하는 작업이 될 수 있습니다. 프런트패널과 블록 다이어그램을 만드는 방법을 배운 후에는 자신만의 VI와 subVI, 배포 VI를 생성하고 독립 어플리케이션과 공유 라이브러리를 만들 수 있습니다.



관련 정보

[예제, VI 템플릿, 프로젝트 템플릿, 샘플 프로젝트](#) (293 페이지 참조)

[VI의 섹션에서 SubVI 생성하기](#) (127 페이지 참조)

[아이콘 사용하기](#) (141 페이지 참조)

[VI 저장하기](#) (145 페이지 참조)

[VI 사용자 정의하기](#) (148 페이지 참조)

[블록다이어그램에 SubVI 놓기](#) (1347 페이지 참조)

[SubVI 열기 및 편집하기](#) (1348 페이지 참조)

[VI 저장하기](#) (887 페이지 참조)

[VI 사용자 정의하기](#) (898 페이지 참조)

내장된 VI와 함수 사용하기

LabVIEW에는 데이터 수집 VI와 함수, 다른 VI에 접근하는 VI, 다른 어플리케이션과 통신하는 VI 등과 같이 특수한 어플리케이션을 만드는데 도움을 주는 내장 VI와 함수가 있습니다. 이러한 VI를 어플리케이션에 SubVI로 사용하여 개발 시간을 줄일 수 있습니다. 새 VI를 만들기 전에 함수 팔레트를 검색하여 비슷한 VI와 함수를 찾고 기존 VI를 새 VI의 시작점으로 사용하는 것을 고려하십시오.



관련 정보

[모듈러 코드 생성하기](#) (129 페이지 참조)

LabVIEW의 SubVI

VI의 섹션에서 SubVI 생성하기

위치 도구를 사용하여 재사용하려는 블록다이어그램 부분을 선택하고 편집>>SubVI 생성을 선택하여 VI의 일부를 SubVI로 변환합니다. 새로운 SubVI의 아이콘이 블록다이어그램에서 선택한 부분을 대체합니다. LabVIEW는 새 SubVI에 컨트롤과 인디케이터를 생성하고, 선택한 컨트롤과 인디케이터 터미널의 개수를 기반으로 커넥터 팬을 자동 설정하며, 기존 와이어에 SubVI를 연결합니다.

이렇게 일부 선택한 후 SubVI를 생성하면 편리하지만, 이런 경우 VI의 논리적인 계층구조를 생성하려면 신중한 계획이 요구됩니다. 어떤 객체를 선택할 것인지 고려해야 하고, 새로 생성된 VI의 기능이 바뀌는 경우는 피해야 합니다.

SubVI와 익스프레스 VI를 아이콘 또는 확장 가능 노드로 디스플레이하기

VI와 익스프레스 VI를 아이콘 또는 확장 가능 노드로 나타낼 수 있습니다. 확장이 가능한 노드는 색이 있는 배경 안에 놓인 아이콘으로 나타납니다. 확장 가능 노드로 디스플레이 되는 경우, SubVI는 노란색 배경 안의 아이콘으로 나타납니다, 익스프레스 VI는 파란색 배경 안의 아이콘으로 나타납니다. 블록다이어그램의 공간을 효율적으로 사용하려면 아이콘을 사용하십시오. 확장 가능한 노드를 사용하면 블록다이어그램에서 와이어 연결과 문서화가 용이합니다. 블록다이어그램에서 기본 설정으로 SubVI는 아이콘으로 나타나고, 익스프레스 VI는 확장 가능한 노드로 나타납니다. SubVI 또는 익스프레스 VI를 확장 가능 노드로 디스플레이하려면, SubVI 또는 익스프레스 VI에서 마우스 오른쪽 버튼을 클릭한 후 아이콘으로 보기 바로 가기 메뉴 아이템 옆의 확인 표시를 제거합니다.



노트

SubVI 또는 익스프레스 VI를 확장 가능 노드로 나타낼 경우, 이 노드는 데이터베이스 접속을 활성화할 수 없습니다.

확장 가능 subVI 또는 익스프레스 VI를 크기 조정할 때, subVI 또는 익스프레스 VI의 입력과 출력 터미널은 아이콘 아래에 나타납니다. 옵션 터미널은 회색 배경으로 나타납니다. 디스플레이하지 않는 권장 또는 필수 입력이나 출력 터미널은 subVI 또는 익스프레스 VI 아이콘의 배경색에 입력 또는 출력 화살표로 나타납니다. SubVI 또는 익스프레스 VI가 확장되었을 때 옵션 터미널에 연결을 한 후 옵션 터미널이 더 이상 확장된 영역에 나타나지 않도록 subVI 또는 익스프레스 VI를 크기 조정할 경우, 옵션 터미널은 배경색에 입력 또는 출력 화살표로 나타납니다. 하지만, 옵션 터미널을 연결하지 않은 경우, 입력 또는 출력 화살표는 나타나지 않습니다.

기본적으로, subVI 또는 익스프레스 VI를 확장할 때 입력은 출력 위에 나타납니다. 확장 가능한 영역에 있는 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력/출력 선택을 선택하여 디스플레이할 입력 또는 출력을 선택합니다. 바로 가기 메뉴에서 라인은 입력과 출력을 나눕니다. 확장 가능한 subVI와 익스프레스 VI의 라벨은 아이콘을 둘러싸는 배경 안에 나타납니다. 확장 가능 노드를 크기 조정하여 각 터미널의 이름이 확장 가능한 영역의 한 라인에 나타나게 하려면, subVI 또는 익스프레스 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 텍스트 크기에 맞춤을 선택합니다.

다음 그림은 10개의 입력과 출력 터미널 중에서 4개를 디스플레이하는 확장 가능한 SubVI를 보여줍니다.



모듈러 코드 생성하기

LabVIEW에서는 SubVI를 사용하여 다른 VI에서 사용할 코드를 생성합니다. 지나치게 큰 블록다이어그램은 읽기 및 관리하기가 어렵습니다. 코드에서 일반적으로 다시 사용되는 부분을 확인하고 이러한 부분을 SubVI로 대체하면 공간이 절감됩니다. 또한 이후 해당 SubVI를 편집하게 되는 경우, 변경 사항이 현재 인스턴스뿐만 아니라 SubVI의 모든 호출에 영향을 끼치기 때문에 이후 업데이트 수행이 간단해집니다. 여러 동일한 SubVI 노드를 포함하는 블록다이어그램은 매번 동일한 SubVI를 호출합니다.



노트

SubVI는 텍스트 기반 프로그래밍 언어의 서브루틴 호출에 해당합니다.

VI를 작성한 후 SubVI로 사용하려면, 반드시 커넥터 팬을 작성하고 아이콘을 생성하여야 합니다. 이후 다른 VI의 블록다이어그램에 해당 SubVI를 놓습니다. SubVI 컨트롤과 인디케이터는 호출하는 VI의 블록다이어그램에서 데이터를 받고 데이터를 반환합니다.



관련 정보

[선택에 의해 SubVI 생성하기](#) (1348 페이지 참조)

[SubVI 열기 및 편집하기](#) (1348 페이지 참조)

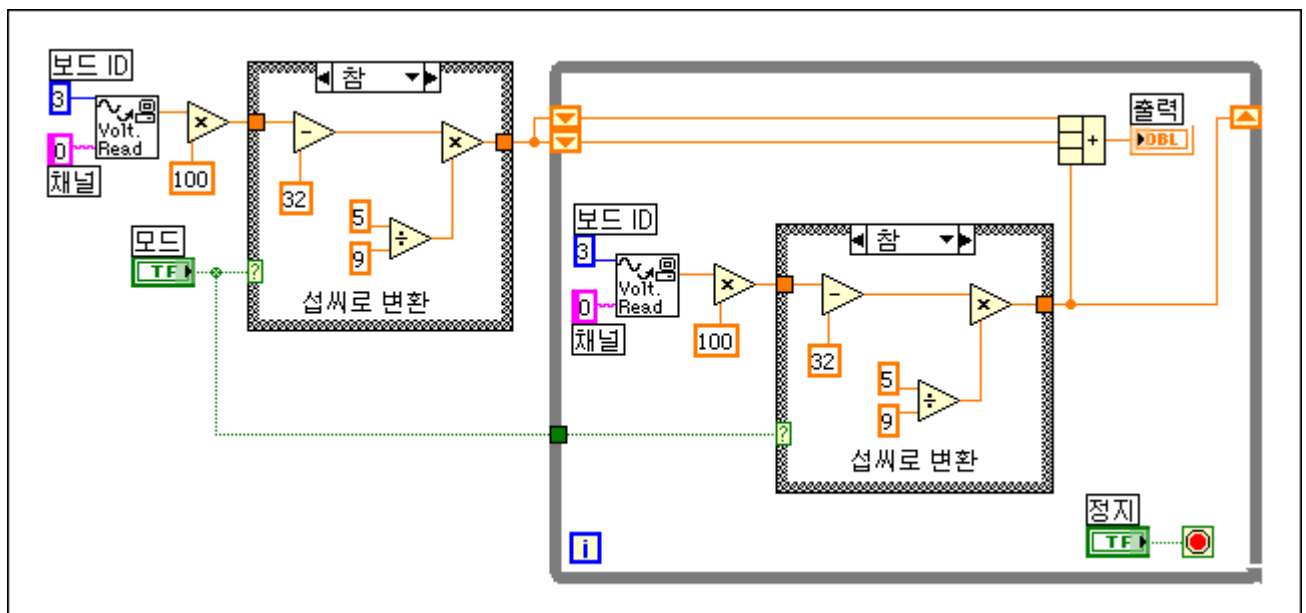
[커넥터 팬 만들기](#) (130 페이지 참조)

[아이콘 사용하기](#) (141 페이지 참조)

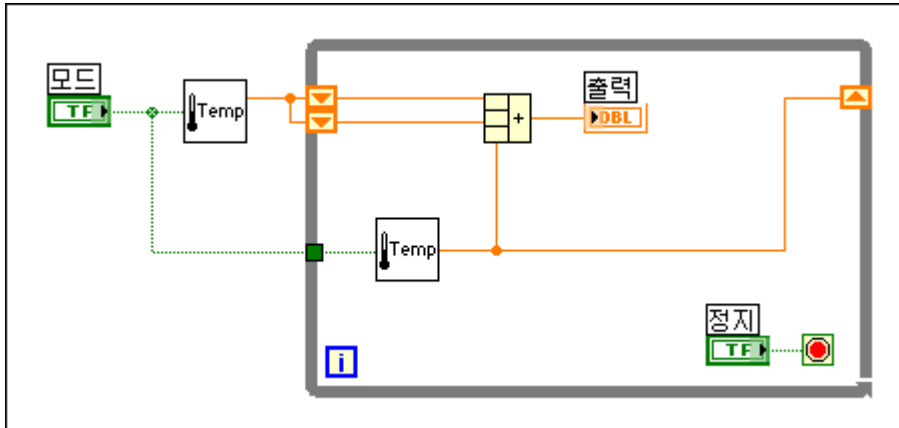
[블록다이어그램에 SubVI 놓기](#) (1347 페이지 참조)

공통된 작업 주목하기

VI를 생성하다 보면 특정 작업을 자주 수행하는 것을 발견하게 됩니다. SubVI 또는 루프를 사용하여 이러한 작업을 반복적으로 수행할 수 있습니다. 예를 들어, 다음 블록다이어그램 안에는 2 개의 동일한 작업이 반복되고 있습니다.



다음 블록 다이어그램에서 보이는 것과 같이, 해당 작업을 수행하는 subVI를 생성하여 subVI를 두 번 호출할 수 있습니다.



이 SubVI는 다른 VI에서도 다시 사용할 수 있습니다. 또한 루프를 사용하여 공통된 작업을 결합할 수 있습니다.

커넥터 팬 만들기

VI를 SubVI로 사용하려면 커넥터 팬을 만들어야 합니다. 모든 VI는 다음과 같이 프런트패널 윈도우 오른쪽 위 코너의 VI 아이콘 옆에 커넥터 팬을 디스플레이합니다.



커넥터 팬은 해당 VI의 컨트롤과 인디케이터에 대응하는 터미널의 세트이며, 텍스트 기반 프로그래밍 언어에서 함수 호출의 파라미터 리스트와 유사합니다. 커넥터 팬에서 VI에 연결할 수 있는 입력과 출력을 정의하면 VI를 SubVI로 사용할 수 있게 됩니다. 커넥터 팬은 입력 터미널에서 데이터를 받고, 프런트패널 컨트롤을 통해 데이터를 블록 다이어그램 코드에 전달하며, 프런트패널 인디케이터의 출력 터미널에서 결과를 받습니다.

커넥터 팬의 각 사각형은 터미널을 나타냅니다. 커넥터 팬 패턴을 선택한 후, 사각형을 사용하여 입력 및 출력으로 프런트패널 컨트롤과 인디케이터를 지정합니다.

커넥터 팬 만들기에 대한 권장사항

다음 리스트는 커넥터 팬을 만들 때 고려해야 할 권장사항을 설명합니다. 커넥터 팬 생성에 대한 추가적인 팁은 [LabVIEW Style Checklist](#)를 참조하십시오.

- National Instruments는 사용하는 터미널이 16개가 넘지 않을 것을 권장합니다. 터미널 개수가 16개가 넘는 커넥터 팬은 와이어로 연결하기 어렵습니다. 더 많은 데이터를 전달할 필요가 있는 경우에는, 클러스터를 사용하십시오.
- VI의 변경으로 새 입력이나 출력이 필요한 경우를 대비해, 여분의 터미널을 할당하지 않은 채로 남겨둡니다.
- 터미널을 입력 및 출력으로 지정할 때, 반드시 커넥터 팬의 터미널을 일관성 있게 나누도록 합니다. $4 \times 2 \times 2 \times 4$ 패턴에서 중간에 있는 4개의 터미널을 사용해야 하

는 경우, 수평 또는 수직으로 터미널을 나눕니다. 예를 들어, 위쪽 두 터미널을 입력으로 지정하고 아래쪽 두 터미널을 출력으로 지정하거나, 왼쪽 두 터미널은 입력으로 지정하고 오른쪽 두 터미널을 출력으로 지정할 수 있습니다.

- 자주 함께 사용하는 SubVI를 모아 그룹을 만드는 경우, 각 입력을 쉽게 찾을 수 있도록 해당 SubVI들의 공통된 입력을 같은 위치에 배치함으로써 일관성 있는 커넥터 팬을 만듭니다. 참조, 태스크 ID, 에러 클러스터 등과 같이, 다른 subVI가 입력으로 사용하는 것을 출력으로 하는 subVI를 생성하는 경우, 입력과 출력 연결을 정렬하여 와이어링 패턴을 단순화시킵니다.
- 프론트패널의 왼쪽 아래 코너에 에러 입력 클러스터를 놓고 오른쪽 아래 코너에 에러 출력 클러스터를 놓습니다. 다음 그림은 부적절하게 정렬된 에러 클러스터와 적절하게 정렬된 에러 클러스터의 예제를 보여줍니다.



1	부적절함
2	적절함

필수, 권장, 옵션 입력과 출력 설정하기

어떤 입력과 출력이 필수, 권장, 옵션인지를 지정하여, 사용자가 SubVI의 터미널의 연결을 잊지 않도록 도와줄 수 있습니다.

터미널 입력이 필수 입력인 경우, SubVI를 놓은 블록 다이어그램에서 해당 필수 입력을 와이어로 연결하지 않으면 블록 다이어그램이 깨집니다. 그러나, 터미널 출력은 필수로 설정할 수 없습니다. 터미널 입력과 출력의 경우, 권장 및 옵션이란 터미널 연결이 선택 사항임을 의미합니다. 터미널을 연결하지 않은 경우, 블록 다이어그램에 subVI를 놓으면 어떤 경고도 생성되지 않습니다.

vi.lib에 있는 VI의 입력과 출력은 이미 필수, 권장, 옵션으로 표시되어 있습니다. LabVIEW에서 생성하는 VI의 입력 및 출력은 권장으로 기본 설정됩니다. VI가 제대로 실행되려면 반드시 입력 또는 출력이 있어야 하는 경우에만, 해당 터미널을 필수로 설정하십시오.

어떤 입력과 출력이 필수, 권장 또는 옵션인지 지정하려면, 커넥터 팬의 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이 연결을 선택합니다. LabVIEW 설정을 통해 커넥터 팬에서 설정하는 모든 새로운 터미널을 권장이 아닌 필수로 설정할 수 있습니다. 도구 >> 옵션 >> 프론트패널을 선택한 후 커넥터 팬 터미널 기본을 필수로 확인란에 확인 표시를 합니다. 이 옵션은 또한 편집 >> SubVI 생성 메뉴 아이টে를 사용한 연결에 적용됩니다.

기본 도움말 윈도우에서 필수 터미널 라벨은 굵은체로, 권장 터미널 라벨은 보통체로, 옵션 터미널 라벨은 회색 글씨체로 표시됩니다. 기본 도움말 윈도우에서 아래에 보이는 것과 같은 옵션 터미널과 전체 경로 숨기기 버튼을 클릭하면, 옵션 터미널의 라벨이 나타나지 않습니다.



SubVI 프론트패널 디자인하기

사용자가 SubVI의 프론트패널을 볼 필요가 없을 경우, 색과 폰트를 포함하여 프론트패널 모양에 시간을 많이 소모하지 않아도 됩니다. 하지만, VI를 디버그하는 동안 프론트패널을 봐야하기 때문에 프론트패널 구성은 여전히 중요합니다.

커넥터 팬에 나타나는 것처럼 프론트패널에 컨트롤과 인디케이터를 추가합니다. 프론트패널의 왼쪽에 컨트롤을 놓고 오른쪽에 인디케이터를 추가합니다. 프론트패널의 왼쪽 아래에 에러 입력 클러스터를 놓고 오른쪽 아래에 에러 출력 클러스터를 놓습니다.



팁

특정 SubVI가 에러를 처리하지 않더라도 모든 SubVI에 에러 입력과 에러 출력 클러스터를 포함시키는 것이 좋습니다. 에러 입력과 에러 출력 클러스터는 실행 흐름을 조절하는데 유용하게 사용됩니다.

컨트롤에 기본값과 측정 단위를 포함하는 **의미있는 라벨**을 붙이십시오.

SubVI 프론트패널 디스플레이하기

LabVIEW에서 SubVI가 호출될 때, SubVI는 프론트패널을 디스플레이하지 않고 실행됩니다. LabVIEW에서 SubVI가 호출되었을 때 SubVI의 프론트패널이 디스플레이되도록 하려면, 다음 대화 상자를 사용합니다:

- SubVI 노드 설정 대화 상자—SubVI의 현재 인스턴스에 대해서만 옵션을 설정합니다.
- 윈도우 모양 사용자 정의 대화 상자—SubVI의 모든 인스턴스에 대해 옵션을 설정합니다.

VI의 계층구조 보기

VI 계층구조 윈도우는 메모리 상에 있는 모든 VI에 대한 호출 계층구조뿐만 아니라, 열려 있는 LabVIEW 프로젝트 및 타겟을 모두 그래픽으로 디스플레이합니다. 윈도우에 디스플레이되는 아이템은 다음과 같습니다:

- 프로젝트 및 타겟
- 정적 및 다이나믹 SubVI 호출
- 타입 정의
- 글로벌 변수
- 공유 변수
- LabVIEW 클래스
- X컨트롤

- Statechart
- 프로젝트 라이브러리 .lvlib
- 익스프레스 VI
- 정적 VI 참조

보기»VI 계층구조를 선택하여 VI 계층구조 윈도우를 디스플레이합니다. 이 윈도우를 사용하여 메모리 상의 VI를 구성하는 **SubVI와 다른 노드 보기**를 하고 **VI 계층구조 검색**을 합니다.

VI 계층구조 윈도우는 최상위 아이콘을 사용하여 주요 LabVIEW **어플리케이션 인스턴스**를 나타내며, 그 아래에 프로젝트에 속하지 않거나 프로젝트의 어플리케이션 인스턴스에 속하지 않는 모든 객체가 나타납니다. 프로젝트를 추가하면 VI 계층구조 윈도우는 다른 상위 레벨 아이콘을 디스플레이하여 프로젝트를 나타냅니다. 추가한 각 타겟은 프로젝트 아래에 나타납니다.

커서를 VI 계층구조 윈도우의 객체 위로 움직이면, LabVIEW는 각 객체의 이름을 팁 상자로 디스플레이합니다. 위치 도구를 사용하여 VI 계층구조 윈도우에 있는 객체를 다른 VI의 프런트패널 또는 블록다이어그램으로 끌어놓고, 그 VI에서 해당 객체를 사용할 수 있습니다. 또한, 1개의 노드 또는 여러 노드를 선택하고 클립보드에 복사하여 다른 프런트패널이나 블록다이어그램에 붙여넣을 수 있습니다. VI 계층구조 윈도우의 VI에서 더블 클릭하여 그 VI의 프런트패널을 디스플레이합니다.

VI 계층구조 윈도우는 메모리에 있는 내용을 동적으로 보여줍니다. 계층구조 안에 있는 아이콘을 재배치하는 경우, LabVIEW는 아이콘의 새 위치를 저장하지 않습니다. VI 계층구조 윈도우를 닫고 다시 열면, 원래의 위치가 재생됩니다.

SubVI를 가지는 VI에는 아래쪽 경계에 화살표 버튼이 있습니다. 이 화살표 버튼을 클릭하여 SubVI를 보이거나 숨깁니다. 최소한 1개의 SubVI가 숨겨진 상태일 때, 빨간색 화살표 버튼이 나타납니다. 검은색 화살표 버튼은 모든 SubVI가 디스플레이되었을 때 나타납니다. VI가 **재귀 호출**을 포함하는 경우, VI 계층구조 윈도우는 재귀 VI 사이에 점선을 그려서 이러한 재귀적인 관계를 표시합니다.

재귀 VI

재귀 VI는 자기 블록다이어그램에서 또는 SubVI의 블록다이어그램에서 스스로를 호출할 수 있습니다. 같은 프로세스의 출력 작업을 여러 번 실행하려는 경우, 재귀 VI가 유용합니다. **어떤 VI든지 재귀 VI로 만들** 수 있으며, 한 VI 계층구조에서 여러 재귀 VI를 사용할 수 있습니다.

VI 계층구조 전체에서 재귀 VI를 사용하려는 경우, 다음 방법 중에서 하나를 사용하여 계층구조에 있는 모든 VI를 설정할 수 있습니다.

- VI 계층구조에 있는 모든 VI를 **재호출**로 설정하고, 그 중에서 최소한 하나의 VI가 스스로의 복제본을 여러 호출 인스턴스 사이에서 공유하도록 만듭니다.
- VI 계층구조에 있는 모든 VI를 **다이나믹 디스패치** 멤버 VI로 설정합니다.
- VI 계층구조에 있는 VI 중에서 최소한 하나의 VI를 다이나믹 디스패치 멤버 VI 또는 재호출로 설정하고, 스스로의 복제본을 여러 호출 인스턴스 사이에서 공유하도록 만듭니다. 나머지 VI를 재호출로 설정하고 다만 각 호출 인스턴스에 대해 복제본이 미리 할당되어 있도록 만듭니다.

(32 비트) 32 비트 플랫폼에서 LabVIEW는 최대 15,000 개의 재귀 호출을 지원합니다. (64 비트) 64 비트 플랫폼에서 LabVIEW는 최대 35,000 개의 재귀 호출을 지원합니다.

Real-Time Module LabVIEW Real-Time Module은 재귀를 지원합니다. 그러나, 재귀는 리얼 타임 확정적(real-time deterministic)이지 않으며, 사용자가 LabVIEW Real-Time에서 재귀 VI를 디버깅할 수 없습니다.

FPGA Module 현재 LabVIEW FPGA Module은 재귀를 지원하지 않습니다.

다형성 VI

다형성 VI는 서로 다른 데이터 타입에 적용됩니다. 다형성 VI는 같은 커넥터 팬 패턴을 가지는 VI의 모음입니다. 각 VI는 다형성 VI의 인스턴스로서, 각각의 인스턴스는 다른 인스턴스에 있는 다른 데이터 타입을 받는 입력 또는 출력 터미널을 1 개 이상 갖고 있습니다.

예를 들어, [키 읽기] VI는 다형성입니다. 이 VI의 기본값 터미널은 불리언; 배정도 부동소수; 32 비트 부호있는 정수; 경로; 문자열; 또는 32 비트 부호없는 정수 데이터를 받습니다.

대부분의 다형성 VI에서, 다형성 VI의 입력에 연결된 데이터 타입은 사용할 인스턴스를 결정합니다. 다형성 VI가 데이터 타입에 호환되는 인스턴스를 포함하지 않는 경우, 깨진 와이어가 나타납니다. 다형성 VI 입력에 연결한 데이터 타입이 사용할 인스턴스를 결정하지 않는다면, 그 인스턴스를 수동으로 선택해야 합니다. 다형성 VI의 인스턴스를 수동으로 선택하면, 그 VI는 선택한 인스턴스의 데이터 타입만을 받고 반환하기 때문에 이 VI는 더이상 다형성 VI로서 작동하지 않습니다.

인스턴스를 수동으로 선택하기 위해서, 다형성 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 선택을 선택하고 사용할 인스턴스를 선택합니다. 또한 수행 도구를 사용하여 다음에서 보는 것과 같이 다형성 VI 선택자를 클릭하고 바로 가기 메뉴에서 인스턴스를 선택할 수 있습니다.

읽기 전용 ▼

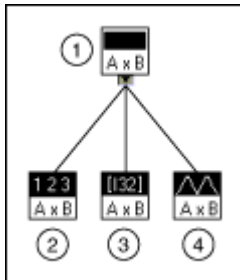
블록 다이어그램의 다형성 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목>>다형성 VI 선택자를 선택하여 선택자를 디스플레이합니다. 다루어지는 모든 데이터 타입을 다시 받도록 다형성 VI를 변경하기 위해, 다형성 VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 타입 선택>>자동을 선택하거나, 수행 도구를 사용하여 다형성 VI 선택자를 클릭하고 바로 가기 메뉴에서 자동을 선택합니다.

여러 다른 데이터 타입에서 같은 연산을 하고, 각 데이터 타입에서 연산이 다르게 구현되어야 하는 경우 다형성 VI를 만듭니다. 특정 데이터 타입을 위해 작업을 크게 바꾸지 않고도 여러 데이터 타입에서 작업이 비슷하게 구현될 수 있는 경우, 이 VI 대신 **가변 VI**를 생성합니다.

예를 들어, 단정도 부동소수, 숫자값의 배열, 또는 웨이브폼에 동일한 수학 연산을 수행하려는 경우에는 [숫자 계산], [배열 계산], [웨이브폼 계산]이라는 3 개의 분리된 VI를 생성할 수 있습니다. 연산을 수행할 때, 입력으로 사용하려는 데이터 타입에 따라서 3 개의 VI 중에서 하나를 블록 다이어그램에 놓습니다.

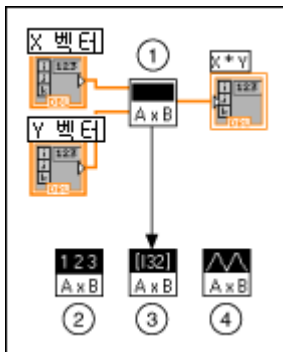
수동으로 VI의 버전을 블록다이어그램에 놓는 대신에, 단일 다형성 VI를 생성하고 사용할 수 있습니다.

다형성 계산 VI는 다음 그림에서와 같이 VI에 3 개의 인스턴스를 가집니다.



1	계산
2	숫자 계산
3	배열 계산
4	웨이브폼 계산

계산 VI는 다음 그림에서와 같이 블록다이어그램의 계산 SubVI에 연결한 데이터 타입에 따라서 올바른 VI의 인스턴스에 연결됩니다.



1	계산
2	숫자 계산
3	배열 계산
4	웨이브폼 계산

일반 VI와는 달리 다형성 VI에는 블록다이어그램 또는 프런트패널이 없습니다.

다형성 VI를 만들 때에는 다음 사항을 고려합니다:

- 다형성 VI에 포함되는 모든 VI는 동일한 커넥터 팬 패턴을 가져야만 합니다. 왜냐하면, 다형성 VI의 커넥터 팬은 다형성 VI를 생성하기 위해 사용하는 VI의 커넥터 팬과 일치하기 때문입니다.
- VI의 커넥터 팬에는 대응하는 입력과 출력 터미널이 있어야 합니다. 예를 들어, 한 VI의 커넥터 팬에 있는 터미널이 입력인 경우, 다른 VI의 커넥터 팬에 있는 대응하는 터미널 역시 입력이어야 하며 그렇지 않은 경우 사용하지 않아야 합니다.
- 각 VI의 커넥터 팬의 터미널 연결 수는 각각 다릅니다.
- 다형성 VI에 포함하는 여러 VI는 동일한 SubVI와 함수로 구성할 필요는 없습니다.
- 다른 다형성 VI에 다형성 VI를 사용할 수 없습니다.
- **가변 VI(.vim)**는 다형성 VI 안에 포함될 수 없습니다.

사용할 인스턴스를 결정하기 위해서 다형성 VI의 입력에 사용자가 연결하는 데이터 타입을 원하고 사용자가 VI가 다형성인 것을 알지 못하게 하려는 경우, 다형성 VI를 만들 때 다음 지침을 고려합니다.

- 사용자가 인스턴스를 수동으로 선택할 필요가 없기 때문에 다형성 VI 선택자를 숨깁니다.
- 다형성 VI에 대한 아이콘을 생성하여 사용자가 다형성 VI 위로 커서를 움직일 때 각 인스턴스에 대한 아이콘이 기본 도움말 윈도우에 나타나지 않도록 합니다.
- 다형성 VI의 설명을 생성하여 사용자가 다형성 VI 위로 커서를 움직일 때 각 인스턴스에 대한 설명이 기본 도움말 윈도우에 나타나지 않도록 합니다. 사용되는 인스턴스에 개의치 않고 다형성 VI를 설명하는 설명을 포함합니다.

블록 다이어그램에 다형성 VI를 놓으면, LabVIEW는 다형성 VI를 메모리에 로드합니다. 다형성 VI 윈도우에서 다형성 VI를 수정하고 변경된 사항을 저장하지 않고 윈도우를 닫으면, 변경된 사항은 계속해서 메모리에 남아있습니다. VI를 닫고 다시 열면, 다형성 VI의 변경된 사항은 더 이상 나타나지 않습니다.

다형성 SubVI를 포함하는 VI에 대한 완전한 설명을 생성할 때, 다형성 VI와 인스턴스는 문서의 SubVI 섹션 리스트에 나타납니다.

관련 정보

다형성 VI 만들기

가변 VI와 다형성 VI 중 선택하기

가변 VI

가변 VI(.vim)은 호출 VI로 인라인되고 각 터미널을 대응되는 입력 데이터 타입으로 변경할 수 있는 VI입니다. 가변 VI를 사용하면, 각 데이터 타입에 대해 별도의 VI를 저장하지 않고, 사용 가능한 모든 데이터 타입에 대해 같은 작업을 수행하는 VI를 생성할 수 있습니다. LabVIEW의 가변 VI는 주황색 배경색을 가집니다.

가변 VI 사용 예제는 labview\examples\Malleable VIs\Basics\Malleable VIs Basics.lvproj를 참조하십시오.



관련 정보

[VI 실행 속도](#) (191 페이지 참조)

[다양한 데이터 타입에서 가변 VI 사용하기](#) (137 페이지 참조)

[다양한 클래스에서 가변 VI 사용하기](#) (137 페이지 참조)

[가변 VI와 다형성 VI 중 선택하기](#) (138 페이지 참조)

[가변 VI 만들기](#) (886 페이지 참조)

[가변 VI 디버그하기](#) (887 페이지 참조)

다양한 데이터 타입에서 가변 VI 사용하기

가변 VI는 입력 데이터 타입에 맞도록 변경됩니다. 가변 VI의 모든 터미널은 다형성입니다. 터미널에 입력 데이터 타입을 연결할 때마다, 타입 전달이 실행되어 연결된 타입이 사용 가능한 타입인지 확인합니다. 타입 전달 에러가 감지되지 않으면, 가변 VI는 이 터미널을 입력 데이터 타입에 맞도록 변경됩니다. 예를 들어 [2D 배열 정렬]은 가변 VI입니다. 2D 배열 입력은 참조 번호를 제외하고 모든 데이터 타입의 배열을 받습니다. 인덱스 입력은 부호있는 32비트 정수 또는 부호있는 32비트 정수의 배열을 받습니다.

여러 데이터 타입에서 가변 VI를 사용하는 예제는 labview\examples\Malleable VIs\Basics\Malleable VIs Basics.lvproj를 참조하십시오.



관련 정보

[가변 VI](#) (136 페이지 참조)

[다형성 함수](#) (106 페이지 참조)

[타입 전달](#) (632 페이지 참조)

[다양한 클래스에서 가변 VI 사용하기](#) (137 페이지 참조)

[가변 VI와 다형성 VI 중 선택하기](#) (138 페이지 참조)

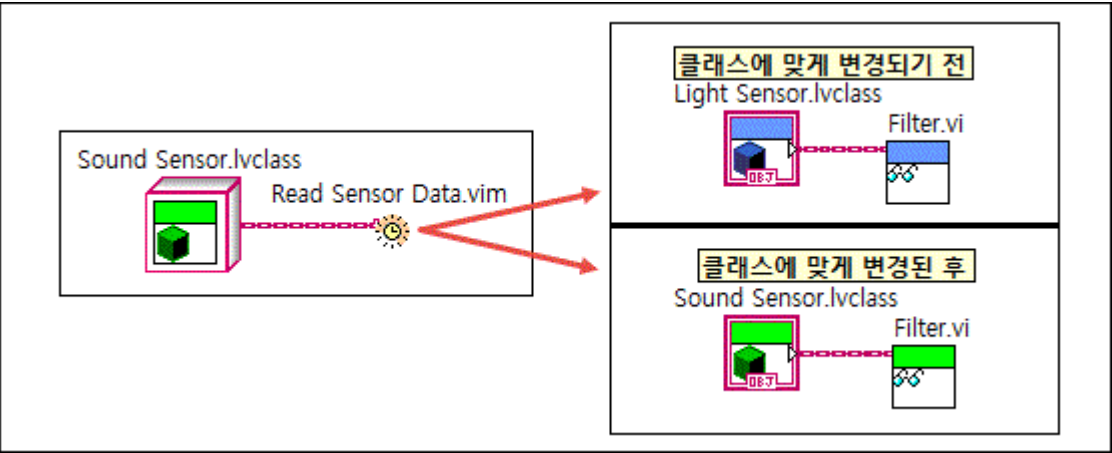
다양한 클래스에서 가변 VI 사용하기

가변 VI는 입력 클래스에 맞도록 변경됩니다. 클래스 A의 메소드를 호출하는 가변 VI는 클래스 A의 메소드와 동일한 이름, 커넥터 팬 패턴 및 터미널 방향을 가진 퍼블릭 메소드가 클래스에 포함되면 관련 없는 클래스의 데이터를 받아들입니다. 가변 VI는 클래스 A의 메소드에 대한 호출을 입력 클래스의 퍼블릭 메소드에 대한 호출로 대체하여 입력 클래스에 맞게 변경됩니다(그러한 메소드가 존재하는 경우).

다이나믹 디스패치 VI와 비교할 때, 가변 VI는 입력 클래스에 더 유연하게 대응합니다. 다이나믹 디스패치 VI는 VI의 클래스 또는 하위 클래스의 데이터만 받습니다. 가변 VI는 또한 경우에 따라 연관되지 않는 클래스의 데이터도 받을 수 있습니다.

가변 VI의 유연성은 메소드 이름, 커넥터 팬 패턴 및 터미널 방향이 있는 메소드 호출의 인터페이스를 정의합니다. 가변 VI는 메소드 호출의 인터페이스를 충족하는 모든 클래스의 데이터를 받아들입니다. 이로 인해 하나의 가변 VI를 사용하여 관련 없는 클래스의 데이터에 대한 연산을 수행하거나 하나의 클래스에서 다양한 가변 VI로 정의된 여러 인터페이스를 구현할 수 있습니다.

다음 그림에서 Read Sensor Data.vim은 Sound Sensor.lvclass와 공통 상위 데이터를 가지지 않는 Light Sensor.lvclass의 Filter.vi 메소드를 호출합니다. Read Sensor Data.vim은 Sound Sensor.lvclass가 Filter.vi와 같은 이름, 커넥터 팬 패턴 및 터미널 방향이 같은 퍼블릭 메소드를 가지고 있는 경우, Sensor.lvclass에 맞도록 변경합니다.



가변 VI를 사용하여 메소드 호출의 인터페이스를 정의하는 예제는 labview \examples\Malleable VIs\Class Adaptation\Malleable VIs - Class Adaptation.lvproj를 참조하십시오.

관련 정보

다양한 데이터 타입에서 가변 VI 사용하기

가변 VI와 다형성 VI 중 선택하기

가변 VI와 다형성 VI 중 선택하기

다양한 입력 데이터 타입에 가변 VI 또는 다형성 VI를 사용할 수 있습니다. 다음 테이블에서는 두 타입의 VI를 비교합니다.

외형	가변 VI	다형성 VI
구현	데이터 타입이 사용 가능한 타입인지 계산하는 단일 VI.	사용 가능한 데이터 타입 세트가 미리 지정된 인스턴스 VI의 모음.
커넥터 팬	모든 데이터 타입에서 커넥터 팬이 같음.	데이터 타입에 따라 커넥터 팬이 다를 수 있음.

외형	가변 VI	다형성 VI
빌드하기	.vim 파일 확장자를 가진 VI를 생성하거나 표준 VI의 파일 확장자를 .vim으로 변경하여 가변 VI로 변환합니다.	각 데이터 타입에 대해 인스턴스 VI를 생성하고 다형성 VI에 이 인스턴스 VI들을 포함시킵니다.
사용자 정의하기	타입 특수화 구조를 사용하여 특정 데이터 유형에 대한 가변 VI의 코드 섹션을 사용자 정의합니다.	특정 인스턴스 VI에서 코드를 변경하여 특정 데이터 타입에 대해 다르게 동작하도록 합니다.
클래스에 적합하도록 변경?	예 — 클래스 A의 메소드를 호출하는 가변 VI는 클래스 A의 메소드와 이름, 커넥터 팬 패턴 및 터미널 방향이 같은 퍼블릭 메소드가 있는 모든 클래스를 받습니다.	아니오 — 클래스 A의 메소드를 호출하는 다형성 VI는 클래스 A 또는 클래스 A의 하위 클래스만 받습니다.

다음 테이블을 사용하여 어떤 VI 타입을 선택할지 결정합니다.

타입	사용하는 경우
가변 VI	- 특정 데이터 타입을 위해 작업을 크게 바꾸지 않고도 여러 데이터 타입에서 작업이 비슷하게 구현될 수 있습니다. - VI가 입력 클래스에 적합하게 변경되기를 원합니다.
다형성 VI	- 작업이 데이터 유형에 따라 다르게 구현되어야 합니다. - 데이터 타입에 따라 커넥터 팬이 다릅니다.

VI 아이콘

아이콘 생성하기

아이콘을 생성하는 가장 기본적인 방법은 아이콘 편집기 대화 상자를 활용하는 것입니다. 또한, **파일 시스템에 있는 그래픽을 끌어와서 놓는 방법**도 있고 VI 클래스의 VI 아이콘 메소드를 사용하여 아이콘을 생성하는 방법도 있습니다.

아이콘 편집기 대화 상자로 아이콘 생성하기

아이콘 편집기 대화 상자를 사용하여 **아이콘을 생성 또는 편집**할 수 있습니다. 아이콘 편집기 대화 상자는 **아이콘 템플릿**, ni.com의 아이콘 라이브러리에 있는 **문양**, 아이콘 텍스트를 추가 및 포맷할 때의 선택 옵션, **레이어**를 통한 편집 지원 기능을 제공합니다.

VI 또는 사용자 정의 컨트롤의 아이콘을 편집하려는 경우에는 프런트패널 윈도우, 블록 다이어그램, 또는 **컨트롤 편집기**의 오른쪽 위에 있는 아이콘에서 마우스 오른쪽 버튼을 클릭하여 아이콘 편집기 대화 상자를 디스플레이합니다. 또한, 아이콘에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이콘 편집을 선택하여 아이콘 편집기 대화 상자를 디스플레이할 수 있습니다. **프로젝트 라이브러리**, statechart, **클래스**, 또는 **X컨트롤**의 아

이콘을 편집하려는 경우, 대응하는 프로퍼티 대화 상자의 일반 셋팅 페이지에서 편집 버튼을 클릭하여 아이콘 편집기 대화 상자를 디스플레이합니다.



노트

아이콘 편집기 대화 상자는 24 비트 색으로 아이콘을 디스플레이합니다. 하지만, 블록다이어그램에서 LabVIEW는 오직 8 비트 색으로 아이콘을 디스플레이합니다. 따라서, 이 대화 상자에서 생성하는 아이콘은 대응하는 블록다이어그램 아이콘에 다소 다르게 나타날 수 있습니다.

그래픽을 끌어와서 아이콘 생성하기

또한, **파일 시스템 상에 있는 그래픽을 끌어와서** 프론트패널 윈도우의 오른쪽 상단 모서리에 놓고 이 그래픽을 VI 아이콘으로 사용할 수 있습니다. 끌어와서 사용할 수 있는 파일에는 .png, .bmp, .jpg 파일이 있습니다.

LabVIEW는 사용자가 끌어 놓는 그래픽을 32 x 32 픽셀 아이콘으로 변환합니다. 32 x 32 픽셀보다 훨씬 큰 그래픽, 특히 더욱 정밀한 색상표를 사용하는 그래픽의 변환이 자연스럽게 지 않을 수도 있습니다.



노트 VI의 아이콘을 이보다 작게 만들면 VI를 선택하고 연결할 때 불편할 뿐만 아니라 보기에 좋지 않습니다. 아이콘 생성에 대한 추가적인 팁은 **LabVIEW Style Checklist**를 참조하십시오.

아이콘 디자인 가이드라인

아이콘에 그림과 텍스트를 모두 사용할 수 있습니다. 텍스트 아이콘의 크기 및 폰트로는 모두 대문자로 통일된 9 포인트 크기의 작은 폰트가 적절합니다.

어플리케이션을 현지화할 때에는 아이콘에 있는 텍스트도 현지화해야 합니다. 정확한 번역이 힘들기 때문에 아이콘을 만들 때 구어체를 사용하지 않는 것이 좋습니다. 또한, 다른 언어에서 특정 단어와 유사한 의미를 갖는 단어가 이미지와 대응하지 않는 경우에는 그 이미지를 해당 단어 대신에 사용하지 않을 것을 권장합니다.

배너를 사용하여 관련 VI를 식별할 수 있습니다. National Instruments는 **배너를 생성하고 템플릿으로 저장**하는 것을 권장합니다. 그러면 관련 VI 아이콘에서 이 템플릿을 사용할 수 있고, **이 VI 아이콘을 수정하여** 특정 VI에 대한 정보를 제공할 수 있습니다.

LabVIEW는 라이브러리 아이콘을 해당 라이브러리 안에 있는 모든 객체의 아이콘에 적용합니다. National Instruments는 **라이브러리 아이콘에서 오직 배너만을 생성할 것**을 권장합니다. 그러면, 라이브러리 안에 있는 모든 객체의 아이콘은 이 배너를 포함하게 되며, **아이콘을 수정**하여 특정 객체에 대한 정보를 제공할 수 있습니다.

예를 들어, 프로젝트 라이브러리 아이콘으로 다음과 같은 배너를 생성할 수 있습니다.



프로젝트 라이브러리 아이콘을 생성한 이후에는, 해당 프로젝트 라이브러리 내에 있는 VI에 대해 아이콘을 생성할 수 있습니다. 아이콘 편집 대화 상자에서 편집 >> 자신의 라이

브러리에서 아이콘 반입을 선택하여 소유 프로젝트 라이브러리의 아이콘을 반입합니다. 이렇게 하면 프로젝트 라이브러리가 NI Library라고 불리는 VI 아이콘의 사용자 레이어로 디스플레이됩니다. 같은 이름의 사용자 레이어가 이미 존재할 경우, 새 사용자 레이어가 기존의 사용자 레이어를 덮어씁니다.

아이콘 편집기 대화 상자의 오른쪽에 있는 아이콘 텍스트, 문양, 편집 도구를 사용하여 VI 아이콘을 수정합니다. 예를 들어, 다음 아이콘과 유사한 VI 아이콘을 생성할 수 있습니다.



초록색 MY LIB 배너를 통해 해당 VI가 프로젝트 라이브러리의 일부라는 것을 구별할 수 있습니다.

아이콘 사용하기

아이콘은 VI 또는 프로젝트 라이브러리를 그래픽으로 표현한 것입니다.

모든 VI는 프런트패널과 블록다이어그램 윈도우 오른쪽 위 코너에 다음에 보이는 것처럼 아이콘을 디스플레이합니다.



기본 VI 아이콘은 LabVIEW를 실행한 이후로 얼마나 많은 새 VI를 열었는지, 최대 9 개의 VI를 숫자로 나타냅니다. 이 숫자 표시를 비활성화하려면, 도구 >> 옵션 >> 프런트패널을 선택한 후 새 VI의 아이콘에 숫자(1-9) 사용 확인란에서 확인 표시를 제거합니다.

아이콘에는 텍스트나 이미지를 사용할 수 있습니다. VI를 SubVI로 사용하는 경우, 블록다이어그램에서는 아이콘으로 해당 SubVI를 식별할 수 있습니다. VI를 팔레트에 추가하는 경우, VI 아이콘은 함수 팔레트에도 나타납니다.

아이콘 편집기 대화 상자를 사용하여 VI 아이콘을 편집할 수 있습니다. 프런트패널 또는 블록다이어그램 윈도우의 오른쪽 위 모서리 부분에 있는 아이콘에서 더블 클릭하여, 아이콘 편집기 대화 상자를 디스플레이합니다.

배너를 사용하여 관련 VI를 식별할 수 있습니다. National Instruments는 배너를 생성하고 템플릿으로 저장하는 것을 권장합니다. 일단 템플릿으로 저장해 놓은 후에는, 템플릿을 사용하여 관련된 VI의 아이콘의 모양을 수정함으로써 특정 VI에 대한 정보를 전달할 수 있습니다.

사용자 컨트롤을 위한 아이콘은 VI의 아이콘과 유사합니다. 다음에 보이는 것처럼, 각각의 사용자 컨트롤은 컨트롤 편집기 윈도우의 오른쪽 위 모서리에 아이콘을 디스플레이합니다.



컨트롤 편집기 윈도우의 오른쪽 위 모서리에 있는 아이콘에서 더블 클릭하여 아이콘 편집기를 디스플레이하면, 이를 사용하여 아이콘을 편집할 수 있습니다.

프로젝트 라이브러리의 아이콘은 vi나 사용자 컨트롤의 아이콘과는 다릅니다. LabVIEW는 라이브러리 아이콘을 해당 라이브러리 안에 있는 모든 객체의 아이콘에 적용합니다. National Instruments는 라이브러리 아이콘에서 오직 배너만을 생성할 것을 권장합니다. 예를 들어, 프로젝트 라이브러리 아이콘으로 다음과 같은 배너를 생성할 수 있습니다.

MY LIB

그러면, 라이브러리 안에 있는 모든 객체의 아이콘은 이 배너를 포함하게 되며, 아이콘을 수정하여 특정 객체에 대한 정보를 제공할 수 있습니다.

프로젝트 라이브러리 프로퍼티 대화 상자의 일반 셋팅 페이지를 사용하여 대응하는 프로젝트 라이브러리의 아이콘을 편집하거나 봅니다.

statechart, 클래스, x컨트롤의 아이콘은 프로젝트 라이브러리 아이콘과 유사합니다. 프로젝트 라이브러리 프로퍼티 대화 상자, 클래스 프로퍼티 대화 상자 또는 x컨트롤 프로퍼티 대화 상자의 일반 셋팅 페이지를 사용하여 각각 대응하는 statechart, 클래스 또는 x컨트롤의 아이콘을 보거나 편집합니다.



관련 정보

[LabVIEW 프로젝트에서 라이브러리 사용하기](#) (296 페이지 참조)

[함수 또는 컨트롤 팔레트에서 vi나 객체 추가하기](#) (794 페이지 참조)

[vi 아이콘 생성하기](#) (1357 페이지 참조)

[vi 아이콘 생성하기](#) (1357 페이지 참조)

[vi 아이콘 생성하기](#) (1357 페이지 참조)

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

[LabVIEW 프로젝트에서 라이브러리 사용하기](#) (296 페이지 참조)

[라이브러리 아이콘 생성하기](#) (1354 페이지 참조)

[라이브러리 아이콘 생성하기](#) (1354 페이지 참조)

[라이브러리 아이콘 생성하기](#) (1354 페이지 참조)

[LabVIEW 클래스 생성하기](#) (520 페이지 참조)

[x컨트롤 사용하기](#) (101 페이지 참조)

아이콘 저장하기

LabVIEW는 [레이어 정보](#)를 포함하여 [아이콘](#)을 이에 대응하는 vi 또는 라이브러리와 함께 저장합니다. 아이콘 편집기 대화 상자는 아이콘을 256 색 (8 비트) 및 흑백 (1 비트) 포맷으로 저장할 수 있습니다.

아이콘 편집기 대화 상자에서 생성하는 아이콘을 추후에 사용할 수 있도록 템플릿 또는 문양으로 저장할 수 있습니다. 파일 >> 다른 이름으로 저장 >> 템플릿 또는 파일 >> 다른 이름으로 저장 >> 문양을 선택하여, 다른 이름으로 아이콘 저장 대화 상자를 디스플레이하

고 아이콘을 저장할 위치를 지정할 수 있습니다. 아이콘 템플릿은 반드시 LabVIEW Data\Icon Templates 디렉토리 안에 저장해야 합니다. 문양은 LabVIEW Data\Glyphs 디렉토리에 저장해야 합니다. LabVIEW는 아이콘 편집기 대화 상자에서 생성한 아이콘 템플릿과 문양을 256색 .png 파일로 저장합니다.

타사의 VI를 열 때, 이 VI에 대응하는 아이콘 템플릿을 추후에 사용하기 위해 템플릿으로 저장할지의 여부를 선택할 수 있습니다. 타사 아이콘 템플릿 저장을 위한 설정을 설정하려면, 아이콘 편집기 프로퍼티 대화 상자의 템플릿 페이지를 사용하십시오.

문양 사용하기

아이콘 편집기 대화 상자를 사용하여 문양을 생성하거나 **아이콘**에 문양을 추가할 수 있습니다. 아이콘 편집기 대화 상자의 문양 페이지는 LabVIEW Data\Glyphs 디렉토리에 있는 .png, .bmp, .jpg 파일을 모두 디스플레이합니다. 이 디렉토리에 사용자가 생성한 문양을 저장하여 아이콘 편집기 대화 상자의 문양 페이지에 나타나도록 할 수 있습니다. 기본 설정에서 문양 페이지는 ni.com의 아이콘 라이브러리에 있는 모든 문양을 포함합니다. 도구>>ni.com 아이콘 라이브러리에 동기화를 선택하면, 아이콘 라이브러리에 동기화 대화 상자를 디스플레이하고 아이콘 라이브러리에 있는 최신 문양으로 LabVIEW Data\Glyphs 디렉토리를 동기화할 수 있습니다.

도구>>문양 및 아이콘 템플릿 열거를 선택하여 사용 가능한 모든 아이콘 템플릿과 문양 및 디스크에서 해당 파일의 위치를 디스플레이할 수 있습니다.

편집>>파일로부터 문양 반입을 선택하여 특정한 문양을 찾고 반입할 수 있는 파일 대화 상자를 디스플레이합니다.

아이콘 템플릿 사용하기

아이콘 편집기 대화 상자를 사용하여 **아이콘** 템플릿을 생성 또는 편집할 수 있습니다. 아이콘 편집기 대화 상자의 템플릿 페이지는 LabVIEW Data\Icon Templates 디렉토리에 있는 .png, .bmp, .jpg 파일을 모두 디스플레이합니다. 이 디렉토리에 사용자가 생성한 아이콘 템플릿을 저장하여 아이콘 편집기 대화 상자의 템플릿 페이지에 나타나도록 할 수 있습니다. LabVIEW Data\Icon Templates\VI 디렉토리에 있는 템플릿은 VI와 사용자 컨트롤에서 사용합니다. LabVIEW Data\Icon Templates\Library 디렉토리에 있는 템플릿은 프로젝트 라이브러리, 클래스, X컨트롤에서 사용합니다.



노트

템플릿을 사용하지 않는 경우, 아이콘 둘레에 경계선 추가하는 것이 좋습니다. 아이콘 편집기 대화 상자의 오른쪽에 있는 사각형 도구에서 더블 클릭하여, 라인 컬러에 있는 아이콘에 1 픽셀 두께의 경계선을 추가할 수 있습니다.

디스크 상에서 아이콘 템플릿이 변경되는 경우, 이후에 사용자가 아이콘 편집기 대화 상자에서 템플릿을 참조하는 아이콘을 디스플레이할 때 자동으로 업데이트됩니다.

타사의 VI를 열 때, 이 VI에 대응하는 아이콘 템플릿을 추후에 사용하기 위해 템플릿으로 저장할지의 여부를 선택할 수 있습니다. 타사 아이콘 템플릿 저장을 위한 설정을 설정하려면, 아이콘 편집기 프로퍼티 대화 상자의 템플릿 페이지를 사용하십시오.

도구»문양 및 아이콘 템플릿 열거를 선택하여 사용 가능한 모든 아이콘 템플릿과 문양 및 디스크에서 해당 파일의 위치를 디스플레이할 수 있습니다.

편집»파일로부터 문양 반입을 선택하여, 템플릿이나 아이콘으로 사용할 수 있는 문양을 찾고, 반입할 수 있는 대화 상자를 디스플레이합니다.

레이어 사용하기

레이어를 사용하여 **아이콘**에서 서로 겹치는 부분을 개별적으로 변경할 수 있습니다. LabVIEW는 아이콘의 레이어 정보를 이에 대응하는 VI 또는 라이브러리와 함께 저장합니다.

아이콘 편집기 대화 상자의 레이어 페이지를 사용하여, 아이콘의 레이어 이름, 불투명도, 보이는 정도, 순서를 설정할 수 있습니다. 레이어 페이지를 디스플레이하려면, 아이콘 편집기 대화 상자에서 레이어»레이어 페이지 보이기를 선택하십시오.

아이콘 편집기 대화 상자의 레이어 페이지는 3개의 섹션으로 나누어져 있습니다. 첫번째 섹션은 아이콘 텍스트 레이어를 포함합니다. 이 레이어는 아이콘 편집기 대화 상자의 아이콘 텍스트 페이지에 사용자가 입력한 모든 텍스트를 나타냅니다. 아이콘 텍스트 레이어는 아이콘에서 맨위에 덮어쓰는 레이어입니다.

레이어 페이지의 두번째 섹션은 모든 사용자 레이어를 포함합니다. 아이콘 편집기 대화 상자는 아이콘에 문양을 추가할 때마다, 그리고 대화 상자 오른쪽에 있는 편집 도구 사용하여 일련의 편집 내용이 적용될 때마다 새 사용자 레이어를 하나씩 생성합니다. 레이어»새 레이어 생성을 선택하면, 편집 도구를 연속적으로 사용하여 일련의 편집 작업을 수행하는 도중에 새 사용자 레이어를 생성할 수 있습니다.

레이어 페이지에서 미리 보기를 선택하거나 아이콘 편집기 대화 상자의 미리 보기 영역에 있는 레이어를 이동 도구를 통해 선택하여, 해당 사용자 레이어를 선택할 수 있습니다. 레이어를 선택하면, 나머지 아이콘은 비활성화되어 선택한 레이어를 쉽게 구분할 수 있습니다. 레이어를 선택한 후에 아이콘 편집기 대화 상자 오른쪽에 있는 편집 도구를 사용하여, 다른 레이어와는 별도로 선택한 레이어를 변경할 수 있습니다.

또한, 사용자 레이어의 순서를 변경할 수 있습니다. 1 개 이상의 사용자 레이어를 선택한 후에 위로 이동 또는 아래로 이동 버튼을 클릭하여 레이어 순서를 바꿀 수 있습니다. 레이어 메뉴에 있는 옵션을 사용하여, 사용자가 선택한 사용자 레이어를 통합하거나 아이콘에 있는 모든 사용자 레이어를 통합할 수 있습니다. 아이콘에 있는 사용자 레이어를 모두 삭제하려면, 편집»사용자 레이어 지우기를 선택합니다. 아이콘 편집기 프로퍼티 대화 상자의 레이어 페이지에 있는 적용시 모든 레이어 병합 확인 상자에 확인 표시를 해서, 아이콘 편집기 대화 상자를 닫을 때 모든 레이어를 통합시킬 수 있습니다. LabVIEW가 아이콘의 레이어 정보를 관련 VI 및 라이브러리와 함께 저장하기 때문에, 아이콘 레이어를 병합하면 디스크 상에서 VI 또는 라이브러리의 크기를 줄일 수 있습니다.

레이어 페이지의 세번째 섹션은 아이콘 템플릿 레이어를 포함합니다. 이 레이어는 아이콘 편집기 대화 상자의 템플릿 페이지에서 사용자가 선택한 아이콘 템플릿을 나타냅니다. 아이콘 템플릿 레이어는 아이콘에서 맨 아래에 있는 레이어입니다.

레이어 페이지는 각 레이어의 미리 보기, 이름, 불투명도, 보이는 정도를 디스플레이합니다. 아이콘 텍스트나 아이콘 템플릿 레이어의 이름은 바꿀 수 없습니다. 하지만, 모든 사용자 레이어의 이름은 변경할 수 있습니다. 문양 레이어의 기본 이름은 해당 문양의 파일 이름에서 확장자명을 제외한 부분입니다. 편집 도구의 사용에 따라 생성되는 레이어의 기본 이름은 도구입니다.

레이어의 파란색 불투명도 슬라이더를 클릭하고 움직이면서 레이어를 더 투명하게 또는 더 불투명하게 만들 수 있습니다. 레이어를 숨기거나 보이게 하려면 보이기? 아이콘  을 클릭하십시오.

VI 저장하기

파일 » 저장을 선택하여 VI를 저장합니다. VI를 개별 파일로 저장하거나 몇 개의 VI를 그룹화하여 LLB에 저장할 수 있습니다. LLB 파일은 .llb의 확장자로 끝납니다. National Instruments는 특히 많은 개발자들이 같은 프로젝트를 작업하는 경우, VI를 디렉토리 안에 개별적인 파일로 저장하는 것을 권장합니다.

National Instruments는 **특정 디렉토리**에 사용자 VI를 저장하는 것을 권장합니다. VI에 대한 접근성을 높이려면, 프로젝트에서 사용하는 디렉토리의 개수와 레벨을 제한하십시오.

VI를 개별 파일로 저장할 때의 장점

다음 리스트는 VI를 개별 파일로 저장해야 하는 이유를 설명합니다:

- 파일 시스템을 사용하여 개별 파일을 관리할 수 있습니다.
- 서브디렉토리를 사용할 수 있습니다.
- VI와 컨트롤을 개별 파일로 저장하는 것이 전체 프로젝트를 동일한 파일에 저장하는 것보다 안전합니다.
- 타사 소스 컨트롤 제공자를 사용할 수 있습니다.

하나의 어플리케이션에서 사용하는 모든 VI에 대한 디렉토리를 생성하여, Main VI를 이 디렉토리 안에 저장하고 SubVI는 서브디렉토리에 저장합니다. SubVI가 다른 SubVI를 보유하고 있는 경우, 해당 디렉토리에 대한 하위 계층구조를 만드는 방식으로 만들어갑니다. 디렉토리를 생성할 때, SubVI의 기능에 따라 모듈화된 구조로 VI와 SubVI를 구성합니다.

VI를 LLB로 저장할 때의 장점

다음 리스트는 VI를 LLB로 저장해야 하는 이유를 설명합니다:

- 파일 이름에 최대 255개 문자를 사용할 수 있습니다.
- LLB를 전송할 수 있는 속도보다 더 쉽게 여러 개별 VI를 다른 플랫폼으로 전송 수 있습니다. 또한 사용자가 모든 필요한 파일을 받을 수 있습니다.
- LLB는 디스크 공간을 줄이기 위해서 압축되기 때문에 프로젝트의 파일 크기를 약간 줄일 수 있습니다.

- LLB에 VI를 최상위로 표시할 수 있기 때문에 LLB를 열 때, LabVIEW는 자동으로 해당 LLB의 모든 최상위 VI를 엽니다.



노트

LabVIEW는 많은 내장 VI와 예제를 LLB에 보관하여 모든 플랫폼에서 일관된 저장 위치를 갖도록 합니다.

LLB를 사용할 경우, 어플리케이션을 여러 LLB로 나누는 것을 고려합니다. 최상위 VI를 하나의 LLB에 넣고 다른 LLB에는 함수에 의해 분리되는 VI를 넣습니다. OS가 더 큰 파일에 변화를 저장해야만 하기 때문에, 변화를 개별 VI에 저장하는 것보다 LLB에 있는 VI에 변화를 저장하는 것이 시간이 더 걸립니다. 큰 LLB에 변화를 저장하면 메모리가 많이 필요하고 성능이 떨어집니다. 각 LLB의 크기를 약 1MB로 제한합니다.

LLB 안의 VI 관리하기

도구>>LLB 관리자를 선택하여 이용할 수 있는 LLB 관리자 윈도우를 사용하여 LLB 안의 파일을 쉽게 복사하고, 이름을 변경하고, 삭제할 수 있습니다. 또한, 이 도구를 사용하여 새 LLB와 디렉토리를 생성하고 LLB를 디렉토리로 변환하거나 디렉토리를 LLB로 변환할 수 있습니다. 새로운 LLB와 디렉토리를 생성하고 LLB를 디렉토리로 또는 그 반대로 변환하는 것은 소스 컨트롤 도구를 사용하여 VI를 관리하려는 경우 중요합니다.

LLB 관리자 윈도우를 사용하기 전에, 이미 메모리에 있는 VI에 대한 파일 작업 수행을 피하는데 영향을 줄 수 있는 모든 VI를 닫습니다.



노트 LLB 관리자 윈도우에서 작업한 변경 내용은 취소할 수 없습니다.

(Windows) 또한 Windows 탐색기의 .llb 파일을 더블 클릭하고 LLB 관리자 윈도우를 디스플레이하여 LLB의 파일을 열고, 이동하고, 복사하고, 이름을 바꾸고, 삭제할 수 있도록 합니다.

VI 이름 붙이기

VI를 저장할 때, 설명적인 이름을 사용합니다. Temperature Monitor.vi와 Serial Write & Read.vi와 같은 설명적인 이름은 VI와 사용법의 식별을 쉽게 합니다. VI_1.vi와 같이 불분명한 이름을 사용할 경우, 특히 여러 VI를 저장한 경우 VI를 식별하는 것이 힘들 것입니다.

사용자가 다른 플랫폼에서 VI를 사용할지를 고려합니다. \ : / ? * < > #과 같이 일부 OS에서 특별한 경우에 사용하는 문자의 사용은 피합니다.



노트 컴퓨터에 같은 이름으로 저장된 VI가 여러 개 있을 경우, LabVIEW가 최상위 레벨 VI를 실행할 때 잘못된 SubVI를 참조하지 않도록 주의 깊게 VI를 다른 **프로젝트 라이브러리**나 LLB에 구성합니다. VI, LLB, 디렉토리에 이름을 붙이는 방법에 대한 추가적인 팁은 **LabVIEW Style Checklist**를 참조하십시오.

이전 버전으로 저장하기

LabVIEW의 이전 버전으로 VI, LabVIEW 프로젝트 및 프로젝트 라이브러리를 저장하여 LabVIEW를 편리하게 업그레이드하고 필요한 경우 하나 이상의 LabVIEW 버전으로 파일을 관리할 수 있습니다. 파일 >> 이전 버전으로 저장을 선택하여 LabVIEW의 **이전 버전으로 저장**합니다.



노트 이전 LabVIEW 버전으로 **묶음 프로젝트 라이브러리**를 저장하려면, 묶음 라이브러리에서 이전 버전으로 파일을 저장한 후, 원하는 LabVIEW 버전으로 **묶음 라이브러리를 다시 만듭니다..**

VI를 이전 버전으로 저장할 때, LabVIEW는 VI 뿐만 아니라 labview\vi.lib 디렉토리의 파일을 제외한 계층구조의 모든 VI를 변환합니다.

VI, 프로젝트 또는 프로젝트 라이브러리가 이전 버전에는 없는 최신 기능을 사용하는 경우, LabVIEW는 가능한 많은 것을 저장하고 변환할 수 없는 부분에 대해서는 리포트를 생성합니다. 리포트는 이전 버전으로 저장 경고 요약 대화 상자에 즉시 나타납니다. 이전 버전으로 저장 경고 요약 대화 상자에 있는 세부사항 보이기 버튼을 클릭하여, 로드 및 저장 경고 리스트 대화 상자를 열고 각 경고에 대한 세부사항을 확인합니다. 또한, 무시 버튼을 클릭하여 경고를 확인하고 요약 대화 상자를 닫을 수 있습니다. 로드 및 저장 경고 리스트 대화 상자에 있는 파일에 저장 버튼을 클릭하여 나중에 볼 수 있도록 텍스트 파일에 경고를 저장합니다.

이전 LabVIEW 버전에서 암호로 보호된 VI를 저장하려면, 암호를 입력해야 합니다. 또한, [VI 참조 열기] 함수의 입력으로 프로그램적으로 암호를 입력할 수 있습니다.

복구를 위한 자동 저장

비정상적으로 종료되거나 시스템 실패의 경우, LabVIEW는 종료 또는 실패 시 열려있던 모든 변경된 VI(.vi), VI 템플릿(.vit), 컨트롤(.ctl), 컨트롤 템플릿(.ctt), 프로젝트(.lvproj), 프로젝트 라이브러리(.lvlib), X컨트롤(.xctl), 또는 LabVIEW 클래스(.lvclass) 파일을 임시 장소에 백업합니다.



노트 자동 저장 시 VI 파일에 텍스트를 입력하던 도중이었던 경우, LabVIEW는 해당 파일을 저장하지 않습니다. 이는 자동 저장이 텍스트 입력을 중단시키는 것을 방지합니다.

도구 >> 옵션을 선택하고 항목 리스트에서 환경을 선택하여 복구를 위한 저장을 활성화 또는 비활성화할 수 있고, LabVIEW가 얼마나 자주 파일을 백업하는지 지정할 수 있습니다. 자동 저장을 활성화하면, LabVIEW는 VI를 실행하기 전에 파일을 백업합니다. 또한 LabVIEW가 주기적인 간격으로 파일을 백업하도록 설정할 수 있습니다.

LabVIEW는 백업된 파일을 **기본 데이터 디렉토리**의 LVAutoSave서브디렉토리에 저장합니다. 수동으로 LVAutoSave 디렉토리 파일에 접근하거나 수정하지 마십시오.

LabVIEW가 비정상적인 종료나 시스템 실패 전에 자동으로 파일을 저장하는 경우, LabVIEW를 다시 시작할 때 복구할 파일 선택 윈도우가 나타납니다. 복구하려는 파일을 선택한 후 복구 버튼을 클릭합니다. 파일을 복구하지 않으려는 경우, 모든 파일의 선택을 해제하고 버림 버튼을 클릭합니다. 취소 버튼을 클릭하면 모든 선택된 파일이 기본 데이

터 디렉토리의 LVAutoSave\archives 서브디렉토리로 이동됩니다. 백업 파일이 더 이상 필요하지 않게 되면 이 서브디렉토리의 내용을 삭제할 수 있습니다.



노트 복구할 프로젝트나 라이브러리를 선택해도 해당 프로젝트나 라이브러리에 포함된 파일의 변경사항은 자동으로 복구되지 않습니다. 변경된 프로젝트 또는 라이브러리 및 모든 변경된 사항을 복구하려면 현재 .lvproj 또는 .lvlib 파일 및 프로젝트 또는 라이브러리에 포함된 각 파일 옆에 꼭 확인 표시를 하십시오.

자동적으로 저장된 파일은 수동적으로 저장된 기존 파일을 덮어쓰지 않습니다. LabVIEW는 사용자가 수동으로 파일을 저장, 삭제, 기록할 때까지 자동적으로 저장된 파일을 유지합니다.

VI 사용자 정의하기

어플리케이션에 맞춰서 사용하기 위해서 VI와 SubVI를 설정할 수 있습니다. 예를 들어, 사용자 입력을 요구하는 VI를 SubVI로 사용하려는 경우, VI를 설정하여 호출될 때마다 프런트패널이 나타나도록 합니다.

VI 자체적으로 또는 **VI 서버**를 사용한 프로그램적인 방법으로, VI를 다양하게 설정할 수 있습니다.

VI의 모양과 동작 설정하기

파일>>VI 프로퍼티를 선택하여 VI의 모양과 특성을 설정합니다. VI 프로퍼티 대화 상자 상단의 항목 풀다운 메뉴를 사용하여 여러 다른 옵션 항목 중 하나를 선택합니다.

메뉴 사용자 정의하기

사용자 메뉴는 어느 VI에서나 만들 수 있으며, VI 설정에 따라 메뉴 모음을 보이거나 숨기도록 할 수 있습니다. 파일>>VI 프로퍼티를 선택하고 항목 풀다운 메뉴에서 윈도우 모양을 선택한 후 사용자 정의 버튼을 클릭하고 메뉴 모음 보이기 확인란에 확인 표시를 하거나 제거합니다.

메뉴 설정하기는 메뉴의 생성과 사용자가 다양한 메뉴 아이템을 선택할 때 실행되는 블록 다이어그램 코드의 제공을 포함합니다.



노트

사용자 메뉴는 VI가 실행되는 동안에만 나타납니다.

메뉴 만들기

VI를 편집할 때 정적으로 사용자 메뉴를 만들거나 또는 기본 LabVIEW 메뉴를 수정할 수 있으며, 또는 VI를 실행할 때 프로그램식으로 사용자 메뉴를 만들거나 또는 기본 LabVIEW 메뉴를 수정할 수 있습니다.

기본 메뉴 모음이 아닌 VI에 사용자 메뉴 모음을 추가하려면, 편집>>런타임 메뉴를 선택하고 메뉴 편집기 대화 상자에 메뉴를 생성합니다. LabVIEW는 런타임 메뉴 .rtm 파일을 생성합니다. .rtm 파일을 생성하고 저장한 후에는 반드시 VI와 .rtm 파일간에 동일한 상대 경로를 유지해야만 합니다. 또한, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 고급>>런타임 바로 가기 메뉴>>편집을 선택하여 사용자 런타임 바로 가기 메뉴를 생성할 수 있습니다. 이 옵션은 바로 가기 메뉴 편집기를 엽니다.

메뉴 편집기와 바로 가기 메뉴 편집기 대화 상자를 사용하여 LabVIEW가 기본 메뉴에 제공하는 메뉴 아이템인 어플리케이션 아이템으로 사용자 메뉴를 만들거나 또는 사용자가 추가하는 메뉴 아이템인 사용자 아이템으로 사용자 아이템을 만들 수 있습니다. 어플리케이션 아이템의 특성은 LabVIEW가 정의하지만, 사용자 아이템의 특성은 블록 다이어그램에서 사용자가 제어합니다. 또한, 메뉴 편집기와 바로 가기 메뉴 편집기 대화 상자를 사용하여 사용자 .rtm 파일을 VI 또는 컨트롤에 연계할 수 있습니다. VI가 실행될 때, VI는 .rtm 파일에서 메뉴를 로드합니다.

메뉴 편집기와 바로 가기 메뉴 편집기 대화 상자를 사용하여 VI를 편집할 때 메뉴를 사용자 정의합니다. 메뉴 함수를 사용하여 실행 중에 프로그램적으로 메뉴를 사용자 정의합니다. 함수는 사용자 아이템의 속성을 삽입, 삭제, 수정할 수 있도록 허용합니다. LabVIEW가 어플리케이션 아이템의 특성과 상태를 정의하기 때문에 어플리케이션 아이템은 추가하거나 삭제만 할 수 있습니다.

실행 모드의 바로 가기 메뉴

VI가 실행 중이거나, 또는 실행 모드에 있을 때, 모든 프런트패널의 객체는 기본적으로 바로 가기 메뉴 아이템의 축소된 세트를 갖습니다. 축소된 바로 가기 메뉴 아이템으로 객체의 내용을 자르고, 복사하고, 붙여넣고, 객체의 기본값으로 설정하거나 객체의 설명을 읽습니다.



노트 실행 모드에서 인디케이터의 데이터를 다시 초기화하거나 잘라내기, 붙여넣기할 수 없습니다. 기본값으로 다시 초기화, 데이터 잘라내기, 데이터 붙여넣기 옵션은 실행 모드의 컨트롤에서만 사용 가능합니다.

일부 복잡한 기능의 컨트롤은 추가 옵션을 갖습니다. 예를 들어, 노브 바로 가기 메뉴는 지침에 추가하고 스케일 마커의 디스플레이를 변경하는 아이템을 포함합니다.

VI에 포함하는 각 컨트롤에 대해 런타임 바로 가기 메뉴를 **사용자 정의**할 수 있습니다. 바로 가기 메뉴를 사용자 정의 하려면, 컨트롤에서 오른쪽 마우스 버튼을 클릭하고 바로 가기 메뉴에서 고급>>런타임 바로 가기 메뉴>>편집을 선택하여 바로 가기 메뉴 편집기 대화 상자를 디스플레이합니다. 바로 가기 메뉴 편집기 대화 상자를 사용하여 기본 바로 가기 메뉴나 컨트롤을 가진 **사용자 정의된 바로 가기 메뉴 파일(.rtm)**을 연결합니다. 바로 가기 메뉴를 **프로그램적으로** 사용자 정의할 수 있습니다.

또한, 프런트패널에 **바로 가기 메뉴를 추가**할 수 있습니다. 프런트패널에 바로 가기 메뉴를 추가하기 위해, 바로 가기 메뉴 활성화와 바로 가기 메뉴 선택 팬 이벤트를 사용합니다.

또한, 컨트롤의 런타임 바로 가기 메뉴를 **비활성화**할 수 있습니다.



노트 사용자 런타임 바로 가기 메뉴는 VI가 실행될 때만 나타납니다.

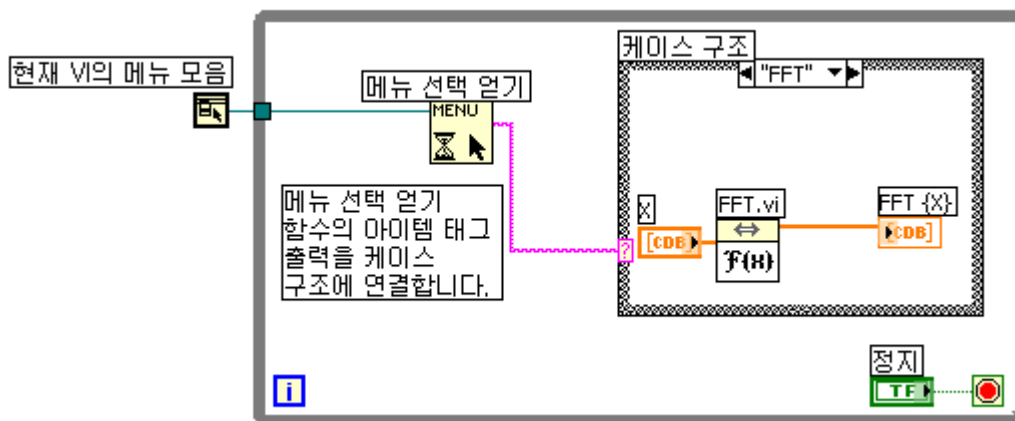
또한 마우스 다운? 및 바로 가기 메뉴 활성화? 필터 이벤트를 사용하여 컨트롤에서 마우스 왼쪽 버튼을 클릭할 때 바로 가기 메뉴를 디스플레이할 수 있습니다. 이 동작을 수행하려면, 마우스 다운? 필터 이벤트의 버튼 이벤트 데이터 필드가 반환하는 데이터를 수정합니다. 왼쪽 마우스 버튼의 값은 1이고, 오른쪽 마우스 버튼의 값은 2입니다. 컨트롤에서 마우스 왼쪽 버튼을 클릭해서 바로 가기 메뉴를 디스플레이하려면, 버튼 이벤트 데이터 필드를 2로 변경하여 LabVIEW가 마우스 왼쪽 버튼을 클릭하는 것을 오른쪽 버튼 클릭을 클릭하는 것처럼 취급하도록 합니다.

메뉴 선택 처리하기

사용자 메뉴를 생성할 때, 각 메뉴 아이템에 태그라 불리는 대소문자 구분 없는 유일한 문자열 식별자를 지정합니다. 사용자가 메뉴 아이템을 선택할 때, [메뉴 선택 열기] 함수를 사용하여 프로그램적으로 해당 태그를 가져옵니다. LabVIEW는 블록다이어그램에서 각 메뉴 아이템의 태그 값을 기반으로 각 메뉴 아이템에 대한 핸들러를 제공합니다. 핸들러는 While 루프와 케이스 구조가 조합된 것으로, 어떤 메뉴가 선택되었는지 확인하고 적절한 코드를 실행하게 합니다.

사용자 메뉴를 만든 후, 사용자 메뉴의 각 아이템을 실행하거나 처리하는 케이스 구조를 블록다이어그램에 만듭니다. 이 과정은 메뉴 선택 처리하기로 불립니다. LabVIEW는 모든 어플리케이션 아이템을 내부적으로 처리합니다.

다음 블록다이어그램에서, [메뉴 선택 열기] 함수는 사용자가 선택하는 메뉴 아이템을 읽고 메뉴 아이템이 실행되는 케이스 구조에 그 메뉴 아이템을 전달합니다.



수행 시간이 긴 메뉴 아이템이 있다면, [메뉴 선택 열기] 함수의 메뉴 차단 입력에 불리언 컨트롤을 연결하고 그 불리언 컨트롤을 참으로 설정하여 LabVIEW가 메뉴 아이템을 처리 중일 때 사용자가 다른 메뉴를 선택하지 못하도록 메뉴 모음을 비활성화시킵니다. LabVIEW가 메뉴 아이템을 처리한 후 [메뉴 추적 활성화] 함수에 참값을 연결하여 메뉴 모음을 활성화합니다.

또한, **이벤트 구조를 사용**하여 메뉴 이벤트를 처리할 수 있습니다.

VI 실행하기 및 디버깅하기

VI를 실행하려면 모든 SubVI, 함수, 구조를 터미널의 올바른 데이터 타입에 맞게 연결해야 합니다. 때로는 VI가 예상치 않은 방식으로 데이터를 생성하거나 실행될 수 있습니다. LabVIEW를 사용하여 VI 실행 방법을 설정하고 블록다이어그램의 구성이나 블록다이어그램을 통과하는 데이터에서 문제점을 찾습니다.



관련 정보

[VI 실행하기](#) (151 페이지 참조)

[깨진 VI 수정하기](#) (262 페이지 참조)

[VI 개정 히스토리 설정하기](#) (152 페이지 참조)

[디버깅 기법](#) (254 페이지 참조)

[에러 핸들링](#) (263 페이지 참조)

[VI 실행하기](#) (903 페이지 참조)

VI 실행하기

VI를 실행하면 VI를 디자인한 대로 작업을 수행합니다. 다음과 같이 도구 모음의 실행 버튼이 굵은 흰색 화살표로 나타나면 VI를 실행할 수 있습니다.



또한 굵은 흰색 화살표는 해당 VI에 [커넥터 팬을 생성](#)하는 경우 이 VI를 [SubVI](#)로 사용할 수 있음을 나타냅니다.

블록다이어그램 도구 모음의 실행 또는 연속 실행 버튼을 클릭하거나 단계별 실행 버튼을 클릭하면 VI가 실행됩니다. VI가 실행되는 동안 실행 버튼은 다음과 같이 회색 화살표로 바뀌어 VI가 실행되고 있음을 나타냅니다.



VI가 실행 중일 때는 VI를 변경할 수 없습니다.

실행 버튼을 클릭하면 VI를 한 번 실행합니다. VI가 데이터 흐름을 완료하면 VI는 정지합니다. 다음과 같이 연속 실행 버튼을 클릭하면 수동으로 정지시킬 때까지 VI를 연속적으로 실행합니다.



단계별 실행 버튼을 클릭하면 단계별로 VI를 실행합니다.



노트

실행 강제 종료 버튼 은 VI가 현재 반복을 마치기 전이라도 VI를 즉시 정지시킵니다. 외부 하드웨어와 같은 외부 리소스를 사용하는 VI를 강제 종료하는 경우, 해당 외부 리소스를 적절히 리셋 혹은 해제하지 못하여 리소스가 알지 못하는 상태가 될 수 있습니다. VI에 정지 버튼을 만들면 이러한 문제를 피할 수 있습니다.

VI 실행 방법 설정하기

파일>>VI 프로퍼티를 선택하고 항목 풀다운 메뉴에서 실행을 선택하여 **VI 실행 방법을 설정**합니다. 예를 들어, VI를 열면 즉시 실행되도록 설정하거나 SubVI로 호출될 때 일시 정지하도록 설정할 수 있습니다. 또한, **VI가 다른 우선 순위를 갖고 실행되도록 설정**할 수 있습니다. 예를 들어, 다른 작업이 완료되기를 기다리지 않고 특정 VI가 실행되어야 하는 경우, 그 VI를 시간에 결정적인(가장 높은) 우선 순위로 실행되도록 설정합니다.



노트 서로 다른 우선 순위로 실행되는 VI를 잘못 설정하면 예상치 않게 실행될 수 있습니다. 대부분의 경우 VI를 다른 우선 순위로 실행되도록 설정할 필요가 없습니다.

VI가 즉시 실행되지 않고 사용자가 VI를 편집하지 못하게 하려면 수행>>실행 모드로 변경을 선택하여 VI를 실행 모드로 변경합니다. 실행 모드에서 모든 프런트패널 객체는 바로 가기 메뉴 아이템의 축소된 세트를 가지고 있습니다. 사용자는 VI를 실행하고 정지하는 프런트패널 컨트롤, 도구 모음 버튼 및 메뉴 모음에만 접근할 수 있습니다. 다시 편집을 허용하려면 수행>>편집 모드로 변경을 선택합니다.

VI 개정 히스토리 설정하기

히스토리 윈도우를 사용하여 개정 번호를 포함한 VI의 변경 히스토리를 디스플레이합니다. VI를 변경할 때, 히스토리 윈도우에 변경을 기록하고 추적합니다. 편집>>VI 개정 히스토리를 선택하여 히스토리 윈도우를 디스플레이합니다. 또한 개정 히스토리를 **인쇄**하거나 HTML, RTF, 또는 텍스트 파일로 저장할 수 있습니다.

개정 히스토리의 **옵션 설정**, **주석 추가**, **보기**, **리셋**을 할 수 있습니다.

개정 번호

개정 번호는 VI의 변경을 추적하기 좋은 방법입니다. 이 개정 번호는 1부터 시작해서 VI를 저장할 때마다 단계적으로 증가합니다. VI의 제목 표시줄과 히스토리 윈도우의 제목 표시줄에 현재 개정 번호를 디스플레이하려면, 도구>>옵션을 선택하고, 항목 리스트에서 개정 히스토리를 선택한 후 제목 표시줄에 개정 번호 보이기 확인란에 확인 표시를 합니다.

개정 번호는 히스토리 윈도우의 주석과 무관합니다. 주석 사이의 개정 번호의 공백은 주석 없이 VI만 저장했음을 나타냅니다.

히스토리는 개발 도구이므로, VI의 **블록다이어그램을 제거**하는 경우 LabVIEW는 자동으로 히스토리를 삭제합니다. VI의 런타임 버전에서는 히스토리 윈도우가 지원되지 않습니다. 블록다이어그램이 없는 VI의 경우에도, VI 프로퍼티 대화 상자의 일반 페이지에 개정 번호가 디스플레이됩니다. 히스토리 윈도우의 리셋 버튼을 클릭하여 개정 히스토리를 삭제하고 개정 번호를 리셋합니다.

정의되지 않은 데이터 방지하기

VI가 정의되지 않은 데이터를 생성하는지 확인하기 위하여 NaN, Inf 또는 빈 배열과 같은 특수 값에 의존하지 마십시오. 그 대신, VI가 정의되지 않은 데이터를 생성하는 상황에 처할 경우 에러를 보고하도록 하여 VI가 정의된 데이터를 생성하도록 하십시오.

예를 들어, 입력 배열을 사용하는 VI를 생성하여 For 루프를 오토인덱싱하는 경우, 입력 배열이 비었을 때 VI가 어떻게 작동해야 하는지 결정합니다. 배열이 빈 배열인 경우, 출력 에러 코드를 생성하거나, 루프가 생성한 값을 정의된 데이터로 대체하거나, For 루프를 실행하지 않는 케이스 구조를 사용합니다.



관련 정보

[블록다이어그램 객체](#) (52 페이지 참조)

[배열과 클러스터를 이용한 데이터의 그룹화](#) (417 페이지 참조)

[루프를 사용하여 배열이나 집합의 개별 원소 처리하기](#) (1218 페이지 참조)

[케이스 구조: 입력값에 따라 코드 섹션 실행하기](#) (374 페이지 참조)

잘못된 의존성 방지하기

VI가 다른 파일에 의존하고 있는 경우, 최상위 VI는 SubVI로의 경로를 저장합니다. 이 기능을 사용하면 LabVIEW가 최상위 VI를 로드할 때, 모든 SubVI도 함께 로드됩니다.

LabVIEW가 SubVI를 찾을 수 없을 경우, 디스크의 다른 위치를 검색하여 파일을 찾습니다. LabVIEW가 파일 이름이 같은 다른 SubVI를 찾을 경우, **상호 링크**가 발생합니다.

아래 사항을 참고해서 잘못된 의존성을 로드하는 것을 방지하십시오.

- 모든 VI를 프로젝트에 넣고, **프로젝트 탐색기** 윈도우를 사용하여, 디스크로 파일을 옮깁니다. 각 SubVI를 옮길 때, LabVIEW는 해당 SubVI에 의존하는 VI를 수정하고, 수정된 사항을 저장합니다.
- SubVI를 디스크로 옮겼으나 프로젝트 탐색기 윈도우에 파일이 없을 경우, 어플리케이션과 관련된 모든 VI가 포함된 디렉토리를 옮겼는지 확인하여 해당 파일의 **상대 경로**가 변경되지 않도록 합니다. 그러나 vi.lib, instr.lib, user.lib와 같이 **LabVIEW와 함께 설치된 디렉토리**의 SubVI를 참조하는 경우, 최상위 VI에 링크된 SubVI는 옮기지 않아도 됩니다.
- 같은 파일 이름을 가진 두 개의 VI가 있을 때는, 프로젝트 라이브러리를 사용하여 파일의 이름을 유효화해서 프로젝트 라이브러리의 이름이 파일 이름의 일부가 되도록 합니다. 어플리케이션이 같은 이름을 가진 두 개의 VI를 참조할 경우, LabVIEW는 잘못된 VI를 로드하거나, 두 개의 VI를 모두 로드하려고 시도할 수도 있습니다.

습니다. LabVIEW가 같은 이름을 가진 두 개의 파일을 로드하려고 시도하면, 충돌 해결책 대화 상자가 나타나 모든 충돌을 열거합니다.

빌드 전 또는 후에 VI 실행하기

독립 어플리케이션, .NET interop 어셈블리, 공유 라이브러리 또는 소스 배포는 빌드 전과 후에 VI를 실행할 수 있습니다.

(Real-Time Module) 독립 리얼 타임 어플리케이션은 빌드 전과 후에 VI를 실행할 수 있습니다.

다음 보기는 빌드 전에 실행하는 VI인 빌드 이전 VI를 실행하는 경우를 보여줍니다.

- 대상 디렉토리에서 특정 파일이나 모든 파일 삭제하기.
- LabVIEW 프로젝트의 가장 최신 버전 VI를 **소스 컨트롤**에서 로컬 디렉토리로 복사하기.



노트

VI가 열려있을 때 변경한 사항은 저장되지 않습니다. 소스 컨트롤에서 VI를 복사할 때, 빌드 전 복사본을 만들기 전에 반드시 VI를 닫도록 하십시오.

다음 보기는 빌드 후에 실행하는 VI인 빌드 이후 VI를 실행하는 경우를 보여줍니다.

- 빌드를 처음 저장한 곳에서 반출 디렉토리등과 같은 다른 위치로 복사하기.
- 입력의 일부를 사용해서 빌드 파일에서 테스트를 실행하여 빌드가 적절한 출력을 반환하는지 확인하기.
- 명령 라인 파라미터의 연산을 사용해서 빌드 파일을 실행하여 빌드를 테스트하기.

빌드 전 혹은 후에 실행하려는 VI가 프로젝트에 이미 존재하지 않을 경우, 빌드 이전/이후 작업 페이지에서 적절한 VI 생성 버튼을 클릭해서 VI를 생성합니다.

VI의 원형을 리셋하는 커넥터 팬 변경 내역 (공유 라이브러리)

공유 라이브러리의 빌드 스펙을 설정할 때, VI 원형 정의 대화 상자를 사용하여 LabVIEW가 공유 라이브러리에서 VI를 위해 생성하는 원형을 사용자 정의할 수 있습니다. 그러나, LabVIEW는 사용자가 이 VI의 커넥터 팬 또는 VI에 할당된 컨트롤 또는 인디케이터를 변경할 경우, 이 사용자 원형을 그대로 유지하지 않습니다. 이러한 아이템을 변경하면 LabVIEW가 기본 원형을 복구하고, 사용자가 다음에 원형을 열거나 공유 라이브러리를 만들 때 이 사실을 알려줍니다.

VI를 다음과 같이 변경하면 LabVIEW가 VI의 원형을 리셋합니다:

- 데이터 타입의 컨트롤 또는 인디케이터를 이전 데이터 타입과 호환되지 않는 타입으로 변경하기. 예: 숫자형 인디케이터를 문자열 인디케이터로 변경.

- 컨트롤 또는 인디케이터형 변경하기.
- 커넥터 팬의 컨트롤 또는 인디케이터를 추가하거나 제거하기.
- 열거형의 아이템을 수정하기.

컨트롤 또는 인디케이터의 외관만 변경하는 것은 사용자 원형에 영향을 미치지 않습니다.

그림 플롯 VI

그림 플롯 VI를 사용하여 2D 그림 컨트롤로 일반적인 타입의 그래프를 생성합니다. 이 그래프에는 극좌표 플롯, 웨이브폼 그래프, XY 그래프, 스미스 플롯, 방사형 플롯, 그리고 그래프 스케일 등이 있습니다.

그림 플롯 VI들은 하위 레벨의 그리기 함수를 사용하여 데이터의 그래픽 디스플레이를 생성하고, 그리기 코드를 사용자 정의하여 기능을 추가합니다. 이들 그래픽 디스플레이는 내장 LabVIEW 컨트롤보다 대화식이지는 않지만, 현재 내장 컨트롤이 할 수 없는 방법으로 시각화된 정보를 제공할 수 있습니다. 예를 들어, [웨이브폼 플롯] VI를 이용하여 내장 웨이브폼 그래프보다 약간의 다른 기능을 갖는 플롯을 생성할 수 있습니다.

[포인트 옵션이 있는 극좌표 플롯] VI를 SubVI로 사용하기

[포인트 옵션이 있는 극좌표 플롯] VI를 이용하여 구체적이고 연속적인 극좌표 그래프의 사본면을 그리거나 한 번에 전체의 그래프를 그리십시오. 내장 LabVIEW 그래프처럼, 원소의 색을 조절할 수 있고, 그리드를 추가하며, 스케일의 범위와 포맷을 지정할 수 있습니다.

[포인트 옵션이 있는 극좌표 플롯] VI는 단일 VI에서 많은 기능을 제공합니다. 결과적으로, 이 VI는 복잡한 클러스터 입력을 가지고 있습니다. 기본값을 사용하거나 사용자 정의 컨트롤을 사용하여 이 VI의 복잡성을 줄일 수 있습니다. 블록다이어그램에서 기본 클러스터 입력을 생성하지 말고 labview\examples\Graphics and Sound\Picture Plots 디렉토리에 있는 Simple Polar Plot VI에서 사용자 컨트롤을 복사하여 프런트패널에 놓고 사용하십시오.

웨이브폼 플롯 VI와 XY 플롯 VI를 SubVI로 사용하기

내장 웨이브폼 그래프와 유사하게 동작하는 [웨이브폼 플롯] VI를 사용하여, 점, 연결된 점, 막대바를 포함한 여러 형태의 스타일로 웨이브폼을 그리십시오. 내장 LabVIEW 웨이브폼 그래프처럼, 구성요소의 색을 조절하고, 눈금을 추가하며, 그리고 스케일의 범위와 포맷을 지정할 수 있습니다.

[웨이브폼 플롯] VI는 단일 VI에서 많은 기능을 제공합니다. 결과적으로, 이 VI는 복잡한 클러스터 입력을 가지고 있습니다. 기본값을 사용하거나 사용자 정의 컨트롤을 사용하여 이 VI의 복잡성을 줄일 수 있습니다. 기본 클러스터 입력을 생성하지 말고 labview\examples\Graphics and Sound\Picture Plots 디렉토리의 Waveform and XY Plots VI에서 사용자 컨트롤을 복사하여 프런트패널에 놓고 사용하십시오.

[XY 플롯] VI와 [여러 XY 플롯] VI는 [웨이브폼 플롯] VI와 유사합니다. XY 플롯 VI들은 세 가지 플롯 스타일을 추가적으로 가지고 있으므로 다른 컨트롤을 사용하여 플롯의 외관을 지정할 수 있습니다. 세 가지 플롯 스타일—두 가지의 산포 플롯 스타일과 특정 x 값에 대하여 y 의 최소 최대 값을 표시하도록 개별 x 위치에 라인을 그리는 플롯 스타일이 있습니다.

스미스 플롯 VI를 SubVI로 사용하기

통신 산업에서 사용되는 트랜스미션 라인의 기능에 대하여 배우고자 할 때, [스미스 플롯] VI를 이용하십시오. 트랜스미션 라인은 에너지나 신호를 전송하는 매개체입니다. 트랜스미션 라인은 신호가 전송되는 와이어일 수도 있고 대기일 수도 있습니다. 트랜스미션 라인은 전송되는 신호에 영향을 미칩니다. 트랜스미션 라인의 임피던스라고 불리는 이 현상은 AC 신호의 크기를 감소시키거나 위상 차를 만듭니다.

트랜스미션 라인의 임피던스는 라인의 저항과 리액턴스를 측정한 것입니다. 임피던스 z 는 일반적으로 저항 r 과 리액턴스 x 를 구성요소로 가지는 $z = r + jx$ 형태의 복소수로 나타냅니다.

[스미스 플롯] VI를 사용하여 트랜스미션 라인의 임피던스를 나타내십시오. 플롯은 저항 상수와 리액턴스 상수의 원으로 구성됩니다.

r 원과 x 원의 교차점이 그려짐에 따라, 주어진 임피던스, 즉 $r + jx$ 를 플롯할 수 있습니다. 임피던스를 플롯한 다음, [스미스 플롯] VI를 임피던스 매칭이나 트랜스미션 라인의 반사 계수를 구하는 시각적인 도구로 사용할 수 있습니다.

[스미스 플롯] VI는 각 단일 VI에서 많은 기능을 제공합니다. 결과적으로, 이들 VI는 복잡한 클러스터 입력을 가집니다. 기본값을 사용하거나 사용자 정의 컨트롤을 사용하여 이 VI의 복잡성을 줄일 수 있습니다. 기본 클러스터 입력을 생성하지 말고 `labview\examples\Graphics and Sound\Picture Plots` 디렉토리의 Smith Plot example VI에서 사용자 컨트롤을 복사하여 프런트패널에 놓고 사용하십시오.

스미스 플롯의 세부 요소를 상실하지 않기 위하여, [스미스 플롯 정규화] VI를 이용하여 데이터를 정규화합니다. [스미스 플롯 정규화] VI를 이용하여 정규화한 데이터를 바로 [스미스 플롯] VI로 전달할 수 있습니다. 일반적으로 시스템의 특성 임피던스(Z_0)에 대하여 스미스 플롯 데이터를 스케일합니다.

그림 함수 VI

그림 함수 VI를 사용하여 2D 그림 컨트롤에 형태를 그리고 텍스트를 입력합니다. 점, 선, 형태, 그리고 픽스맵을 그릴 수 있습니다. 패턴화 해제된 데이터의 픽스맵은 2차원 색 배열이며, 각 값은 색 또는 색 농도에 의한 RGB 색 값의 배열 인덱스에 대응합니다.

그림 함수 팔레트의 첫 번째 행은 포인트와 선을 그리는 VI가 위치합니다. 포인트는 픽셀의 x -와 y -좌표를 나타내는 두 개의 16비트 부호있는 정수의 클러스터입니다.

그림 함수 VI를 사용하면, 그림은 그래픽 펜의 위치를 기억합니다. 대부분의 그림 함수 VI는 반드시 절대 좌표로 지정됩니다—즉, 원점 (0, 0)에 상대적입니다. [라인 그리기] VI와 [펜 이동] VI에서는 절대 좌표 또는 상대 좌표를 설정할 수 있습니다. 상대 좌표는 펜의 현재 위치에 연동합니다. [펜 이동] VI를 이용하여 그림을 그리지 않고 펜의 위치를 바꿀 수

있습니다. [포인트 그리기] VI, [펜 이동] VI, [라인 그리기] VI, [여러 라인 그리기] VI만 펜의 위치를 바꿀 수 있습니다.

그림 함수 팔레트의 두 번째 행은 형태를 그리는 VI가 위치합니다. 각 VI는 그림의 사각 영역 안에 형태를 그립니다. 사각형을 사각 영역의 왼쪽, 위쪽, 오른쪽, 그리고 아래쪽 픽셀을 나타내는 4개의 값의 클러스터로 지정합니다.

그림 함수 팔레트의 세 번째 행은 그림 속에 문자를 쓰는 VI가 위치합니다. [텍스트 사각 형태 두리얼기] VI는 문자를 쓰지 않습니다. 대신에 문자열을 둘러싼 사각형의 크기를 계산할 때 사용합니다.

그림 함수 팔레트의 네 번째 행은 그림에 패턴화하거나 패턴화 해제된 픽스맵을 그리는 VI, 이미지에 마스크를 적용하는 VI, 원본 이미지의 일부를 가져오는 VI, 또는 그림 데이터 타입을 패턴화된 이미지 데이터 클러스터로 변환하는 VI가 위치합니다.

그림 함수 팔레트의 마지막 행은 백지 그림 상수가 위치하며, 이는 빈 이미지로 초기화하거나 빈 그림을 변경할 때 사용됩니다. 팔레트의 마지막 행은 빨간색, 녹색, 파란색 값을 대응하는 RGB 색으로 변환하거나 RGB 색을 빨간색, 녹색, 파란색 요소값으로 변환하는 VI가 위치합니다.

그림 함수 VI로 만든 그림은 2D 그림 컨트롤 또는 그림 함수 VI의 그림 입력에만 연결할 수 있습니다. LabVIEW는 사용 중인 프런트패널에 2D 그림 컨트롤이 업데이트될 때 그림을 그립니다.

각 그림 함수 VI는 자신의 그리기 명령을 그림 입력에 연결된 그리기 명령과 연결하며 새 그림 출력단으로 전체 그리기 명령을 반환합니다.

다음 블록다이어그램은 [사각형 그리기] VI를 사용하여 두 개의 겹친 사각형을 만듭니다.

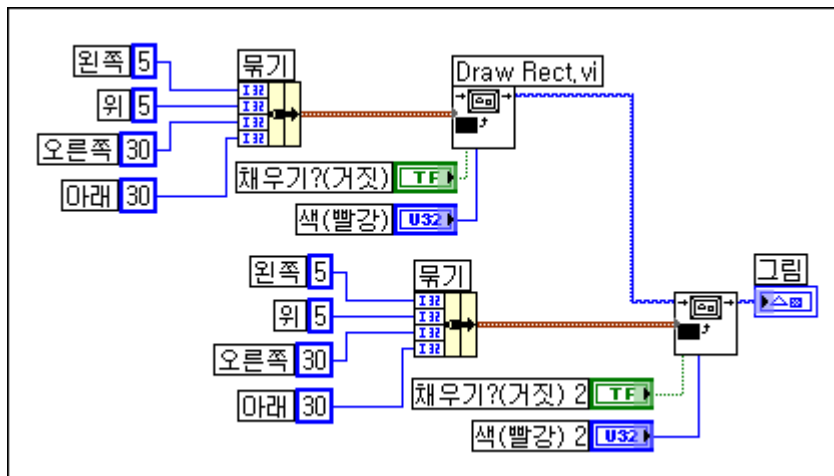


그림 함수 VI로 색 생성하기와 수정하기

대부분의 그림 함수 VI는 형태나 문자의 색을 수정하기 위한 색 입력을 가지고 있습니다. 색을 지정하기 위한 가장 편한 방법은 색 상자 상수를 사용하고 이를 클릭하여 색을 선택하는 것입니다.

색 상자 상수를 사용하지 않고 계산의 결과로 색을 생성하려면, 색 상자가 숫자값으로 어떻게 색을 지정하는지 이해할 필요가 있습니다.

32비트 부호있는 정수가 하나의 색을 나타내며, 아래쪽 세 바이트가 색의 빨간색, 녹색, 파란색 원소를 나타냅니다. 여러 형태의 파란색을 만들기 위하여, 32비트 정수 배열을 만들고, 각 원소의 파란색 값을 변화시켜 주고, 파란색의 값을 빨간색과 녹색보다 크게 만듭니다. 회색을 만들기 위해서, 빨간색, 녹색, 파란색 원소의 값이 동일한 32비트 정수 배열을 만듭니다.

그래픽 포맷 VI

[그래픽 포맷] VI를 사용하여 여러 표준 그래픽 파일 포맷인 비트맵 파일 .bmp, PNG(Portable Network Graphics) 파일 .png, JPEG(Joint Photographic Experts Group) 파일 .jpg에서 데이터를 읽고 씁니다. 이 VI로 다음 태스크를 수행할 수 있습니다:

- 그래픽 파일에서 데이터를 읽어서 2D 그림 컨트롤에 디스플레이합니다.
- 이미지 조작을 위하여 데이터를 읽습니다.
- 다른 어플리케이션에서 볼 수 있도록 이미지를 씁니다.

LabVIEW는 비트맵 데이터를 색상의 농도에 따라 각 포인트가 변하는 2D 배열로 저장합니다. 예를 들어, 흑백 또는 1비트 이미지에서 각 포인트는 불리언입니다. 4비트 그리고 8비트 이미지에서, 각 포인트는 컬러 테이블의 인덱스입니다. 24비트 트루컬러 이미지에서, 각 포인트는 빨간색, 녹색, 파란색(RGB) 값의 혼합입니다.

그래픽 파일을 읽고 쓰는 VI는 그래픽 파일을 디스크에 쓰고 데이터가 1D 배열로 저장되는 방법과 유사하게 단순하고 패턴화된 포맷으로 데이터를 처리합니다. 이러한 그래픽 파일은 픽스맵이며, 비트맵과 비슷한 개념입니다. [패턴화된 픽스맵 그리기] VI를 사용하여 패턴화된 데이터를 직접 디스플레이할 수 있습니다.

2차원 배열로 데이터를 조작하려면, [픽스맵 패턴화 해제] VI와 [픽스맵 패턴화] VI를 사용하여 데이터를 적절한 포맷으로 변환합니다.

사운드 VI

사운드 VI를 사용하여 사운드 파일과 함수를 사용자의 VI에 통합할 수 있습니다. 이 VI로 다음 태스크를 수행할 수 있습니다:

- 사용자의 특정 행동에 대하여 녹음된 경고음과 같은 사운드 파일을 재생하는 VI를 만듭니다.
- VI가 실행을 시작하거나 끝낼 때, 또는 VI의 특정 위치에 도달했을 때 사운드 파일을 재생하는 VI를 만듭니다.
- 사운드 데이터를 수집하기 위해 사운드 입력 디바이스를 설정합니다. 사운드 데이터를 수집하기 위해 사운드 입력 VI를 사용합니다. 또한 디바이스를 통해 들어오는 모든 사운드 정보를 읽을 수 있습니다.

- 다른 사운드 VI에서 사운드 데이터를 받도록 사운드 출력 디바이스를 설정합니다. 장치에 전달되는 사운드의 볼륨을 컨트롤하고, 사운드를 재생 또는 일시 중지하며, 시스템에서 사운드를 제거할 수 있습니다.

(Windows) 사운드 VI를 사용하려면 DirectX 8.0 또는 이후 버전이 설치되어 있어야 합니다. (Linux) 사운드 VI를 사용하려면 Open Sound System(OSS) 드라이버가 있어야 합니다.

사운드 출력 태스크

사운드 출력 VI를 사용하여 스피커나 헤드폰과 같은 출력 사운드 디바이스에서 사운드를 생성하고 컨트롤합니다. 간단한 작업의 경우, 웨이브폼 재생 익스프레스 VI를 사용하여 사운드 출력 디바이스에 사운드 데이터를 재생합니다.

보다 복잡한 작업의 경우, [사운드 출력 설정] VI를 사용하여 새 사운드 출력 태스크 ID를 생성하고 설정합니다. 태스크 ID를 생성한 후, [사운드 출력 시작] VI를 사용하여 사운드가 재생을 시작하는 때를 컨트롤하고, [사운드 출력 중지] VI를 사용하여 사운드를 중지하고, [사운드 출력 볼륨 설정] VI를 사용하여 사운드의 볼륨을 컨트롤하고, [사운드 출력 쓰기] VI를 사용하여 사운드를 사운드 디바이스에 보내고, [사운드 출력 정보] VI를 사용하여 태스크 ID에 대한 정보를 얻기 위해 사운드 디바이스를 쿼리합니다. [사운드 출력 지우기] VI를 사용하여 태스크 ID가 사용한 모든 리소스를 해제합니다.

사운드 입력 태스크

사운드 입력 VI를 사용하여 마이크와 같은 사운드 입력 디바이스에서 LabVIEW로 사운드 데이터를 수집합니다. 사운드 입력 디바이스로부터의 단순한 사운드 수집 작업은 사운드 수집 익스프레스 VI를 사용합니다. 보다 복잡한 작업의 경우, [사운드 입력 설정] VI를 사용하여 새 사운드 입력 태스크 ID를 생성하고 설정합니다.

태스크 ID를 생성한 후, [사운드 입력 시작] VI와 [사운드 입력 중지] VI를 사용하여 각각 사운드의 상태를 실행 또는 유히로 변경할 수 있습니다. [사운드 입력 읽기] VI를 사용하여 사운드 입력 디바이스에서 데이터를 읽습니다. [사운드 입력 지우기] VI를 사용하여 태스크 ID가 사용한 모든 리소스를 해제합니다.

사운드 데이터

LabVIEW는 사운드 데이터를 웨이브폼의 배열로 저장합니다. 배열의 각 웨이브폼은 하나의 채널을 나타냅니다. 예를 들어, 두 웨이브폼으로 구성된 배열은 스테레오 또는 두 채널 데이터를 포함합니다. 각 채널이 동시에 재생되므로, 각 웨이브폼에서 dt 와 t_0 값이 Y 배열의 원소 개수와 일치하도록 합니다.

웨이브폼의 Y 배열은 Pulse Code Modulated (PCM) 데이터를 포함합니다. PCM 데이터에서, Y 배열의 각 원소는 신호의 진폭에 비례합니다. Y 배열 데이터 타입이 부호있는 16 비트 또는 부호있는 32 비트 정수일 경우, 값은 0을 중심으로 하여 데이터 타입에 대한 최소값과 최대값 사이의 범위입니다. 0의 값을 갖는 각 원소의 버퍼는 소리가 없는 상태를 나타냅니다. 배열 데이터 타입이 8비트 부호없는 정수이면, Y의 범위는 127을 중심으로 0에서 255입니다. 127의 값을 가진 각 원소의 버퍼는 0의 값을 갖는 각 원소의 버퍼가 아닌 소리가 없는 상태를 나타냅니다. 이로 인해 8 비트 부호없는 정수에서 처음에 클릭 소리가 들릴 수도 있습니다. 부동소수 데이터의 경우, 값은 -1.0에서 1.0의 범위(경계 포함)입니다. 데이터는 이 범위로 강제 변환됩니다. 스케일링은 없습니다.

태스크 ID

[사운드 출력 설정], [사운드 입력 설정], [사운드 재생] VI는 태스크 ID를 반환합니다. 이 ID는 현재 사운드 태스크를 참조하기 위해 VI가 생성한 숫자입니다. 블록다이어그램에서 이후의 사운드 VI는 태스크 ID를 사용하여 특정한 사운드 작업에서 데이터를 수정합니다. 생성한 VI를 다시 실행하기 전에 반드시 메모리에서 태스크 ID를 삭제해야 합니다. [사운드 출력 지우기] VI를 사용하여 사운드 출력 작업의 태스크 ID를 삭제합니다. [사운드 입력 지우기] VI를 사용하여 사운드 입력 작업의 태스크 ID를 삭제합니다.

사운드 디바이스

사운드 디바이스는 컴퓨터에 연결된 사운드 카드 또는 기타 하드웨어입니다. [사운드 출력 설정]과 [사운드 입력 설정], [사운드 파일 재생] VI의 디바이스 ID 입력값은 사용할 사운드 디바이스를 지정합니다. 컴퓨터의 기본 사운드 디바이스를 사용할 경우 디바이스 ID에 0을 입력합니다. 다른 디바이스를 사용하려면 1, 2, 3 등을 입력합니다.

사운드 버퍼

사운드 입력과 사운드 출력 작업은 LabVIEW와 사운드 디바이스 사이에 버퍼를 사용할 수 있기 때문에 사운드 데이터의 원활한 전송을 제공합니다. 사운드 데이터를 읽고, 쓰고, 설정하는 사운드 VI를 사용하여 사운드 버퍼를 조작합니다. 이러한 VI가 생성하는 태스크는 버퍼와 사운드 디바이스 사이에 사운드 데이터를 전송합니다. 태스크는 사용자가 태스크를 정지하거나 에러가 발생할 때까지 계속 실행됩니다.

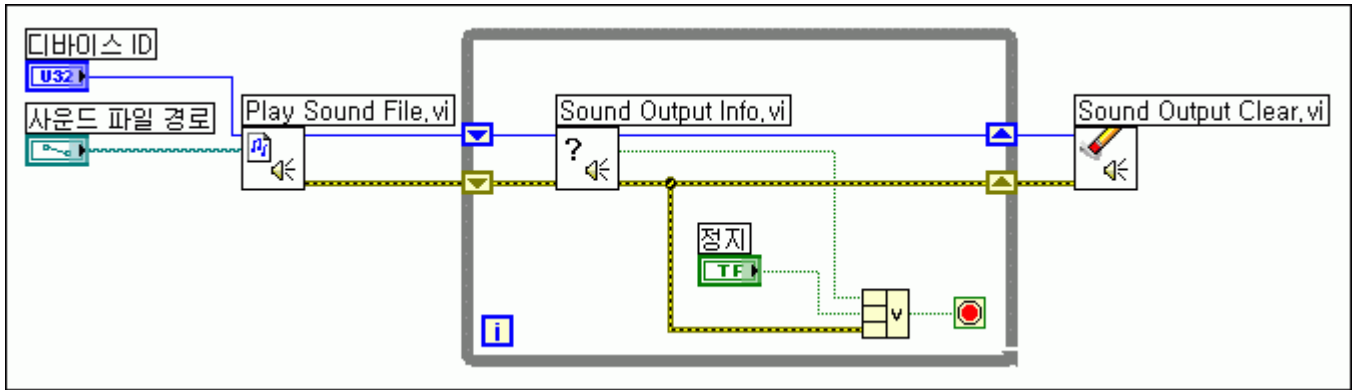
입력 작업은 사운드 데이터를 입력 디바이스에서 버퍼로, 버퍼에서 블록다이어그램으로 전송합니다. [사운드 입력 읽기] VI를 사용하여 디바이스에서 데이터를 수집합니다. 버퍼에 요청된 모든 샘플을 읽을만한 충분한 데이터가 있는 경우, LabVIEW는 즉시 샘플을 읽습니다. 준비된 샘플보다 요청된 샘플이 많은 경우, VI는 요청이 완료될 때까지 실행을 일시 정지합니다.

출력 작업은 사운드 데이터를 블록다이어그램에서 버퍼로, 버퍼에서 출력 디바이스로 전송합니다. [사운드 출력 쓰기] VI를 사용하여 데이터를 사운드 출력 작업의 버퍼에 전송합니다. 버퍼에 충분한 공간이 있는 경우, LabVIEW는 데이터를 버퍼에 복사합니다. 버퍼에 충분한 공간이 없고 버퍼가 연속 모드로 설정되어 있는 경우, VI는 모든 데이터를 버퍼에 복사할 때까지 실행을 일시 정지합니다.

사운드 데이터 설정은 버퍼된 데이터의 포맷을 설정합니다. LabVIEW는 자동으로 LabVIEW 데이터의 포맷을 변환하여 버퍼 데이터 포맷에 맞춥니다. 예를 들어, 출력 태스크를 설정하여 스테레오 데이터를 사용하나 하나의 채널에만 데이터를 쓰는 경우, LabVIEW는 두 스테레오 채널 모두에 사용자가 쓰는 데이터를 보냅니다.

사운드 파일 재생하기

[사운드 파일 재생] VI와 While 루프를 사용하여 다음 블록다이어그램과 같이 연속적으로 사운드 파일을 재생합니다. [사운드 파일 재생] VI는 사운드 파일 경로 입력에 지정된 파일을 읽고 디바이스 ID 입력이 지정하는 사운드 디바이스를 통해 사운드를 재생합니다. VI는 **태스크 ID**를 반환하며 While 루프는 사운드 재생이 완료되거나, 에러가 발생하거나, 사용자가 프런트패널의 정지 버튼을 누를 때까지 연속적으로 태스크 ID를 전달합니다. [사운드 출력 지우기] VI는 컴퓨터 메모리에서 태스크 ID를 삭제합니다.

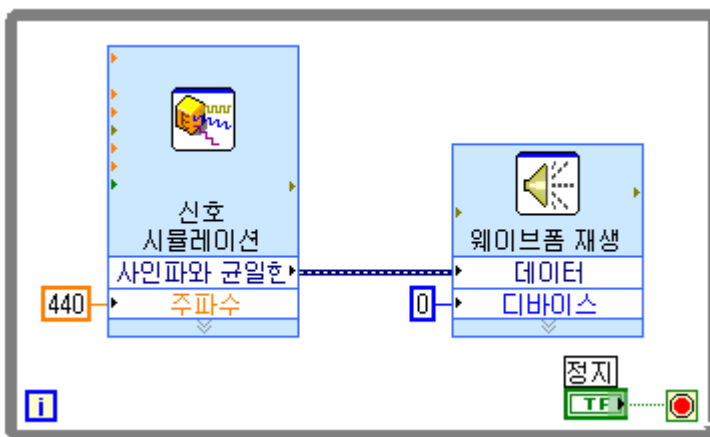


사운드 VI를 사용하여 파일의 비트 속도와 같은 사운드 재생 방법과 사운드가 재생되는 채널(모노, 스테레오 등)을 설정합니다. 또한 런타임에 파일을 일시 정지, 정지, 재생할 수 있습니다.

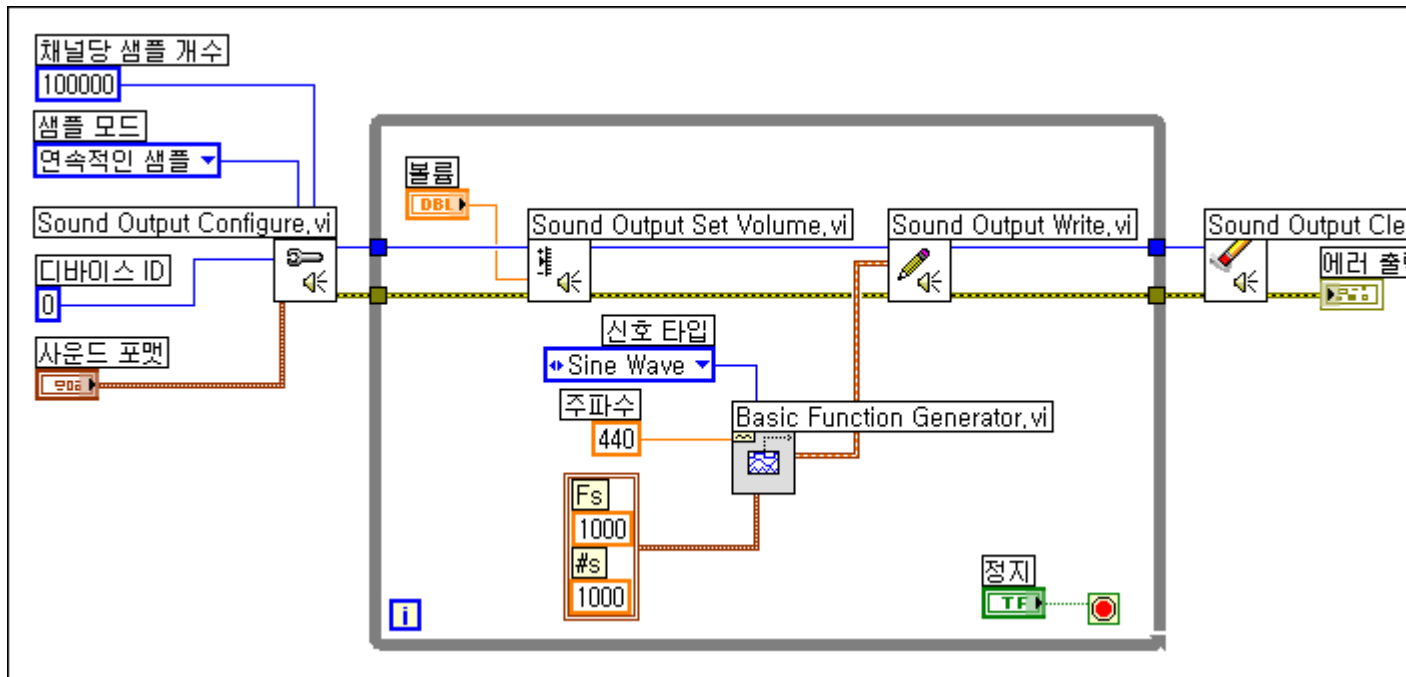
사운드 데이터 생성에 대한 예제는 labview\examples\Graphics and Sound\Sound 디렉토리에 있는 Sound File to Sound Output VI를 참조하십시오.

사운드 데이터 생성하고 재생하기

다음 블록 다이어그램에서 보이는 것처럼, [웨이브폼 생성] VI와 [신호 시뮬레이션] 익스프레스 VI를 사용하여 사운드 데이터를 생성합니다. [신호 시뮬레이션] 익스프레스 VI는 사인파를 생성하며, [웨이브폼 재생] 익스프레스 VI는 디바이스 입력에서 지정한 사운드 디바이스를 통해 생성된 사운드 데이터를 재생합니다.



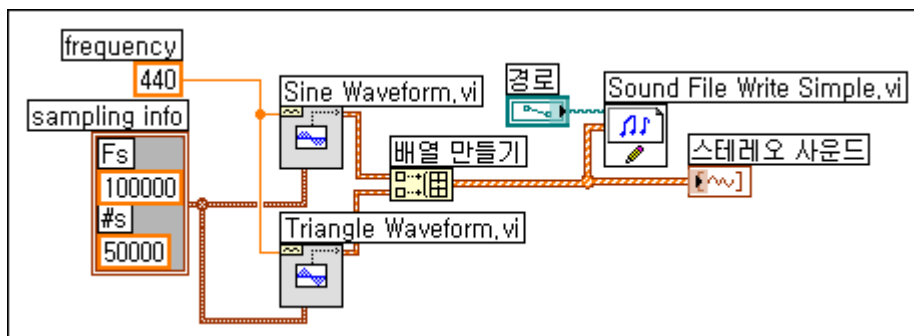
생성한 사운드 데이터를 더욱 정밀하게 컨트롤하려면 [사운드 출력] VI를 사용하십시오. 예를 들어, [사운드 출력 설정] VI를 사용하여 비트 속도, 샘플 속도, 사운드의 채널 개수를 컨트롤하고, [사운드 출력 볼륨 설정] VI를 사용하여 다음 블록 다이어그램과 같이 사운드의 볼륨을 컨트롤합니다.



사운드 데이터 생성에 대한 예제는 labview\examples\Graphics and Sound\Sound 디렉토리에 있는 Generate Sound VI를 참조하십시오.

데이터를 사운드 파일에 쓰기

[단순한 사운드 파일 쓰기]와 [사운드 파일 쓰기] VI를 사용하여 수집하거나 생성한 데이터를 사운드 파일에 씁니다. 다음 블록 다이어그램은 [단순한 사운드 파일 쓰기] VI를 사용하여 각각 50,000 샘플의 사인파 톤과 삼각파 톤으로 구성된 스테레오 사운드를 경로 입력에서 지정된 파일에 저장합니다.



사운드 파일의 생성을 더 세심하게 컨트롤하려면 [사운드 파일 쓰기] VI를 사용하십시오. [사운드 파일 쓰기] VI는 사용자가 [사운드 파일 열기] VI로 기존 사운드 파일의 참조를 열거나 새 사운드 파일을 생성하고 [사운드 파일 닫기] VI로 사운드 파일 참조를 닫을 것을 요구합니다. 사운드 파일을 닫지 않는 경우, 파일은 메모리에 남게되며 다른 어플리케이션이나 다른 사용자가 이에 접근할 수 없습니다.

사운드를 수집하고 사운드 파일에 저장하는 예제는 labview\examples\Graphics and Sound\Sound 디렉토리에 있는 Sound Input to File VI를 참조하십시오.

그래픽 & 사운드 VI

그래픽 & 사운드 VI를 사용하여 VI의 그래픽과 사운드를 디스플레이하거나 수정합니다.



관련 정보

- [그림 플롯 VI](#) (155 페이지 참조)
- [그림 함수 VI](#) (156 페이지 참조)
- [그래픽 포맷 VI](#) (158 페이지 참조)
- [사운드 VI](#) (158 페이지 참조)
- [3D 그림 컨트롤 VI](#) (164 페이지 참조)
- [2D 그림 컨트롤에서 보이지 않는 픽셀 디스플레이하기](#) (1184 페이지 참조)
- [사운드 입력과 사운드 출력 디바이스 사용하기](#) (1185 페이지 참조)
- [3D 그림 컨트롤로 3D 장면 생성하기](#) (1185 페이지 참조)
- [3D 객체간 관계 생성하기](#) (1192 페이지 참조)
- [3D 장면 윈도우 설정하기](#) (1195 페이지 참조)

VI 문서화 및 인쇄하기

LabVIEW를 사용하여 VI 및 VI에 대한 정보를 문서화하고 인쇄할 수 있습니다. VI의 목적을 설명하고 VI의 사용법을 알리기 위해 VI를 문서화합니다. VI에 대한 정보를 기록하기 위해 VI 및 VI 문서를 인쇄합니다. 또한 VI 및 VI 문서를 HTML, RTF, 또는 텍스트 파일로 저장할 수 있습니다.

LabVIEW에는 VI를 인쇄하는 3가지 방법이 있습니다. 활성 윈도우만을 인쇄할지, VI 문서와 같은 VI에 대한 보다 종합적인 정보를 포함할지 또는 VI가 반환하는 데이터를 프로그램적으로 인쇄할지 선택할 수 있습니다. 인쇄 대화 상자 및 VI 프로퍼티 대화 상자에는 LabVIEW가 인쇄하는 내용을 설정하는 옵션이 포함되어 있습니다.



관련 정보

- [VI 문서화하기](#) (164 페이지 참조)
- [VI 인쇄하기](#) (165 페이지 참조)
- [VI 인쇄하기](#) (165 페이지 참조)
- [VI 문서화하기](#) (164 페이지 참조)
- [VI 인쇄하기](#) (165 페이지 참조)
- [VI 문서화하기](#) (1391 페이지 참조)
- [리포트 생성하기](#) (1409 페이지 참조)

3D 그림 컨트롤 VI

3D 그림 컨트롤 VI는 그래픽 & 사운드 팔레트에 나타납니다. 3D 그림 컨트롤 VI를 사용하여 객체, 빛, 변환, 객체에 첨가하는 다른 특성들이 포함된 장면을 디자인합니다. 장면을 만든 후, 3D 그림 프론트패널 컨트롤에 객체 혹은 객체들을 연결하고 VI를 실행하여 장면을 생성합니다.

3D 그림 컨트롤 VI와 함께 다음의 프로퍼티와 메소드를 사용하여 3D 장면을 생성합니다:

- 장면 객체 프로퍼티와 장면 객체 메소드는 3D 객체를 장면에 놓고 기하, 변환, 다른 특성들을 객체에 부여합니다.
- 장면 절단 평면 프로퍼티는 객체가 나타나거나 절단되는 장면의 평면을 지정합니다.
- 장면 기하 프로퍼티와 장면 기하 메소드는 객체가 갖는 기하 형태를 지정합니다.
- 장면 텍스처 프로퍼티와 장면 텍스처 메소드는 3D 객체에 텍스처를 첨가합니다.
- 장면 윈도우 프로퍼티는 분리된 윈도우에서 장면을 제공하고, 윈도우를 설정하며, 장면과 카메라의 상호작용을 셋팅합니다.
- 장면 빛 프로퍼티는 장면의 빛 소스를 설정합니다.

3D 그림 컨트롤 VI에서 3D 장면을 생성하는 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.



관련 정보

[프론트패널 컨트롤과 인디케이터](#) (544 페이지 참조)

VI 문서화하기

LabVIEW를 사용하여 완성된 VI를 문서화하고 VI의 사용자를 위한 설명을 생성할 수 있습니다. LabVIEW에서 문서를 보고, 인쇄하고, HTML, RTF, 또는 텍스트 파일로 저장할 수 있습니다.

VI를 효과적으로 문서화하려면, VI 또는 객체의 용도와 사용방법에 대한 설명을 제공합니다.

파일>>VI 프로퍼티를 선택하고 항목 풀다운 메뉴에서 문서를 선택하여 VI 설명을 생성하고, 편집하고, 봅니다. 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 설명과 팁을 선택하여 객체 설명을 생성하고, 편집하고, 봅니다. 팁 상자는 VI가 실행되는 동안 커서를 프론트패널 객체 위로 움직일 때 나타나는 간략한 설명입니다. 설명과 팁 대화 상자에 팁을 입력하지 않은 경우, 팁 상자는 나타나지 않습니다. 커서를 객체 위로 이동할 때 객체 설명이 기본 도움말 윈도우에 나타납니다. 설명과 팁 대화 상자에서 VI에 대한 설명을 볼 수 있습니다.

VI 인쇄하기

차후에 참조할 수 있는 기록을 남기기 위해 VI 및 VI 문서를 인쇄하거나 저장하십시오. 다음의 기본적인 방법을 사용하여 VI를 인쇄할 수 있습니다:

- 파일>>원도우 인쇄를 선택하여 활성 윈도우의 내용을 인쇄합니다.
- 파일>>인쇄를 선택하여 프런트패널, 블록다이어그램, SubVI, 컨트롤, VI 히스토리 등을 포함한 VI에 대한 보다 포괄적인 정보를 인쇄합니다.
- VI 문서나 VI가 반환하는 데이터를 포함하는 VI 또는 리포트를 프로그램적으로 인쇄하거나 저장합니다.

파일>>원도우 인쇄를 선택할 때 또는 **프로그램적으로 인쇄**할 때 LabVIEW가 VI를 인쇄하는 방법은 파일>>VI 프로퍼티를 선택하여 설정할 수 있습니다.

활성 윈도우 인쇄하기

파일>>원도우 인쇄를 선택하여 활성 프런트패널 또는 블록다이어그램 윈도우의 내용을 인쇄할 수 있습니다. LabVIEW는 윈도우의 보이지 않는 부분의 객체를 포함하여 활성 윈도우의 작업 공간을 인쇄합니다. LabVIEW는 제목 표시줄, 메뉴 모음, 도구 모음, 또는 스크롤 막대를 인쇄하지 않습니다.

문서 인쇄하기

파일>>인쇄를 선택하여, VI 문서를 인쇄하거나 HTML, RTF, 또는 텍스트 파일로 저장할 수 있습니다. 하나의 VI 또는 여러 VI의 문서를 인쇄할지를 선택할 수 있습니다. 또한, 문서에 대해 내장된 문서 스타일을 선택하거나 사용자 정의된 스타일을 생성할 수 있습니다..

생성하는 문서는 다음 아이템을 포함할 수 있습니다:

- 아이콘과 커넥터 팬
- 프런트패널과 블록다이어그램
- 컨트롤, 인디케이터, 데이터 타입 터미널
- 컨트롤과 인디케이터의 라벨과 캡션
- VI와 객체 설명
- VI 계층구조
- SubVI의 리스트
- 개정 히스토리



노트

특정 타입의 VI에 대해서 생성하는 문서는 위의 모든 아이템을 포함할 수 없습니다. 예를 들어, 다형성 VI에는 프런트패널 또는 블록다이어그램이 없기 때문에 다형성 VI에 대해서 문서를 생성할 때에는 이러한 아이템을 포함할 수 없습니다.

문서를 HTML, RTF, 또는 텍스트 파일로 저장하기

VI 문서를 HTML, RTF, 또는 텍스트 파일로 저장할 수 있습니다. HTML과 RTF 파일은 대부분의 워드 프로세싱 어플리케이션으로 반입할 수 있으며, **컴파일된 도움말 파일을 생성**할 때 사용할 수 있습니다. 또한 LabVIEW가 생성하는 HTML 파일을 사용하여 웹에 VI 문서를 디스플레이할 수 있습니다. 또한 **프로그램적으로** HTML, RTF, 또는 텍스트 파일로 문서를 인쇄하고 저장할 수 있습니다.

RTF 파일로 문서를 저장할 때, 도움말 파일과 워드 프로세싱 중 어느 것에 적합한 파일을 생성할지 지정합니다. 도움말 파일 포맷인 경우, LabVIEW는 그래픽을 외부 비트맵 파일로 저장합니다. 워드 프로세싱 파일 포맷인 경우, LabVIEW는 그래픽을 문서에 내장시킵니다. HTML 파일인 경우, LabVIEW는 모든 그래픽을 외부 JPEG, PNG, 또는 GIF 포맷으로 저장합니다.

HTML 파일을 위한 그래픽 포맷 선택하기

문서를 HTML 파일로 저장할 때, 그래픽 파일의 포맷과 색 농도를 선택할 수 있습니다.

JPEG 포맷은 그래픽 압축을 잘 하지만 정밀한 그래픽의 일부가 손상될 수 있습니다. 이 포맷은 사진에 가장 적합합니다. 라인 아트, 프론트패널, 블록다이어그램에서, JPEG 압축은 희미한 그래픽과 균일하지 않은 색을 가져올 수 있습니다. JPEG 그래픽은 항상 24비트 그래픽입니다. 흑백과 같은 낮은 색 품질의 해상도를 선택할 경우, 그래픽은 요청한 농도로 저장하지만 결과는 여전히 24비트 그래픽입니다.

PNG 포맷 또한 그래픽을 잘 압축하지만, JPEG 포맷만큼 항상 잘 하지는 못합니다. 하지만, PNG 압축시 그래픽의 정밀한 부분이 손상되지 않습니다. 또한, 1비트, 4비트, 8비트, 24비트 그래픽을 지원합니다. 낮은 색 품질의 해상도인 경우, 결과 그래픽은 JPEG보다 압축률이 높습니다.

LabVIEW는 그래픽을 압축된 GIF 파일로 저장하지 않습니다. 그래픽 포맷 변환기를 사용하여, LabVIEW가 저장한 압축되지 않은 GIF 파일을 압축된 GIF 파일로 변환할 수 있습니다. 더 좋은 질의 압축 GIF 파일을 위해서, 문서를 저장하고 그래픽 포맷 변환기를 사용하여 LabVIEW에서 저장한 PNG 파일을 GIF 파일로 바꿀 때 PNG 포맷을 선택합니다. PNG 포맷은 원본 그래픽과 동일한 복사본이기 때문에, PNG 포맷은 고화질의 그래픽을 만듭니다. .png 확장자로 GIF 파일을 나타내기 위해 LabVIEW가 생성한 HTML 파일을 수정합니다.

그래픽 파일의 명명 규칙

외부 그래픽을 포함하는 HTML 또는 RTF 문서를 생성할 때, LabVIEW는 컨트롤과 인디케이터 데이터 타입 터미널을 일관된 이름을 갖는 그래픽 파일로 저장합니다. VI에 동일한 타입의 여러 터미널이 있을 경우, LabVIEW는 해당 타입에 대해서 오직 하나의 그래픽 파일을 생성합니다. 예를 들어, VI에 세 개의 32비트 부호있는 정수 입력이 있을 경우, LabVIEW는 한 개의 ci32. x 파일을 생성하며, 이때 x는 그래픽 포맷에 해당하는 확장자입니다.

사용자 도움말 파일 생성하기

LabVIEW가 생성한 HTML 또는 RTF 파일로 **컴파일된 사용자 정의 도움말 파일을 생성**할 수 있습니다. (Windows) LabVIEW로 생성한 개별의 **HTML 파일**을 HTML 도움말 파일로 컴파

일할 수 있습니다. (macOS) LabVIEW가 생성하는 개별 HTML 파일을 Apple 도움말에 사용할 수 있습니다.

LabVIEW로 생성한 **RTF 파일**을 (Windows) WinHelp 또는 (Linux) HyperHelp 파일로 컴파일할 수 있습니다.

VI와 어플리케이션에 사용자 문서 제공하기

도움말 메뉴에 사용자 문서를 제공하려면, labview\help 디렉토리에 사용자 정의된 도움말 파일을 놓습니다. (Windows) 사용자 문서에는 텍스트 파일을 제외한 모든 타입의 파일을 사용할 수 있습니다. (macOS와 Linux) HTML (.html 및 .htm) 파일과 PDF 파일만 사용할 수 있습니다.

	노트 (Linux) 사용자 문서에 PDF 파일을 사용하려면 반드시 다음 중 하나의 PDF 뷰어가 설치되어 있어야 합니다. • Acroread • Xpdf • KPDF • GGV

다음 단계를 완료하여 도움말 메뉴에 사용자 도움말을 디스플레이합니다.

1. 사용자 도움말 파일을 labview\help 디렉토리에 놓습니다.
2. 사용자 도움말 파일과 동일한 이름을 갖는 텍스트 파일을 labview\help 디렉토리에 놓습니다. 도움말 메뉴에 디스플레이하려는 도움말 제목을 텍스트 파일 안에 포함시킵니다. 예를 들어, 도움말 파일의 이름이 My Custom Help.html 라면, 메뉴 제목을 포함하는 텍스트 파일의 이름은 My Custom Help.txt가 되어야 합니다.

	노트 텍스트 파일에는 도움말 메뉴에 디스플레이하려는 사용자 도움말 제목만 포함해야 합니다.

VI에서 웹 기반 도움말 링크하기

기본 도움말 윈도우에서 **VI를 웹 기반 문서에 링크**하려면, VI 프로퍼티 대화 상자의 문서 페이지에 있는 드롭다운 메뉴에서 웹 기반 도움말 옵션을 선택합니다. 도움말 URL 텍스트 박스를 사용하여 링크하려는 토픽의 URL을 지정합니다.

또한 웹 URL 사용과 웹 URL 문서 프로퍼티를 사용하여 프로그램적으로 VI를 웹 기반 문서에 링크할 수 있습니다.

VI에서 온라인 도움말 링크하기

기본 도움말 윈도우에서 **VI를 로컬 문서에 링크**하려면, VI 프로퍼티 대화 상자의 문서 페이지에 있는 드롭다운 메뉴에서 로컬 도움말 파일 옵션을 선택합니다. 도움말 경로 텍스트 상자를 사용하여 로컬 컴퓨터에 있는 사용자 문서를 탐색합니다. 선택적으로, 도움말 태그 텍스트 박스를 사용하여 컴파일된 도움말 파일의 **특정 토픽에 VI를 링크**하거나, HTML 도움말 파일의 북마크에 링크할 수 있습니다. 또는 도움말 태그 텍스트 박스를 비워둘 수 있습니다. 예를 들어, 도움말 경로 텍스트 박스에 My Custom Help.chm라는 이름의 컴파일된 도움말 파일을 사용하도록 지정한 경우, 도움말 태그 텍스트 박스에 GettingStarted.html#Intro를 입력하여 기본 도움말 윈도우의 상세 도움말 링크가 사용자가 제공한 컴파일된 도움말의 기본 정보로 바로 연결되도록 할 수 있습니다. 그러나, 도움말 경로 텍스트 박스에 My Custom Help.pdf라는 이름의 PDF 파일을 지정하는 경우, 토픽이 HTML 파일 또는 컴파일된 도움말의 HTML 파일인 경우에만 VI를 특정 토픽에 링크할 수 있기 때문에 도움말 태그를 비워 두어야 합니다.

또한 문서 태그와 문서 경로 프로퍼티를 사용하여 프로그램적으로 VI를 로컬 도움말 파일에 링크할 수 있습니다.

프로그램적으로 VI 인쇄하기

다음 방법 중 하나로 VI나 VI가 반환하는 데이터를 프로그램적으로 인쇄합니다.

- VI가 매번 실행을 마칠 때마다 프론트패널을 자동적으로 인쇄하도록 VI를 설정합니다.
- VI를 인쇄할 subVI를 생성합니다.
- [리포트 생성] VI를 사용하여 리포트를 인쇄하거나 문서화된 VI 또는 VI가 반환한 데이터를 포함한 HTML 리포트를 저장합니다.
- **VI 서버**를 사용하여 원하는 시간에 프로그램적으로 VI 윈도우 또는 VI 문서를 인쇄하거나 HTML, RTF, 또는 텍스트 파일로 저장합니다.



노트 독립 어플리케이션에서 VI 문서를 인쇄하는 경우, 프론트패널만 인쇄할 수 있습니다.

VI 실행 후 VI의 프론트패널 인쇄하기

수행>>완료 시 인쇄를 선택하여 실행을 끝낼 때 VI의 프론트패널을 인쇄합니다. 또한 파일>>VI 프로퍼티를 선택하고 항목 풀다운 메뉴에서 인쇄 옵션을 선택한 후 VI가 실행을 완료할 때마다 자동적으로 패널 인쇄 확인란에 확인 표시를 합니다.

이 옵션을 선택하는 것은 프론트패널이 활성 윈도우일 때 파일>>윈도우 인쇄를 선택하는 것과 유사합니다.

이 VI를 SubVI로 사용할 경우, LabVIEW는 subVI의 실행이 완료되고 subVI가 호출차로로 복귀하기 전에 인쇄합니다.

SubVI를 사용하여 상위 레벨 VI의 데이터 인쇄하기

어떤 경우, VI가 수행을 끝낼 때마다 VI를 인쇄하지 않으려 할 수 있습니다. 사용자가 버튼을 클릭하거나 테스트 실패와 같이 어떤 조건이 발생할 경우에만 인쇄하려는 경우도 있습니다. 또한, 인쇄의 포맷을 더 많이 컨트롤하거나 컨트롤의 일부분만 인쇄하려 할 수도 있습니다. 이런 경우, 완료 시 인쇄로 설정된 SubVI를 사용할 수 있습니다.

SubVI를 생성하고 LabVIEW에서 인쇄하고자 하는 방향으로 프런트패널을 포맷합니다. 상위 VI에서 수행>>완료 시 인쇄를 선택하는 대신에 subVI에서 선택합니다. 인쇄하고자 할 때, subVI를 호출하고 subVI로 인쇄하고자 하는 데이터를 연결합니다.

리포트 생성 및 인쇄하기

[리포트 생성] VI를 사용하여 리포트를 인쇄하거나 VI 문서 또는 VI가 반환하는 데이터를 포함하는 HTML 리포트를 저장합니다. [VI 패널 또는 문서 인쇄] VI를 사용하여 VI 문서를 포함하는 기본 리포트를 생성합니다. [간단한 텍스트 리포트 생성] VI를 사용하여 VI가 반환하는 데이터를 포함하는 기본 리포트를 생성합니다. 다른 리포트 생성 VI를 사용하여 보다 복잡한 리포트를 생성합니다.

리포트 익스프레스 VI를 사용하여 VI 문서, VI가 반환한 데이터, 그리고 만든이, 회사, 페이지 수 같은 리포트 프로퍼티를 포함하는 미리 포맷된 리포트를 생성합니다.

리포트 생성 VI를 사용하여 다음 작업을 수행합니다:

- 텍스트, 그래픽, 테이블, 또는 VI 문서를 리포트에 추가합니다.
- 텍스트 폰트, 크기, 스타일, 색을 설정합니다.
- 리포트 방향을 가로 또는 세로 방향으로 설정합니다.
- 리포트의 머리글과 바닥글을 설정합니다.
- 여백과 탭을 설정합니다.

추가적인 인쇄 기술

표준 LabVIEW 인쇄 방식으로 부족한 경우, 다음의 추가적인 기술을 사용할 수 있습니다:

- 데이터를 Microsoft Excel과 같은 다른 어플리케이션으로 반출하고 데이터를 파일로 저장한 후 다른 어플리케이션에서 인쇄를 합니다.
- [시스템 실행] VI를 사용합니다.
- (Windows) **ActiveX**를 사용하여 다른 어플리케이션에서 데이터를 인쇄합니다.

프로그램식으로 VI 제어하기

VI 서버를 사용하여 프로그램적으로 VI 및 LabVIEW를 컨트롤할 수 있습니다. 블록 다이어그램, ActiveX 기술, TCP 프로토콜을 통해 VI 서버에 접근할 수 있습니다. 로컬 컴퓨터에서 또는 네트워크를 통하여 원격으로 VI 서버 작업을 수행할 수 있습니다.

프로그램적으로 VI 컨트롤하기 관련 예제는 labview\examples 디렉토리를 참조하십시오.

관련 예제 찾기



관련 정보

[VI 서버의 기능](#) (184 페이지 참조)

[VI 서버 어플리케이션 만들기](#) (185 페이지 참조)

[어플리케이션과 VI 셋팅 변경하기](#) (186 페이지 참조)

[다이나믹하게 VI 로딩 및 호출하기](#) (170 페이지 참조)

[비동기적으로 VI 호출하기](#) (171 페이지 참조)

[비동기적으로 VI 호출하기](#) (934 페이지 참조)

[VI 서버 설정하기](#) (942 페이지 참조)

[VI 호출 설정하기](#) (942 페이지 참조)

[SubVI로부터 프런트패널 객체를 프로그램적으로 컨트롤하기](#) (943 페이지 참조)

[VI 서버 어플리케이션 생성하기](#) (945 페이지 참조)

다이나믹하게 VI 로딩 및 호출하기

정적으로 링크된 SubVI 호출을 사용하는 대신 다이나믹하게 VI를 로드할 수 있습니다. 정적으로 링크된 SubVI는 호출자 VI의 블록 다이어그램에 직접 놓는 것입니다. 이것은 호출자 VI가 로드될 때 동시에 로드됩니다.

정적으로 링크된 SubVI와는 달리, 다이나믹하게 로드된 VI는 호출자 VI가 [VI 참조 열기]로 해당 VI를 로드하기 전에는 로드되지 않습니다. 호출자 VI가 클 경우, VI를 다이나믹하게 로드하면 로드 시간과 메모리를 절약할 수 있습니다. VI는 호출자 VI가 필요로 하기 전에는 로드되지 않으며 작업이 완료된 후 VI를 메모리에서 해제할 수 있기 때문입니다. 또한, [SubVI를 로드할 때 설정](#)을 하기 위해 VI 호출 설정 대화 상자를 사용할 수 있습니다.

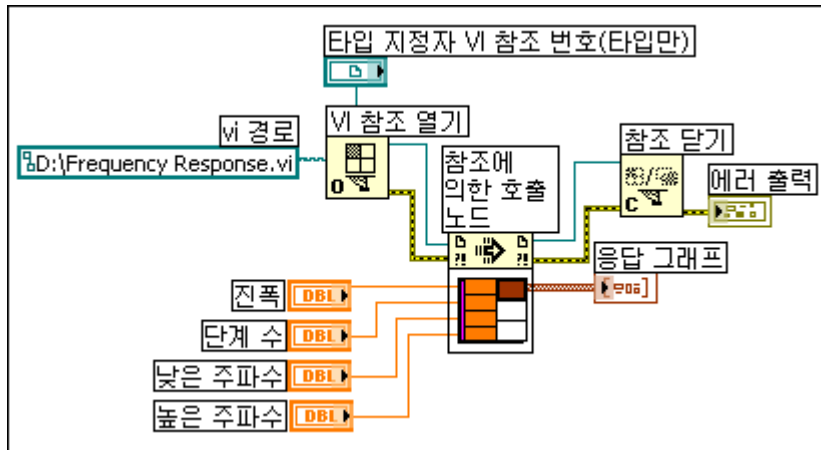
참조에 의한 호출 노드와 엄격한 타입의 VI 참조 번호

참조에 의한 호출 노드나 비동기화 호출 시작 노드를 사용하여 VI를 다이나믹하게 호출합니다. 두 노드는 모두 참조 입력에 의해 지정된 SubVI로 호출을 시작합니다. 그러나 비동기화 호출 시작 노드는 호출 VI에서 데이터 흐름이 지속되는 동안 [비동기적으로 subVI를 호출](#)합니다.

참조에 의한 호출 노드는 엄격한 타입의 VI 참조 번호를 필요로 합니다. 엄격한 타입의 VI 참조는 호출하는 VI와 그 커넥터 팬을 모두 식별합니다. 이는 VI와 영구적인 연관을 만들지 않거나 이름 및 위치와 같은 VI 정보를 포함하지 않습니다. VI를 연결하는 것과 같은 방법으로 참에 의한 호출 노드 입력과 출력을 연결할 수 있습니다.

다음 블록 다이어그램은 참조에 의한 호출 노드를 사용하여 다이나믹하게 [주파수 응답] VI를 어떻게 호출하는지 보여줍니다. 참조에 의한 호출 노드는 프로퍼티 노드와 인보크

노드에서 사용하는 함수와 비슷한 [VI 참조 열기] 및 [참조 닫기] 함수의 사용을 필요로 합니다.



엄격한 타입의 참조 번호에 지정한 VI는 커넥터 팬 정보만을 제공합니다. 즉, 참조 번호와 VI 사이에는 영구적인 관계가 생성되지 않습니다. 특히 VI 커넥터 팬 선택과 선택된 VI의 참조 번호 열기를 혼동하지 마십시오. [VI 참조 열기] 함수의 vi 경로 입력을 사용하여 특정한 VI를 지정합니다.

비동기적으로 VI 호출하기

SubVI나 참조에 의한 호출 노드로 타겟 VI에 대해 표준 호출을 수행하면, 타겟 VI가 결과를 반환할 때까지 해당 노드에서 데이터 흐름이 멈추고, 결과가 반환되면 노드의 출력 터미널로부터 데이터 흐름이 이어집니다.

반면, SubVI의 비동기화 호출은 호출 VI에서 데이터 흐름이 지속되는 상태에서 SubVI의 실행을 시작합니다. 호출 VI가 SubVI의 결과를 즉시 필요로 하지 않으면 SubVI를 비동기적으로 호출하는 것이 좋습니다. 그런 경우, 비동기화 호출 시 SubVI가 호출 VI와 병렬 실행되어 호출 VI의 실행 시간이 현저하게 개선될 수 있습니다. VI 비동기화 호출이 유익하게 쓰일만한 프로그래밍 예는 다음 리스트를 참조하십시오:

- 실행을 지속하면서 SubVI에 작업 할당하기
- 같은 타겟 VI의 여러 인스턴스에 단일 작업 배분하기, 특히 실행 시까지 작업에 필요한 인스턴스의 수를 알 수 없을 때
- 병렬 실행과 결과의 동시 수집이 필요한 개별 VI로 컨트롤되는 구성요소를 지닌 복잡한 사용자 인터페이스 시작하기

labview\examples\Application Control\VI Server\Asynchronous Call By Reference

VI 비동기화 호출 생성의 개요

LabVIEW에서, 모든 VI 비동기화 호출은 다음 단계의 조합으로 이루어집니다.

1. VI 참조가 비동기적으로 호출되도록 준비—참조를 [VI 참조 열기] 함수로 얻을 때 비동기화 실행을 대비해 VI 참조를 준비하고 비동기화 옵션 플래그 0x80 또는 0x100 중 하나를 옵션 입력의 일부로 포함시킵니다. 이 단계를 통해 LabVIEW는 비동기화 호출에 필요한 리소스를 할당합니다.
2. VI 비동기화 호출 시작—VI 참조를 비동기화 호출 시작 노드에 연결하여 타겟 VI의 실행을 시작합니다. 타겟 VI가 실행되는 동안, 호출 VI에서 데이터 흐름이 지속됩니다.
3. (옵션) VI 비동기화 호출의 결과 수집—비동기적으로 호출하는 타겟 VI의 출력이 어플리케이션에서 필요한 경우, VI 참조를 비동기화 호출 기다림 노드에 연결하여 결과를 수집합니다. 비동기화 호출 기다림 노드는 관련 subVI가 실행을 마칠 때까지 또는 비동기화 호출 기다림 노드가 타임아웃될 때까지 기다립니다.
4. VI 참조 닫기—[참조 닫기] 함수를 사용하여 VI 참조의 사용이 끝났음을 LabVIEW에 알립니다. 그러면 LabVIEW가 관련 리소스를 해제할 수 있습니다.

VI 비동기화 호출의 타입

VI 비동기화 호출을 실행하려면, 다음 사용 모델 중 사용자 용도에 맞는 것이 무엇인지 결정해야 합니다. 각 모델의 실행에 대한 추가 정보는 다음 토픽을 참조하십시오:

- **호출 및 무시**—호출 VI가 subVI의 실행을 시작하지만 subVI의 출력을 필요로 하지는 않습니다.
- **호출 및 수집**—호출 VI가 subVI의 실행을 시작하며, 나중에 결과를 수집할 필요가 있습니다.

병렬 실행을 위해 동일한 VI로 여러 호출을 비동기적으로 시작

LabVIEW의 VI 비동기화 호출 관리 방법: 비동기화 호출 풀

비동기적으로 VI를 호출하는 대부분의 경우 LabVIEW가 어떻게 VI 비동기화 호출을 관리하는지 알아야 할 필요는 없지만, 비동기화 호출 풀을 이해하면 성능을 컨트롤하고 타임링 결정성이 있는 어플리케이션을 디자인하는데 도움이 됩니다. 비동기화 옵션 플래그 하나를 [VI 참조 열기] 함수의 옵션 입력의 일부로 포함하여 VI 참조가 비동기적으로 실행될 수 있도록 준비하면, LabVIEW는 해당 VI의 개별 호출에 대한 데이터를 추적하는 데이터 공간 풀을 생성합니다. LabVIEW가 단일 데이터 공간이 아닌 데이터 공간 풀을 VI 참조에 연관시키기 때문에, 같은 VI 참조에서 여러 비동기화 호출을 시작할 수 있습니다. 비동기화 호출 시작 노드를 사용하여 VI 참조로 비동기화 호출을 시작할 때마다, LabVIEW는 데이터 공간 풀에서 공간 하나를 확보합니다. VI가 실행을 마치면, LabVIEW는 데이터 공간에 출력을 저장하고 VI 호출을 완료로 표시합니다. 비동기화 호출 기다림 노드가 호출된 VI의 출력을 기다리는 경우, LabVIEW는 데이터 공간으로부터 결과를 반환하고, 이로써 같은 VI에 호출을 더 할 수 있도록 데이터 공간이 비워집니다.

비동기화 호출 풀의 성능적 영향

각 VI 참조에 대해, LabVIEW는 타겟 컴퓨터의 각 CPU 코어에 있는 비동기화 호출 풀에서 하나의 데이터 공간을 생성합니다. 사용 가능한 데이터 공간보다 더 많은 동시 호출을 타겟 VI로 보내기 시작하면, LabVIEW는 데이터 공간을 더 할당해서 호출 풀의 크기를 증가시킵니다. 데이터 공간을 할당하는 프로세스가 정해진 시간 안에 끝나는 것이 아니기 때

문에, 호출 VI는 비동기화 호출 시작 노드 어디에서나 할당을 유발하는 예상치 못한 지연을 겪을 수 있습니다.

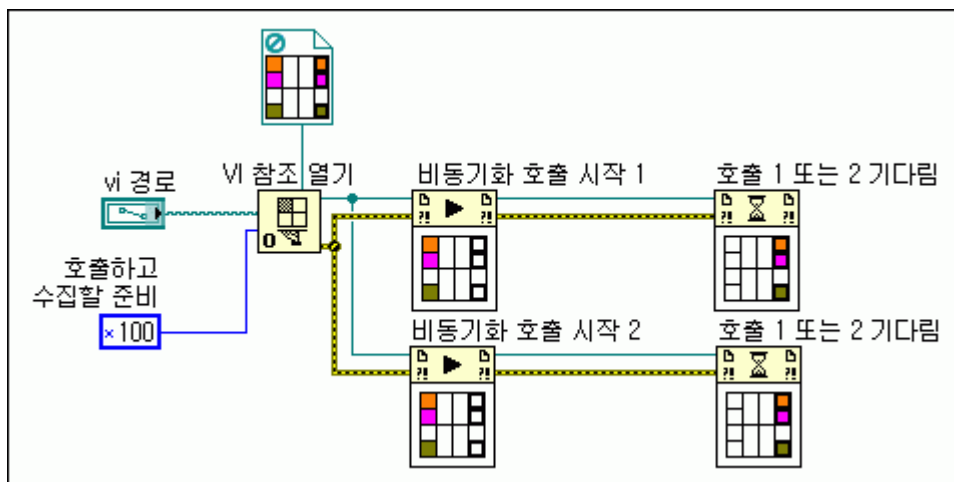
비동기화 호출 시작 노드에서 예상치 못한 지연 방지하기

Real-Time OS에서 실행되어야 하는 VI의 예상치 못한 지연을 방지하려면, 비동기화 호출을 시작하기 전에 비동기화 호출 풀 채우기 메소드를 호출하여 주어진 VI 참조에 대해 호출 풀에서 데이터 공간의 수를 미리 할당합니다. 데이터 공간을 미리 할당하면 비동기화 호출 시작 노드에서 지연이 발생되지 않게 하고 지연을 프로그램의 시작으로 제한합니다.

비동기화 호출 풀이 성능에 미치는 영향에 대한 더 자세한 정보는 labview\examples\Application Control\VI Server\Asynchronous Call By Reference 디렉토리의 Populating the Asynchronous Call Pool VI를 참조하십시오. 관련 예제 찾기

같은 VI에 대한 여러 호출의 출력 수집에 미치는 비동기화 호출 풀의 영향

VI 비동기화 호출의 결과를 수집하려면, 호출된 VI의 참조를 비동기화 호출 기다림 노드로 전달해야 합니다. 비동기화 호출 기다림 노드는 특정한 데이터 공간의 참조가 아닌 VI의 참조를 받기 때문에, 노드는 참조된 VI에 대한 특정 호출의 출력을 수집할 수 없습니다. 그 대신 노드는 해당 VI에 대해 완료된 모든 호출의 출력을 반환합니다. 이 동작에 대한 더 자세한 설명은 다음 블록 다이어그램을 참조하십시오.



라벨에서 보이는 것처럼, 여러 비동기화 호출 기다림 노드가 동일한 VI 참조에 의존할 때, 어떤 노드가 해당 VI 참조에 대한 특정 호출의 출력을 반환할지 예상할 수 없습니다. 앞의 다이어그램에서, 위쪽에 있는 비동기화 호출 기다림 노드는 비동기화 호출 시작 1에 연결되어 있지만 비동기화 호출 시작 2의 출력을 반환할 수 있습니다. 같은 VI에 대해 여러 호출을 실행하려 하지만 비동기화 호출 시작 노드를 특정한 비동기화 호출 기다림 노드와 짝을 지워야 하는 경우, 다음 두 솔루션 중 하나를 사용합니다.

- 실행하려는 모든 비동기화 호출에 대해 타겟 VI에 대한 고유한 VI 참조를 생성합니다. 비동기화 호출 기다림 노드와 비동기화 호출 시작 노드가 고유한 VI 참조를 공유하는 경우, 이 두 노드는 언제나 동일한 VI 인스턴스를 참조합니다. 이 솔루션은 **고유한 VI 참조에 대한 여러 호출의 출력을 수집**하는 것과 유사합니다.

- 타겟 VI가 고유한 숫자나 문자열을 입력으로 받아들이고 동일한 식별자를 출력으로 반환하도록 수정합니다. 타겟 VI를 비동기적으로 호출할 때, 특정한 비동기화 호출 시작 노드로 전달하는 식별자와 비동기화 호출 기다림 노드가 반환하는 식별자를 비교합니다. 그것이 일치하면, 비동기화 호출 기다림 노드가 반환한 호출을 알 수 있습니다.

콜백 VI

콜백 VI는 사용자가 지정한 ActiveX 또는 .NET 이벤트를 핸들링하기 위해 작성한 코드를 포함합니다. ActiveX 컨트롤 또는 .NET 객체가 등록된 이벤트를 생성하면 **콜백 VI를 생성**하여 해당 이벤트를 처리해야 합니다. 콜백 VI는 이벤트가 발생하면 실행됩니다. 사용자가 콜백 VI를 생성할 때, LabVIEW는 **재호출 VI**를 생성하여 사용자가 이벤트를 처리하기 위해 이를 열고 편집할 수 있게 합니다. 콜백 VI에는 다음이 포함됩니다.

- 이벤트 공통 데이터는 다음과 같은 원소들을 포함합니다.
 - 이벤트 소스는 LabVIEW, ActiveX 또는 .NET과 같은 이벤트의 소스를 지정하는 숫자형 컨트롤입니다. 1의 값은 ActiveX 이벤트를 나타냅니다. 2의 값은 .NET 이벤트를 나타냅니다.
 - 이벤트 타입은 어떤 이벤트가 발생했는지 지정합니다. 이는 사용자 인터페이스 이벤트의 경우 열거형 타입이며 ActiveX, .NET 및 기타 이벤트 소스의 경우 32비트 부호 없는 정수 타입입니다. ActiveX 이벤트의 경우, 이벤트 타입은 등록된 이벤트의 메소드 코드 또는 ID를 나타냅니다. .NET의 경우 이 원소를 무시해도 됩니다.
 - 타임스탬프는 이벤트가 생성된 시간을 밀리초 단위로 지정하는 타임스탬프입니다.
- 컨트롤 참조는 이벤트가 발생한 ActiveX, 오토메이션 참조 번호 또는 .NET 객체에 대한 참조입니다.
- 이벤트 데이터는 콜백 VI가 처리하는 이벤트에 특정한 파라미터의 클러스터입니다. LabVIEW는 [이벤트 콜백 등록] 함수로부터 이벤트를 선택할 때 적절한 이벤트 데이터를 결정합니다. 이벤트에 관련된 데이터가 없는 경우, LabVIEW는 콜백 VI에서 이 컨트롤을 생성하지 않습니다.
- 이벤트 데이터 출력은 콜백 VI가 처리한 이벤트에 대한 변경 가능한 파라미터의 클러스터입니다. 이 원소는 ActiveX 또는 .NET 이벤트가 출력 파라미터를 가지고 있을 때에만 사용 가능합니다.
- (옵션) 사용자 파라미터는 .NET 또는 ActiveX 객체가 이벤트를 생성할 때 콜백 VI에 전달하려는 데이터입니다.



노트

사용하려는 VI의 커넥터 팬이 이벤트 데이터의 커넥터 팬과 일치하는 한 기존의 VI를 콜백 VI로 사용할 수 있습니다. 콜백 VI는 재호출이어야 하고 콜백 VI에 대한 참조는 엄격한 타입이어야 합니다.

콜백 VI가 중단 없이 실행될 수 있도록 LabVIEW는 콜백 VI의 실행이 정지되거나 사용자가 모달 윈도우를 로드할 때까지 OS 메시지의 처리를 지연시킵니다. LabVIEW가 OS 메시지의 처리를 지연하면, LabVIEW 프론트패널을 편집하거나 작동시킬 수 없습니다. 모달 윈도우는 윈도우를 닫거나 다른 모달 윈도우를 열 때까지 활성화 상태로 있거나 다른 LabVIEW 윈도우보다 상위에 있는 윈도우 타입입니다. 모달 윈도우가 열려 있는 동안에

는 다른 윈도우와 상호작용을 할 수 없습니다. 대부분의 LabVIEW 대화 상자는 모달 윈도우입니다.

다른 프로세스가 실행 중일 때에는 LabVIEW 콜백 VI나 DLL에서 모달 윈도우가 아닌 윈도우를 열 수 없습니다. 콜백 VI 또는 DLL에서의 모달이 아닌 윈도우 호출에 대한 추가적인 정보는 [프로그램적으로 모달이 아닌 윈도우 호출하기](#)를 참조하십시오.



노트 ActiveX 객체는 ActiveX 콜백 VI를 동시에 호출합니다. 이로 인해 콜백을 실행하는 동안 객체가 다른 코드를 실행할 수 없어서 교착이 발생할 수도 있습니다. 콜백 다이어그램과 다른 코드 사이의 상호작용을 주의깊게 고려해야 합니다. 예를 들어, 서브루틴이 이미 실행 중인 동안에 ActiveX 콜백 VI를 호출하는 경우에 ActiveX 콜백 VI 안에서의 재호출이 아닌 서브루틴을 호출하면 응답 없음 상태가 될 수 있습니다.

VI에서 데이터를 E-메일하기

[SMTP E-메일] VI를 사용하면, Simple Mail Transfer Protocol (SMTP)을 통해 데이터와 파일이 첨부된 E-메일을 보낼 수 있습니다. [SMTP E-메일] VI는 정보를 받는 용도로 사용할 수 없습니다. [SMTP E-메일] VI는 여러 목적의 인터넷 메일 확장자(MIME) 포맷을 이용하여 메시지를 인코딩합니다. 이 포맷으로 2진 데이터 파일을 포함한 여러 개의 파일을 한 통의 E-메일로 보낼 수 있습니다.

수신자의 E-메일 주소와 함께 SMTP 서버의 웹 주소를 반드시 알아야 합니다. [E-메일 보내기] 익스프레스 VI를 사용할 때에는, 발신 메일 서버 (SMTP) 텍스트 박스에 SMTP 서버 주소를 입력하여 메일 서버를 지정할 수 있습니다. 서버 주소는 [SMTP E-메일] VI를 실행하는 컴퓨터로부터 E-메일 서비스를 요청할 수 있는 외부 서버 컴퓨터의 호스트명 또는 IP 주소이어야 합니다. 어느 메일 서버를 사용해야 할지 알고 있지 않다면, 유효한 서버의 이름을 위해 네트워크 관리자에게 연락하기 바랍니다. 유효한 메일 서버가 정해지면, [SMTP E-메일] VI는 서버와 연결을 개통하고 수신인이 누구며 E-메일의 내용을 설명하는 명령을 서버에 전달합니다. 서버는 모든 수신인에게 메시지를 전송하거나 다른 SMTP 서버로 그 메시지를 전달합니다.

다른 방법으로, 서버 주소를 [핸들 열기] VI에 연결하여 메일 서버를 지정할 수도 있습니다. [핸들 열기] VI는 다른 SMTP 이메일 VI에 연결하여 이메일 스펙을 설정할 수 있도록 SMTP 클라이언트 핸들을 생성합니다. [핸들 닫기] VI를 사용하여 핸들을 삭제하고 시스템 리소스를 해제합니다. SMTP E-메일 VI를 사용하는 예제를 확인하려면 labview\examples\Data Communication\Protocols\SMTP\Send Email using SMTP Client.vi를 참조하십시오.

문자 세트와 UTF-8 인코딩

LabVIEW는 E-메일 메시지에서 8비트 Unicode Transmission Format (UTF-8) 문자 인코딩을 지원합니다. UTF-8은 전세계의 다양한 언어의 문자를 나타내기 위해 바이트의 그룹을 사용하는 손실이 없고 길이가 가변적인 문자 인코딩 방식입니다. 시스템 문자열 세트가 이미 UTF-8 포맷으로 인코딩되어 있지 않은 경우, SMTP E-메일 VI는 이메일을 전송하기 전 시스템 문자열 세트에서 인코딩된 입력 문자열을 UTF-8 포맷으로 변환합니다. 이 변환을 이용하면 다양한 OS에서 읽을 수 있는 E-메일 메시지를 보낼 수 있습니다.

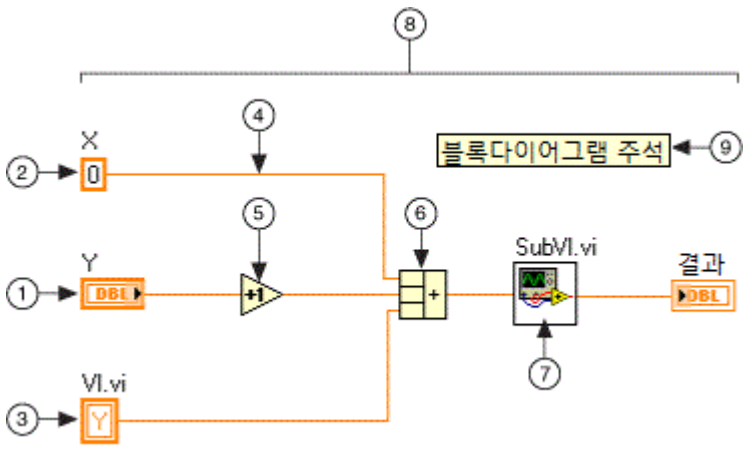
VI와 일반적인 VI 객체의 조직

VI 스크립팅을 사용하는 VI를 생성하거나 수정하려면, VI를 구성하는 객체들이 VI 서버 클래스 계층구조 및 가까운 객체와 어떤 관련이 있는지 이해해야 합니다. 이 토픽에서는 다양한 용도의 편의를 위해 몇몇 일반적인 VI에 대한 세 가지 유형의 필수 정보를 제시합니다.

- VI와 일반적인 VI 객체의 조직 이 토픽의 이미지는 VI 스크립팅으로 조작이 가능한 서로 다른 타입의 객체뿐만 아니라 객체의 일부 구성요소 및 상호관계를 보여줍니다. 예를 들어, For 루프는 여러 구성요소를 포함하는 일반적인 VI 객체입니다. For 루프로 들어가는 새 와이어를 생성하려는 경우, 루프 자체, 루프 터널, 루프 터널의 외부 및 내부 터미널과 같은 몇 가지 구성요소들과 상호작용해야 합니다. For 루프 및 기타 일반적인 VI 객체의 조직을 조사하여, 특정 스크립팅 동작이 영향을 줄 수 있는 모든 객체 원소를 식별할 수 있습니다.
- 일반 VI 객체의 VI 서버 클래스 몇 가지 스크립팅 함수에서는 상호작용하려는 객체의 클래스 지정이 필요합니다. 이 토픽에서 각 조직 테이블의 클래스 계층구조 열은 관련 다이어그램 내의 객체에 대한 클래스 정보를 보여줍니다. 클래스 지정자 상수를 사용하여 이 정보를 함수나 VI로 전달합니다. 클래스 계층구조 열의 정보를 사용하여 다음 태스크에서 클래스를 지정합니다:
 - [새 VI 객체] 함수로 새 객체 생성하기
 - [VI 객체 참조 열기] 함수로 기존 객체에 대한 참조 얻기
 - [G객체의 탐색] VI로 특정 클래스 객체의 인스턴스 검색하기
 - [보다 구체적인 클래스로] 함수로 참조 다운캐스트하기
 - 특정 객체에 대해 VI 서버 클래스 계층구조에서 어떤 프로퍼티와 메소드가 사용 가능한지 식별하기
- 일반 VI 객체의 특정 인스턴스에 대한 참조를 반환하는 프로퍼티 일부 스크립팅 작업에서는 한 객체에 대한 참조를 사용해야만 다른 객체에 대한 참조를 얻을 수 있습니다. 이런 경우, 원래 객체의 어떤 프로퍼티가 원하는 참조를 반환하는지 알아야 합니다. 각 조직 테이블의 객체 참조 반환 열은 객체 타입:프로퍼티의 표기법을 사용하여 이러한 프로퍼티 정보를 보여줍니다. 예를 들어, For 루프:루프 카운트 표기는 For 루프 객체에 대한 참조에서 루프 카운트 프로퍼티를 호출 중임을 나타냅니다. 이를 위해, 우선 For 루프에 대한 참조를 얻어야 합니다. 그런 다음 For 루프 참조를 프로퍼티 노드에 연결하고 해당 프로퍼티 노드에 대해 루프 카운트 프로퍼티를 선택합니다. 객체 참조 반환 열의 정보를 사용하여 다음 작업에 필요한 참조를 얻습니다:
 - VI 탐색하기
 - VI 편집하기
 - 새 VI 객체 간 연결하기

블록다이어그램 조직

다음 이미지는 VI 스크립팅으로 생성하거나 수정할 수 있는 몇몇 일반적인 블록다이어그램 객체를 보여줍니다.

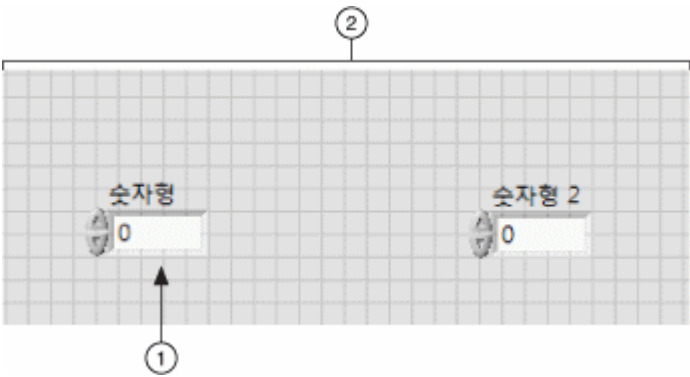
		
1 컨트롤/인디케이터 터미널	4 와이어	7 SubVI
2 숫자형 상수	5 [증가] 함수	8 최상위 레벨 다이어그램
3 로컬 변수	6 [복합 연산] 함수	9 블록다이어그램 주석

다음 테이블은 위 이미지에서 객체에 대한 참조를 얻는데 사용 가능한 클래스 및 프로퍼티를 나열합니다.

클래스 계층 구조	다음 객체 참조 반환	
1	G객체 >> 터미널 >> 컨트롤터미널	컨트롤:터미널, 다이어그램:모든 객체[]
2	G객체 >> 상수 >> 숫자형상수	다이어그램:모든 객체[]
3	G객체 >> 노드 >> 로컬	컨트롤:로컬 변수[]
4	G객체 >> 와이어	터미널:연결된 와이어, 다이어그램:와이어[]
5	G객체 >> 노드 >> 함수	다이어그램:노드[], 다이어그램:모든 객체[]
6	G객체 >> 노드 >> 크기 조정 가능한 함수 >> 복합 연산	다이어그램:노드[], 다이어그램:모든 객체[]
7	G객체 >> 노드 >> SubVI	다이어그램:SubVI[], 다이어그램:노드[], 다이어그램:모든 객체[]
8	G객체 >> 추상 다이어그램 >> 다이어그램 >> 최상위 다이어그램	VI:블록다이어그램
9	G객체 >> 장식 >> 텍스트	다이어그램:장식[], 다이어그램:모든 객체[]

프론트패널 조직

다음 이미지는 일부 일반적인 프론트패널 객체를 보여줍니다.

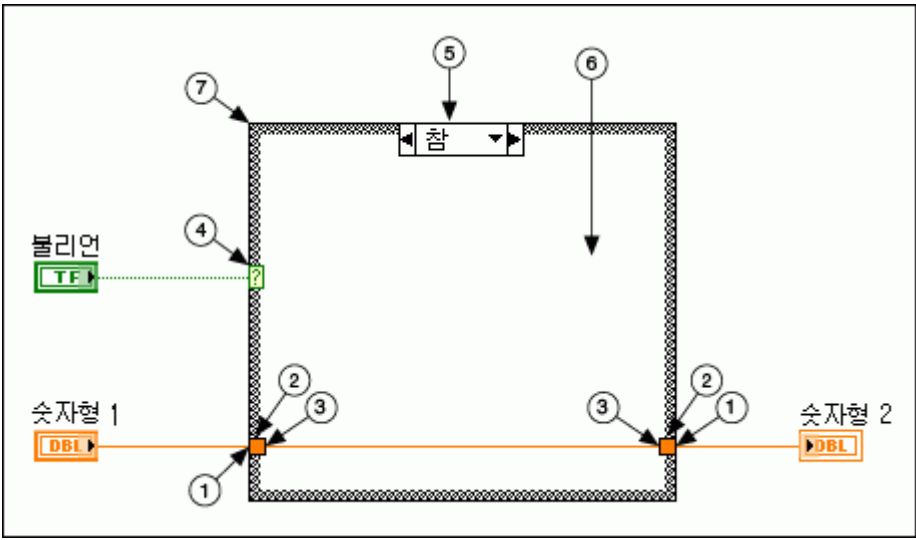
	
1 컨트롤/인디케이터	2 프론트패널

다음 테이블은 위 이미지에서 객체에 대한 참조를 얻는데 사용 가능한 클래스 및 프로퍼티를 나열합니다.

클래스 계층구조	다음 객체 참조 반환	
1	G객체>>컨트롤>>숫자형	패널:컨트롤[], 컨트롤 터미널:컨트롤
2	G객체>>패널	VI:프론트패널

케이스 구조 조직

다음 이미지는 케이스 구조를 구성하는 다양한 객체를 보여줍니다.

1 터미널	4 선택자 터널	7 케이스 구조
2 터널	5 선택자 라벨	
3 내부 터미널	6 다이어그램	

다음 테이블은 위 이미지에서 객체에 대한 참조를 얻는데 사용 가능한 클래스 및 프로퍼티를 나열합니다.

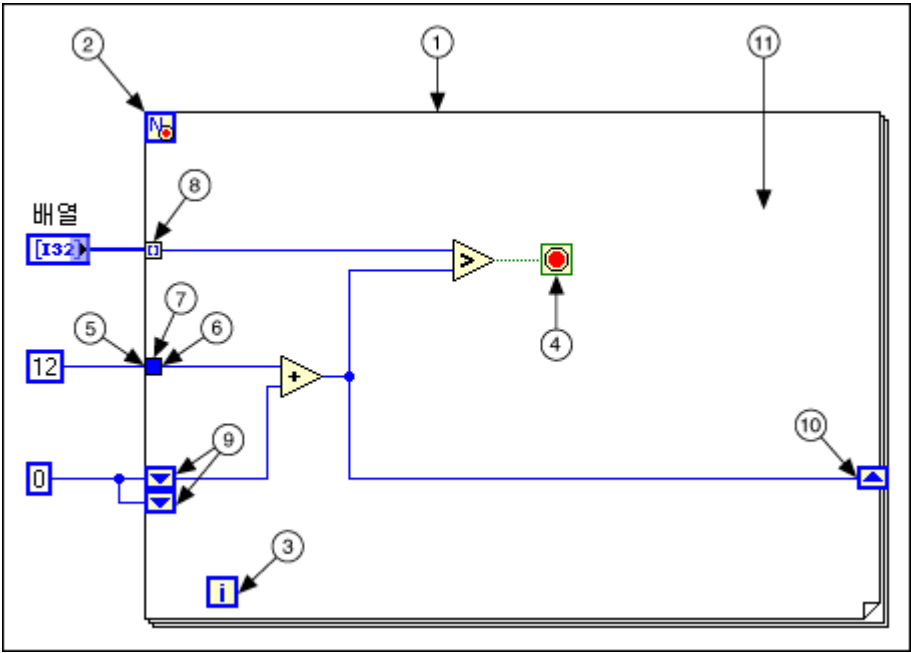
클래스 계층 구조	다음 객체 참조 반환	
1	G객체»터미널»외부 터미널	터널:외부 터미널
2	G객체»터널	구조:터널[]
3	G객체»터미널»내부 터미널	터널:내부 터미널[]
4	G객체»터널»조건 터널»선택자 터널	구조:터널[], 케이스 구조:선택자
5	해당 없음	케이스 구조:프레임 이름 *
6	G객체»추상 다이어그램»다이어그램	구조:다이어그램[], 멀티프레임 구조:프레임[]
7	G객체»노드»구조»멀티프레임 구조»케이스 구조	다이어그램:모든 객체[], 다이어그램:노드[]

* LabVIEW는 케이스 구조에서 선택자 라벨에 대한 참조는 제공하지 않습니다. 케이스 구조:프레임 이름 프로퍼티는 다른 케이스와 관련된 다이어그램에 대한 참조 배열보다는 문자열 배열을 반환합니다. 케이스 구조:프레임 인덱스 얻기 메소드와 함께 이러한 이름 문자열을 사용하여, 멀티프레임 구조:프레임[] 프로퍼티에 의해 반환된 배열 내에 있는 특정 다이어그램의 인덱스를 얻을 수 있습니다.

Structures - Case Structure VI는 VI 스크립팅을 사용하여 케이스 구조 조직의 구성요소에 접근하는 방법을 보여줍니다.

For 루프 조직

다음 이미지는 For 루프를 구성하는 다양한 객체를 보여줍니다.



1 For 루프

5 외부
터미널

9 왼쪽
시프트
레지스
터

2 카운트 터미널 (N)

6 내부
터미널

10
오른쪽
시프트
레지스
터

3 터미널 (i)

7 루프
터널

11
다이어
그램

4 조건 터미널

8
오토인
덱싱
루프
터널

다음 테이블은 위 이미지에서 객체에 대한 참조를 얻는데 사용 가능한 클래스 및 프로퍼티를 나열합니다.

클래스 계층 구조	다음 객체 참조 반환	
1	G객체>>노드>>구조>>루프>>For 루프	다이어그램:모든 객체[], 다이어그램:노드[]
2	G객체>>터널	For 루프:루프 카운트

3	G객체»터미널	루프:루프 카운터
4	G객체»터미널	For 루프:루프 끝 참조
5	G객체»터미널»외부 터미널	터널:외부 터미널
6	G객체»터미널»내부 터미널	터널:내부 터미널[]
7	G객체»터널»루프 터널	구조:터널[]
8	G객체»터널»루프 터널	구조:터널[], 루프 터널:인덱스 모드 = 1 (오토인덱싱일 때)
9	G객체»터널»왼쪽 시프트 레지스터	오른쪽 시프트 레지스터:왼쪽 레지스터[]
10	G객체»터널»오른쪽 시프트 레지스터	루프:시프트 레지스터[], 왼쪽 시프트 레지스터:오른쪽 시프트 레지스터
11	G객체»추상 다이어그램»다이어그램	루프:다이어그램, 구조:다이어그램[]

Structures - For Loop VI는 VI 스크립팅을 사용하여 For 루프 조직의 구성요소에 접근하는 방법을 보여줍니다.

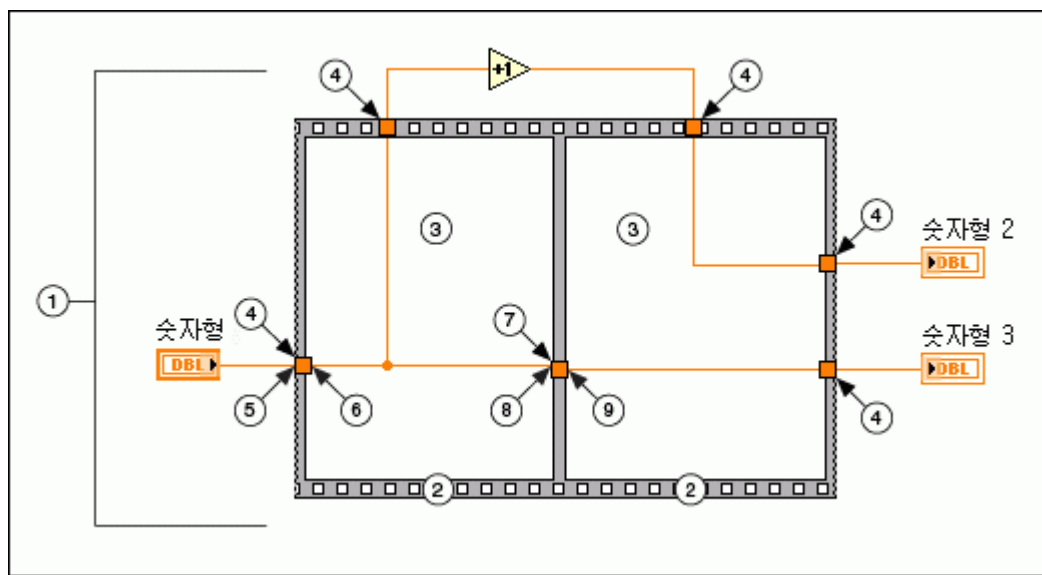
플랫 시퀀스 구조 조직



노트

플랫 시퀀스 구조에서는 한 개의 프레임을 통해 구조의 밖으로 와이어를 뺀 후, 다른 프레임을 통해 다시 와이어를 구조 안으로 연결하는 것이 가능합니다. 이러한 데이터 흐름은 플랫 시퀀스 구조만이 가지고 있는 특징으로, 이와 같은 이유로 VI 서버 클래스는 플랫 시퀀스 구조에서 모두 다릅니다.

다음 이미지는 플랫 시퀀스 구조를 구성하는 다양한 객체를 보여줍니다.



1 플랫폼 시퀀스

4 플랫시퀀스바깥쪽터널

7 플랫시퀀스안쪽터널

2 플랫폼 시퀀스 프레임

5 플랫폼 시퀀스 바깥쪽 터널의 외부 터미널

8 플랫폼스핀아웃의원쪽터미널

3 다이어그램	6 플랫폼 시퀀스 바깥쪽 터널의 내부 터미널	9 플랫폼 시퀀스 안쪽 터널의 오른쪽 터미널

다음 테이블은 위 이미지에서 객체에 대한 참조를 얻는데 사용 가능한 클래스 및 프로퍼티를 나열합니다.

클래스 계층 구조	다음 객체 참조 반환	
1	G객체 >> 플랫폼 시퀀스	다이어그램: 모든 객체[]
2	일반 >> 플랫폼 시퀀스 프레임	플랫폼 시퀀스: 프레임[], 다이어그램: 발생 순서[], 플랫폼 시퀀스 바깥쪽 터널: 프레임, 플랫폼 시퀀스 안쪽 터널: 왼쪽 프레임, 플랫폼 시퀀스 안쪽 터널: 오른쪽 프레임
3	G객체 >> 추상 다이어그램 >> 다이어그램	플랫폼 시퀀스 프레임: 다이어그램
4	G객체 >> 플랫폼 시퀀스 터널	플랫폼 시퀀스 프레임: 바깥쪽 터널[]
5	G객체 >> 터미널	플랫폼 시퀀스: 터미널[], 플랫폼 시퀀스 바깥쪽 터널: 외부 터미널
6	G객체 >> 터미널	플랫폼 시퀀스 바깥쪽 터널: 내부 터미널
7	G객체 >> 플랫폼 시퀀스 안쪽 터널	플랫폼 시퀀스 프레임: 왼쪽 안쪽 터널[], 플랫폼 시퀀스 프레임: 오른쪽 안쪽 터널[]
8	G객체 >> 터미널	플랫폼 시퀀스 터널: 왼쪽 터미널
9	G객체 >> 터미널	플랫폼 시퀀스 터널: 오른쪽 터미널

Structures - Flat Sequence Structure VI는 VI 스크립팅을 사용하여 플랫폼 시퀀스 구조 조직의 구성요소에 접근하는 방법을 보여줍니다.

추가 정보

이 토픽에서는 가장 일반적인 VI 객체의 일부만 집중적으로 다루어집니다. 여기서 다루지 않은 객체에 대한 정보는 다음 리소스를 참고하십시오:

- **일반:클래스 이름** — 이 프로퍼티를 사용하여 모든 VI 프로젝트의 클래스 이름을 찾습니다. 그런 다음 VI 서버 클래스 계층구조 내에서 클래스 이름을 찾아 객체에 대한 전체 클래스 계층구조를 식별합니다.
- **VI 서버 클래스 계층구조** — 특정 객체의 클래스를 아는 경우, VI 서버 클래스 계층구조를 통해 해당 객체의 다른 구성요소에 대한 참조를 제공하는 프로퍼티나 메소드를 알 수 있습니다.

VI 스크립팅

필수: VI 스크립팅

여러 일반 스크립팅 태스크의 수행 방법을 알아봅니다.

- VI 스크립팅 활성화하기에 대한 설명
- 타겟 VI 검사하기, 편집하기 및 생성하기에 대한 설명과 예제
- VI 스크립팅을 사용한 새 VI 생성, 블록다이어그램에 객체 추가, 추가한 객체의 위치 변경, 객체 간 연결에 대한 기본 길라잡이



관련 정보

[VI 스크립팅 활성화하기](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅 활성화하기](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅 길라잡이](#) (978 페이지 참조)

VI 서버의 기능

VI 서버는 함수 세트로서, 이를 통해 사용자가ダイナ믹하게 프론트패널 객체, VI, LabVIEW 환경을 컨트롤할 수 있습니다. 관련 함수는 어플리케이션 컨트롤 서브팔레트에 있습니다. VI 서버를 사용하여 다음 프로그램 작업을 수행합니다:

- VI를 열 때 모든 subVI를 로드하기보다, 다른 VI가 호출해야 할 때에만 메모리에 **다이나믹하게 VI를 로드**합니다.
- 원격에서 VI를 호출합니다.
- LabVIEW **어플리케이션 인스턴스**를 설정하여 웹 상에서 LabVIEW의 다른 인스턴스로부터 호출할 수 있는 VI를 반출하는 서버로 만듭니다. 예를 들어, 원격으로 데이터를 수집하고 로그하는 데이터 수집 어플리케이션이 있는 경우, 때때로 로컬 컴퓨터에서 데이터를 샘플링할 수 있습니다. 옵션 대화 상자에서 VI 서버 환경을 변

경하여, 일부 VI를 웹에서 접근할 수 있도록 만들 수 있습니다. 이렇게 되면 최신 데이터 전송이 다이나믹 VI 호출만큼 쉬워집니다. VI 서버가 네트워킹 세부사항을 처리합니다. 또한, VI 서버는 다양한 플랫폼에서 작동하므로, 클라이언트와 서버는 서로 다른 플랫폼에서도 실행됩니다.

- LabVIEW와 VI의 프로퍼티를 편집합니다. 예를 들어, 다이나믹하게 VI 윈도우의 위치를 결정하거나 프런트패널을 스크롤하여 일부가 보이도록 만들 수 있습니다. 또한, 변경된 내용을 프로그램적으로 디스크에 저장할 수 있습니다.
- 각 VI의 VI 프로퍼티 대화 상자를 일일이 사용하는 것보다는 여러 VI의 프로퍼티를 업데이트합니다.
- 버전 번호와 개정판과 같은 어플리케이션 인스턴스 정보를 불러옵니다. 또한, LabVIEW가 실행되는 플랫폼과 같은 환경 정보도 불러올 수 있습니다.
- 플러그인 구조를 생성하여 고객에게 어플리케이션을 배포한 후에도 어플리케이션에 기능을 추가할 수 있도록 합니다. 예를 들어, 동일한 파라미터를 받는 데이터 필터링 VI 세트가 있습니다. 이러한 VI를 플러그인 디렉토리에서 다이나믹하게 로드하도록 어플리케이션을 설계하여, 우선 필터링 VI들 중에서 일부만 포함한 어플리케이션을 배달하고, 더 많은 필터링 기능은 사용자가 새로운 필터링 VI들을 플러그인 디렉토리에 위치시켜야만 사용할 수 있도록 만들 수 있습니다.
- **프런트패널 객체를 컨트롤합니다.**

VI 서버 어플리케이션 만들기

VI 서버 어플리케이션의 프로그래밍 모델은 **참조 번호**를 기초로 합니다. 또한, 참조 번호는 LabVIEW에서 파일 I/O, 네트워크 연결, 기타 객체에서 사용됩니다.

일반적으로, **어플리케이션 인스턴스** 또는 VI의 참조 번호를 엽니다. 그 후 이 참조 번호는 다른 VI의 파라미터로 사용됩니다. 이 VI는 프로퍼티를 얻기(읽기) 또는 설정(쓰기)하거나, 메소드를 실행하거나, 다이나믹하게 참조된 VI를 로드하고 호출합니다. 끝으로, 참조 번호를 닫습니다. 이는 메모리에서 참조되는 VI를 해제합니다.

다음 어플리케이션 컨트롤 함수와 노드를 사용하여 VI 서버 어플리케이션을 만듭니다:

- 어플리케이션 참조 열기—로컬이나 원격 어플리케이션 인스턴스에 대한 참조를 엽니다.
- VI 참조 열기—로컬이나 원격 컴퓨터에서 VI에 대한 참조를 열거나 디스크에서 다이나믹하게 VI를 로드합니다.
- 프로퍼티 노드—VI, 객체, 또는 어플리케이션의 **프로퍼티**를 받거나 설정합니다.
- 인보크 노드—VI, 객체, 또는 어플리케이션에 **메소드**를 실행합니다.
- 참조에 의한 호출 노드—다이나믹하게 로드된 VI를 호출합니다.
- 비동기화 호출 시작 노드—다이나믹하게 로드된 VI에 대한 비동기화 호출을 시작합니다.
- 비동기화 호출 기다림 노드—비동기적으로 호출된 VI의 결과를 수집합니다.
- 참조 닫기—VI 서버를 사용하여 접근한 VI, 객체, 또는 어플리케이션의 열린 참조를 닫습니다.

어플리케이션과 VI 참조

두 가지의 주요 객체 클래스, 어플리케이션 객체와 VI 객체에 대한 참조를 통해 VI 서버 기능에 접근할 수 있습니다. 두 객체 중 하나의 참조를 생성한 후, 해당 객체에 동작을 수행하는 VI 또는 함수에 참조를 전달할 수 있습니다.

어플리케이션 참조는 로컬 또는 원격 어플리케이션 인스턴스를 참조합니다. 어플리케이션 프로퍼티와 메소드를 사용하여 LabVIEW 환경을 변경하고 시스템 정보를 반환할 수 있습니다. VI 참조 번호는 어플리케이션 인스턴스의 VI를 참조합니다.

어플리케이션 인스턴스의 참조로 LabVIEW가 실행되는 플랫폼, 버전 번호, 또는 현재 메모리에 있는 모든 VI의 리스트와 같은 LabVIEW 환경에 대한 정보를 가져올 수 있습니다. 또한, 다른 어플리케이션 인스턴스로 반출된 VI의 리스트와 같은 정보를 설정할 수 있습니다. 한 번에 여러 어플리케이션 인스턴스를 열 수 있기 때문에, 하나의 어플리케이션 인스턴스에서 VI 서버 프로퍼티와 메소드를 사용하면서 다른 어플리케이션 인스턴스와 상호 작용을 하고자 할 때에는 어플리케이션 참조를 사용해야 합니다. 예를 들어, LabVIEW는 사용자가 LabVIEW 프로젝트 또는 LabVIEW 프로젝트를 위한 타겟을 생성할 때 새로운 어플리케이션 인스턴스를 엽니다.

VI에 대한 참조 번호를 생성하면 LabVIEW는 이 VI를 메모리에 로딩합니다. 참조를 닫고 VI가 다음 조건을 만족할 때까지 이 VI는 메모리에 남아있습니다.

- 참조된 VI에 다른 열린 참조가 없습니다.
- VI의 프런트패널이 열려있지 않습니다.
- VI가 메모리에 있는 다른 VI의 SubVI가 아닙니다.
- VI는 열린 프로젝트 라이브러리의 멤버가 아닙니다.

VI의 참조 번호로, 소유 구획의 위치와 같은 다이나믹 프로퍼티와 VI 프로퍼티 대화 상자에서 사용 가능한 VI의 모든 프로퍼티를 업데이트할 수 있습니다. 또한, 프로그램적으로 **VI 문서를 인쇄하고**, 다른 위치에 VI를 저장하며, **문자열을 반출 및 반입하여** 다른 언어로 번역할 수 있습니다.

어플리케이션과 VI 셋팅 변경하기

프로퍼티는 VI, 객체 또는 어플리케이션의 속성입니다. 메소드는 VI, 객체, 또는 어플리케이션에서 동작을 수행합니다. [프로퍼티 노드]와 [인보크 노드]를 사용하여 로컬 또는 원격 **어플리케이션 인스턴스**, VI, 객체에서 프로퍼티와 메소드를 얻고 설정할 수 있습니다. 여러 어플리케이션, VI, 객체의 셋팅은 오직 프로퍼티와 메소드를 통해 가져오고 설정할 수 있습니다.

어플리케이션과 VI 클래스 프로퍼티 및 메소드 사용에 대한 예제는 `labview\examples\viserver` 디렉토리를 참조하십시오.

관련 예제 찾기

프로퍼티 노드

프로퍼티 노드를 사용하여 어플리케이션 또는 VI의 다양한 프로퍼티를 얻고 설정합니다. 수행 도구로 프로퍼티 터미널을 클릭하거나 노드의 흰색 영역에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 노드에서 프로퍼티를 선택합니다.

다음은 프로퍼티를 사용하여 어플리케이션이나 VI의 사용을 간편하게 만든 예입니다:

- 텍스트 포커스를 가장 많이 사용하는 주요 컨트롤로 설정함.
- 현재 관련이 없거나 유효하지 않은 컨트롤을 비활성화 시키거나 숨김.
- 컨트롤을 하이라이트하여 단계별로 사용자에게 정보를 제공함.
- 색을 변경하여 에러 조건을 인식하도록 유도함.

하나의 노드를 사용하여 여러 프로퍼티를 읽거나 쓸 수 있습니다. 그러나, 일부 프로퍼티를 읽을 수 없고 일부 프로퍼티는 쓸 수 없습니다. 위치 도구로 프로퍼티 노드의 크기를 조정하여 새로운 터미널을 추가합니다. 프로퍼티의 오른쪽에 위치한 작은 화살표는 읽는 프로퍼티를 나타냅니다. 프로퍼티의 왼쪽에 위치한 작은 화살표는 쓰는 프로퍼티를 나타냅니다. 프로퍼티에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 읽기로 변경 또는 쓰기로 변경을 선택하여 프로퍼티의 작동을 변경합니다.

노드는 위에서 아래로 실행됩니다. 실행 전에 에러가 발생하는 경우 프로퍼티 노드는 실행되지 않으므로 항상 에러 발생 가능성을 확인하십시오. 프로퍼티에서 에러가 발생하는 경우, LabVIEW는 나머지 프로퍼티를 무시하고 에러를 반환합니다. 프로퍼티 노드에서 마우스 오른쪽 버튼을 클릭한 후 노드 내의 에러 무시를 선택하면, LabVIEW는 프로퍼티 노드에 남아있는 프로퍼티를 실행합니다. 오직 첫번째 에러가 프로퍼티 노드에 의해서 반환됩니다. 에러 출력 클러스터는 어떤 프로퍼티가 에러를 발생시켰는지에 대한 정보를 포함합니다.

프로퍼티 노드가 열려있고 객체의 참조를 반환하는 경우, [참조 닫기] 함수를 사용하여 참조를 닫습니다.

내부적으로 연결된 프로퍼티 노드

프론트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 생성» 프로퍼티 노드를 선택하고 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티를 생성할 때, LabVIEW는 프론트패널 객체에 내부적으로 링크된 블록 다이어그램에 프로퍼티 노드를 생성합니다. 이러한 프로퍼티 노드는 생성된 프론트패널 객체에 내부적으로 연결되므로, 노드에는 참조 입력이 없으며 이 노드를 프론트패널 객체의 터미널이나 **컨트롤 참조**에 연결할 필요가 없습니다. 프론트패널 객체로의 링크를 제거하려면, 블록 다이어그램의 프로퍼티 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 컨트롤로부터 연결 끊기를 선택합니다.

인보크 노드

인보크 노드를 사용하여 어플리케이션 또는 VI에서 동작이나 메소드를 수행합니다. 프로퍼티 노드와는 달리 하나의 인보크 노드는 어플리케이션 또는 VI에 하나의 메소드만을 수행합니다. 메소드를 선택하려면 수행 도구로 메소드 터미널을 클릭하거나 노드의 흰색 영역에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 메소드를 선택합니다. 또한, 프론트패널 객체에서 마우스 오른쪽 버튼을 클릭하고 생성» 인보크 노드를 선택

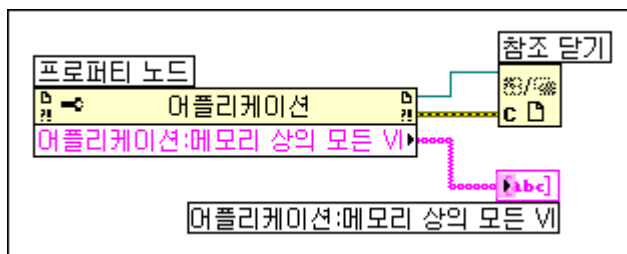
한 후, 바로 가기 메뉴에서 메소드를 선택하여 내부적으로 연결된 인보크 노드를 생성할 수 있습니다.

메소드의 이름은 항상 인보크 노드의 파라미터 리스트에서 첫번째 터미널입니다. 메소드가 값을 반환하는 경우, 메소드 터미널은 반환값을 디스플레이합니다. 그렇지 않은 경우, 메소드 터미널은 값을 갖지 않습니다.

인보크 노드는 파라미터를 위에서 아래로 나열합니다. 메소드의 이름은 위에 나열하고 비활성화된 옵션 파라미터는 아래쪽에 나열합니다.

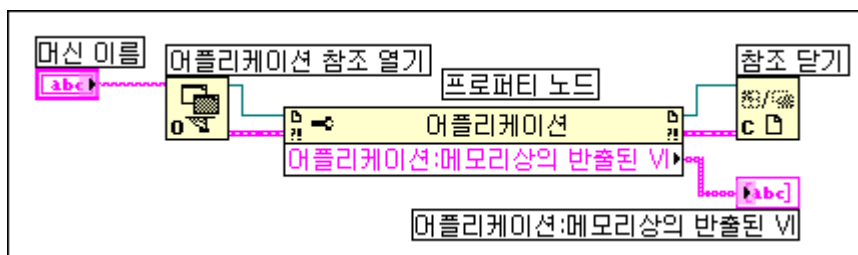
어플리케이션 클래스 프로퍼티와 메소드 변경하기

LabVIEW의 로컬 또는 원격 어플리케이션 인스턴스에서 프로퍼티를 얻거나 설정할 수 있고, LabVIEW에서 메소드를 수행할 수 있고, 두 가지 모두를 수행할 수 있습니다. 다음 블록 다이어그램은 로컬 컴퓨터의 메모리에 위치한 모든 VI를 프런트패널에서 문자열 배열로 디스플레이하는 방법을 보여줍니다.



참조 번호를 참조 입력에 연결하지 않는 경우, 프로퍼티 노드 또는 인보크 노드는 현재의 어플리케이션 인스턴스에 대한 참조를 사용합니다. 다른 어플리케이션 인스턴스의 프로퍼티나 메소드를 변경하려는 경우, 참조 입력에 어플리케이션 참조를 연결해야 합니다. 여러 어플리케이션 인스턴스가 동시에 열려 있는 경우, 어플리케이션 참조를 참조 입력에 와이어로 연결하는 것을 잊지 마십시오. 예를 들어, LabVIEW는 사용자가 LabVIEW 프로젝트 또는 LabVIEW 프로젝트를 위한 타겟을 생성할 때마다 새로운 어플리케이션 인스턴스를 엽니다. 또한, LabVIEW는 주요 어플리케이션 인스턴스를 생성합니다. 여기에는 프로젝트의 일부가 아닌 열린 VI와 프로젝트에서 열지 않은 VI가 포함됩니다.

반출된 VI를 원격 컴퓨터의 메모리에서 찾으려면, 다음 블록 다이어그램에서 보이는 것과 같이, 문자열 컨트롤을 [어플리케이션 참조 열기] 함수의 머신 이름 입력에 연결하고, IP 주소나 도메인을 입력하십시오. 또한, 이전 블록 다이어그램에서 사용된 메모리상의 모든 VI 프로퍼티는 로컬 어플리케이션 인스턴스에만 적용되므로 메모리상의 VI 반출 프로퍼티를 선택해야 합니다.



또한 기본:어플리케이션 프로퍼티를 사용하여 기본 어플리케이션 참조를 프로그램적으로 반환할 수 있습니다. 어플리케이션 프로퍼티를 사용하여 타겟 어플리케이션 인스턴스를 프로그램적으로 엽니다.

VI 클래스 프로퍼티와 메소드 변경하기

VI의 프로퍼티를 받거나 설정할 수 있고, VI에서 메소드를 수행할 수 있고, 이 두 가지 모두를 수행할 수 있습니다. 어플리케이션 클래스 프로퍼티와 메소드를 조작할 때 [VI 참조 열기] 함수를 사용하여 VI 참조를 반환한 후, 프로퍼티 노드와 인보크 노드를 사용합니다.

참조 입력에 참조 번호를 연결하지 않는 경우, 프로퍼티 또는 메소드는 프로퍼티 노드나 인보크 노드를 포함하는 VI의 참조를 사용합니다. 다른 VI의 프로퍼티나 메소드를 변경하려는 경우, VI 참조 번호를 참조 입력에 연결해야 합니다.

프로퍼티 노드는 인보크 노드와 유사하게 작동합니다. VI 참조 번호를 프로퍼티 노드에 연결한 후, 모든 VI 클래스 프로퍼티에 접근할 수 있습니다.

어플리케이션과 VI 클래스 프로퍼티 및 메소드 변경하기

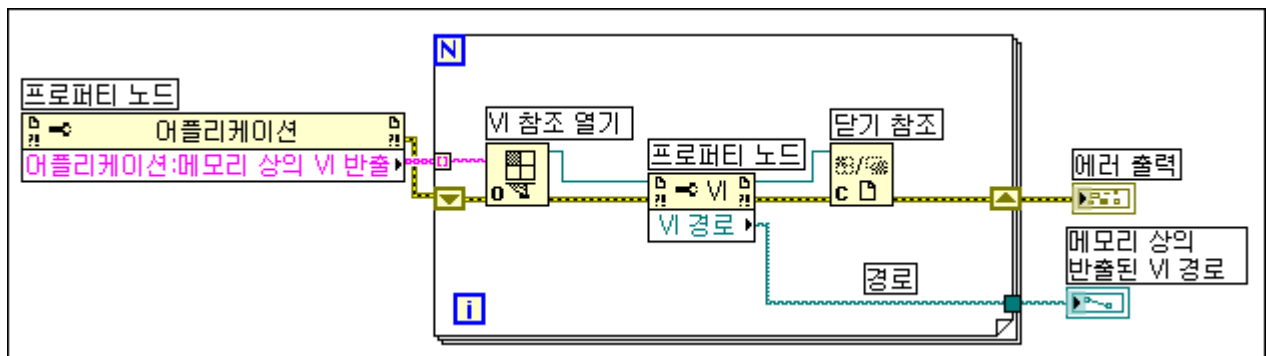
일부 VI에서는, 어플리케이션과 VI 클래스의 프로퍼티 및 메소드에 둘 다 접근해야 합니다. 다음 블록 다이어그램과 같이, 어플리케이션과 VI 클래스 참조 번호를 따로 열고 닫아야 합니다.



노트

이 블록 다이어그램의 어플리케이션 참조는 함축적 참조이므로, 닫지 않아도 됩니다. LabVIEW에서 참조를 닫아야 하는 경우에 대한 더 자세한 정보는 LabVIEW에서 참조 닫기 길라잡이를 참조하십시오.

다음 블록 다이어그램은 로컬 컴퓨터 메모리에 있는 반출 VI를 구분하는 방법과 프런트패널의 각 VI 경로를 디스플레이하는 방법을 보여줍니다. 메모리 안에서 반출 VI를 찾으려면 어플리케이션 클래스 프로퍼티에 접근해야 합니다. 각 VI의 경로를 지정하려면, 우선 VI 클래스 프로퍼티에 접근해야 합니다. 메모리 안의 VI 반출 개수가 [For 루프]의 실행 횟수를 결정합니다. 메모리에 있는 각 VI에 VI 참조 번호가 필요하므로 [For 루프] 안에 [VI 참조 열기]와 [참조 닫기] 함수를 놓습니다.



원격 컴퓨터에서 VI 편집하기와 실행하기

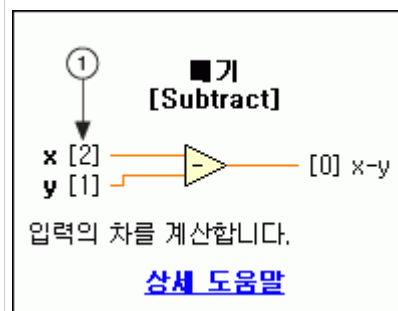
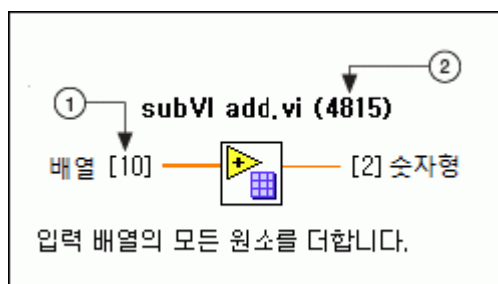
어플리케이션과 VI 참조 번호의 중요한 장점은 네트워크 투명성입니다. 이는 사용자 컴퓨터에서 객체의 참조 번호를 여는 것과 동일한 방법으로 원격 컴퓨터에서도 객체의 참조 번호를 열 수 있음을 의미합니다.

원격 객체의 참조 번호를 얻 후, 몇가지 제한을 제외하고 로컬 객체와 같은 방식으로 처리할 수 있습니다. 원격 객체에서 작업을 할 경우, VI 서버가 네트워크를 통해 작업에 대한 정보를 보내고 결과를 돌려받습니다. 어플리케이션은 작업이 원격 또는 로컬에서 일어나는지에 관계없이 비슷하게 보입니다.

기본 도움말 윈도우의 VI 스크립팅 정보

노드나 커넥터 팬의 터미널과 관련된 VI 스크립팅 태스크를 완료하려면, 터미널 및 커넥터 팬 패턴 인덱스를 사용해야 합니다. 기본 도움말 설정으로 대부분의 노드와 VI에 대해 이러한 인덱스를 디스플레이할 수 있으며, 이로써 LabVIEW에서 프로그램을 만드는 동안 이 정보를 쉽게 얻을 수 있습니다.

기본 도움말 윈도우에서 터미널 및 커넥터 팬 패턴 인덱스를 디스플레이하려면, VI 스크립팅을 활성화하고 옵션 대화 상자의 VI 서버 페이지에 있는 기본 도움말 윈도우에 추가적인 VI 스크립팅 정보 디스플레이 확인란에 확인 표시를 합니다. 이렇게 선택하면, 기본 도움말 윈도우가 다음 그림 중 하나처럼 보입니다:



1 터미널 인덱스 — 디스플레이된 노드의 노드:터미널[] 프로퍼티에 의해 반환된 터미널 참조의 배열 내에서 관련 터미널의 위치를 확인합니다. [배열 인덱스] 함수와 더불어 터미널[] 배열 및 터미널 인덱스를 사용하여 특정한 터미널에 대한 참조를 얻을 수 있습니다. VI 스크립팅을 사용하여 노드에서 연결된 와이어로 탐색하거나 원하는 터미널로 데이터를 연결하려면 일반적으로 이 작업을 수행해야 합니다.

2 패턴 인덱스 — 디스플레이된 VI의 커넥터 팬 패턴을 식별합니다. 패턴 인덱스를 사용하여 VI 스크립팅으로 VI의 커넥터 팬을 설정할 수 있습니다. VI만 기본 도움말 윈도우에서 패턴 인덱스를 디스플레이합니다

각 패턴 인덱스와 관련된 커넥터 팬 패턴은 `labview\examples\Application Control\VI Scripting\Connector Pane` 디렉토리에 있는 `Connector Pane Pattern Reference VI`를 참조하십시오.



관련 정보

[VI 스크립팅 활성화하기](#) (948 페이지 참조)

[VI 스크립팅을 사용하여 노드에서 와이어로 탐색하기](#) (972 페이지 참조)

[길라잡이: VI 스크립팅을 사용해서 객체 연결하기](#) (984 페이지 참조)

[VI 스크립팅을 사용하여 커넥터 팬 설정하기](#) (963 페이지 참조)

VI 실행 속도

LabVIEW가 VI를 컴파일하고 일반적으로 매우 빠르게 실행하는 코드를 생성한다고 해도, 사용자는 시간에 결정적인 VI를 다룰 때 가능한 최상의 성능을 얻고자 할 것입니다. 이 섹션은 실행 속도에 영향을 주는 요소를 다루고 가능한 최상의 성능을 얻도록 도움을 주는 몇몇 프로그래밍 기술을 제안합니다.

다음 아이템을 조사하여 느린 성능의 원인을 결정합니다:

- 입력/출력(파일, GPIB, 데이터 수집, 네트워크)
- 화면 디스플레이(큰 컨트롤, 컨트롤 오버래핑, 너무 많은 디스플레이)
- 메모리 관리(배열과 문자열의 불충분한 사용, 불충분한 데이터 구조)
- 컴파일러 최적화(편집기 응답이 VI 실행 속도와 균형을 맞추도록 컴파일러 조절)

실행 오버헤드와 SubVI 호출 오버헤드와 같은 다른 요소는 일반적으로 실행 속도에 최소한의 영향을 줍니다.

입력/출력

입력/출력(I/O) 호출은 일반적으로 많은 오버헤드를 초래합니다. 입출력 호출은 종종 계산 작동 시간보다 훨씬 더 많은 시간이 걸립니다. 예를 들면, 간단한 시리얼 포트 읽기 작동은 상당한 밀리초의 오버헤드를 가질 수 있습니다. 이 오버헤드는 시리얼 포트를 사용하는 모든 어플리케이션에서 발생하는데, 그 이유는 I/O 호출이 OS의 몇개의 계층을 통해 정보를 전달하는 것을 포함하기 때문입니다.

아주 많은 오버헤드를 전달하는 가장 좋은 방법은 만들어내는 I/O 호출의 수를 줄이는 것입니다. 더 작은 데이터 양을 전달하기 위해 여러 I/O 호출을 만들어내는 것보다 각 호출마다 많은 양의 데이터를 전달하도록 VI를 구성한다면 성능은 향상됩니다.

예를 들어, 데이터 수집(NI-DAQmx) VI를 생성하는 경우, 데이터를 읽는데 두가지 옵션이 있습니다. [AI 샘플 채널] VI와 같은 단일 포인트 데이터 전달 함수를 사용할 수 있거나, [AI 웨이브폼 읽기] VI와 같은 여러 포인트 데이터 전달 함수를 사용할 수 있습니다. 100 포인트를 수집해야 한다면, 타이밍을 만들기 위해 기다림 함수를 가진 루프에서 [AI 샘플 채널] VI를 사용합니다. 또한 100 포인트를 원한다는 것을 나타내는 입력을 가진 [AI 웨이브폼 읽기] VI를 사용할 수 있습니다.

[AI 웨이브폼 얻기] VI를 사용하여 훨씬 더 크고 더 정확한 데이터 샘플링 속도를 만들 수 있습니다. 이 VI가 데이터 샘플링을 관리하기 위해 하드웨어 타이머를 사용하기 때문입니다. 또한, [AI 웨이브폼 얻기] VI가 훨씬 더 많은 데이터를 전달하지만, 이 VI에 대한 오버헤드는 [AI 샘플 채널] VI를 한번 호출하는 것에 대한 오버헤드와 대략적으로 같습니다.

화면 디스플레이

프론트패널에서 컨트롤을 자주 업데이트하는 것은 어플리케이션에서 가장 시간을 많이 소비하는 작동 중 하나일 수 있습니다. 특히 그래프 및 차트와 같은 더 복잡한 디스플레이 중 일부를 사용하는 경우가 이에 해당됩니다. 대부분의 인디케이터는 오래된 데이터와 같은 새 데이터를 받을 때 인디케이터를 다시 그리지 않지만, 그래프와 차트는 항상 다시 그립니다. 다시 그리는 속도가 문제가 되면, 최상의 해결책은 프론트패널 객체의 수를 줄이고 프론트패널 디스플레이를 가능한 한 간단하게 유지하는 것입니다. 그래프와 차트의 경우에, 오토스케일링, 스케일 마커, 엘리어스 제거 라인 그리기, 눈금을 꺼서 디스플레이의 속도를 증가시킬 수 있습니다.

다른 종류의 I/O에서처럼, 컨트롤을 디스플레이할 때 특정한 양의 고정된 오버헤드가 있습니다. 차트와 같은 특정한 컨트롤을 사용하여 여러 포인트를 한 번에 인디케이터로 전달할 수 있습니다. 매번 차트에 더 많은 데이터를 전달하여 차트 업데이트의 수를 최소화할 수 있습니다. 데이터가 들어올 때마다 각 포인트를 디스플레이하는 대신, 차트 데이터를 배열에 모아 여러 포인트를 한 번에 디스플레이하면 데이터 디스플레이 속도를 크게 향상할 수 있습니다.

실행하는 동안 프론트패널을 닫는 SubVI를 디자인할 때, 디스플레이 오버헤드는 걱정하지 않아도 됩니다. 프론트패널을 닫으면 컨트롤에 대한 그리기 오버헤드를 가지지 않으므로, 그래프는 배열보다 더 많은 계산을 요구하지 않습니다.

멀티스레드 실행 시스템에서, 고급 >> 동기화된 디스플레이 바로 가기 메뉴 아이템을 사용하여 컨트롤과 인디케이터 업데이트를 연기할 것인지 설정합니다. 기본적으로, 컨트롤과 인디케이터는 비동기 디스플레이를 사용하는데, 이것은 실행 시스템이 프론트패널 컨트롤과 인디케이터로 데이터를 전달한 후에, 실행을 즉시 계속할 수 있다는 것을 의미합니다. 그 후에 어떤 포인트에서, 사용자 인터페이스 시스템은 컨트롤이나 인디케이터를 업데이트할 필요가 있다는 것을 인식하고 다시 그려 새 데이터를 보여줍니다. 실행 시스템이 빠르게 연속적으로 컨트롤을 여러번 업데이트하려고 시도하면 중간에 개입하는 몇개의 업데이트를 못 볼 수도 있습니다.

대부분의 어플리케이션에서 비동기 디스플레이는 사용자가 보는 것에 영향을 주지 않고 실행 속도를 상당히 끌어올립니다. 예를 들면, 불리언 값을 1초에 수백 번 업데이트할 수 있는데, 이는 인간의 눈으로 구별할 수 있는 것보다 더 많은 업데이트입니다. 비동기 디스플레이는 사용자 인터페이스 스레드에 의해 업데이트가 자동적으로 더 느린 속도로 줄도록 하여 실행 시스템이 VI를 실행하는데 더 많은 시간을 사용하도록 허용합니다.

동기화된 디스플레이를 원하면, 컨트롤과 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급 >> 동기화된 디스플레이를 선택하여 메뉴 아이템 옆에 확인 표시를 합니다.



노트

모든 데이터 값을 디스플레이하기 위해 필요한 경우에만 동기화된 디스플레이를 활성화합니다. 동기화된 디스플레이를 사용하면 멀티스레드 시스템 성능에 큰 불이익이 발생합니다.

또한 패널 업데이트 연기 프로퍼티를 사용하여 프런트패널 업데이트를 위한 새로운 모 든 요청을 연기할 수 있습니다.

또한, 프런트패널에 놓는 모니터 셋팅과 컨트롤은 VI의 성능을 향상시킬 수 있습니다. 모 니터의 색 농도와 해상도를 더 낮추고 모니터를 위한 하드웨어 가속을 활성화합니다. 하 드웨어 가속에 대한 추가적인 정보는 해당 OS의 문서를 참조하십시오. 또한 일반 팔레트 의 컨트롤 대신에 클래식 팔레트의 컨트롤을 사용하면 VI의 성능을 향상시킬 수 있습 니다.

메모리 관리

VI의 메모리를 관리하여 최적의 실행 속도를 내려면 다음 사항을 참조하십시오.

- **데이터 크기를 계속하여 바꾸는 것을 피하면** LabVIEW가 해당 데이터의 복사본을 만들지 않아도 됩니다.
- 조작하기 쉬운 데이터 타입을 사용하여 **효과적인 데이터 구조를 개발하면** 추가적 인 데이터 복사본의 생성을 피할 수 있습니다.
- 컴파일된 코드를 분리하지 않고 LabVIEW의 현재 버전에 VI를 저장합니다.

컴파일러 최적화

큰 VI의 경우 LabVIEW 컴파일러는 VI의 실행 속도를 빠르게하는 최적화를 제한하여 편집 기 응답을 향상합니다. 큰 VI의 개발을 마칠 때 **실행 속도가 편집기 응답보다 우선하도록** LabVIEW를 설정할 수 있습니다. **전체 빌드 스펙에 컴파일러 최적화 세트 모두를 적용** 하도록 LabVIEW를 설정할 수 있습니다.

어플리케이션 안에서 데이터 전달하기

LabVIEW 어플리케이션 내에서 데이터를 전달하는 기술은 여러가지가 있으며 그 중 하나 를 선택할 수 있습니다. 다음의 리스트는 가장 일반적인 방법을 효율적인 순서대로 나타 냅니다.

1. 와이어—와이어를 사용하여 데이터를 전달하면 LabVIEW가 성능이 최적화되도록 컨트롤할 수 있습니다. 데이터에 하나의 쓰기와 하나 이상의 읽기가 있기 때문에 데이터 흐름 언어에서 가장 빠른 기술입니다.
2. 피드백 노드—피드백 노드를 사용하여 이전 VI 또는 루프 실행에서 온 데이터를 저장합니다. subVI, 함수, 또는 subVI와 함수로 이루어진 그룹의 출력을 같은 VI, 함수, 또는 그룹의 입력에 연결하고 옵션 대화 상자의 블록 다이어그램 페이지에서 사이 클내에 피드백 노드 자동 삽입을 활성화하면 피드백 노드가 자동으로 나타납니 다. LabVIEW는 기본설정으로 사이클내에 피드백 노드 자동 삽입을 활성화합니다. 피드백 노드는 **초기화 터미널을 이동** 하는지 여부에 따라 VI, 함수, 또는 그룹의 마 지막 실행에서 나온 데이터를 기억하고 있다가 이를 다음 실행으로 전달하거나 새로운 초기값을 설정합니다. 루프에서 시프트 레지스터는 피드백 노드와 같은 방식으로 데이터를 저장하지만 루프에 와이어를 확장해야 합니다.
3. 시프트 레지스터—루프 안에 스토리지나 피드백을 필요로 한다면 **시프트 레지스 터**를 사용합니다. 시프트 레지스터는 쓰기와 읽기 밖으로 그리고 쓰기와 읽기 안 으로 데이터를 전달합니다. 데이터 접근에 있어서 이러한 밀접한 스코프는

LabVIEW의 효율을 최대화합니다. 루프에 와이어를 확장하지 않고도 같은 기능을 사용하려면 시프트 레지스터를 피드백 노드로 대체합니다.

4. 글로벌 변수와 기능적인 글로벌 변수—단순한 데이터와 단순한 접근을 위해 **글로벌 변수**를 사용합니다. 크거나 복잡한 데이터에 대해, 글로벌 변수는 모든 데이터를 읽고 전달합니다. **기능적인 글로벌 변수**를 사용하여 LabVIEW가 얼마나 많은 데이터를 반환하는지 컨트롤합니다.

컨트롤, 컨트롤 참조, 프로퍼티 노드를 변수로서 사용하기

VI 사이에 데이터를 전달하기 위해 컨트롤, 컨트롤 참조, 프로퍼티 노드를 사용한다 할지라도, 그들은 사용자 인터페이스를 통해 작동하기 때문에 변수로 사용하도록 디자인되지 않았습니다. 사용자 인터페이스 동작을 수행할 때 또는 병렬 루프를 멈출 때만 로컬 변수와 그 값 프로퍼티를 사용합니다.

사용자 인터페이스 동작은 전통적으로 컴퓨터를 느리게 합니다. LabVIEW는 나노초 동안 와이어를 통해 더블로 선언된 값을 전달하고, 텍스트 일부를 수백 마이크로초에서 밀리초 사이에 씁니다. 예를 들면, LabVIEW는 100K 배열을 와이어를 통해 0 나노초에서 몇 마이크로초 안에 전달합니다. 이 100K 배열의 그래프를 그리는데 수십 밀리초가 걸립니다. 컨트롤은 사용자 인터페이스가 덧붙여져 있기 때문에, 컨트롤을 사용하여 데이터를 전달하면 컨트롤을 다시 그리게 되므로 메모리를 추가적으로 쓰게 되고 성능이 떨어집니다. 컨트롤이 숨겨져 있으면, LabVIEW는 데이터를 더 빠르게 전달하지만, 컨트롤이 항상 디스플레이될 수 있기 때문에 LabVIEW는 여전히 컨트롤을 업데이트할 필요가 있습니다.

멀티스레딩이 사용자 인터페이스 동작에 영향을 주는 방식

LabVIEW는 실행 스레드로부터 사용자 인터페이스 스레드로 전환하기 때문에 사용자 인터페이스 동작을 완성하기 위해 더 많은 메모리를 사용합니다. 예를 들면, 값 프로퍼티를 설정할 때, LabVIEW는 사용자가 컨트롤의 값을 변경하고, 실행 스레드를 멈추고, 사용자 인터페이스 스레드로 스위치하여 값을 변경하는 것을 시뮬레이트합니다. 그 후, LabVIEW는 컨트롤에 데이터를 저장하는 동작 버퍼를 업데이트하고, 프론트패널이 열려 있을 경우, 컨트롤을 다시 그립니다. 그리고 LabVIEW는 전달 버퍼라고 불리는 메모리의 안전한 영역의 실행 스레드로 데이터를 되돌려 보냅니다. 그 후 LabVIEW는 실행 스레드로 다시 되돌아 갑니다. 그 다음에 실행 스레드가 컨트롤을 읽을 때, LabVIEW는 전달 버퍼에서 데이터를 찾고, 또한 블록 다이어그램에 데이터를 저장하는 실행 버퍼의 새로운 값을 받습니다.

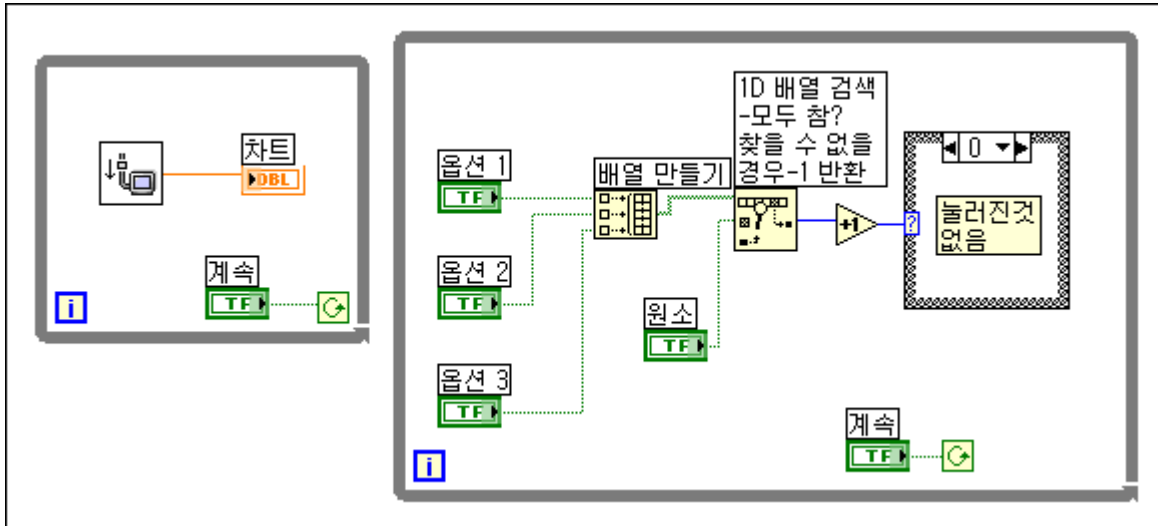
로컬 또는 글로벌 변수에 쓸 때, LabVIEW는 사용자 인터페이스 스레드로 즉시 전환하지 않습니다. 대신에 LabVIEW는 전달 버퍼에 값을 씁니다. 사용자 인터페이스는 다음의 계획된 업데이트 시간에 업데이트합니다. 단일 스레드 스위치나 사용자 인터페이스 업데이트가 발생하기 전에 변수를 여러번 업데이트하는 것이 가능합니다. 이것은 변수가 단지 실행 스레드에서 작동하기 때문에 가능합니다.

기능적인 글로벌 변수는 전달 버퍼를 사용하지 않기 때문에 일반적인 글로벌 변수보다 더 효과적일 수 있습니다. 기능적인 글로벌 변수는 실행 스레드 안에서만 존재하고 그 값을 열려 있는 프론트패널에 디스플레이하지 않으면 전달 버퍼를 사용하지 않습니다.

병렬 블록다이어그램

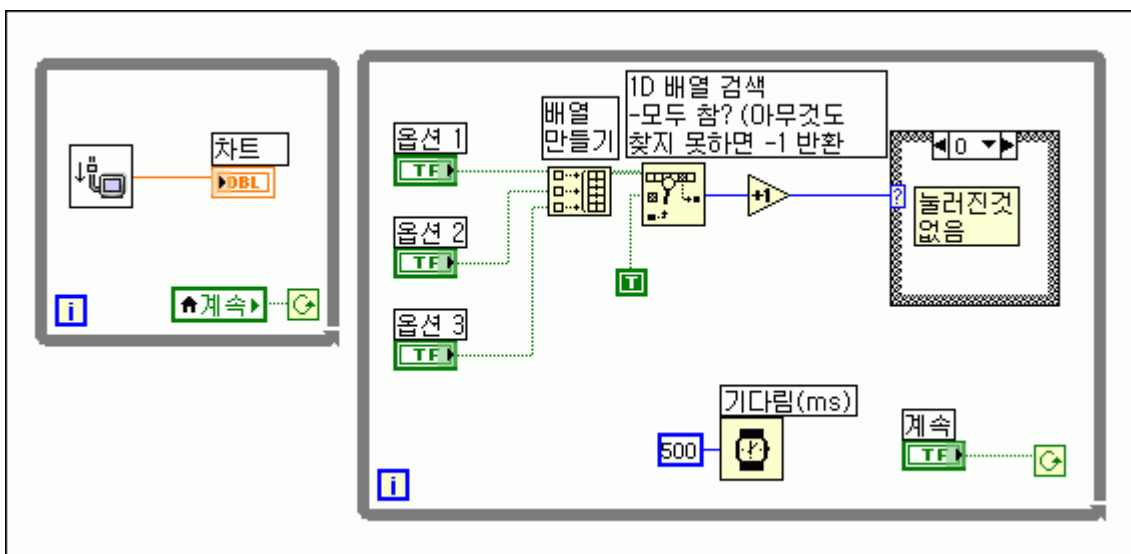
병렬로 실행되고 있는 여러 블록다이어그램을 가질 때, 실행 시스템은 그 사이에서 정기적으로 전환합니다. 이 루프 중의 일부가 다른 것보다 덜 중요하다면, [기다림(ms)] 함수를 사용하여 덜 중요한 루프가 더 적은 시간을 사용하도록 만듭니다.

예를 들어, 다음의 블록다이어그램을 생각해 봅니다.



병렬로 된 두 개의 루프가 있습니다. 루프 중 하나는 데이터를 수집하며 가능한 한 자주 실행해야 합니다. 다른 루프는 사용자 입력을 모니터합니다. 이 프로그램이 구성되는 방식 때문에 루프는 같은 시간을 할당받습니다. 사용자 동작을 모니터하는 루프는 초당 몇 번 실행할 기회를 가집니다.

실제로, 버튼을 모니터하는 루프가 매 1/2초당 한번씩 또는 그보다 덜 자주 실행해도 일반적으로 받아들입니다. 사용자 인터페이스 루프에서 [기다림(ms)] 함수를 호출하여 다른 루프에 상당히 더 많은 시간을 할당합니다.



병렬 For 루프 반복

For 루프를 사용할 때, LabVIEW는 순차적으로 루프 반복을 실행합니다. 계산 집약적인 For 루프인 경우, **루프를 병렬 반복 실행**하여 성능을 향상시킬 수 있습니다. 병렬 루프 반복을 사용하면 LabVIEW는 여러 프로세서의 장점을 활용하여 For 루프를 더욱 빠르게 실행할 수 있습니다. 하지만, 병렬 루프 반복은 다른 반복 모드에 대해 독립적이어야 합니다. 병렬 가능한 루프 찾기 결과 대화 상자를 사용하여 병렬 가능한 For 루프를 구분할 수 있습니다. 병렬 루프 반복을 활성화하려면 For 루프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 반복 병렬 처리 설정을 선택하여 For 루프 반복 병렬 처리 대화 상자를 디스플레이합니다.

병렬 루프 반복을 활성화하면, 병렬 인스턴스 터미널이  카운트 터미널  아래에 나타납니다. 이 터미널을 사용하여 특정 반복을 어떤 인스턴스가 실행할지 확인하고 LabVIEW가 사용할 루프 인스턴스의 수를 지정할 수 있습니다.

For 루프의 병렬 반복을 활성화하기 전에 **성능 고려사항**을 검토해보십시오.

National Instruments는 병렬 루프 반복에 의해 VI에 문제가 발생하면 LabVIEW가 경고를 나타낼 수 있도록 에러 리스트 윈도우에 **경고를 디스플레이**할 것을 권장합니다.

SubVI 오버헤드

SubVI를 호출할 때, 그 호출과 연관된 상당한 양의 오버헤드가 있습니다. 이 오버헤드는 특히 밀리초에서 수십 밀리초 단위의 범위를 가질 수 있는 I/O 오버헤드 및 디스플레이 오버헤드에 비해 꽤 작습니다(수십 마이크로초). 그러나, 몇몇 경우에 이 오버헤드가 커질 수 있습니다. 예를 들면, SubVI를 루프에서 10,000번 호출할 수 있다면, 이 오버헤드는 실행 속도에 상당한 영향을 줄 수 있습니다. 이 경우에, SubVI에 루프를 넣을 것을 고려할 수 있습니다.

subVI 오버헤드를 최소화할 수 있는 다른 방법은 VI 프로퍼티 대화 상자의 항목 풀다운 메뉴에서 실행을 선택한 후 우선순위 풀다운 메뉴에서 서브루틴을 선택하여 subVI를 서브루틴으로 만드는 것입니다. 그러나, 이에 따른 결점도 있습니다. 서브루틴은 프론트패널 데이터를 디스플레이할 수 없거나, 타이밍 또는 대화 상자 함수를 호출할 수 없거나, 디버깅 또는 자동 에러 핸들링을 허용할 수 없거나, 다른 VI와 멀티태스킹을 할 수 없습니다. 서브루틴은 사용자 상호작용이 필요하지 않으면서 짧게, 자주 실행되는 태스크인 VI에 일반적으로 적합합니다.

세번째로 SubVI 오버헤드를 최소화 하는 방법은 SubVI와 호출자 VI를 인라인 시키는 것입니다. SubVI를 인라인 시킬때 LabVIEW는 SubVI의 컴파일 코드를 호출자 VI의 컴파일 코드에 삽입합니다. 그후에 SubVI를 변경하려면 LabVIEW는 SubVI의 변경사항을 호출자 VI에 포함할 수 있도록 재컴파일합니다. 기본적으로, SubVI를 인라인하면 실행 시 SubVI를 호출할 필요가 없습니다. 대신에 LabVIEW는 SubVI 코드를 호출자 VI의 컴파일 코드안에서 실행합니다.

SubVI 인라인하기는 작은 SubVI, 루프 안에 있는 SubVI, 연결되지 않은 출력을 가진 SubVI와 단 한번만 호출되는 SubVI에 사용할 때 가장 유용합니다. SubVI를 인라인하려면, VI 프로퍼티 대화 상자의 실행 페이지에서 호출하는 VI로 SubVI 인라인 확인란에 확인 표시를 합니다. 또한 대화 상자의 같은 페이지에 있는 미리 할당된 복제 재호출 실행을 반드시 선택해야 합니다. LabVIEW는 SubVI를 인라인할 때 각 인스턴스의 복제본을 자동으로 미리 할당합니다.

다음과 같은 특성을 지닌 SubVI는 인라인할 수 없습니다.

- 재귀 기능 포함
- 자동 에러 핸들링 사용
- 프로퍼티 노드 및 인보크 노드와 같은 특정 블록 다이어그램 노드 포함. SubVI 인라인 기능을 활성화하면 이러한 노드는 편집 시 에러 메시지를 생성합니다.
- 디버깅 가능. 인라인하려는 SubVI에 대해 **디버깅을 비활성화**해야 합니다.
- 다음 VI 타입 중 하나입니다:
 - LabVIEW 클래스의 다이나믹 디스패치 VI
 - CGen 또는 FPGA용 VI
 - 컨트롤 VI – 사용자 컨트롤 또는 인디케이터를 정의하는 VI
 - 다형성 VI
 - 글로벌 변수 VI
 - X컨트롤 VI
 - X컨트롤 라이브러리의 메소드 VI
 - LabVIEW Control Design and Simulation Module의 시뮬레이션 다이어그램에만 놓을 수 있는 VI

SubVI가 위의 특성을 지니지 않은 경우라도, LabVIEW는 다음과 같은 종류의 호출로 subVI를 인라인하지 않습니다:

- SubVI에 대한 다이나믹 호출
- 데이터베이스 접근을 활성화하는 subVI에 대한 호출

마지막으로, SubVI를 인라인할 때, LabVIEW는 우선순위나 선호하는 실행 설정을 무시합니다.

SubVI를 인라인하여 성능 개선이 이루어지는 예제는 `examples\Performance\VI Properties`의 VI Execution Properties VI를 참조하십시오.

공유 변수 오버헤드

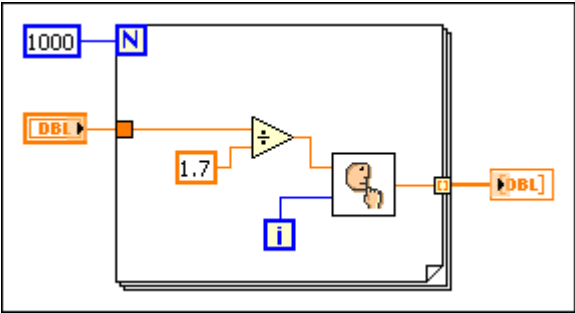
네트워크 공유된 변수는 CPU와 메모리에서 상당한 오버헤드를 초래합니다. 사용하는 어플리케이션이 다수의 네트워크 공유 변수를 이용할 경우 단순 메시지 참조 라이브러리(Simple Messaging Reference Library (STM))의 이용을 고려해 보십시오. STM에 대한 보다 더 자세한 내용은 `ni.com`에서 관련 문서를 참조하십시오.

하나의 연산 디바이스에서 다른 연산 디바이스로 연속적으로 스트링되는 데이터를 보내려면 네트워크 스트림 함수를 이용하십시오.

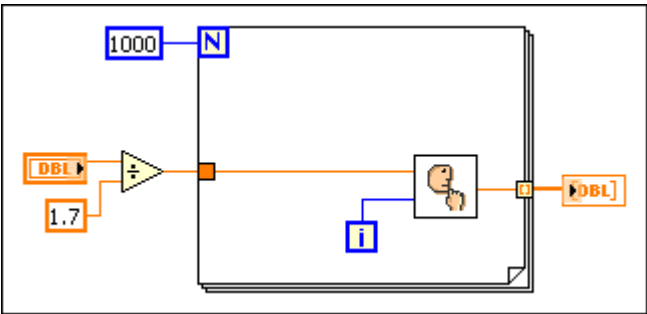
루프에서 불필요한 계산

계산이 매 반복마다 같은 값을 생성한다면 루프에 계산을 넣지 않습니다. 대신에, 루프 밖으로 그 계산을 이동해서 그 결과를 루프 안에 전달합니다.

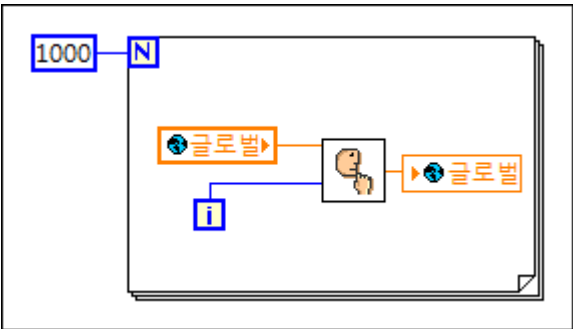
예를 들면, 다음의 블록다이어그램을 살펴봅니다.



나누기의 결과값은 루프에서 매번 같습니다. 그러므로 다음 블록다이어그램에서 보는 것처럼 루프 밖으로 그 나누기를 이동시켜서 성능을 향상시킬 수 있습니다.

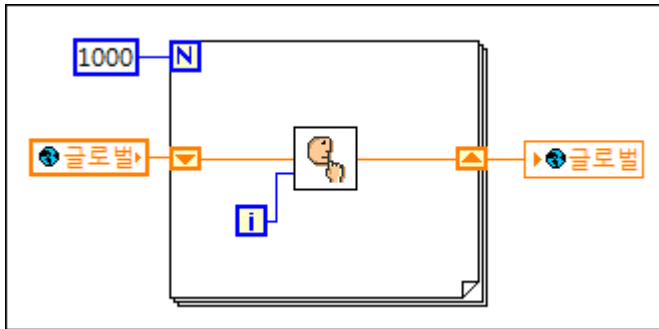


다음 블록다이어그램을 참조하십시오.

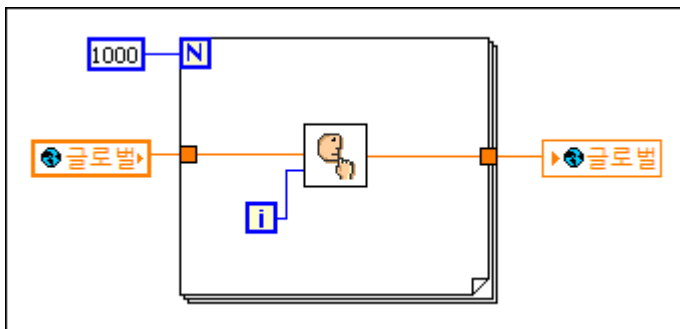


이 루프에서 글로벌 변수의 값이 다른 동시에 실행되는 블록다이어그램이나 VI에 의해 바뀌지 않는다는 것을 안다면, 이 블록다이어그램은 글로벌 변수로부터 값을 읽어들이고 루프를 통해 매번 글로벌 변수에 쓰는 시간을 낭비하는 것입니다.

이 루프에서 다른 블록다이어그램으로부터 읽어 들인 또는 다른 블록다이어그램에 의해 쓰인 글로벌 변수를 요구하지 않는다면, 다음 블록다이어그램을 대신 사용할 수 있습니다.



시프트 레지스터는 SubVI로부터 루프의 다음 반복으로 새로운 값을 전달해야 합니다. 다음 블록 다이어그램은 초보자가 저지를 수 있는 일반적인 실수를 보여줍니다. 시프트 레지스터가 없기 때문에, SubVI로부터의 결과값은 새로운 입력값처럼 SubVI로 다시 전달되지 않습니다.



VI 메모리 사용

LabVIEW는 텍스트 기반 프로그래밍 언어에서 다루어야만 하는 많은 세부사항을 처리해 줍니다. 텍스트 기반의 언어의 주요한 어려운 점 중 하나가 메모리 사용입니다. 텍스트를 기반으로 한 언어에서는 프로그래머가 메모리를 사용하기 전에 할당하고 그리고 끝마치면 할당을 해제하는 것을 관리해야 합니다. 또한 처음에 할당한 메모리의 한계를 넘어서 쓰지 않도록 주의할 기울여야 합니다. 메모리를 할당하지 못하거나 충분한 메모리를 할당하지 못하는 것이 프로그래머가 텍스트 기반의 언어에서 저지르는 가장 큰 실수 중의 하나입니다. 또한 부적절한 메모리 할당은 디버그하기 어려운 문제입니다.

LabVIEW의 데이터 흐름 패러다임은 메모리 관리의 어려움을 상당 부분 제거합니다. LabVIEW에서는 변수를 할당하지도 않고, 변수에 값을 지정하거나 변수로부터 값을 받지도 않습니다. 대신, 데이터의 전달을 보여주는 연결이 있는 블록 다이어그램을 생성합니다.

데이터를 생성하는 함수가 그 데이터의 스토리지 할당을 관리합니다. 데이터가 더 이상 사용되지 않을 때, 관련된 메모리는 할당을 해제합니다. 새로운 정보를 배열이나 문자열에 추가할 때, 충분한 메모리가 자동적으로 할당되어 새로운 정보를 관리합니다.

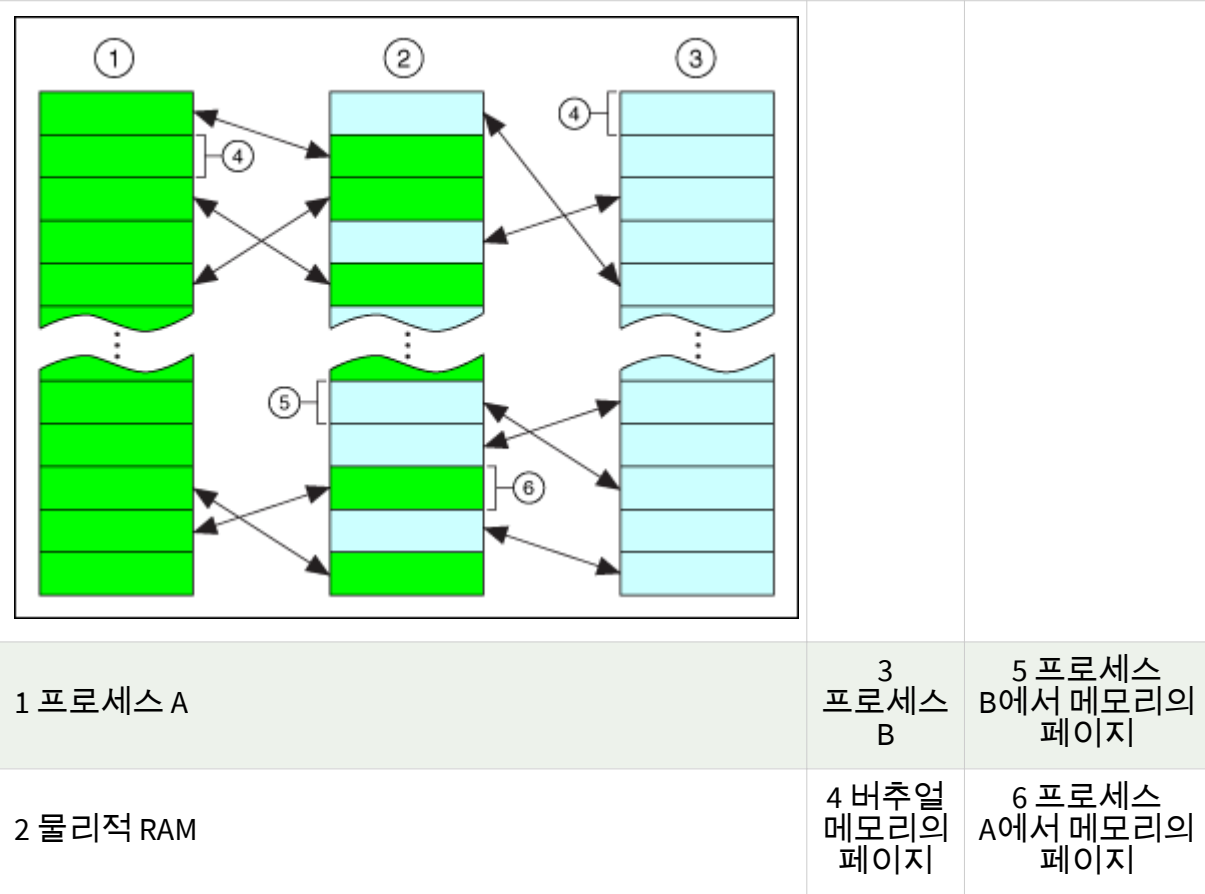
이 자동 메모리 핸들링은 LabVIEW의 주요한 장점 중의 하나입니다. 그러나 자동이기 때문에, 사용자는 언제 발생하는지 제어하기 어렵습니다. 프로그램이 큰 데이터 세트를 가지고 작동할 때, 언제 메모리 할당이 발생할지를 이해하는 것은 중요합니다. 관련된 원리를 이해하면 약간 더 작은 메모리를 요구하는 프로그램도 만들 수 있습니다. 또한, 메모리 할당과 데이터를 복사하는 일은 상당한 시간이 걸릴 수 있기 때문에, 메모리 사용을 최소화하는 방법을 이해하면 VI 실행 속도를 증가시킬 수 있습니다.

가상 메모리

OS는 버추얼 메모리를 사용하여 어플리케이션이 물리적인 RAM 용량 이상의 메모리에 접근할 수 있도록 합니다. OS는 물리적 RAM을 페이지라고 불리는 블록으로 나눕니다. 어플리케이션이나 프로세스가 주소를 메모리 블록에 할당하면 주소는 물리적 RAM의 직접적인 위치를 참조하지 않고 페이지의 메모리를 참조합니다. OS는 이러한 페이지를 물리적 RAM과 하드 디스크 사이에서 스왑할 수 있습니다.

어플리케이션이나 프로세스가 물리적 RAM에 없는 특정 블록이나 메모리 페이지에 접근해야 하는 경우, OS는 어플리케이션이나 프로세스가 사용하지 않는 물리적 RAM의 페이지를 하드 디스크로 옮기고 그 자리를 필요한 페이지로 채웁니다. OS는 메모리에 있는 페이지를 계속 추적하며 어플리케이션이나 프로세스가 특정한 메모리에 접근해야 할 때 페이지의 버추얼 주소를 물리적 RAM의 실제 주소로 해석합니다.

다음 이미지는 두 개의 프로세스가 어떻게 물리적 RAM의 안팎으로 페이지를 교환하는지 보여줍니다. 이 예제에서 프로세스 A와 프로세스 B는 동시에 실행됩니다.



어플리케이션이나 프로세스가 사용하는 페이지의 개수는 사용 가능한 물리적 RAM이 아니라 사용 가능한 디스크 공간에 따라 달라지기 때문에 어플리케이션은 물리적 RAM의 실제 메모리보다 더 많은 메모리를 사용할 수 있습니다. 어플리케이션이 사용하는 메모리 주소의 크기가 어플리케이션이 접근할 수 있는 버추얼 메모리의 양을 제한합니다.

LabVIEW 버추얼 메모리 사용

LabVIEW (32비트)는 32비트 주소를 사용하며 메모리 사용 확장이 가능(Large Address Aware)합니다.

LabVIEW (64비트)는 Windows 버전에 따라 최대 8 TB 또는 128 TB까지 버추얼 메모리를 사용할 수 있습니다. 또한 LabVIEW (64비트)가 사용할 수 있는 버추얼 메모리의 실제 크기는 물리적 RAM의 크기와 최대 페이징 파일 크기에 따라 달라집니다.

VI 구성요소 메모리 관리

VI는 다음의 4가지 주요한 성분을 가집니다:

- 프론트패널
- 블록다이어그램
- 코드(기계 코드로 컴파일되는 다이어그램)
- 데이터(컨트롤과 인디케이터 값, 기본 데이터, 다이어그램 상수 데이터 등)

VI를 로드할 때, 그 VI의 프론트패널, 코드(플랫폼에 일치한다면), 데이터가 메모리에 로드됩니다. VI가 플랫폼에 있어서 변경이나 SubVI의 인터페이스에서 변경 때문에 리컴파일할 필요가 있다면, 블록다이어그램 또한 메모리에 로드됩니다.

또한 VI는 메모리에 그 SubVI의 코드와 데이터 공간도 로드합니다. 특정 상황에서, 일부 SubVI의 프론트패널도 메모리에 로드될 수 있습니다. 예를 들면, 프로퍼티 노드는 프론트패널 컨트롤의 상태 정보를 다루기 때문에, 이는 SubVI가 프로퍼티 노드를 사용하는 경우 발생할 수 있습니다.

VI 일반적으로 VI의 한 섹션을 SubVI로 변경할 때, 그다지 많은 메모리를 사용하지 않는다는 것입니다. SubVI가 없는 거대한 단일 VI를 생성할 때, 메모리에 그 상위 레벨 VI를 위한 프론트패널, 코드, 데이터를 가지게 됩니다. 그러나, VI를 여러 SubVI로 나누면, 상위 레벨 VI를 위한 코드는 더 작게 되고 SubVI의 코드와 데이터는 메모리에 있게 됩니다. 일부 경우에, 더 낮은 런타임 **메모리 사용**을 볼 수 있습니다.

큰 VI는 편집에 걸리는 시간도 더 깁니다. 다음의 방법으로 이 문제를 피할 수 있습니다:

- (권장) VI를 여러 SubVI로 나눕니다. 편집기는 작은 VI를 더 효율적으로 다룰 수 있을 뿐 아니라, 일반적으로 계층구조에 계층이 많을 수록 VI 조직이 유지하고 읽기 쉽습니다.
- 큰 VI의 **컴파일된 코드 복잡도**에 근거하여 LabVIEW가 **실행 시간보다 편집기 응답을 우선**하도록 설정합니다.



노트

주어진 VI의 프론트패널이나 블록다이어그램이 화면보다 훨씬 더 크면, 더 쉽게 접근할 수 있도록 VI를 여러 SubVI로 나눌 수 있습니다.

데이터 흐름 프로그래밍과 데이터 버퍼

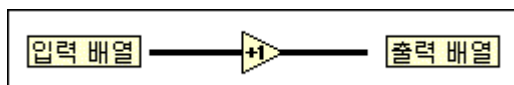
데이터 흐름 프로그래밍에서, 일반적으로 변수를 사용하지 않습니다. 데이터 흐름 모델은 일반적으로 노드를 데이터 입력을 소비하고 데이터 출력을 생성하는 것으로 설명합니다. 이 모델을 그대로 구현하면 많은 양의 메모리를 사용하고, 성능이 느린 어플리케이션

션이 생성됩니다. 모든 함수는 출력이 전달되는 모든 대상에 데이터의 복사본을 생성합니다. LabVIEW는 inplaceness를 통해 이 부분의 실행을 개선시킬 수 있습니다.

LabVIEW는 inplaceness를 사용하여 언제 메모리를 다시 사용할지, 또한 각 출력 터미널의 복사본을 만들지를 결정합니다. LabVIEW가 데이터를 입력에서 출력으로 복사하지 않을 때, 해당 입력과 출력의 데이터는 in-place라고 할 수 있습니다.

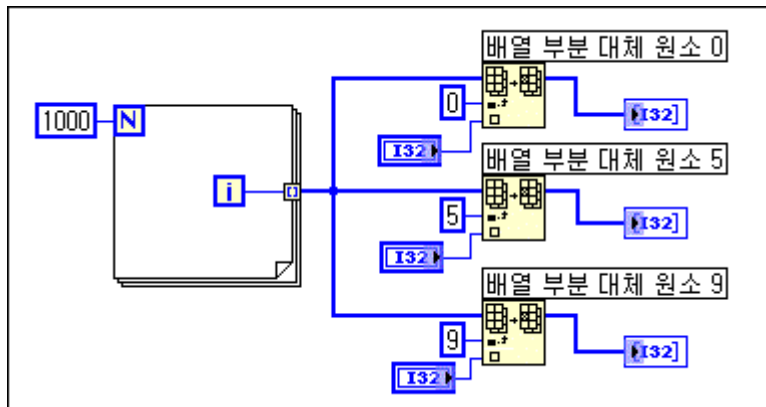
버퍼 할당 보이기 윈도우를 사용하여 LabVIEW가 어디에 데이터의 복사본을 생성할 수 있는지 알려줍니다.

예를 들면, 컴파일러에 대한 더 전통적인 접근 방식으로, 다음 블록다이어그램은 두 개의 데이터 메모리 블록을 사용하는데, 하나는 입력으로 또 다른 하나는 출력으로 사용합니다.



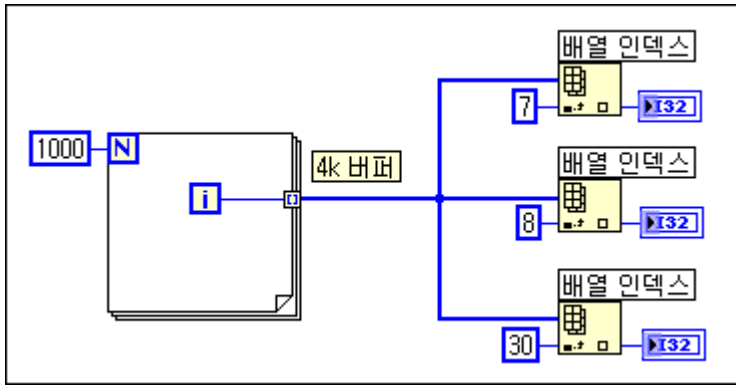
입력 배열과 출력 배열은 같은 개수의 원소를 가지고 있고, 이 두 배열의 데이터 타입은 같습니다. 입력 배열을 데이터의 버퍼로서 생각합니다. 출력에 대해 새로운 버퍼를 생성하는 대신에, 컴파일러는 입력 버퍼의 메모리를 출력 버퍼에 다시 사용합니다. 이와 같이 inplaceness를 이용하면 실행 시 메모리를 할당할 필요가 없기 때문에 메모리가 절약되고 더 빠른 실행이 가능합니다.

그러나 컴파일러는 다음 블록다이어그램에서 보는 것처럼 모든 경우에 메모리 버퍼를 다시 사용할 수는 없습니다.



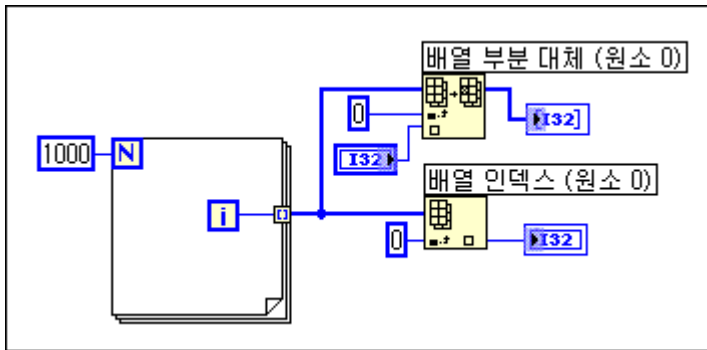
신호는 단일 데이터 소스를 여러 대상으로 전달합니다. [배열 부분 대체] 함수는 입력 배열을 변경하여 출력 배열을 생성합니다. 이 경우, 이 함수 중 한 개의 데이터는 in-place이며, 이때 출력은 다른 출력과는 달리 입력 배열의 데이터를 다시 사용합니다. 이렇게 하여, 컴파일러는 두 개의 함수에 대해 새로운 데이터 버퍼를 생성하고 버퍼에 배열 데이터를 복사합니다. 이 블록다이어그램은 약 12 KB(원래 배열에 4KB 그리고 2개의 여분의 데이터 버퍼 각각에 4KB)를 사용합니다.

이제 다음의 블록다이어그램을 살펴봅시다.



이전처럼, 입력은 세 개의 함수로 갈라집니다. 그러나, 이 경우에 [배열 인덱스] 함수는 입력 배열을 변경하지 않습니다. 데이터를 여러 위치로 전달하고 그 위치에서 데이터를 변경하지 않고 읽으면, LabVIEW는 데이터의 복사본을 만들지 않습니다. 이 결과, 모든 데이터가 in-place입니다. 이 블록 다이어그램은 약 4 KB의 메모리를 사용합니다.

마지막으로, 다음의 블록 다이어그램을 살펴봅시다.



이 경우에, 입력은 두 개의 함수로 나뉘는데, 그 중 하나는 데이터를 변경합니다. 그 두 개의 함수 사이에 의존성은 없습니다. 그러므로, 적어도 하나의 복사본은 만들어 둘 필요가 있다는 것을 예상할 수 있으므로 [배열 부분 대체] 함수가 안전하게 데이터를 변경할 수 있습니다. 그러나 이 경우에, 컴파일러는 데이터를 읽는 함수가 먼저 실행되고 데이터를 변경하는 함수가 마지막으로 실행되는 방식으로 함수의 실행 일정을 잡습니다. 이러한 방식으로 [배열 부분 대체] 함수는 배열 복사를 생성하지 않고, 입력 배열 버퍼를 다시 사용하여 이 함수의 데이터를 in-place로 만듭니다. 노드의 순서는 중요하기 때문에, 다른 노드의 입력을 위해 한 노드의 순서나 출력을 사용하여 순서를 명시해 놓습니다.

실제로, 컴파일러에 의한 블록 다이어그램의 분석은 완전하지 않습니다. 몇가지 경우에, 컴파일러는 블록 다이어그램 메모리를 다시 사용하기 위한 최적의 방법을 결정할 수 없을 수 있습니다.

조건적 인디케이터와 데이터 버퍼

사용자가 블록 다이어그램을 만드는 방법은 LabVIEW가 데이터 버퍼를 다시 사용하는 것을 막을 수 있습니다. SubVI에서 조건적 인디케이터를 사용하면 LabVIEW가 데이터 버퍼 사용을 최적화하는 것을 막습니다. 조건적 인디케이터는 케이스 구조나 For 루프 안의 인디케이터입니다. 조건적으로 실행되는 코드 경로에 인디케이터를 놓으면 시스템을 통한 데이터의 흐름을 깨고 LabVIEW는 입력으로부터 데이터 버퍼를 다시 사용할 수 없을 것입니다. 그러나 그 대신 데이터를 인디케이터에 강제로 복사해 놓도록 합니다. 케이스 구조와 For 루프의 밖에 인디케이터를 놓을 때, LabVIEW는 루프나 구조 안에서 데이터를 직접 변경하고, 데이터 복사본을 생성하는 대신에 인디케이터에 데이터를 전달합니다.

다. 케이스 구조 안에 인디케이터를 놓는 대신에 다른 경우에 대해 상수를 생성할 수 있습니다.

메모리 사용 모니터하기

여러가지 방법을 통해 메모리 사용 정보를 확인할 수 있습니다.

현재 VI에서 메모리 사용을 보려면, 파일»VI 프로퍼티를 선택하고 최상위 풀다운 메뉴에서 메모리 사용을 선택합니다. 이때, SubVI의 메모리 사용에 대한 정보는 포함하지 않습니다. 성능과 메모리 프로파일 윈도우를 사용하여 메모리에 있는 모든 VI가 사용하는 메모리를 모니터할 수 있고, 성능에 문제가 있는 SubVI를 파악할 수 있습니다. VI 성능과 메모리 프로파일 윈도우는 매번 실행에서 각 VI가 사용하는 바이트와 블록의 최소, 최대, 평균의 **통계값을 나타냅니다**.

노트 정확한 VI 런타임 성능을 모니터하려면 **컴파일된 코드를 분리하지 않고** 모든 VI를 LabVIEW의 현재 버전에 저장해야 하며 그 이유는 다음과 같습니다:

- 실행 취소 기능은 객체 및 데이터의 임시적인 복사본을 생성하며, 이는 VI의 메모리 요구를 증가시킬 수 있습니다. VI를 저장하면 실행 취소에 의해 생성된 복사본이 제거되고 그 결과 정확한 메모리 정보를 알게 됩니다.
 - 이전 버전의 LabVIEW에서 만든 VI를 저장하지 않고 실행하면, VI의 메모리 요구가 증가되고 VI 런타임 성능은 감소합니다.
-

버퍼 할당 보이기 윈도우를 사용하여 블록다이어그램에서 LabVIEW가 버퍼의 형태로 메모리 공간을 할당하는 영역을 파악할 수 있습니다.

노트 버퍼 할당 보이기 윈도우는 LabVIEW Full과 Professional Development Systems에서만 제공됩니다.

도구»프로파일»버퍼 할당 보이기를 선택하여 버퍼 할당 보이기 윈도우를 디스플레이합니다. 버퍼를 확인할 데이터 타입의 옆에 확인 표시를 하고 새로 고침 버튼을 클릭하십시오. 블록다이어그램에 나타나는 검은 사각형은 LabVIEW가 데이터 공간 할당을 위해 버퍼를 생성하는 위치를 나타냅니다.

LabVIEW가 각 버퍼에 할당하는 메모리의 크기는 가장 큰 **데이터 타입**의 크기와 같습니다. VI를 실행할 때, LabVIEW는 생성된 버퍼를 사용하여 데이터를 저장할 수도 있으나, 항상 그렇지는 않습니다. LabVIEW가 데이터의 복사본을 만들지는 런타임 때 결정 지어지고, 때로는 VI가 다이내믹 데이터에 의존하기 때문에 복사본을 만들지 미리 알 수는 없습니다.

VI가 버퍼 할당에 메모리를 필요로 하면, LabVIEW는 해당 버퍼의 데이터 복사본을 만듭니다. 버퍼가 데이터의 복사본을 필요로 하는지 확실치 않을 때에도, LabVIEW는 버퍼의 복사본을 만들기도 합니다.

노트 버퍼 할당 보이기 윈도우는 블록다이어그램 상에서 검정색 사각형을 사용해 실행 버퍼만을 식별합니다. 이 윈도우를 사용하여 프론트패널에서 동작 버퍼를 식별할 수는 없습니다.

LabVIEW의 버퍼 생성 위치를 알면, LabVIEW가 **보다 적은 양의 메모리로 VI를 실행할 수 있도록 VI를 편집**할 수 있습니다. LabVIEW는 VI를 실행하기 위해 메모리를 할당해야 하므로 모든 버퍼를 지울 수는 없습니다.

LabVIEW가 VI를 다시 컴파일할 만큼 VI를 변경하게 되면, 버퍼 정보가 더 이상 정확하지 않으므로 검정색 사각형은 사라집니다. VI를 다시 컴파일하고 검정색 사각형을 디스플레이하기 위해서 버퍼 할당 보이기 윈도우의 새로 고침 버튼을 클릭하십시오. 버퍼 할당 보이기 윈도우를 닫게 되면 검정색 사각형은 사라집니다.

노트 또한 LabVIEW Desktop Execution Trace Toolkit을 이용하여 스레드 사용과 메모리 손실, 기타 LabVIEW 프로그래밍의 영역을 모니터링할 수 있습니다.

도움말 >> LabVIEW에 대하여를 선택하여 어플리케이션에 의해 사용되는 총 메모리의 양을 요약하는 통계값을 보여줍니다. 이것은 VI를 위한 메모리와 어플리케이션이 사용하는 메모리를 포함합니다. VI 세트의 실행 전과 후에 이 메모리 양을 검사하여 VI가 얼마나 많은 메모리를 사용하게 되는가를 대략적으로 알 수 있습니다.

더 효과적인 메모리 사용을 위한 규칙

이전 섹션의 주요 포인트는 컴파일러가 메모리를 지능적으로 다시 사용하려는 시도에 대한 것입니다. 컴파일러가 언제 메모리를 다시 사용할 수 있고 사용할 수 없는지에 대한 규칙은 매우 복잡합니다. 실제로, 다음 규칙은 메모리를 효과적으로 사용하는 VI를 생성하는데 도움을 줄 수 있습니다:

- VI를 SubVI로 나누어도 일반적으로 메모리 사용에 나쁜 영향을 주지는 않습니다. 많은 경우에, SubVI가 실행되지 않을 때 실행 시스템이 SubVI 데이터 메모리를 다시 요청할 수 있기 때문에 메모리 사용은 향상됩니다.
- 스칼라 값의 복사본에 대해 너무 염려하지 마십시오. 메모리 사용에 부정적인 영향을 미칠 정도가 되려면 스칼라가 엄청나게 많아야 합니다.
- 배열이나 문자열을 가지고 작업할 때 글로벌과 로컬 변수를 지나치게 사용하지 않습니다. 글로벌이나 로컬 변수를 읽으면 LabVIEW는 데이터를 복사하는 일이 발생합니다.
- 필요하지 않다면 열려 있는 프런트패널에 큰 배열과 문자열을 디스플레이하지 않습니다. 열려 있는 프런트패널의 컨트롤과 인디케이터는 디스플레이하는 데이터를 복사해서 가지고 있습니다.

팁 차트 인디케이터가 필요한 경우, 차트 히스토리가 디스플레이되는 데이터의 복사본을 보유하게 된다는 점을 염두에 두십시오. 차트 히스토리가 채워질 때까지 지속되는데, 이후 LabVIEW는 더 이상의 메모리를 차지하지 않게 됩니다. LabVIEW는 VI가 다시 시작될 때 자동으로 차트 히스토리를 지우지 않고, 프로그램이 실행되는 과정에서 차트 히스토리가 지워질 수 있도록 합니다. 이를 위해, 차트에 대한 히스토리 데이터 속성 노드에 빈 배열을 작성하십시오.

- 패널 업데이트 연기 프로퍼티를 사용합니다. 이 프로퍼티를 참으로 설정할 때, 컨트롤 값을 변경할지라도 프런트패널 인디케이터 값은 바뀌지 않습니다. 인디케이터를 새 값으로 채우기 위해 OS는 메모리를 사용할 필요는 없습니다.

노트 일반적으로 LabVIEW는 SubVI를 호출할 때 SubVI의 프런트패널을 열지 않습니다.

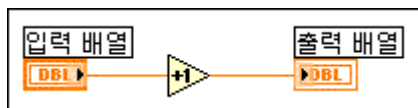
- SubVI의 프런트패널이 디스플레이되지 않는다면, SubVI에 사용되지 않은 프로퍼티 노드를 남겨놓지 마십시오. 프로퍼티 노드는 SubVI의 프런트패널이 메모리에 남아있도록 하여 불필요한 메모리 사용을 만들 수 있습니다.
- 블록다이어그램을 만들 때, 입력의 크기가 출력의 크기와 다른 위치를 확인합니다. 예를 들면, [배열 만들기] 또는 [문자열 연결] 함수를 사용하여 자주 배열이나 문자열의 크기를 증가시키는 위치에서는 데이터의 복사본을 생성하는 것입니다.
- 데이터를 SubVI와 함수로 전달할 때 배열에 대해 **일관된 데이터 타입**을 사용하고 강제 변환점을 찾습니다. 데이터 타입을 변경할 때, 실행 시스템은 데이터의 복사본을 만듭니다.
- 큰 배열이나 문자열을 가진 클러스터나 클러스터의 배열과 같은 복잡하고 계층구조적인 데이터 타입을 사용하지 마십시오. 더 많은 메모리를 사용하게 될 것입니다. 가능하면, **더 효과적인 데이터 타입을 사용**합니다.
- 필요하지 않는 한 투명하고 겹치는 프런트패널 객체를 사용하지 않습니다. 그런 객체는 더 많은 메모리를 사용할 수 있습니다.

VI의 성능을 최적화하는 방법에 대한 구체적인 방법은 **LabVIEW Style Checklist**를 참조하십시오.

프런트패널에서 메모리 문제

프런트패널이 열려 있을 때, 컨트롤과 인디케이터는 디스플레이하는 데이터의 컨트롤과 인디케이터 자체의 복사본을 가지고 있습니다.

다음 블록다이어그램은 프런트패널 컨트롤과 인디케이터가 추가된 증가 함수를 보여줍니다.



VI를 실행할 때, 프런트패널 컨트롤의 데이터가 블록다이어그램에 전달됩니다. 증가 함수가 입력 버퍼를 다시 사용합니다. 그리고 나서 인디케이터는 디스플레이할 목적으로 데이터의 복사본을 만듭니다. 이렇게 해서 버퍼의 3개의 복사본이 생깁니다.

프런트패널 컨트롤의 데이터 보호는 이어지는 노드에 데이터가 전달될 때 데이터를 컨트롤에 입력하는 것, 관련된 VI를 실행하는 것, 컨트롤에서 데이터 변경을 보는 것을 막습니다. 마찬가지로, 인디케이터의 경우에도 데이터는 보호되므로 인디케이터가 새로운 데이터를 받을 때까지 이전 내용을 디스플레이할 수 있습니다.

SubVI와 함께, 입력과 출력으로서 컨트롤과 인디케이터를 사용할 수 있습니다. 다음 조건에 따라 실행 시스템은 SubVI의 컨트롤과 인디케이터 데이터의 복사본을 만듭니다:

- 프런트패널이 메모리에 있습니다. 이것은 다음의 이유 때문에 발생할 수 있습니다:
 - 프런트패널이 열려 있습니다.
 - VI가 변경되었지만 저장되지 않았습니다(VI의 모든 구성요소가 VI가 저장될 때까지 메모리에 있습니다).
- 패널이 데이터 인쇄를 사용합니다.
- 블록다이어그램이 프로퍼티 노드를 사용합니다.

- VI가 로컬 변수를 사용합니다.
- 패널이 데이터 로깅을 사용합니다.

프로퍼티 노드가 달한 패널을 가진 SubVI에서 차트 히스토리를 읽을 수 있도록 하기 위해서, 컨트롤이나 인디케이터는 전달되는 데이터를 디스플레이해야 합니다. 이와 같이 다른 많은 속성들이 있기 때문에, SubVI가 프로퍼티 노드를 사용한다면 실행 시스템은 SubVI 패널을 메모리에 두어야 합니다.

프론트패널이 프론트패널 데이터 로깅이나 데이터 데이터 인쇄를 사용한다면, 컨트롤과 인디케이터가 그 데이터의 복사본을 유지합니다. 또한, 프론트패널은 데이터 인쇄를 위해 메모리에 남아있기 때문에 패널을 인쇄할 수 있습니다.

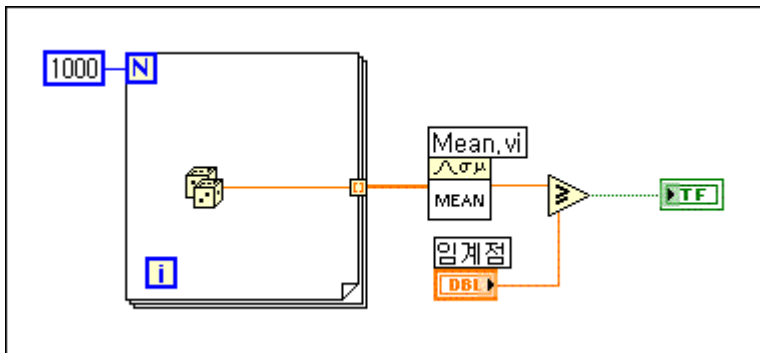
VI 프로퍼티 대화 상자나 SubVI 노드 설정 대화 상자를 사용하여 SubVI를 호출할 때 그 프론트패널을 디스플레이하도록 설정하면, 그 SubVI를 호출할 때 프론트패널이 메모리에 로드됩니다. 원래 달혀있으면 실행 후 달기 아이템을 설정한다면, LabVIEW는 SubVI가 실행을 끝낼 때 메모리에서 그 프론트패널을 제거합니다.

데이터 메모리를 다시 사용할 수 있는 SubVI

일반적으로, SubVI의 블록다이어그램을 상위 레벨에서 복사하는 것만큼 쉽게 SubVI는 그 호출자로부터 데이터 버퍼를 사용할 수 있습니다. 대부분의 경우에, 블록다이어그램의 한 섹션을 SubVI로 변환한다고 더 많은 메모리를 사용하지는 않습니다. 특별한 디스플레이가 요구되는 VI에 대해, 이전 섹션에서 설명된 것처럼, 프론트패널과 컨트롤을 위한 일부 추가된 메모리 사용이 있을 수 있습니다.

언제 메모리의 할당이 해제되는지 이해하기

다음의 블록다이어그램을 살펴봅니다.



평균 VI를 실행한 후에 데이터의 배열은 더 이상 필요 없습니다. 데이터가 더 이상 필요 없는 때를 결정하는 것은 더 큰 블록다이어그램에서 매우 복잡할 수 있기 때문에, 실행되는 동안 특정한 VI의 데이터 버퍼 할당을 해제하지 않습니다.

macOS에서 실행 시스템이 작은 메모리를 차지하면, 현재 실행되지 않는 모든 VI에 의해 사용되는 데이터 버퍼의 할당을 해제합니다. 실행 시스템은 프론트패널 컨트롤, 인디케이터, 글로벌 변수, 초기화되지 않은 시프트 레지스터에 의해 사용된 메모리의 할당을 해제하지 않습니다.

이제 같은 VI를 더 큰 VI의 SubVI로 생각합니다. 데이터의 배열은 SubVI에서만 생성되고 사용됩니다. macOS에서, SubVI를 실행하지 않고 시스템이 낮은 메모리를 차지하면, SubVI에서 데이터의 할당을 해제할 수 있습니다. 이러한 경우에 SubVI를 사용하여 메모리 사용을 절약할 수 있습니다.

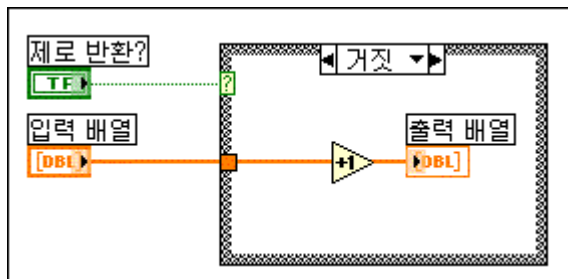
Windows와 Linux 플랫폼에서, VI가 닫히고 메모리로부터 제거되지 않으면 데이터 버퍼는 정상적으로 할당을 해제하지 않습니다. 메모리는 필요할 때마다 OS로부터 할당받고 가상 메모리가 이 플랫폼에서 잘 작동합니다. 조각난 메모리 때문에, 어플리케이션은 실제 사용하는 것보다 더 많은 메모리를 사용하는 것으로 나타날 수 있습니다. 메모리를 할당받고 비워가면서, 어플리케이션은 메모리 사용을 통합하여 사용되지 않은 블록을 OS에 반환할 수 있습니다.

[할당 해제 요청] 함수를 사용하여 이 함수를 포함하는 VI가 실행된 후에 사용하지 않는 메모리 할당을 해제할 수 있습니다. 이 함수는 고급 성능 최적화에만 사용됩니다. 일부 경우, 사용되지 않은 메모리의 할당을 해제하면 성능이 향상됩니다. 그러나 지나치게 메모리의 할당을 해제하면 LabVIEW가 해당 메모리를 재사용하기보다는 반복해서 메모리 공간을 재할당하게 됩니다. 사용자 VI가 많은 양의 데이터를 할당하지만 이러한 할당분을 재사용하지 않을 때 이 함수를 사용하십시오. 최상위 VI가 SubVI를 호출할 때, LabVIEW는 해당 SubVI가 실행되는 메모리의 데이터 공간을 할당합니다. SubVI가 실행을 완료해도, LabVIEW는 최상위 VI가 실행을 마치거나 전체 어플리케이션이 종료될 때까지 데이터 공간 할당을 해제하지 않기 때문에 메모리 부족 상태가 되거나 성능이 저하될 수 있습니다. [할당 해제 요청] 함수를 메모리 할당 해제하려는 SubVI에 위치시키십시오. 플래그 볼리언 입력을 참으로 설정하면, LabVIEW는 subVI의 데이터 공간 할당을 해제하여 메모리 사용을 감소시킵니다.

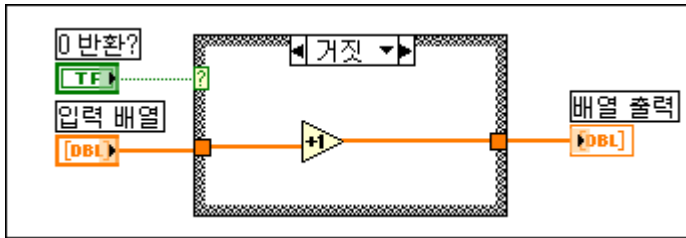
출력이 입력 버퍼를 언제 다시 사용할 수 있는지 결정하기

출력이 입력과 같은 크기와 데이터 타입을 가지고 다른 곳에서 입력이 요청이 없으면, 출력은 입력 버퍼를 다시 사용할 수 있습니다. 이전에 언급했던 것처럼, 몇몇 경우에는 입력이 다른 곳에서 사용될 때에도 컴파일러와 실행 시스템이 출력 버퍼를 위해 입력을 다시 사용할 수 있는 방식으로 코드 실행을 명령할 수 있습니다. 그러나, 이러한 규칙은 복잡합니다. 그것에 의존하지 마십시오.

버퍼 할당 보이기 윈도우를 사용하여 출력 버퍼가 입력 버퍼를 다시 사용할 것인지 알 수 있습니다. 다음의 블록 다이어그램에서, 케이스 구조의 각 경우에 인디케이터를 놓으면 데이터 흐름을 막게 되는데 그 이유는 LabVIEW가 각 인디케이터에서 데이터의 복사본을 생성하기 때문입니다. LabVIEW는 입력 배열을 위해 생성한 버퍼를 사용하지 않지만 대신에 출력 배열을 위한 데이터의 복사본을 생성합니다.



인디케이터를 케이스 구조 밖으로 이동시키면, 출력 버퍼는 입력 버퍼를 다시 사용하는 데, 그 이유는 LabVIEW가 인디케이터가 디스플레이하는 데이터의 복사본을 만들 필요가 없기 때문입니다. LabVIEW는 VI에서 나중에 입력 배열의 값을 요청하지 않기 때문에, 증가 함수는 직접적으로 입력 배열을 변경하고 출력 배열에 그 값을 전달할 수 있습니다. 이때, LabVIEW는 데이터를 복사할 필요가 없으므로 다음 블록 다이어그램에서 보는 것처럼 버퍼가 출력 배열에 나타나지 않습니다.



데이터 타입 일치로 메모리 사용 최적화하기

데이터 타입을 일관성있게 사용하는 경우, LabVIEW는 출력을 생성할 때 입력에 할당했던 메모리 버퍼를 다시 사용할 수 있습니다. 이 경우 VI의 메모리 사용 및 실행 시간이 향상됩니다. 그러나 입력과 출력의 데이터 타입이 다른 경우, LabVIEW에서 해당 입력에 할당된 메모리를 출력에서 다시 사용할 수 없게 됩니다.

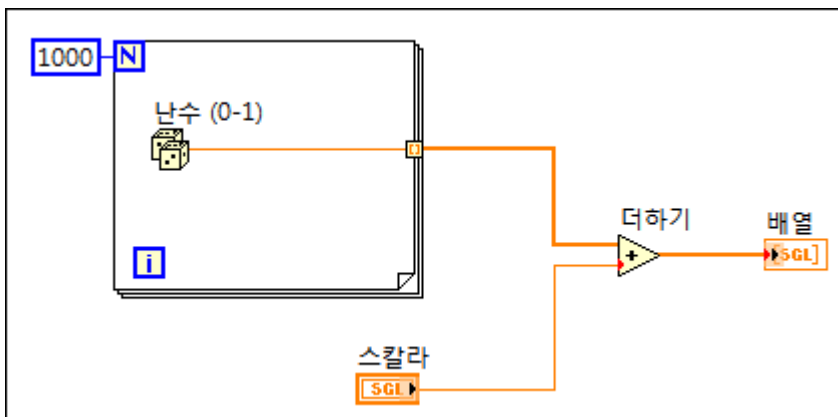
예를 들어 32 비트 정수를 16 비트 정수에 더하는 경우, LabVIEW는 16 비트 정수를 32 비트 정수로 강제 변환시킵니다. 컴파일러는 강제 변환된 데이터에 대해 새로운 버퍼를 생성하고, LabVIEW는 [더하기] 함수에 **강제 변환점**을 놓습니다. 입력이 다른 모든 요구사항을 충족한다고 가정하면, 이 예에서 LabVIEW는 32비트 정수 입력을 출력 버퍼로 사용할 수 있습니다. 그러나 LabVIEW가 16 비트 정수를 강제 변환하였기 때문에, LabVIEW는 해당 입력에 할당된 메모리를 다시 사용할 수 없습니다.

메모리 사용을 최소화하기 위해, 가능한 모든 곳에서 일관된 데이터 타입을 사용합니다. 데이터 타입을 일관성있게 사용하면, 더 큰 크기의 데이터로 강제 변환해야 하는 경우를 피하게 되어 데이터의 복제본을 더 적게 생성하게 됩니다. 또한 일부 어플리케이션에서는 더 작은 데이터 타입의 사용으로 데이터 사용을 최소화시킬 수도 있습니다. 예를 들면 8 바이트 배정도 숫자 대신에 4 바이트 단정도 숫자를 사용하는 것을 고려할 수 있습니다. 그러나 SubVI에 연결하려는 데이터 타입과 SubVI가 받기를 예상하는 데이터 타입을 일치시키면 불필요한 강제 변환을 피할 수 있습니다.

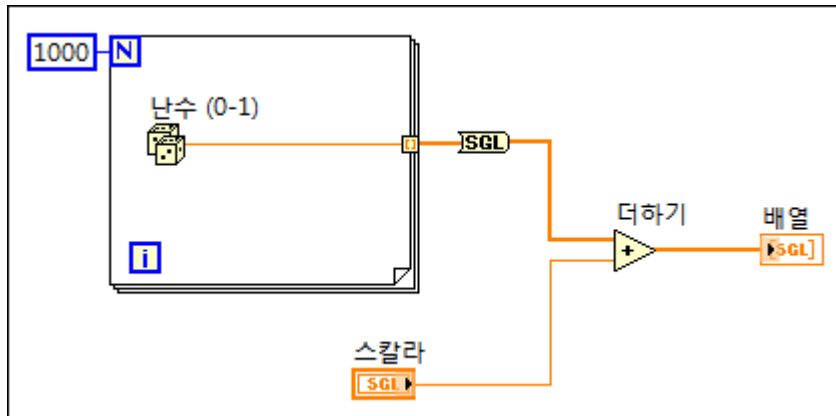
특정 타입의 데이터 생성시 메모리 사용 최적화하기

특정 데이터 타입의 데이터를 생성할 때, 블록 다이어그램에서 데이터 타입을 변환해야 합니다. 그러나 변환 함수를 사용하면 LabVIEW가 큰 배열을 생성하기 전에 데이터 타입을 변환하여 메모리 사용을 최적화할 수 있습니다.

다음의 예에서 다른 VI 입력의 데이터 타입과 일치시키려면 출력 데이터가 단정도 부동소수가 되어야 합니다. LabVIEW는 1000개의 임의의 값의 배열을 생성하고 그 배열을 스칼라 값에 더합니다. LabVIEW는 단정도 부동소수 스칼라 데이터 타입을 임의의 배정도 부동소수와 일치시키기 위해 강제 변환시키고, [더하기] 함수에 강제 변환점을 놓습니다.

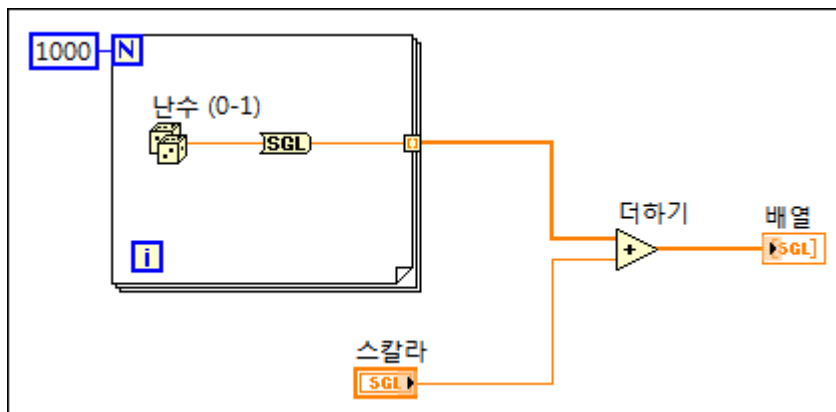


다음의 블록다이어그램에서는 [단정도 부동소수로] 함수를 사용하여 배정도 부동소수를 변환하여 이 문제를 해결하려 합니다.



이 함수는 LabVIEW가 배열을 생성한 이후 큰 배열의 데이터 타입을 변환하기 때문에, VI가 사용하는 메모리의 양은 강제 변환점이 있는 경우와 같은 양입니다.

다음 블록다이어그램은 LabVIEW가 배열을 생성하기 전에 임의의 수를 변환하여 메모리 사용 및 실행 속도를 최적화하는 방법을 보여줍니다.



변환이 불가피한 경우 LabVIEW에서 큰 배열이 생성되기 전에 [변환] 함수를 사용하여, 배열에 할당된 큰 데이터 버퍼의 데이터 타입이 변환되는 것을 피합니다. LabVIEW에서 큰 배열이 생성되기 전에 데이터를 변환하면 VI의 메모리 사용을 최적화할 수 있습니다.

계속해서 데이터 크기가 바뀌는 것을 피하기

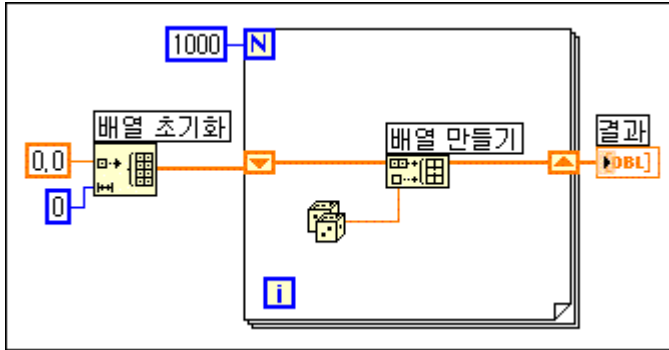
출력 크기가 입력 크기와 다르면, 출력은 입력 데이터 버퍼를 다시 사용하지 않습니다. 이것이 배열이나 문자열의 크기를 증가시키거나 감소시키는 [배열 만들기], [문자열 연결], [배열 잘라내기]와 같은 함수의 경우입니다. 배열과 문자열을 가지고 작동할 때, 이 함수를 계속 사용하는 것은 피해야 합니다. 왜냐하면 이들은 데이터를 계속해서 복사하기 때문에 사용자 프로그램이 더 많은 데이터 메모리를 사용하고 더 느리게 실행되기 때문입니다.

예제 1: 배열 만들기

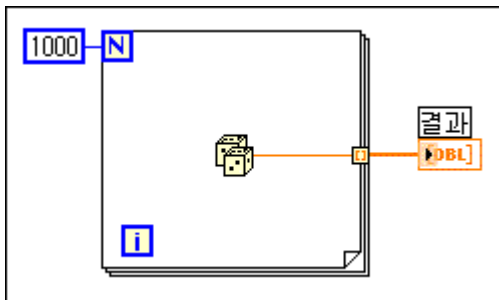
데이터의 배열을 생성하는데 사용되는 다음 블록다이어그램을 살펴봅니다. 이 블록다이어그램은 새 원소를 연결하기 위해 계속해서 [배열 만들기]를 호출하여 루프에 배열을 생성합니다. 입력 배열은 [배열 만들기]에 의해 다시 사용됩니다. VI는 각 반복마다 계속

해서 버퍼의 크기를 다시 조정하여 새로운 배열을 위한 공간을 확보하고 새 원소를 추가합니다. 결과적으로 실행 속도는 느려지는데, 특히 루프가 여러 번 반복해서 실행될 때 느려집니다.

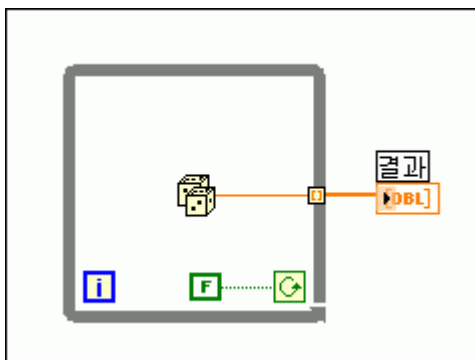
노트 배열, 클러스터, 웨이브폼, 배리언트를 조작할 때 In Place 원소 구조를 사용하여 VI에서의 메모리 사용을 향상시킬 수 있습니다.



루프의 매 반복마다 배열에 값을 추가하려면, 루프 경계선의 오토인덱싱을 사용하는 것이 가장 좋은 성능을 냅니다. For 루프를 가지고 VI는 배열의 크기를 미리 결정하고(N에 연결된 값을 기본으로) 버퍼의 크기를 한번만 바꿉니다.



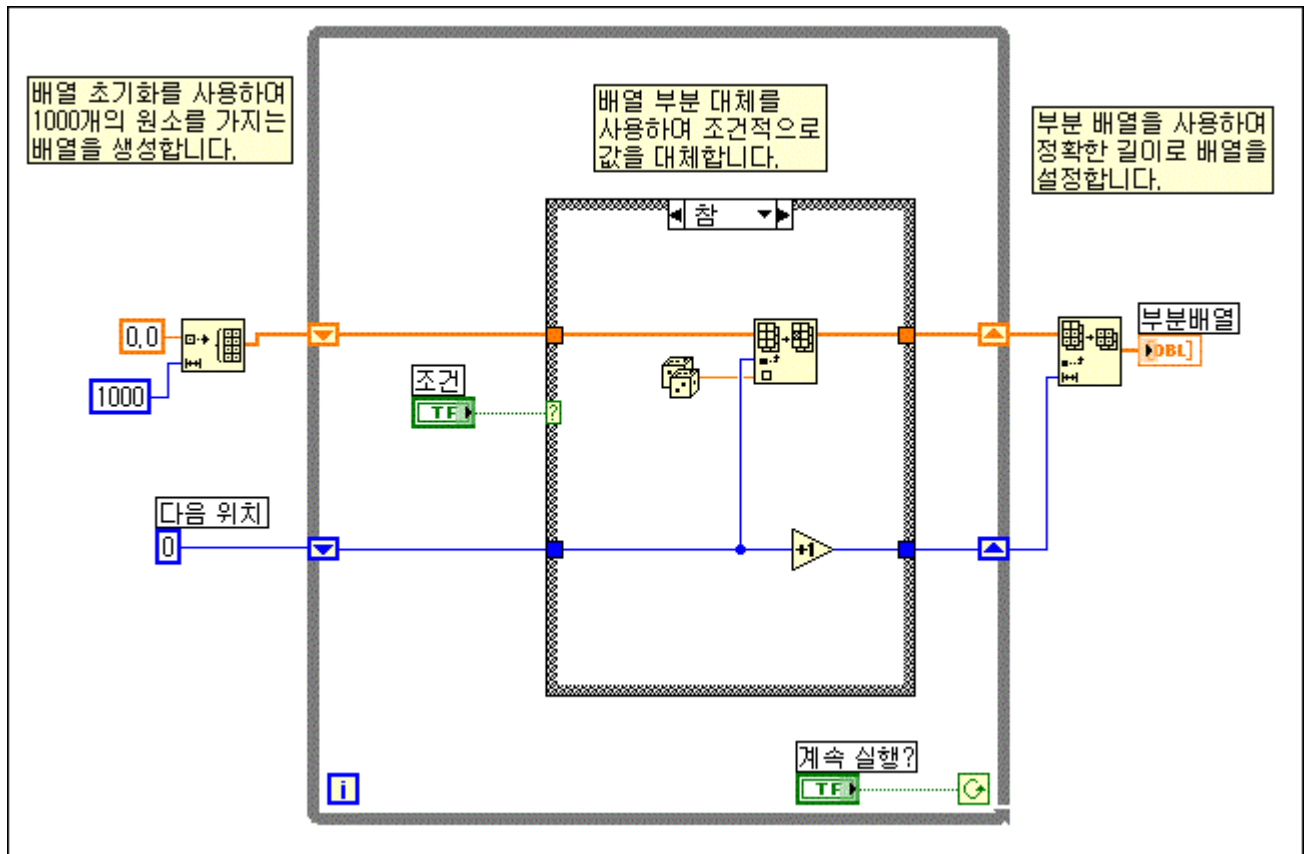
While 루프에서 오토인덱싱은 그리 효과적이지가 못하는데, 그 이유는 배열의 마지막 크기를 알 수 없기 때문입니다. 그러나 While 루프에서 오토인덱싱은 출력 배열 크기를 상당히 많이 증가시켜 출력 배열의 크기 조절을 피하도록 해줍니다. 루프가 끝날 때, 출력 배열의 크기는 적당한 크기로 조절됩니다. While 루프 오토인덱싱의 성능은 For 루프 오토인덱싱과 거의 일치합니다.



오토인덱싱은 루프의 매 반복마다 그 배열 결과에 값을 추가할 것이라고 가정합니다. 조건적으로 값을 배열에 추가해야 하지만 배열 크기에서 상위 리미트를 결정할 수 있는 경우, 배열을 미리 할당하고 배열을 채우기 위해 [배열 부분 대체]를 사용하는 것을 고려해 볼 수 있습니다.

배열 값을 채우기를 끝마칠 때, 배열의 크기를 올바른 크기로 조절할 수 있습니다. 배열은 한번만 생성되고 [배열 부분 대체]는 출력 버퍼를 위해 입력 버퍼를 다시 사용할 수 있습니다. 이것의 성능은 오토인덱싱을 사용하는 루프의 성능과 매우 비슷합니다. 이 기술을 사용할 때, [배열 부분 대체]는 배열의 크기를 조절하지 않기 때문에 값을 대체하는 배열이 그 결과 데이터를 가지고 있을 만큼 충분히 커야 합니다.

이 과정의 예제가 다음 블록다이어그램에 나타나 있습니다.

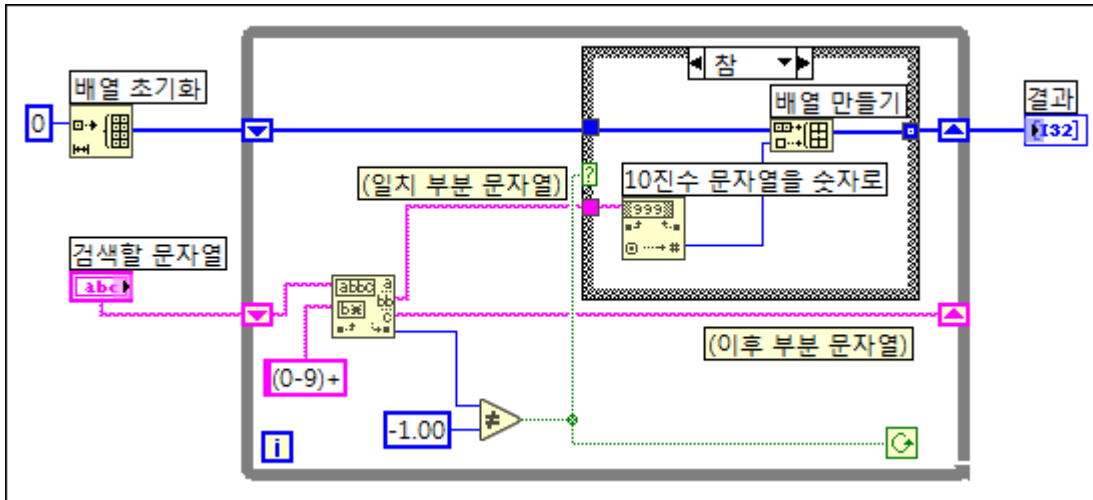


예제 2: 문자열을 통해 검색하기

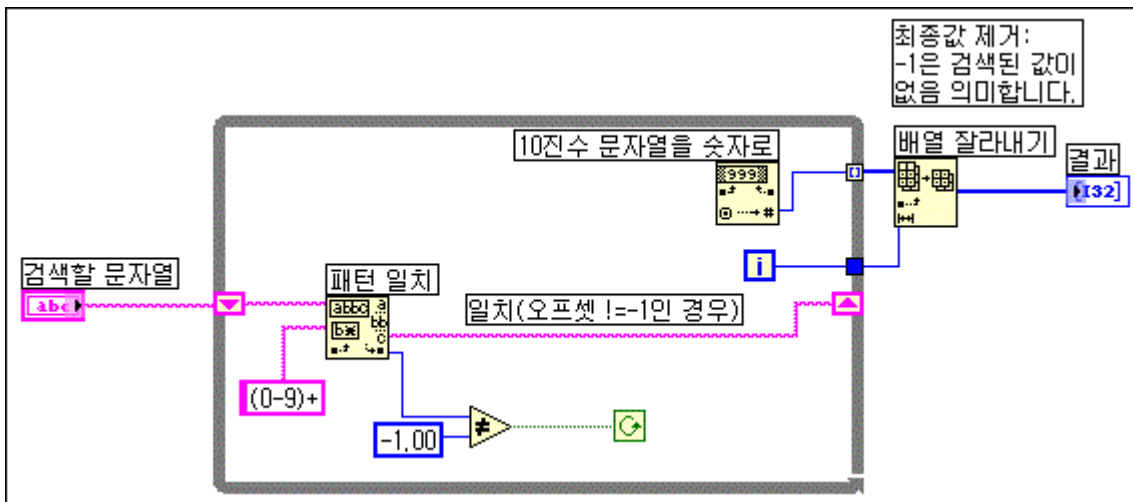
[패턴 일치] 함수를 사용하여 패턴에 맞는 문자열을 검색할 수 있습니다. 그것을 사용하는 방법에 따라 문자열 데이터 버퍼를 불필요하게 생성하여 성능을 낮출 수 있습니다.

문자열에서 정수를 일치시키려고 하는 경우, [0-9]+를 이 함수에 대한 정규식 입력으로 사용할 수 있습니다. 문자열에서 모든 정수 배열을 생성하려면, 루프를 사용하고 반환되는 오프셋 값이 -1이 될 때까지 [정규식 일치]를 반복해서 호출합니다.

다음 블록다이어그램은 문자열에서 정수의 모든 어커런스를 스캔하는 한가지 방법입니다. 빈 배열을 생성한 후 루프의 각 반복마다 남아 있는 문자열에서 숫자 패턴을 검색합니다. 패턴이 발견되면(오프셋이 -1이 아님), 이 블록다이어그램은 [배열 만들기]를 사용하여 결과 숫자 배열에 숫자를 추가합니다. 문자열에 값이 남아 있지 않으면, [정규식 일치]는 -1을 반환하고 블록다이어그램은 실행을 완료합니다.



이 블록 다이어그램에서 한가지 문제점은 루프에서 [배열 만들기]를 사용하여 새 값을 이전 값에 연결한다는 것입니다. 대신에, 오토인덱싱을 사용하여 값을 루프의 경계선에 축적할 수 있습니다. 배열에서 [정규식 일치]가 일치되는 것을 찾지 못한 루프의 마지막 반복으로부터 온 여분의 원하지 않는 값을 보게 됩니다. 해결책은 [배열 잘라내기]를 사용하여 여분의 원하지 않는 값을 제거하는 것입니다. 이것이 다음 블록 다이어그램에 나와 있습니다.



이 블록 다이어그램의 다른 문제는 루프를 통해 매번 남아있는 문자열의 불필요한 복사본을 생성한다는 것입니다. 정규식 일치 ha

VI 구성요소 메모리 관리

VI는 다음의 4가지 주요한 성분을 가집니다:

- 프론트패널
- 블록다이어그램
- 코드(기계 코드로 컴파일되는 다이어그램)
- 데이터(컨트롤과 인디케이터 값, 기본 데이터, 다이어그램 상수 데이터 등)

VI를 로드할 때, 그 VI의 프런트패널, 코드(플랫폼에 일치한다면), 데이터가 메모리에 로드됩니다. VI가 플랫폼에 있어서 변경이나 SubVI의 인터페이스에서 변경 때문에 리컴파일할 필요가 있다면, 블록다이어그램 또한 메모리에 로드됩니다.

또한 VI는 메모리에 그 SubVI의 코드와 데이터 공간도 로드합니다. 특정 상황에서, 일부 SubVI의 프런트패널도 메모리에 로드될 수 있습니다. 예를 들면, 프로퍼티 노드는 프런트패널 컨트롤의 상태 정보를 다루기 때문에, 이는 SubVI가 프로퍼티 노드를 사용하는 경우 발생할 수 있습니다.

VI 일반적으로 VI의 한 섹션을 SubVI로 변경할 때, 그다지 많은 메모리를 사용하지 않는다는 것입니다. SubVI가 없는 거대한 단일 VI를 생성할 때, 메모리에 그 상위 레벨 VI를 위한 프런트패널, 코드, 데이터를 가지게 됩니다. 그러나, VI를 여러 SubVI로 나누면, 상위 레벨 VI를 위한 코드는 더 작게 되고 SubVI의 코드와 데이터는 메모리에 있게 됩니다. 일부 경우에, 더 낮은 런타임 **메모리 사용**을 볼 수 있습니다.

큰 VI는 편집에 걸리는 시간도 더 깁니다. 다음의 방법으로 이 문제를 피할 수 있습니다:

- (권장) VI를 여러 SubVI로 나눕니다. 편집기는 작은 VI를 더 효율적으로 다룰 수 있을 뿐 아니라, 일반적으로 계층구조에 계층이 많을 수록 VI 조직이 유지하고 읽기 쉽습니다.
- 큰 VI의 **컴파일된 코드 복잡도**에 근거하여 LabVIEW가 **실행 시간보다 편집기 응답을 우선**하도록 설정합니다.



노트 주어진 VI의 프런트패널이나 블록다이어그램이 화면보다 훨씬 더 크면, 더 쉽게 접근할 수 있도록 VI를 여러 SubVI로 나눌 수 있습니다.

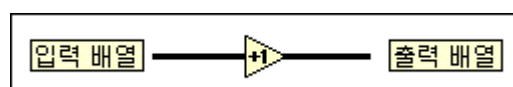
데이터 흐름 프로그래밍과 데이터 버퍼

데이터 흐름 프로그래밍에서, 일반적으로 변수를 사용하지 않습니다. 데이터 흐름 모델은 일반적으로 노드를 데이터 입력을 소비하고 데이터 출력을 생성하는 것으로 설명합니다. 이 모델을 그대로 구현하면 많은 양의 메모리를 사용하고, 성능이 느린 어플리케이션이 생성됩니다. 모든 함수는 출력이 전달되는 모든 대상에 데이터의 복사본을 생성합니다. LabVIEW는 inplaceness를 통해 이 부분의 실행을 개선시킬 수 있습니다.

LabVIEW는 inplaceness를 사용하여 언제 메모리를 다시 사용할지, 또한 각 출력 터미널의 복사본을 만들지를 결정합니다. LabVIEW가 데이터를 입력에서 출력으로 복사하지 않을 때, 해당 입력과 출력의 데이터는 in-place라고 할 수 있습니다.

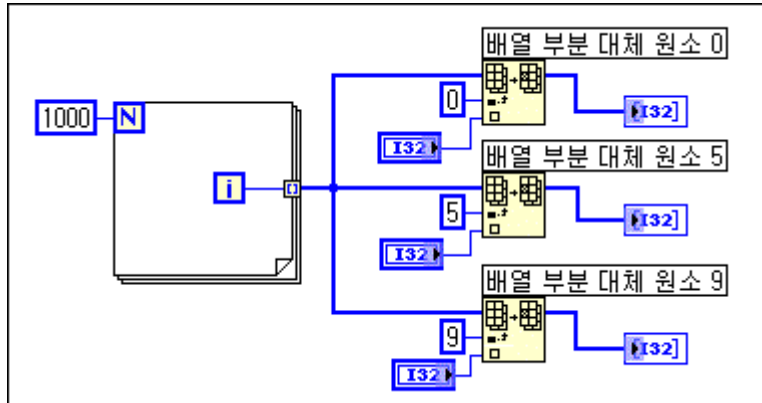
버퍼 할당 보이기 윈도우를 사용하여 LabVIEW가 어디에 데이터의 복사본을 생성할 수 있는지 알려줍니다.

예를 들면, 컴파일러에 대한 더 전통적인 접근 방식으로, 다음 블록다이어그램은 두 개의 데이터 메모리 블록을 사용하는데, 하나는 입력으로 또 다른 하나는 출력으로 사용합니다.



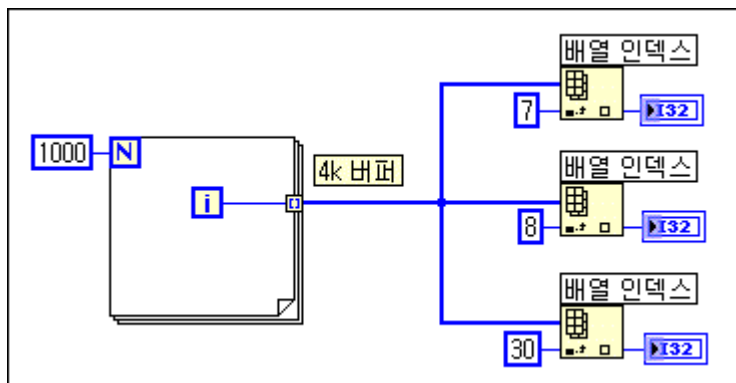
입력 배열과 출력 배열은 같은 개수의 원소를 가지고 있고, 이 두 배열의 데이터 타입은 같습니다. 입력 배열을 데이터의 버퍼로서 생각합니다. 출력에 대해 새로운 버퍼를 생성하는 대신에, 컴파일러는 입력 버퍼의 메모리를 출력 버퍼에 다시 사용합니다. 이와 같이 inplaceness를 이용하면 실행 시 메모리를 할당할 필요가 없기 때문에 메모리가 절약되고 더 빠른 실행이 가능합니다.

그러나 컴파일러는 다음 블록다이어그램에서 보는 것처럼 모든 경우에 메모리 버퍼를 다시 사용할 수는 없습니다.



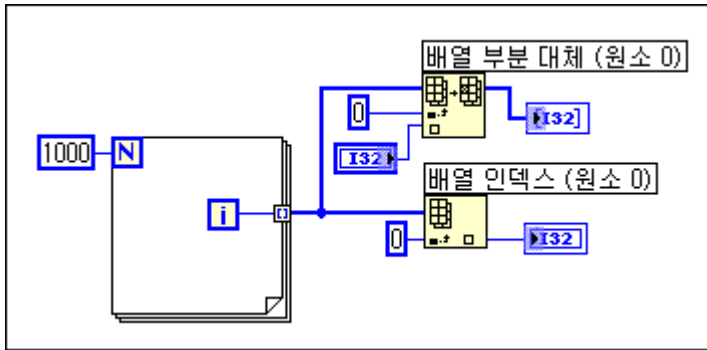
신호는 단일 데이터 소스를 여러 대상으로 전달합니다. [배열 부분 대체] 함수는 입력 배열을 변경하여 출력 배열을 생성합니다. 이 경우, 이 함수 중 한 개의 데이터는 in-place이며, 이때 출력은 다른 출력과는 달리 입력 배열의 데이터를 다시 사용합니다. 이렇게 하여, 컴파일러는 두 개의 함수에 대해 새로운 데이터 버퍼를 생성하고 버퍼에 배열 데이터를 복사합니다. 이 블록다이어그램은 약 12 KB(원래 배열에 4KB 그리고 2개의 여분의 데이터 버퍼 각각에 4KB)를 사용합니다.

이제 다음의 블록다이어그램을 살펴봅니다.



이전처럼, 입력은 세 개의 함수로 갈라집니다. 그러나, 이 경우에 [배열 인덱스] 함수는 입력 배열을 변경하지 않습니다. 데이터를 여러 위치로 전달하고 그 위치에서 데이터를 변경하지 않고 읽으면, LabVIEW는 데이터의 복사본을 만들지 않습니다. 이 결과, 모든 데이터가 in-place입니다. 이 블록다이어그램은 약 4 KB의 메모리를 사용합니다.

마지막으로, 다음의 블록다이어그램을 살펴봅니다.



이 경우에, 입력은 두 개의 함수로 나뉘는데, 그 중 하나는 데이터를 변경합니다. 그 두 개의 함수 사이에 의존성은 없습니다. 그러므로, 적어도 하나의 복사본은 만들어 둘 필요가 있다는 것을 예상할 수 있으므로 [배열 부분 대체] 함수가 안전하게 데이터를 변경할 수 있습니다. 그러나 이 경우에, 컴파일러는 데이터를 읽는 함수가 먼저 실행되고 데이터를 변경하는 함수가 마지막으로 실행되는 방식으로 함수의 실행 일정을 잡습니다. 이러한 방식으로 [배열 부분 대체] 함수는 배열 복사를 생성하지 않고, 입력 배열 버퍼를 다시 사용하여 이 함수의 데이터를 in-place로 만듭니다. 노드의 순서는 중요하기 때문에, 다른 노드의 입력을 위해 한 노드의 순서나 출력을 사용하여 순서를 명시해 놓습니다.

실제로, 컴파일러에 의한 블록다이어그램의 분석은 완전하지 않습니다. 몇가지 경우에, 컴파일러는 블록다이어그램 메모리를 다시 사용하기 위한 최적의 방법을 결정할 수 없을 수 있습니다.

조건적 인디케이터와 데이터 버퍼

사용자가 블록다이어그램을 만드는 방법은 LabVIEW가 데이터 버퍼를 다시 사용하는 것을 막을 수 있습니다. SubVI에서 조건적 인디케이터를 사용하면 LabVIEW가 데이터 버퍼 사용을 최적화하는 것을 막습니다. 조건적 인디케이터는 케이스 구조나 For 루프 안의 인디케이터입니다. 조건적으로 실행되는 코드 경로에 인디케이터를 놓으면 시스템을 통한 데이터의 흐름을 깨고 LabVIEW는 입력으로부터 데이터 버퍼를 다시 사용할 수 없을 것입니다. 그러나 그 대신 데이터를 인디케이터에 강제로 복사해 놓도록 합니다. 케이스 구조와 For 루프의 밖에 인디케이터를 놓을 때, LabVIEW는 루프나 구조 안에서 데이터를 직접 변경하고, 데이터 복사본을 생성하는 대신에 인디케이터에 데이터를 전달합니다. 케이스 구조 안에 인디케이터를 놓는 대신에 다른 경우에 대해 상수를 생성할 수 있습니다.

메모리 사용 모니터링

여러가지 방법을 통해 메모리 사용 정보를 확인할 수 있습니다.

현재 VI에서 메모리 사용을 보려면, 파일>>VI 프로퍼티를 선택하고 최상위 풀다운 메뉴에서 메모리 사용을 선택합니다. 이때, SubVI의 메모리 사용에 대한 정보는 포함하지 않습니다. 성능과 메모리 프로파일 윈도우를 사용하여 메모리에 있는 모든 VI가 사용하는 메모리를 모니터링할 수 있고, 성능에 문제가 있는 SubVI를 파악할 수 있습니다. VI 성능과 메모리 프로파일 윈도우는 매번 실행에서 각 VI가 사용하는 바이트와 블록의 최소, 최대, 평균의 **통계값을 나타냅니다.**

	노트 정확한 VI 런타임 성능을 모니터하려면 컴파일된 코드를 분리하지 않고 모든 VI를 LabVIEW의 현재 버전에 저장해야 하며 그 이유는 다음과 같습니다: - 실행 취소 기능은 객체 및 데이터의 임시적인 복사본을 생성하며, 이는 VI의 메모리 요구를 증가시킬 수 있습니다. VI를 저장하면 실행 취소에 의해 생성된 복사본이 제거되고 그 결과 정확한 메모리 정보를 알게 됩니다. - 이전 버전의 LabVIEW에서 만든 VI를 저장하지 않고 실행하면, VI의 메모리 요구가 증가되고 VI 런타임 성능은 감소합니다.

버퍼 할당 보이기 윈도우를 사용하여 블록다이어그램에서 LabVIEW가 버퍼의 형태로 메모리 공간을 할당하는 영역을 파악할 수 있습니다.

	노트 버퍼 할당 보이기 윈도우는 LabVIEW Full과 Professional Development Systems에서만 제공됩니다.

도구 >> 프로파일 >> 버퍼 할당 보이기를 선택하여 버퍼 할당 보이기 윈도우를 디스플레이합니다. 버퍼를 확인할 데이터 타입의 옆에 확인 표시를 하고 새로 고침 버튼을 클릭하십시오. 블록다이어그램에 나타나는 검은 사각형은 LabVIEW가 데이터 공간 할당을 위해 버퍼를 생성하는 위치를 나타냅니다.

LabVIEW가 각 버퍼에 할당하는 메모리의 크기는 가장 큰 **데이터 타입**의 크기와 같습니다. VI를 실행할 때, LabVIEW는 생성된 버퍼를 사용하여 데이터를 저장할 수도 있으나, 항상 그렇지는 않습니다. LabVIEW가 데이터의 복사본을 만들지는 런타임 때 결정지어지고, 때로는 VI가 다이내믹 데이터에 의존하기 때문에 복사본을 만들지 미리 알 수는 없습니다.

VI가 버퍼 할당에 메모리를 필요로 하면, LabVIEW는 해당 버퍼의 데이터 복사본을 만듭니다. 버퍼가 데이터의 복사본을 필요로 하는지 확실치 않을 때에도, LabVIEW는 버퍼의 복사본을 만들기도 합니다.

	노트 버퍼 할당 보이기 윈도우는 블록다이어그램 상에서 검정색 사각형을 사용하여 실행 버퍼만을 식별합니다. 이 윈도우를 사용하여 프런트패널에서 동작 버퍼를 식별할 수는 없습니다.

LabVIEW의 버퍼 생성 위치를 알면, LabVIEW가 **보다 적은 양의 메모리로 VI를 실행할 수 있도록 VI를 편집**할 수 있습니다. LabVIEW는 VI를 실행하기 위해 메모리를 할당해야 하므로 모든 버퍼를 지울 수는 없습니다.

LabVIEW가 VI를 다시 컴파일할 만큼 VI를 변경하게 되면, 버퍼 정보가 더 이상 정확하지 않으므로 검정색 사각형은 사라집니다. VI를 다시 컴파일하고 검정색 사각형을 디스플레이하기 위해서 버퍼 할당 보이기 윈도우의 새로 고침 버튼을 클릭하십시오. 버퍼 할당 보이기 윈도우를 닫게 되면 검정색 사각형은 사라집니다.

	노트 또한 LabVIEW Desktop Execution Trace Toolkit을 이용하여 스레드 사용과 메모리 손실, 기타 LabVIEW 프로그래밍의 영역을 모니터할 수 있습니다.

도움말 >> LabVIEW에 대하여를 선택하여 어플리케이션에 의해 사용되는 총 메모리의 양을 요약하는 통계값을 보여줍니다. 이것은 VI를 위한 메모리와 어플리케이션이 사용한

는 메모리를 포함합니다. VI 세트의 실행 전과 후에 이 메모리 양을 검사하여 VI가 얼마나 많은 메모리를 사용하게 되는가를 대략적으로 알 수 있습니다.

더 효과적인 메모리 사용을 위한 규칙

이전 섹션의 주요 포인트는 컴파일러가 메모리를 지능적으로 다시 사용하려는 시도에 대한 것입니다. 컴파일러가 언제 메모리를 다시 사용할 수 있고 사용할 수 없는지에 대한 규칙은 매우 복잡합니다. 실제로, 다음 규칙은 메모리를 효과적으로 사용하는 VI를 생성하는데 도움을 줄 수 있습니다:

- VI를 SubVI로 나누어도 일반적으로 메모리 사용에 나쁜 영향을 주지는 않습니다. 많은 경우에, SubVI가 실행되지 않을 때 실행 시스템이 SubVI 데이터 메모리를 다시 요청할 수 있기 때문에 메모리 사용은 향상됩니다.
- 스칼라 값의 복사본에 대해 너무 염려하지 마십시오. 메모리 사용에 부정적인 영향을 미칠 정도가 되려면 스칼라가 엄청나게 많아야 합니다.
- 배열이나 문자열을 가지고 작업할 때 글로벌과 로컬 변수를 지나치게 사용하지 않습니다. 글로벌이나 로컬 변수를 읽으면 LabVIEW는 데이터를 복사하는 일이 발생합니다.
- 필요하지 않다면 열려 있는 프런트패널에 큰 배열과 문자열을 디스플레이하지 않습니다. 열려 있는 프런트패널의 컨트롤과 인디케이터는 디스플레이하는 데이터를 복사해서 가지고 있습니다.



팁 차트 인디케이터가 필요한 경우, 차트 히스토리가 디스플레이되는 데이터의 복사본을 보유하게 된다는 점을 염두에 두십시오. 차트 히스토리가 채워질 때까지 지속되는데, 이후 LabVIEW는 더 이상의 메모리를 차지하지 않게 됩니다. LabVIEW는 VI가 다시 시작될 때 자동으로 차트 히스토리를 지우지 않고, 프로그램이 실행되는 과정에서 차트 히스토리가 지워질 수 있도록 합니다. 이를 위해, 차트에 대한 히스토리 데이터 속성 노드에 빈 배열을 작성하십시오.

- 패널 업데이트 연기 프로퍼티를 사용합니다. 이 프로퍼티를 참으로 설정할 때, 컨트롤 값을 변경할지라도 프런트패널 인디케이터 값은 바뀌지 않습니다. 인디케이터를 새 값으로 채우기 위해 OS는 메모리를 사용할 필요는 없습니다.



노트 일반적으로 LabVIEW는 SubVI를 호출할 때 SubVI의 프런트패널을 열지 않습니다.

- SubVI의 프런트패널이 디스플레이되지 않는다면, SubVI에 사용되지 않은 프로퍼티 노드를 남겨놓지 마십시오. 프로퍼티 노드는 SubVI의 프런트패널이 메모리에 남아있도록 하여 불필요한 메모리 사용을 만들 수 있습니다.
- 블록 다이어그램을 만들 때, 입력의 크기가 출력의 크기와 다른 위치를 확인합니다. 예를 들면, [배열 만들기] 또는 [문자열 연결] 함수를 사용하여 자주 배열이나 문자열의 크기를 증가시키는 위치에서는 데이터의 복사본을 생성하는 것입니다.
- 데이터를 SubVI와 함수로 전달할 때 배열에 대해 **일관된 데이터 타입**을 사용하고 강제 변환점을 찾습니다. 데이터 타입을 변경할 때, 실행 시스템은 데이터의 복사본을 만듭니다.

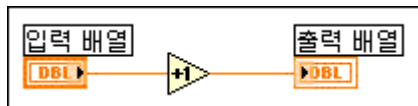
- 큰 배열이나 문자열을 가진 클러스터나 클러스터의 배열과 같은 복잡하고 계층구조적인 데이터 타입을 사용하지 마십시오. 더 많은 메모리를 사용하게 될 것입니다. 가능하면, **더 효과적인 데이터 타입을 사용**합니다.
- 필요하지 않는 한 투명하고 겹치는 프런트패널 객체를 사용하지 않습니다. 그런 객체는 더 많은 메모리를 사용할 수 있습니다.

VI의 성능을 최적화하는 방법에 대한 구체적인 방법은 [LabVIEW Style Checklist](#)를 참조하십시오.

프런트패널에서 메모리 문제

프런트패널이 열려 있을 때, 컨트롤과 인디케이터는 디스플레이하는 데이터의 컨트롤과 인디케이터 자체의 복사본을 가지고 있습니다.

다음 블록다이어그램은 프런트패널 컨트롤과 인디케이터가 추가된 증가 함수를 보여줍니다.



VI를 실행할 때, 프런트패널 컨트롤의 데이터가 블록다이어그램에 전달됩니다. 증가 함수가 입력 버퍼를 다시 사용합니다. 그리고 나서 인디케이터는 디스플레이할 목적으로 데이터의 복사본을 만듭니다. 이렇게 해서 버퍼의 3개의 복사본이 생깁니다.

프런트패널 컨트롤의 데이터 보호는 이어지는 노드에 데이터가 전달될 때 데이터를 컨트롤에 입력하는 것, 관련된 VI를 실행하는 것, 컨트롤에서 데이터 변경을 보는 것을 막습니다. 마찬가지로, 인디케이터의 경우에도 데이터는 보호되므로 인디케이터가 새로운 데이터를 받을 때까지 이전 내용을 디스플레이할 수 있습니다.

SubVI와 함께, 입력과 출력으로서 컨트롤과 인디케이터를 사용할 수 있습니다. 다음 조건에 따라 실행 시스템은 SubVI의 컨트롤과 인디케이터 데이터의 복사본을 만듭니다:

- 프런트패널이 메모리에 있습니다. 이것은 다음의 이유 때문에 발생할 수 있습니다:
 - 프런트패널이 열려 있습니다.
 - VI가 변경되었지만 저장되지 않았습니다(VI의 모든 구성요소가 VI가 저장될 때까지 메모리에 있습니다).
- 패널이 데이터 인쇄를 사용합니다.
- 블록다이어그램이 프로퍼티 노드를 사용합니다.
- VI가 로컬 변수를 사용합니다.
- 패널이 데이터 로깅을 사용합니다.

프로퍼티 노드가 닫힌 패널을 가진 SubVI에서 차트 히스토리를 읽을 수 있도록 하기 위해, 컨트롤이나 인디케이터는 전달되는 데이터를 디스플레이해야 합니다. 이와 같이 다른 많은 속성들이 있기 때문에, SubVI가 프로퍼티 노드를 사용한다면 실행 시스템은 SubVI 패널을 메모리에 두어야 합니다.

프론트패널이 프론트패널 데이터 로깅이나 데이터 데이터 인쇄를 사용한다면, 컨트롤과 인디케이터가 그 데이터의 복사본을 유지합니다. 또한, 프론트패널은 데이터 인쇄를 위해 메모리에 남아있기 때문에 패널을 인쇄할 수 있습니다.

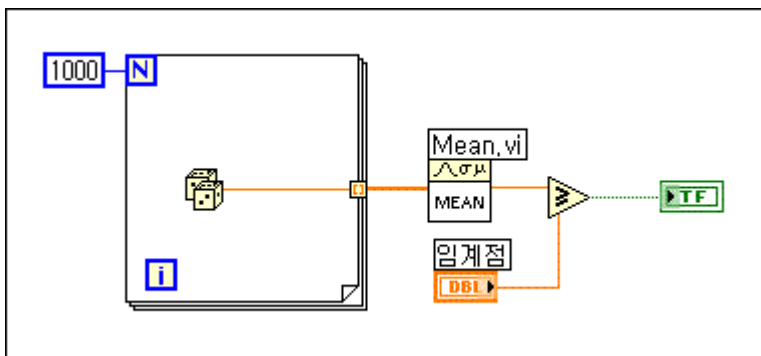
VI 프로퍼티 대화 상자나 SubVI 노드 설정 대화 상자를 사용하여 SubVI를 호출할 때 그 프론트패널을 디스플레이하도록 설정하면, 그 SubVI를 호출할 때 프론트패널이 메모리에 로드됩니다. 원래 닫혀있으면 실행 후 닫기 아이템을 설정한다면, LabVIEW는 SubVI가 실행을 끝낼 때 메모리에서 그 프론트패널을 제거합니다.

데이터 메모리를 다시 사용할 수 있는 SubVI

일반적으로, SubVI의 블록다이어그램을 상위 레벨에서 복사하는 것만큼 쉽게 SubVI는 그 호출자로부터 데이터 버퍼를 사용할 수 있습니다. 대부분의 경우에, 블록다이어그램의 한 섹션을 SubVI로 변환한다고 더 많은 메모리를 사용하지는 않습니다. 특별한 디스플레이가 요구되는 VI에 대해, 이전 섹션에서 설명된 것처럼, 프론트패널과 컨트롤을 위한 일부 추가된 메모리 사용이 있을 수 있습니다.

언제 메모리의 할당이 해제되는지 이해하기

다음의 블록다이어그램을 살펴봅시다.



평균 VI를 실행한 후에 데이터의 배열은 더 이상 필요 없습니다. 데이터가 더 이상 필요 없는 때를 결정하는 것은 더 큰 블록다이어그램에서 매우 복잡할 수 있기 때문에, 실행되는 동안 특정한 VI의 데이터 버퍼 할당을 해제하지 않습니다.

macOS에서 실행 시스템이 작은 메모리를 차지하면, 현재 실행되지 않는 모든 VI에 의해 사용되는 데이터 버퍼의 할당을 해제합니다. 실행 시스템은 프론트패널 컨트롤, 인디케이터, 글로벌 변수, 초기화되지 않은 시프트 레지스터에 의해 사용된 메모리의 할당을 해제하지 않습니다.

이제 같은 VI를 더 큰 VI의 SubVI로 생각합니다. 데이터의 배열은 SubVI에서만 생성되고 사용됩니다. macOS에서, SubVI를 실행하지 않고 시스템이 낮은 메모리를 차지하면, SubVI에서 데이터의 할당을 해제할 수 있습니다. 이러한 경우에 SubVI를 사용하여 메모리 사용을 절약할 수 있습니다.

Windows와 Linux 플랫폼에서, VI가 닫히고 메모리로부터 제거되지 않으면 데이터 버퍼는 정상적으로 할당을 해제하지 않습니다. 메모리는 필요할 때마다 OS로부터 할당받고 가상 메모리가 이 플랫폼에서 잘 작동합니다. 조각난 메모리 때문에, 어플리케이션은 실제 사용하는 것보다 더 많은 메모리를 사용하는 것으로 나타날 수 있습니다. 메모리를 할

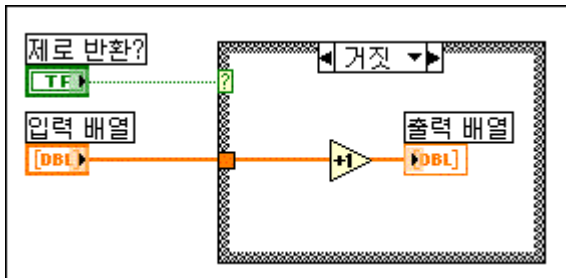
당받고 비워가면서, 어플리케이션은 메모리 사용을 통합하여 사용되지 않은 블록을 OS에 반환할 수 있습니다.

[할당 해제 요청] 함수를 사용하여 이 함수를 포함하는 VI가 실행된 후에 사용하지 않는 메모리 할당을 해제할 수 있습니다. 이 함수는 고급 성능 최적화에만 사용됩니다. 일부 경우, 사용되지 않은 메모리의 할당을 해제하면 성능이 향상됩니다. 그러나 지나치게 메모리의 할당을 해제하면 LabVIEW가 해당 메모리를 재사용하기보다는 반복해서 메모리 공간을 재할당하게 됩니다. 사용자 VI가 많은 양의 데이터를 할당하지만 이러한 할당분을 재사용하지 않을 때 이 함수를 사용하십시오. 최상위 VI가 SubVI를 호출할 때, LabVIEW는 해당 SubVI가 실행되는 메모리의 데이터 공간을 할당합니다. SubVI가 실행을 완료해도, LabVIEW는 최상위 VI가 실행을 마치거나 전체 어플리케이션이 종료될 때까지 데이터 공간 할당을 해제하지 않기 때문에 메모리 부족 상태가 되거나 성능이 저하될 수 있습니다. [할당 해제 요청] 함수를 메모리 할당 해제하려는 SubVI에 위치시키십시오. 플래그 볼리언 입력을 참으로 설정하면, LabVIEW는 subVI의 데이터 공간 할당을 해제하여 메모리 사용을 감소시킵니다.

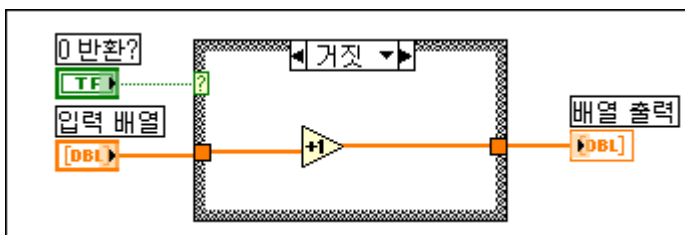
출력이 입력 버퍼를 언제 다시 사용할 수 있는지 결정하기

출력이 입력과 같은 크기와 데이터 타입을 가지고 다른 곳에서 입력이 요청이 없으면, 출력은 입력 버퍼를 다시 사용할 수 있습니다. 이전에 언급했던 것처럼, 몇몇 경우에는 입력이 다른 곳에서 사용될 때에도 컴파일러와 실행 시스템이 출력 버퍼를 위해 입력을 다시 사용할 수 있는 방식으로 코드 실행을 명령할 수 있습니다. 그러나, 이러한 규칙은 복잡합니다. 그것에 의존하지 마십시오.

버퍼 할당 보이기 윈도우를 사용하여 출력 버퍼가 입력 버퍼를 다시 사용할 것인지 알 수 있습니다. 다음의 블록 다이어그램에서, 케이스 구조의 각 경우에 인디케이터를 놓으면 데이터 흐름을 막게 되는데 그 이유는 LabVIEW가 각 인디케이터에서 데이터의 복사본을 생성하기 때문입니다. LabVIEW는 입력 배열을 위해 생성한 버퍼를 사용하지 않지만 대신에 출력 배열을 위한 데이터의 복사본을 생성합니다.



인디케이터를 케이스 구조 밖으로 이동시키면, 출력 버퍼는 입력 버퍼를 다시 사용하는데, 그 이유는 LabVIEW가 인디케이터가 디스플레이하는 데이터의 복사본을 만들 필요가 없기 때문입니다. LabVIEW는 VI에서 나중에 입력 배열의 값을 요청하지 않기 때문에, 증가 함수는 직접적으로 입력 배열을 변경하고 출력 배열에 그 값을 전달할 수 있습니다. 이 때, LabVIEW는 데이터를 복사할 필요가 없으므로 다음 블록 다이어그램에서 보는 것처럼 버퍼가 출력 배열에 나타나지 않습니다.



데이터 타입 일치로 메모리 사용 최적화하기

데이터 타입을 일관성있게 사용하는 경우, LabVIEW는 출력을 생성할 때 입력에 할당했던 메모리 버퍼를 다시 사용할 수 있습니다. 이 경우 VI의 메모리 사용 및 실행 시간이 향상됩니다. 그러나 입력과 출력의 데이터 타입이 다른 경우, LabVIEW에서 해당 입력에 할당된 메모리를 출력에서 다시 사용할 수 없게 됩니다.

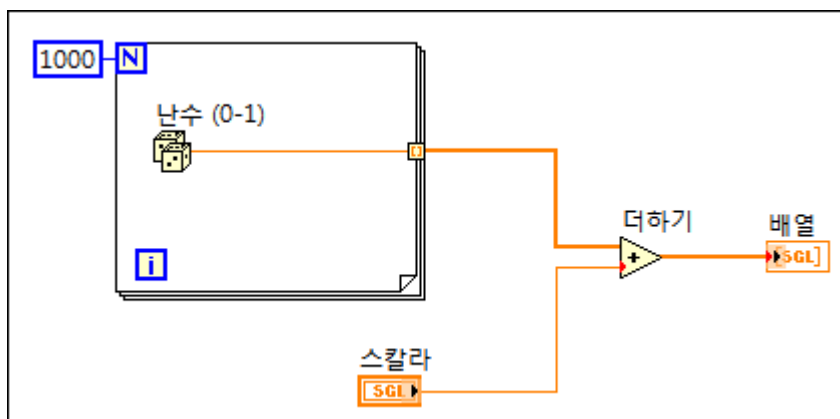
예를 들어 32 비트 정수를 16 비트 정수에 더하는 경우, LabVIEW는 16 비트 정수를 32 비트 정수로 강제 변환시킵니다. 컴파일러는 강제 변환된 데이터에 대해 새로운 버퍼를 생성하고, LabVIEW는 [더하기] 함수에 **강제 변환점**을 놓습니다. 입력이 다른 모든 요구사항을 충족한다고 가정하면, 이 예에서 LabVIEW는 32비트 정수 입력을 출력 버퍼로 사용할 수 있습니다. 그러나 LabVIEW가 16 비트 정수를 강제 변환하였기 때문에, LabVIEW는 해당 입력에 할당된 메모리를 다시 사용할 수 없습니다.

메모리 사용을 최소화하기 위해, 가능한 모든 곳에서 일관된 데이터 타입을 사용합니다. 데이터 타입을 일관성있게 사용하면, 더 큰 크기의 데이터로 강제 변환해야 하는 경우를 피하게 되어 데이터의 복제본을 더 적게 생성하게 됩니다. 또한 일부 어플리케이션에서는 더 작은 데이터 타입의 사용으로 데이터 사용을 최소화시킬 수도 있습니다. 예를 들면 8 바이트 배정도 숫자 대신에 4 바이트 단정도 숫자를 사용하는 것을 고려할 수 있습니다. 그러나 SubVI에 연결하려는 데이터 타입과 SubVI가 받기를 예상하는 데이터 타입을 일치시키면 불필요한 강제 변환을 피할 수 있습니다.

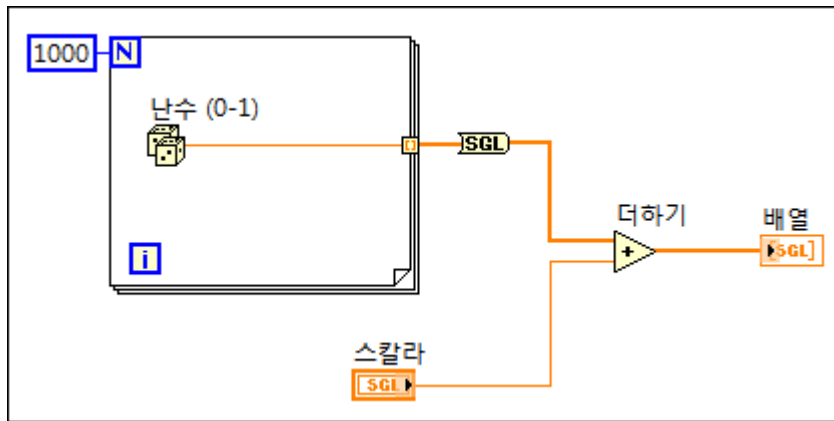
특정 타입의 데이터 생성시 메모리 사용 최적화하기

특정 데이터 타입의 데이터를 생성할 때, 블록다이어그램에서 데이터 타입을 변환해야 합니다. 그러나 변환 함수를 사용하면 LabVIEW가 큰 배열을 생성하기 전에 데이터 타입을 변환하여 메모리 사용을 최적화할 수 있습니다.

다음의 예에서 다른 VI 입력의 데이터 타입과 일치시키려면 출력 데이터가 단정도 부동소수가 되어야 합니다. LabVIEW는 1000개의 임의의 값의 배열을 생성하고 그 배열을 스칼라 값에 더합니다. LabVIEW는 단정도 부동소수 스칼라 데이터 타입을 임의의 배정도 부동소수와 일치시키기 위해 강제 변환시키고, [더하기] 함수에 강제 변환점을 놓습니다.

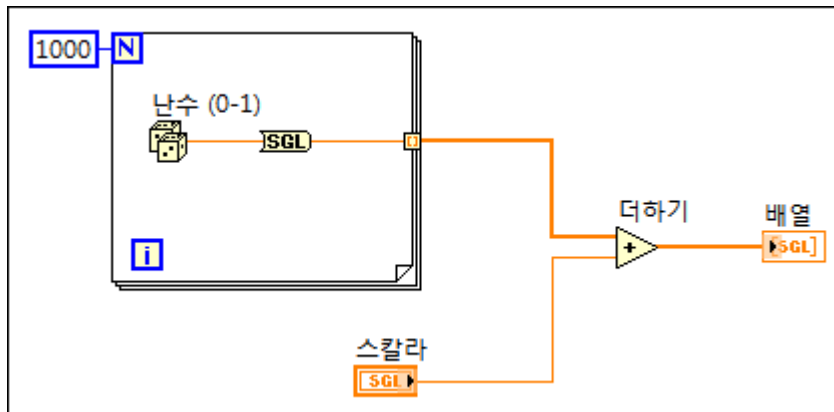


다음의 블록다이어그램에서는 [단정도 부동소수로] 함수를 사용하여 배정도 부동소수를 변환하여 이 문제를 해결하려 합니다.



이 함수는 LabVIEW가 배열을 생성한 이후 큰 배열의 데이터 타입을 변환하기 때문에, VI가 사용하는 메모리의 양은 강제 변환점이 있는 경우와 같은 양입니다.

다음 블록 다이어그램은 LabVIEW가 배열을 생성하기 전에 임의의 수를 변환하여 메모리 사용 및 실행 속도를 최적화하는 방법을 보여줍니다.



변환이 불가피한 경우 LabVIEW에서 큰 배열이 생성되기 전에 [변환] 함수를 사용하여, 배열에 할당된 큰 데이터 버퍼의 데이터 타입이 변환되는 것을 피합니다. LabVIEW에서 큰 배열이 생성되기 전에 데이터를 변환하면 VI의 메모리 사용을 최적화할 수 있습니다.

계속해서 데이터 크기가 바뀌는 것을 피하기

출력 크기가 입력 크기와 다르면, 출력은 입력 데이터 버퍼를 다시 사용하지 않습니다. 이것이 배열이나 문자열의 크기를 증가시키거나 감소시키는 [배열 만들기], [문자열 연결], [배열 잘라내기]와 같은 함수의 경우입니다. 배열과 문자열을 가지고 작동할 때, 이 함수를 계속 사용하는 것은 피해야 합니다. 왜냐하면 이들은 데이터를 계속해서 복사하기 때문에 사용자 프로그램이 더 많은 데이터 메모리를 사용하고 더 느리게 실행되기 때문입니다.

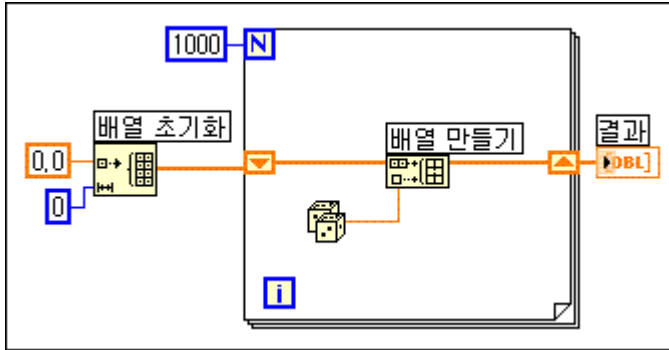
예제 1: 배열 만들기

데이터의 배열을 생성하는데 사용되는 다음 블록 다이어그램을 살펴봅니다. 이 블록 다이어그램은 새 원소를 연결하기 위해 계속해서 [배열 만들기]를 호출하여 루프에 배열을 생성합니다. 입력 배열은 [배열 만들기]에 의해 다시 사용됩니다. VI는 각 반복마다 계속

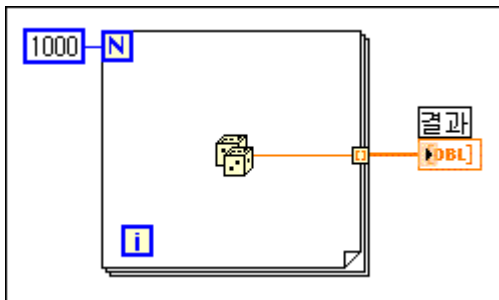
해서 버퍼의 크기를 다시 조정하여 새로운 배열을 위한 공간을 확보하고 새 원소를 추가합니다. 결과적으로 실행 속도는 느려지는데, 특히 루프가 여러 번 반복해서 실행될 때 느려집니다.



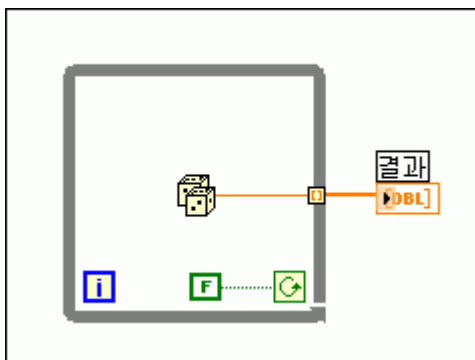
노트 배열, 클러스터, 웨이브폼, 배리언트를 조작할 때 In Place 원소 구조를 사용하여 VI에서의 메모리 사용을 향상시킬 수 있습니다.



루프의 매 반복마다 배열에 값을 추가하려면, 루프 경계선의 오토인덱싱을 사용하는 것이 가장 좋은 성능을 냅니다. For 루프를 가지고 N 은 배열의 크기를 미리 결정하고(N 에 연결된 값을 기본으로) 버퍼의 크기를 한번만 바꿉니다.



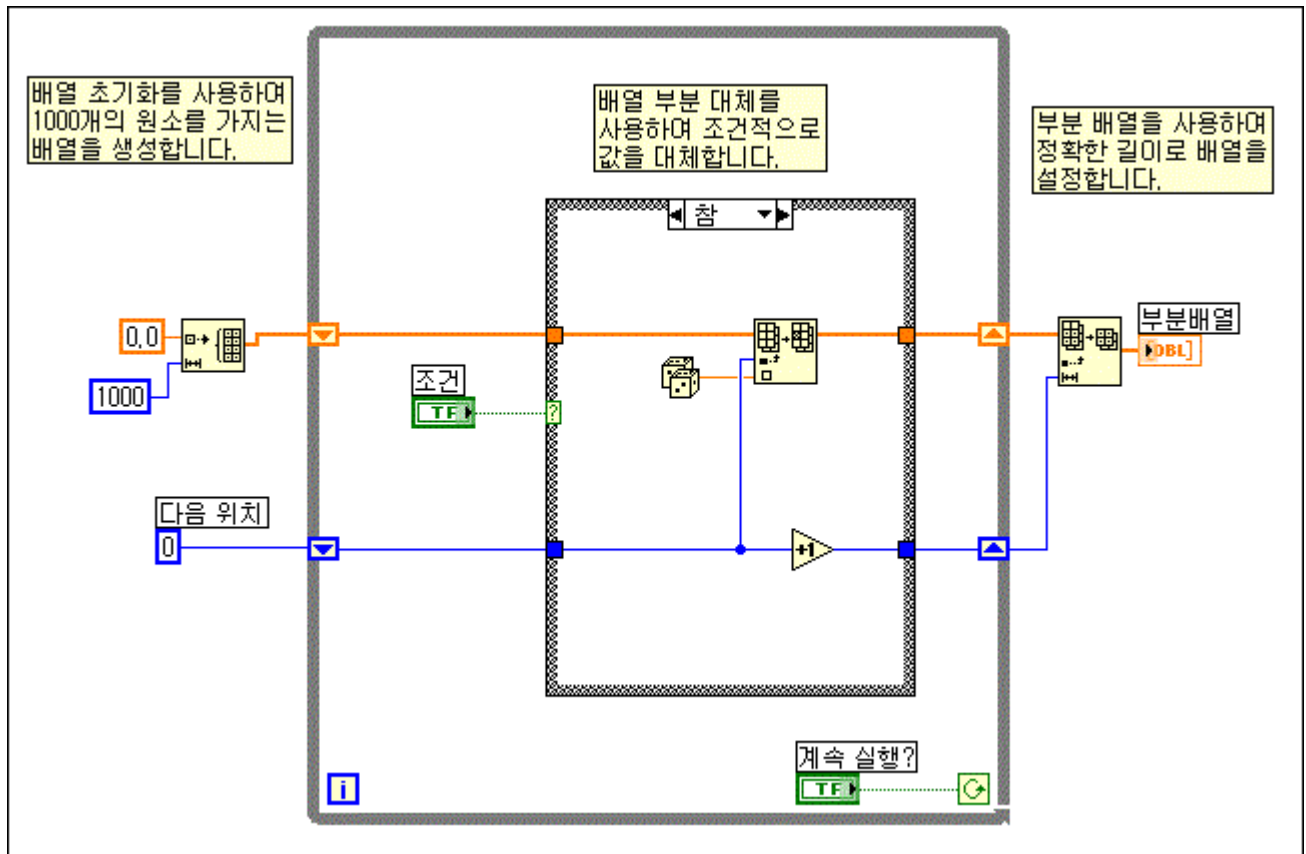
While 루프에서 오토인덱싱은 그리 효과적이지가 못하는데, 그 이유는 배열의 마지막 크기를 알 수 없기 때문입니다. 그러나 While 루프에서 오토인덱싱은 출력 배열 크기를 상당히 많이 증가시켜 출력 배열의 크기 조절을 피하도록 해줍니다. 루프가 끝날 때, 출력 배열의 크기는 적당한 크기로 조절됩니다. While 루프 오토인덱싱의 성능은 For 루프 오토인덱싱과 거의 일치합니다.



오토인덱싱은 루프의 매 반복마다 그 배열 결과에 값을 추가할 것이라고 가정합니다. 조건적으로 값을 배열에 추가해야 하지만 배열 크기에서 상위 리미트를 결정할 수 있는 경우, 배열을 미리 할당하고 배열을 채우기 위해 [배열 부분 대체]를 사용하는 것을 고려해 볼 수 있습니다.

배열 값을 채우기를 끝마칠 때, 배열의 크기를 올바른 크기로 조절할 수 있습니다. 배열은 한번만 생성되고 [배열 부분 대체]는 출력 버퍼를 위해 입력 버퍼를 다시 사용할 수 있습니다. 이것의 성능은 오토인덱싱을 사용하는 루프의 성능과 매우 비슷합니다. 이 기술을 사용할 때, [배열 부분 대체]는 배열의 크기를 조절하지 않기 때문에 값을 대체하는 배열이 그 결과 데이터를 가지고 있을 만큼 충분히 커야 합니다.

이 과정의 예제가 다음 블록다이어그램에 나타나 있습니다.



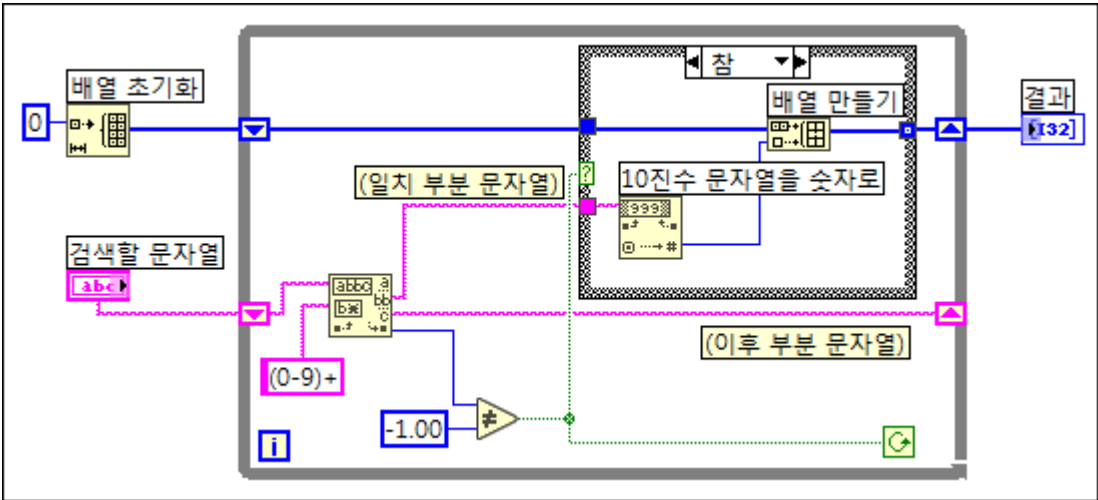
예제 2: 문자열을 통해 검색하기

[패턴 일치] 함수를 사용하여 패턴에 맞는 문자열을 검색할 수 있습니다. 그것을 사용하는 방법에 따라 문자열 데이터 버퍼를 불필요하게 생성하여 성능을 낮출 수 있습니다.

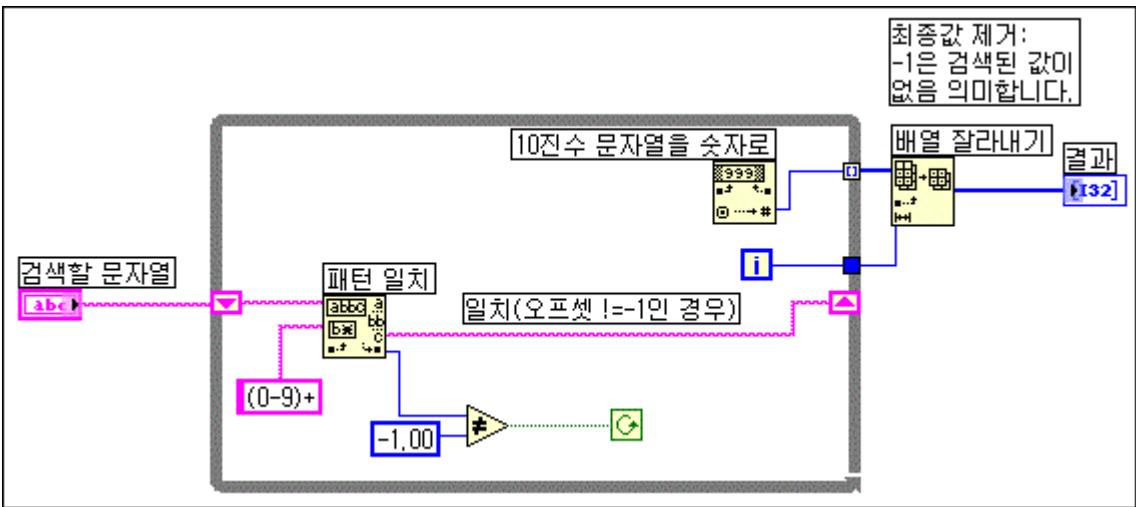
문자열에서 정수를 일치시키려고 하는 경우, $[0-9]^+$ 를 이 함수에 대한 정규식 입력으로 사용할 수 있습니다. 문자열에서 모든 정수 배열을 생성하려면, 루프를 사용하고 반복되는 오프셋 값이 -1이 될 때까지 [정규식 일치]를 반복해서 호출합니다.

다음 블록다이어그램은 문자열에서 정수의 모든 어커런스를 스캔하는 한 가지 방법입니다. 빈 배열을 생성한 후 루프의 각 반복마다 남아 있는 문자열에서 숫자 패턴을 검색합니다. 패턴이 발견되면(오프셋이 -1이 아님), 이 블록다이어그램은 [배열 만들기]를 사용

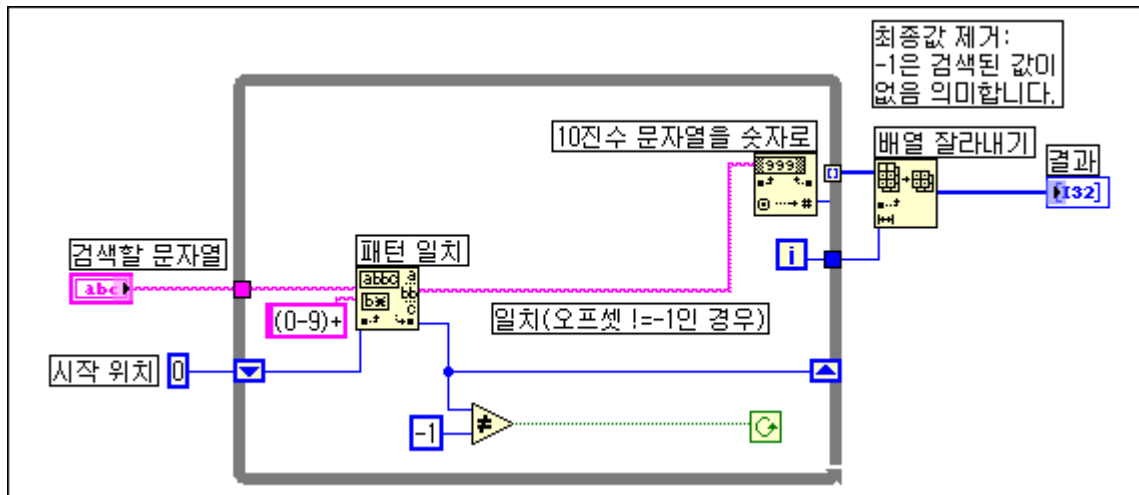
하여 결과 숫자 배열에 숫자를 추가합니다. 문자열에 값이 남아 있지 않으면, [정규식 일치]는 -1을 반환하고 블록 다이어그램은 실행을 완료합니다.



이 블록 다이어그램에서 한가지 문제점은 루프에서 [배열 만들기]를 사용하여 새 값을 이전 값에 연결한다는 것입니다. 대신에, 오토인덱싱을 사용하여 값을 루프의 경계선에 축적할 수 있습니다. 배열에서 [정규식 일치]가 일치되는 것을 찾지 못한 루프의 마지막 반복으로부터 온 여분의 원하지 않는 값을 보게 됩니다. 해결책은 [배열 잘라내기]를 사용하여 여분의 원하지 않는 값을 제거하는 것입니다. 이것이 다음 블록 다이어그램에 나와 있습니다.



이 블록 다이어그램의 다른 문제는 루프를 통해 매번 남아있는 문자열의 불필요한 복사본을 생성한다는 것입니다. [정규식 일치]는 어디에서 검색을 시작할 것인지 나타내는데 사용하는 입력을 가지고 있습니다. 이전 반복으로부터 오프셋을 기억한다면, 이 숫자를 사용하여 다음 반복에서 어디부터 검색을 시작하는지 나타낼 수 있습니다. 이 기술은 다음 블록 다이어그램에 나와 있습니다.



효과적인 데이터 구조를 개발하기

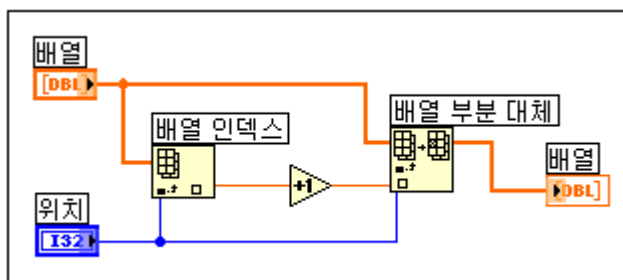
앞의 예제에서 짚고 넘어가야 할 포인트 중 하나는 큰 배열이나 문자열을 가지고 있는 클러스터나 배열 클러스터와 같은 계층구조적인 데이터 구조가 효과적으로 다루어질 수 없다는 것입니다. 이 섹션은 그 이유를 설명하고 더 효과적인 데이터 타입을 선택하는 전략을 설명합니다.

복잡한 데이터 구조가 가지고 있는 문제는 생성하려고 접근하는 원소를 복사하지 않고 데이터 구조 안에서 원소에 접근하고 변경하기는 어렵다는 것입니다. 원소 그 자체가 배열이나 문자열인 경우와 같이 이 원소가 크다면, 이러한 여분의 복사본이 더 많은 메모리를 사용하고 메모리를 복사하는데 시간이 걸립니다.

일반적으로 스칼라 데이터 타입은 매우 효과적으로 다룰 수 있습니다. 마찬가지로, 작은 문자열과 원소가 스칼라인 배열은 효과적으로 다룰 수 있습니다. 스칼라 배열의 경우에, 다음 코드는 배열에서 값을 증가시키기 위해서 무엇을 해야 하는지 보여줍니다.



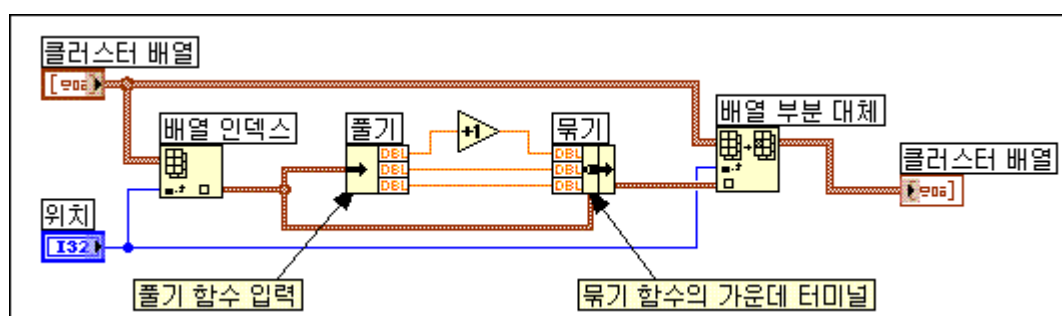
노트 배열, 클러스터, 웨이브폼, 배리언트를 조작할 때 In Place 원소 구조를 사용하여 VI에서의 메모리 사용을 향상시킬 수 있습니다. 많은 LabVIEW 작업의 경우, LabVIEW가 메모리에 데이터를 복사하고 저장해야 합니다. 따라서 실행 속도가 떨어지고 메모리 사용이 늘어납니다. In Place 원소 구조는 LabVIEW가 메모리에 데이터 값을 여러번 복사하지 않고도 일반적인 LabVIEW 작업을 수행합니다. 대신 In Place 원소 구조는 같은 메모리 위치에 있는 데이터 원소에서 작업을 수행하며 해당 원소를 배열, 클러스터, 배리언트, 웨이브폼의 같은 위치에 반환합니다. LabVIEW가 데이터 원소를 메모리의 같은 위치에 반환하므로 LabVIEW 컴파일러는 메모리에 여분의 데이터 복사본을 저장할 필요가 없습니다.



전체 배열의 여분의 복사본을 생성할 필요가 없기 때문에 이것은 꽤 효과적입니다. 또한, [배열 인덱스] 함수에 의해 생성된 원소는 효과적으로 생성되고 다루어지는 스칼라입니다.

클러스터가 스칼라만을 가지고 있다고 가정하면 배열 클러스터도 마찬가지로 해당됩니다. 다음 블록 다이어그램에서, 원소를 다루는 것이 약간 더 복잡해지는데, 그 이유는 [풀기]와 [묶기]를 사용해야 하기 때문입니다. 그러나 이 클러스터는 대부분 작으므로(스칼라는 매우 작은 메모리를 사용), 클러스터 원소에 접근하고 원래 클러스터로 원소를 대체하는데 큰 오버헤드가 발생하지 않습니다.

다음 블록 다이어그램은 풀기, 수행하기, 다시 묶기를 위한 효과적인 패턴을 보여줍니다. 데이터 소스에서 온 와이어는 2가지 대상인 [풀기] 함수 입력과 [묶기] 함수의 중간 터미널을 가져야 합니다. LabVIEW는 이 패턴을 인식하고 더 성능이 좋은 수행 코드를 생성할 수 있습니다.



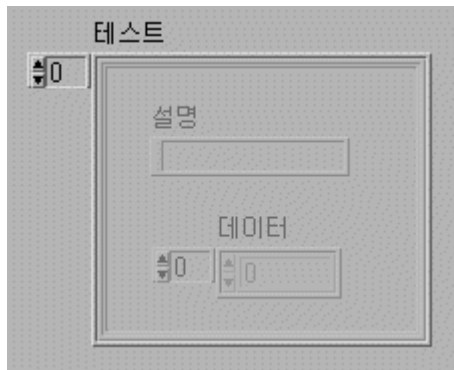
각 클러스터가 큰 부분배열이나 문자열을 포함하는 곳에 배열 클러스터가 있다면, 클러스터에서 원소의 값을 인덱스하고 변경하는 것은 메모리와 속도 면에서 더 많은 로드가 걸릴 수 있습니다.

전체 배열에서 원소를 인덱스할 때, 그 원소의 복사본을 만듭니다. 이렇게 해서, 클러스터의 복사본과 이에 상응하는 큰 부분배열이나 문자열이 만들어집니다. 문자열과 배열의 크기가 다양하기 때문에, 복사하는 과정은 적당한 크기의 문자열이나 부분배열을 만들기 위한 메모리 할당 호출을 포함하고 문자열이나 부분배열의 데이터를 실제로 복사하는 오버헤드를 포함합니다. 이렇게 몇 번만 하려는 계획이면 그리 심각하지는 않을 수 있습니다. 그러나 어플리케이션이 이 데이터 구조에 자주 접근해야 하는 경우, 메모리와 실행 오버헤드는 빠르게 쌓입니다.

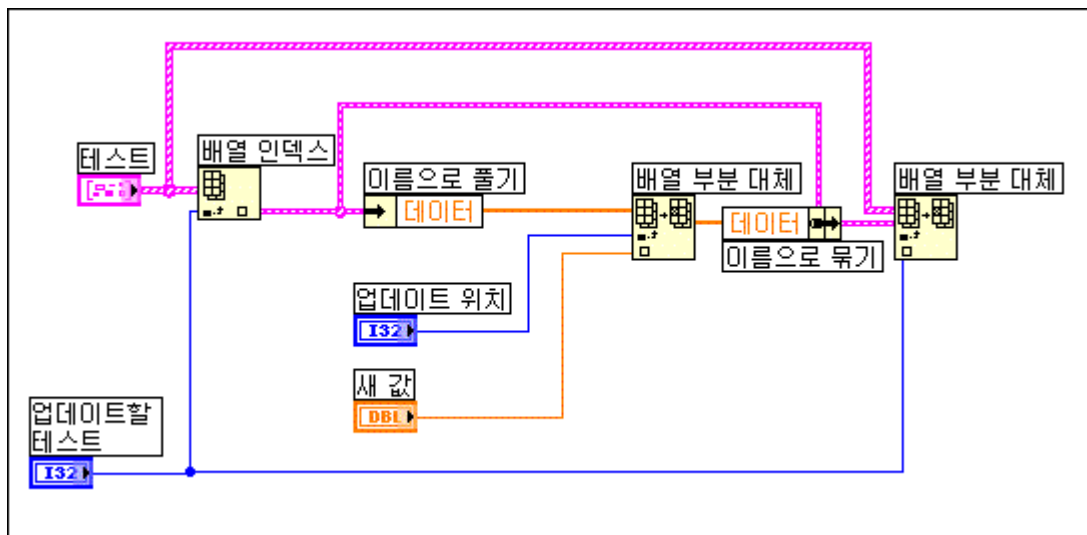
그 해법은 데이터를 다른 형태로 보는 것입니다. 다음의 세 가지 사례 연구는 각 경우에 최선의 데이터 구조를 제안하는 것과 함께 세 가지 다른 어플리케이션을 제시합니다.

사례 연구 1: 복잡한 데이터 타입 피하기

몇가지 테스트의 결과를 기록하려는 어플리케이션을 생각합니다. 결과에서, 사용자는 테스트를 설명하는 문자열과 테스트 결과가 있는 배열을 원합니다. 이 정보를 저장하기 위해 사용하는 하나의 데이터 타입을 다음 프런트패널에서 볼 수 있습니다.



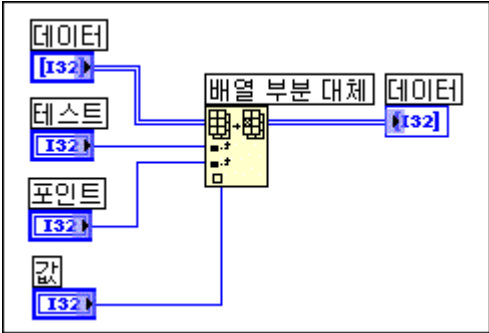
배열에서 원소를 변경하려면 전체 배열 원소에 인덱스를 주어야 합니다. 이제 그 클러스터에 대해 배열을 만들기 위해 원소를 풀어야 합니다. 그 후 배열의 원소를 대체하고 클러스터에 그 결과 배열을 저장합니다. 마지막으로 결과 클러스터를 원래 배열에 저장합니다. 이의 예제가 다음 블록 다이어그램에 나타나 있습니다.



폴기/인덱싱의 각 레벨은 생성되는 데이터를 복사하게 됩니다. 복사본이 반드시 생성될 필요는 없습니다. 데이터를 복사하는 것은 시간과 메모리가 소모됩니다. 해법은 데이터 구조를 가능한 한 평평하게 만드는 것입니다. 예를 들면, 이 사례 연구에서 데이터 구조를 두 개의 배열로 나눕니다. 첫번째 배열은 문자열 배열입니다. 두번째 배열은 2D 배열인데, 여기에서 각 행은 주어진 테스트의 결과입니다. 이 결과가 다음 프런트패널에 나와 있습니다.



이 데이터 구조가 주어져 있으면 다음 블록 다이어그램에서 볼 수 있는 것처럼 [배열 부분 대체] 함수를 사용하여 배열 원소를 직접 대체할 수 있습니다.



사례 연구 2: 혼합된 데이터 타입의 글로벌 테이블

정보 테이블을 유지하려는 또 다른 어플리케이션이 있습니다. 이 어플리케이션에서 글로벌하게 접근할 수 있는 데이터를 원합니다. 이 테이블에는 이득, 하위와 상위 전압 리미트, 채널을 지칭하기 위해 사용되는 이름을 가진 인스트루먼트를 위한 셋팅이 있습니다.

어플리케이션을 통해 데이터에 접근 가능하도록, 다음에 보이는 Change Channel Info VI, Remove Channel Info VI와 같은 SubVI 세트를 생성하여 테이블의 데이터에 접근하는 방법을 고려해볼 수 있습니다.



다음 섹션은 이 VI에 대한 세 가지 다른 실행을 나타냅니다.

명백한 실행

이 함수의 세트와 함께 밀줄 친 테이블을 위해 고려하는 몇 가지 데이터 구조가 있습니다. 먼저 각 클러스터에 이득, 하위 리미트, 상위 리미트, 채널 이름이 포함된 배열 클러스터가 있는 글로벌 변수를 사용할 수 있습니다.

이전 섹션에서 설명된 것처럼 이 데이터 구조는 효과적으로 다루는 것이 어려운데, 그 이유는 일반적으로 데이터에 접근하기 위해 인덱싱하고 풀기하는 몇 개의 단계를 통과해야 하기 때문입니다. 또한, 데이터 구조는 몇 가지 정보가 응축되어 있기 때문에 채널을 찾기 위해 [1D 배열 검색] 함수를 사용할 수 없습니다. [1D 배열 검색]을 사용하여 배열 클러스터에서 특정한 클러스터를 검색할 수 있지만, 단일 클러스터 배열에 일치하는 원소를 찾기 위해 [1D 배열 검색]을 사용할 수 없습니다.

다른 실행 1

이전 사례 연구에서처럼, 데이터를 두 개의 분리된 배열에 저장하는 것을 선택합니다. 하나에는 채널 이름이 있습니다. 다른 것에는 채널 데이터가 있습니다. 이름 배열에서 주어진 채널 이름의 인덱스는 다른 배열에서 상응하는 채널 데이터를 찾기 위해 사용됩니다.

문자열 배열이 데이터로부터 분리되기 때문에 채널을 검색하기 위해 [1D 배열 검색] 함수를 사용할 수 있습니다.

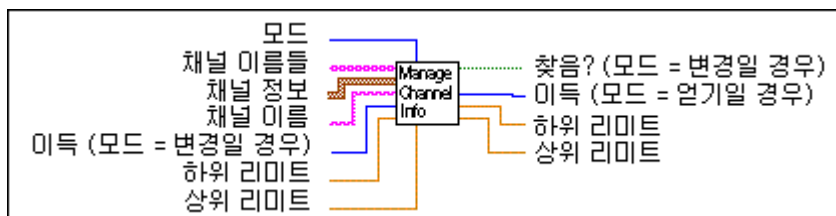
실제로 Change Channel Info VI를 사용하여 1,000 채널의 배열을 생성하려 한다면, 이 실행은 이전 버전보다 거의 2배는 빠릅니다. 이 변경은 성능에 영향을 주는 다른 오버헤드가 있기 때문에 그다지 중요하지는 않습니다.

글로벌 변수로부터 읽어 들일 때, 글로벌 변수의 데이터의 복사본이 생성됩니다. 이렇게 해서 배열에 있는 데이터의 완전한 복사본이 원소에 접근할 때마다 생성됩니다. 다음의 방법은 이 오버헤드를 피하는 훨씬 더 효과적인 방법을 제시합니다.

다른 실행 2

글로벌 데이터를 저장하는 다른 방법이 있으며, 그것은 초기화하지 않은 시프트 레지스터를 사용하는 것입니다. 본질적으로 초기값을 연결하지 않으면, 시프트 레지스터는 그 값을 호출마다 기억합니다.

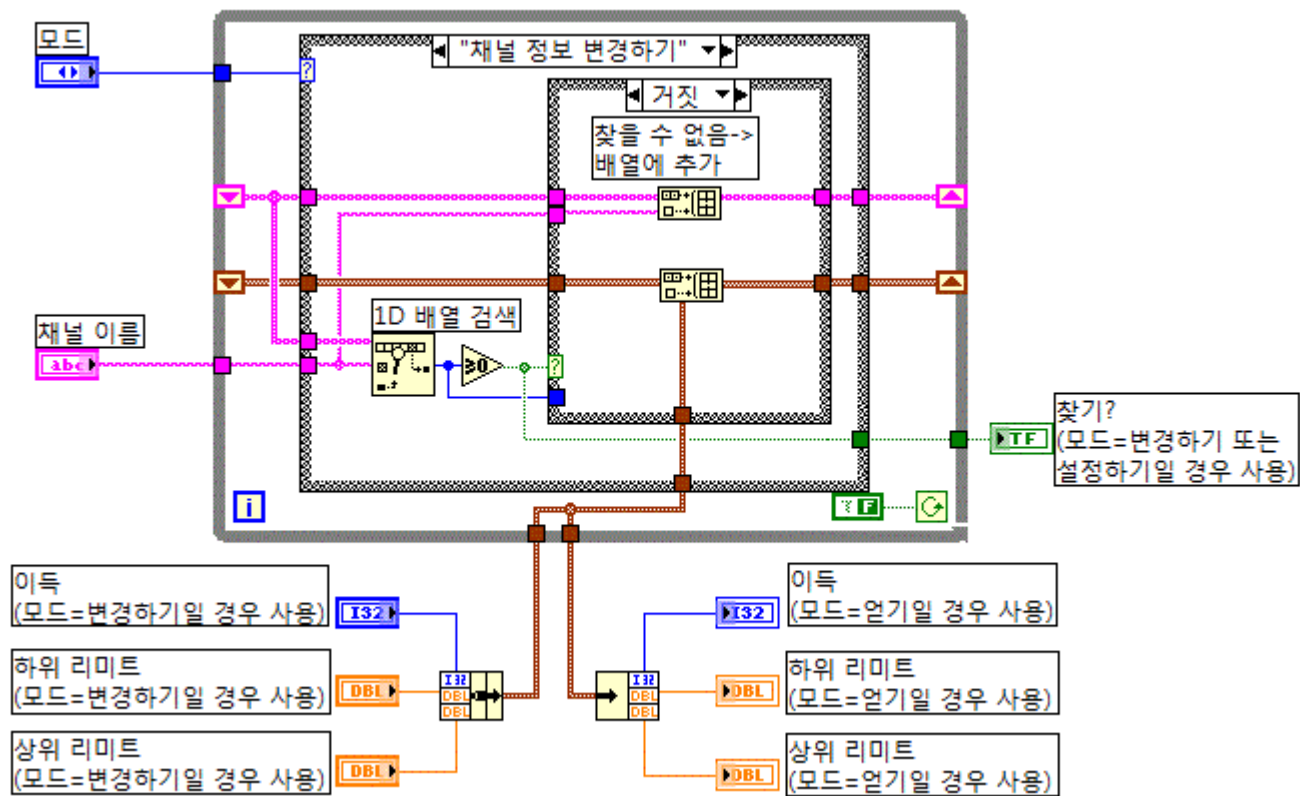
LabVIEW 컴파일러는 시프트 레지스터에 대한 접근을 효과적으로 다룹니다. 시프트 레지스터의 값을 읽는 것이 데이터의 복사본을 반드시 생성하는 것은 아닙니다. 실제로 시프트 레지스터에 저장된 배열에 인덱스를 줄 수 있고 전체 배열의 여분의 복사본을 생성하지 않고도 그 값을 변경하고 업데이트할 수 있습니다. 시프트 레지스터의 문제점은 시프트 레지스터가 있는 VI만이 시프트 레지스터 데이터에 접근할 수 있다는 것입니다. 반면에, 시프트 레지스터는 모듈화라는 이점이 있습니다.



채널을 읽거나 변경하거나 제거하는지 지정하거나 모든 채널에 대해 데이터를 제로로 놓을 것인지 지정하는 모드 입력을 가진 단일 subVI를 만들 수 있습니다.

SubVI에 2 개의 시프트 레지스터를 가진 While 루프가 있습니다. 하나는 채널 데이터를 위한 것이고 다른 하나는 채널 이름을 위한 것입니다. 이 시프트 레지스터는 둘 다 초기화되어 있지 않습니다. 그리고 While 루프 안에 모드 입력에 연결된 케이스 구조를 놓습니다. 모드의 값에 따라 시프트 레지스터에서 데이터를 읽고 변경할 수 있습니다.

다음은 이 3 개의 다른 모드를 다루는 인터페이스를 가진 SubVI의 개략적인 그림입니다. 단지 Change Channel Info(채널 정보 변경하기) 코드만이 나타나 있습니다.

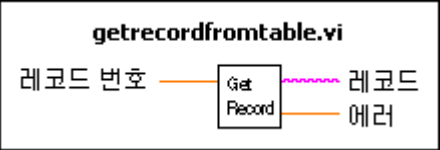
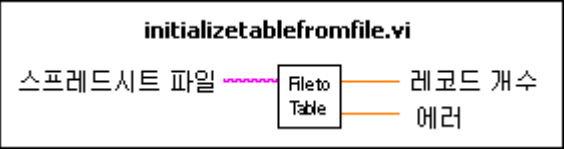


1,000개의 원소에 대해 이 실행은 이전 실행보다 2배가 빠릅니다. 그리고 원래 실행보다 4배가 더 빠릅니다.

사례 연구 3: 문자열의 정적 글로벌 테이블

이전 사례 연구는 데이터 타입이 혼합되어 있는 테이블을 가진 어플리케이션을 보여주고 그 테이블은 자주 변경될 수 있습니다. 많은 어플리케이션에서 일단 생성되면 꽤 정적인 테이블 정보를 가지고 있습니다. 스프레드시트 파일에서 테이블을 읽어올 수 있습니다. 일단 메모리로 읽어오면 정보를 찾기 위해 그것을 주로 사용합니다.

이 경우에 다음의 2가지 함수를 구성하여 구현합니다. 그 두 함수는 Initialize Table From File과 Get Record From Table입니다.



테이블을 구현하는 한가지 방법은 문자열의 2D 배열을 사용하는 것입니다. 컴파일러는 메모리의 분리된 블록의 문자열 배열에 각 문자열을 저장합니다. 많은 문자열이 있으면

(예를 들면, 5,000개 문자열 이상), 메모리 관리자에게 로드를 줍니다. 각 객체의 수가 증가함에 따라 이 로드는 성능에 상당한 손실을 줄 수 있습니다.

큰 테이블을 저장하는 다른 방법은 단일 문자열로서 테이블을 읽는 것입니다. 그리고 나서 문자열에 각 기록의 오프셋이 있는 분리된 배열을 만듭니다. 이것은 수천 개의 작은 메모리 블록을 만드는 대신에 하나의 큰 메모리 블록(문자열)과 더 작게 분리된 메모리 블록(오프셋의 배열)이 생기도록 조직을 변경합니다.

이 방법은 실행하기에 더 복잡할 수 있지만, 큰 테이블에서는 훨씬 더 빨라질 수 있습니다.

VI 코드 복잡도

VI 코드 복잡도는 프런트패널 및 블록 다이어그램 노드의 개수, 타입, 조직 등 VI의 여러 특징을 종합적으로 나타내는 용어입니다. LabVIEW에서 코드 복잡도는 다음과 같이 서로 연관성이 있는 개별적인 항목으로 나타납니다:

복잡도 측정	사용하는 경우	설명
VI 메트릭스 윈도우	- 정량화할 수 있는 메트릭에 근거해 프로젝트 개발에 소요되는 노력을 추정하고 추측하기 - VI의 복잡도를 감소하는 방법을 찾으면 다음과 같은 혜택을 얻을 수 있습니다: - 컴파일 속도 향상 - 컴파일 중 메모리 사용 감소 - 블록 다이어그램 가독성 향상 및 관리 용이	이 윈도우는 다양한 종류의 노드 개수, 블록 다이어그램에서 허용되는 최대 하위 레벨 구조 등 코드 복잡도를 여러 측면에서 보고합니다.

복잡도 측정	사용하는 경우	설명
컴파일된 코드 복잡도	컴파일러 최적화를 제한하기 시작하는 임계점을 조정하여 LabVIEW의 편집기 응답 향상하기	이 값은 전반적인 VI 코드 복잡도를 나타냅니다. 컴파일러는 이 값을 사용하여 VI가 컴파일 과정 중, 컴파일러 최적화를 위해 너무 많은 메모리를 필요로 하는지 결정합니다. LabVIEW는 VI 메트릭 윈도우에서 제공하는 숫자와 관련된 내부 컴파일 세부사항에 근거하여 이 값을 산출합니다. 그러나 이러한 세부사항은 LabVIEW 버전마다 변경될 수 있습니다. 그러므로 VI 메트릭 윈도우의 실제적인 통계와는 달리, 이 코드 복잡도 값은 컴파일 과정의 구현 세부사항과는 직접적으로 관련이 없는 단위를 사용합니다. 대신, 이 단위는 VI 코드 복잡도를 LabVIEW가 최적화를 제한하는데 사용하는 컴파일러 최적화 임계점과 비교할 때 사용하는 기준치로만 작용합니다.

재호출: 같은 SubVI에 동시에 여러 호출을 허용

기본으로, LabVIEW는 VI를 재호출이 아닌 실행으로 설정합니다. 재호출이 아닌 실행에서 여러 VI들이 동시에 하나의 VI를 SubVI로 호출하는 경우, 이러한 호출 중 하나의 호출만이 오직 한 번에 하나씩 해당 SubVI를 실행할 수 있습니다. LabVIEW는 재호출되지 않는 SubVI의 호출자가 순서대로 SubVI를 사용하도록 합니다. 재호출되지 않는 VI에는 호출자의 파라미터, 실행 중 계산된 임시 값 및 재호출되지 않는 VI가 여러 호출 간에 유지하는 모든 상태를 저장할 수 있는 오직 하나의 데이터 공간만이 있습니다. 그러나 VI가 재호출 실행이 가능하도록 설정된 경우, LabVIEW는 동시에 모든 호출자로부터 재호출 VI를 입력할 수 있습니다. 재호출 실행에서 LabVIEW는 데이터 공간의 여러 인스턴스 또는 복제본을 할당하여 각 호출 장소 또는 호출자의 블록 다이어그램 내에서의 SubVI의 용도가 병렬로 동시에 실행되어 각 호출 장소에서 독립된 복제본이 사용될 수 있도록 합니다.



노트

(FPGA Module) FPGA VI는 기본으로 재호출입니다.

재호출 실행의 타입

LabVIEW는 다음과 같은 타입의 재호출 VI를 제공합니다:

- 재호출이 아닌 실행—LabVIEW는 SubVI의 모든 인스턴스에서 사용하도록 단일 데이터 공간을 할당합니다.
- 공유 복제 재호출 실행—LabVIEW는 먼저 한 복제본에서 생성된 복제본의 풀을 할당하여 여러 호출자가 공유할 수 있도록 합니다. 호출자가 공유 복제 VI를 호출하면, 풀에서 복제본을 가져오고 호출이 실행을 마치면 해당 복제본을 반환합니다.

호출자가 공유 복제 VI를 호출할 때 풀이 비어 있는 경우, LabVIEW는 해당 호출에 대한 새 복제본을 생성하고 실행이 완료될 때 해당 복제본을 풀에 놓아 복제본 풀을 확장시킵니다.

- 미리 할당된 복제 재호출 실행—LabVIEW는 SubVI를 사용할 때마다 별도의, 공유할 수 없는 복제본을 할당합니다.

VI의 재호출 설정을 설정하려면, VI 프로퍼티 대화 상자의 실행 페이지를 참조하십시오.

어떤 재호출 실행 타입을 사용할지 선택하기

사용 고려사항	재호출이 아님	공유 복제 재호출	미리 할당된 복제 재호출
초기화되지 않은 시프트 레지스터와 같은 상태를 유지할 수 있음	데이터 공간의 단일 인스턴스를 유지하고 모든 호출 사이트에서 그 상태를 사용합니다.	상태를 유지하지 않음—각 호출 사이트는 복제본의 풀에서 무작위로 복제본의 데이터 공간을 사용합니다. 호출 사이트는 상태를 공유할 수 있습니다.	각 호출 사이트에 대해 상태를 유지함—각 호출 사이트는 각각 별도의 특정한 복제본을 가집니다.
여러 개의 동시 호출 VI에 대한 결정성	호출 사이트는 다른 동시 호출자 뒤에서 대기할 수 있음—SubVI의 인스턴스가 실행을 시작하는데 필요한 시간은 해당 인스턴스 이전에 몇개의 인스턴스가 실행되도록 예정되었는지에 따라 다릅니다.	기다릴 수 있음—복제본의 풀이 비어 있는 경우 호출 사이트는 LabVIEW가 새 복제본을 생성할 때까지 대기해야 할 수 있습니다. 풀이 비어 있지 않은 경우, 풀에서 복제본을 가져오고 풀에 복제본을 반환하면 타이밍 결정성이 높지 않은 지연을 일으킬 수 있습니다.	기다리지 않음—LabVIEW는 실행 전 각 호출 사이트에 대해 복제본을 할당합니다. 이렇게 하면 시간 결정성있는 메모리 사용 및 실행 속도가 허용됩니다.
동시 호출 VI가 존재하지 않는 경우 호출 오버헤드가 발생합니다.	중간—LabVIEW는 반드시 동시 호출자를 확인해야 합니다.	최고—LabVIEW는 반드시 복제본의 풀에서 복제본을 가져오고 복제본의 풀로 복제본을 반환해야 합니다. 호출은 LabVIEW가 풀에 복제본을 추가하도록 기다려야 합니다.	최저—LabVIEW는 각 호출 사이트에 미리 복제본을 할당합니다.
메모리 사용	최저—LabVIEW는 SubVI에 하나의 데이터 공간만 할당합니다.	중간—LabVIEW는 동시에 실행되는 최대 개수의 인스턴스에 대해서만 복제본을 할당해야 합니다.	최고—LabVIEW는 SubVI의 각 인스턴스에 복제본을 할당해야 합니다.



노트 (FPGA Module) FPGA VI를 사용하는 경우, 공유 복제 재호출 VI 및 미리 할당된 복제 재호출 VI는 동일하게 동작합니다.

재호출 VI의 여러 인스턴스를 비동기적으로 호출하기

재호출 VI를 비동기적으로 호출하는 것에 대한 정보는 [병렬 실행을 위해 VI의 여러 인스턴스를 비동기적으로 호출하기](#) 토픽을 참조하십시오.

재호출 VI의 복제본 프런트패널 보기

각 복제본에는 개별적인 프런트패널이 있습니다. 복제본이 실행되지 않을 때 프런트패널을 보려면, 해당 SubVI의 인스턴스를 더블 클릭합니다. 또한 재호출 VI가 [실행 중 프런트패널을 열도록](#) 설정할 수 있습니다. SubVI의 제목 표시줄은 (복제)를 포함하며 이 VI가 원본이 아니라 복제본임을 나타냅니다. 복제본에서 VI 원본에 접근하려면 보기>>관계 탐색>>재호출 아이템을 선택합니다. 또한 SubVI 안에서 프런트패널:열기 메소드를 선택하여 각 복제본의 프런트패널을 실행 중에 프로그램적으로 열 수 있습니다. SubVI의 제목 표시줄은 (복제)를 포함하며 이 VI가 소스 VI의 복제본임을 나타냅니다.

재호출이 아닌 VI의 프런트패널과 상호 작용하는 것과 비슷한 방법으로 재호출 VI의 프런트패널과 상호 작용할 수 있으며, 특히 다음의 방법을 참조하십시오:

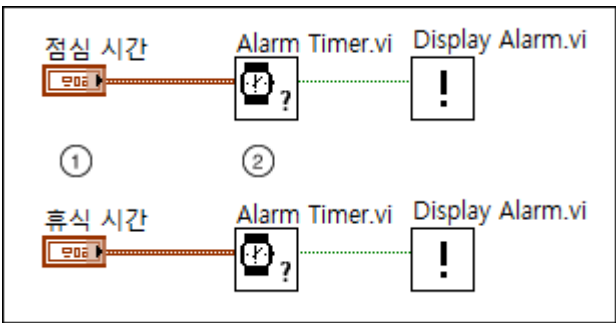
- 이벤트 구조가 프런트패널 객체의 이벤트를 처리하도록 설정하기
- 프런트패널을 [서브패널](#)로 사용하기
- 프로퍼티 및 메소드를 사용하여 [컨트롤과 인디케이터](#) 설정하기

재호출 실행의 예제

다음 두 섹션은 기다리고 데이터를 공유하지 않는 재호출 VI의 예제를 설명합니다.

SubVI의 여러 인스턴스 동시에 실행되도록 하기

다음 블록 다이어그램은 점심 시간 및 휴식 시간 시간에 도달한 것을 사용자에게 알려줍니다. 그러나, 정확한 결과를 얻으려면 Alarm Timer VI를 재호출되도록 설정하는 것이 중요합니다.



①	사용자가 점심 시간 및 휴식 시간을 지정합니다. 이 예제에서 점심 시간은 오후 12시, 휴식 시간은 오전 10시라고 가정해봅니다.
②	Alarm Timer VI는 지정된 입력 시간까지 기다립니다. Alarm Timer VI의 인스턴스가 다이어그램의 병렬 부분에 나타나기 때문에, 어떤 인스턴스가 먼저 실행되는지 예측할 수 없습니다. 이 예제에서는 점심 시간 타이머가 먼저 실행된다고 가정합니다.

Alarm Timer VI가 재호출이 아닌 실행으로 설정된 경우, 휴식 시간 타이머는 점심 시간 타이머가 실행을 종료할 때까지 기다려야 하기 때문에 오후 12시까지는 실행을 시작할 수 없습니다. 이러한 지연으로 인해 휴식 시간 타이머는 오전 10시에 알람을 작동할 수 없습니다. 그렇기 때문에 SubVI의 두 인스턴스가 동시에 실행되고 정해진 시간에 알람이 작동되도록 하려면, Alarm Timer VI가 재호출 실행 타입 중 하나를 사용하도록 설정해야 합니다.

각 SubVI 인스턴스를 여러 번 호출할 때 데이터 저장하기

특정 호출 사이트의 SubVI에 대한 호출 사이의 상태를 유지하려면, 반드시 미리 할당된 복제 재호출 실행을 사용해야 합니다. 예를 들어, VI에 전달된 네개의 데이터 포인트의 실행 지수 평균을 계산하는 SubVI인 ExpAvg를 생성합니다. LabVIEW는 4개의 최신 데이터 포인트를 단일 반복 While 루프의 초기화되지 않은 시프트 레지스터에 유지할 수 있습니다.

다른 VI가 ExpAvg subVI를 사용하여 두 데이터 수집 채널의 실행 평균을 계산합니다. VI는 과정에서 두 포인트의 전압을 모니터하고 스트립 차트에 지수 실행 평균을 디스플레이 합니다. VI의 블록다이어그램은 ExpAvg SubVI의 두 인스턴스를 포함합니다. 번갈아 호출되는데, 하나는 채널 0, 다른 하나는 채널 1입니다. 채널 0이 먼저 실행된다고 가정합니다. ExpAvg subVI가 재호출이 아닌 경우, 채널 1의 호출은 채널 0의 호출에 의해 계산된 평균을 사용합니다. 또한 채널 0은 채널 1의 호출에 의해 계산된 평균을 사용합니다. ExpAvg subVI를 재호출로 만들어, 각 호출은 데이터를 공유하지 않고 독립적으로 실행될 수 있습니다.

Timed 구조에서 SubVI 재호출 셋팅 덮어쓰기

(Real-Time, Windows) **Timed 루프 또는 Timed 시퀀스** 구조 내에서 공유된 복제 재호출 VI의 복제본을 미리 할당하거나 공유하도록 LabVIEW를 설정할 수 있습니다. 예를 들어, 공유된 복제 재호출 VI를 Timed 루프 또는 Timed 시퀀스 구조 안에 놓으면, 구조 안에서 호출하는 재호출 VI의 인스턴스에 대한 복제본을 미리 할당하도록 구조를 설정할 수 있습니다. 구조 밖에서 호출하는 재호출 VI의 인스턴스는 계속해서 복제본을 공유합니다. Timed 루프 또는 Timed 시퀀스 구조 내에서의 VI 할당을 설정하려면, 구조에서 마우스 오른쪽 버튼을 클릭한 후 공유 복제 할당을 선택하고 다음 중에서 하나를 선택합니다:

- 자동—LabVIEW는 구조의 내용에 따라 구조를 미리 할당 또는 필요할 때 할당으로 설정할지 결정합니다. LabVIEW는 바로 가기 메뉴에 텍스트를 추가하여 LabVIEW가 선택하는 셋팅을 표시합니다. 예를 들어, LabVIEW가 미리 할당을 선택하는 경우, 메뉴 아이템은 자동(미리 할당)이 됩니다.
- 미리 할당—LabVIEW가 구조 내의 재호출 VI에 대한 각 호출에 대해 복제 VI를 생성합니다. Timed 구조 또는 Timed 시퀀스 구조는  아이콘을 디스플레이하여 LabVIEW가 구조 내에 있는 모든 재호출 VI에 대한 복제본을 미리 할당하고 있음을 표시합니다.

- 필요할 때 할당—사용자가 구조 내에 있는 재호출 VI를 호출하기 전까지는 LabVIEW가 복제 VI를 생성하지 않습니다. Timed 구조 또는 Timed 시퀀스 구조는  아이콘을 디스플레이하여 LabVIEW가 구조 내에 있는 모든 재호출 VI에 대한 복제본을 공유하고 있음을 표시합니다.

재호출 VI 디버깅

재호출 VI에서 디버깅을 허용하려면, 파일 >> VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이하고, 폴다운 메뉴에서 실행을 선택하고, 디버깅 허용 확인란에 확인 표시를 합니다.

미리 할당된 복제본이 있는 VI의 특정 인스턴스를 디버깅하려면, SubVI에서 해당 인스턴스를 엽니다. 복제본 VI를 편집할 수는 없지만, 이 VI의 블록 다이어그램을 사용하여 디버깅할 수는 있습니다. 예를 들면, **브레이크포인트**를 설정하고, **프로브**를 사용하고, **실행 하이라이트**를 활성화하고, **단계별** 실행도 할 수 있습니다.

복제본을 공유하는 VI의 인스턴스를 디버그하려면, 복제 VI가 아닌 소스 VI에 서 브레이크포인트, 프로브 및 실행 하이라이트를 사용해야 합니다. 복제 VI는 여러 호출에서 이러한 설정을 그래도 유지하지 않습니다.



노트 어플리케이션과 공유 라이브러리를 디버그할 때 [VI 참조 열기] 함수가 생성하는 재호출 패널은 디버그할 수 없습니다. 또한 LabVIEW 생성 공유 라이브러리의 입력 포인트인 재호출 패널도 디버깅할 수 없습니다.

행렬 사용하기

2D 배열 대신 행렬 데이터 타입을 사용하여 행렬 데이터를 나타냅니다. 행렬 데이터 타입은 행렬 연산, 특히 일부 선형 대수 연산에서 실수나 복소수 스칼라 데이터의 행 또는 열을 저장하기 때문입니다. 행렬 연산을 수행하는 수학 VI는 행렬 데이터 타입을 받고 행렬 결과를 반환합니다. 이로 인해 데이터 흐름에서 이후의 다형성 VI와 함수가 행렬 특성의 연산을 수행할 수 있습니다. 수학 VI가 행렬 연산을 수행하지 않으나 행렬 데이터 타입을 받는 경우, 해당 VI는 자동으로 행렬 데이터 타입을 2D 배열로 변환합니다. 2D 배열을 기본으로 행렬 연산을 수행하는 VI에 연결하는 경우, 해당 VI는 2D 배열의 데이터 타입에 따라 자동으로 2D 배열을 실수 또는 복소수 행렬로 변환합니다.

대부분의 숫자 함수는 행렬 데이터 타입과 행렬 연산을 지원합니다. 예를 들어, [곱하기] 함수를 사용하여 행렬에 다른 행렬 또는 숫자를 곱할 수 있습니다. 기본 숫자 데이터 타입과 복소수 선형 대수 함수를 결합하여 정확한 행렬 연산을 수행하는 숫자 알고리즘을 생성할 수 있습니다.



노트

강제 변환점은 VI 또는 함수가 데이터를 행렬 또는 2D 배열로, 또는 그 반대로 변환했을 때 나타납니다. LabVIEW는 행렬을 2D 배열과 같은 방식으로 저장하기 때문에 이런 데이터 변환은 성능에 영향을 미치지 않습니다.

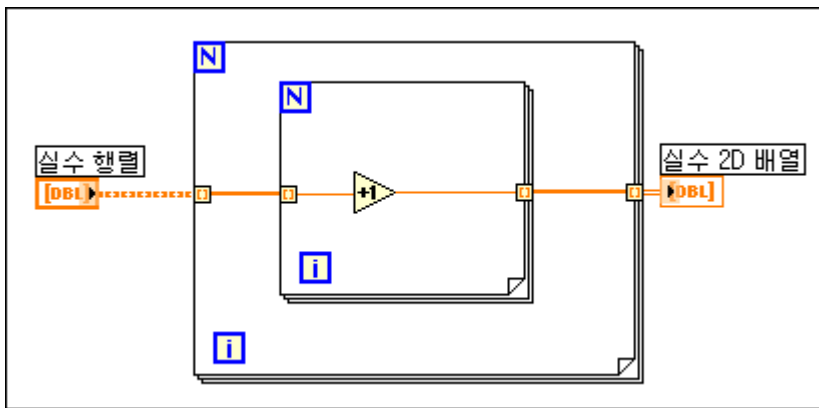
실수 행렬은 배정도 원소를 포함하며 복소수 행렬은 배정도 구성요소를 가진 복소수 원소를 포함합니다. 행렬은 두 차원만 가질 수 있습니다. 행렬의 배열은 생성할 수 없습니

다. [묶기] 함수를 사용하여 둘 이상의 행렬을 결합하여 클러스터를 생성합니다. 행렬은 배열과 같은 제한을 갖습니다.

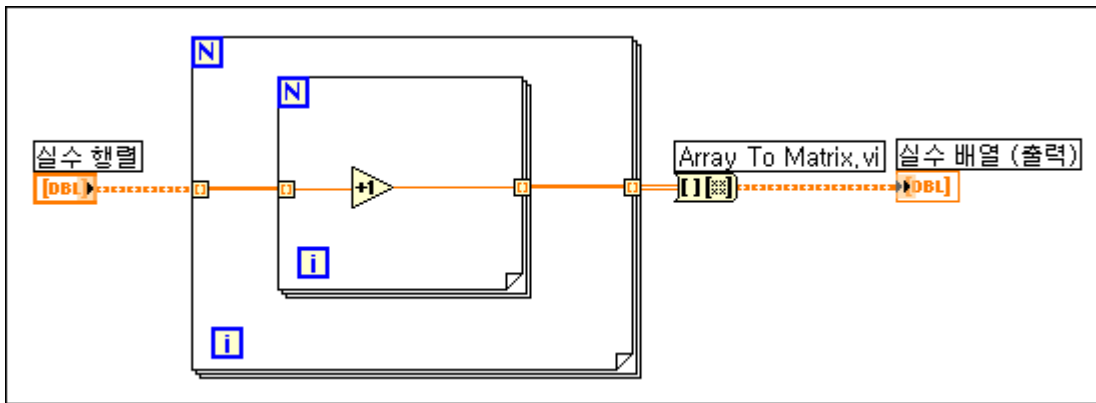
[행렬] 함수를 사용하여 행렬의 원소, 행, 열을 조작합니다. 행렬 함수는 행렬 데이터 타입을 반환합니다. 예를 들어, [행렬 원소 얻기] 함수를 통해 행렬의 행 또는 열을 추출하면, 스칼라 값의 1D 배열이 아닌 단일 행 또는 열의 행렬이 생성됩니다. 스칼라 값의 2D 배열이 아닌 행렬을 생성하려면, 이 행렬을 [행렬 만들기] 함수의 다른 배열과 통합시킵니다.

행렬의 차원을 줄이는 모든 VI는 데이터를 1D 배열 또는 배열도 부동소수나 복소수로 변환하도록 요구합니다. 2D 구조를 다시 만들기 위해 1D 배열 또는 숫자를 사용하는 경우, LabVIEW는 원래 행렬이 아닌 2D 배열을 생성합니다.

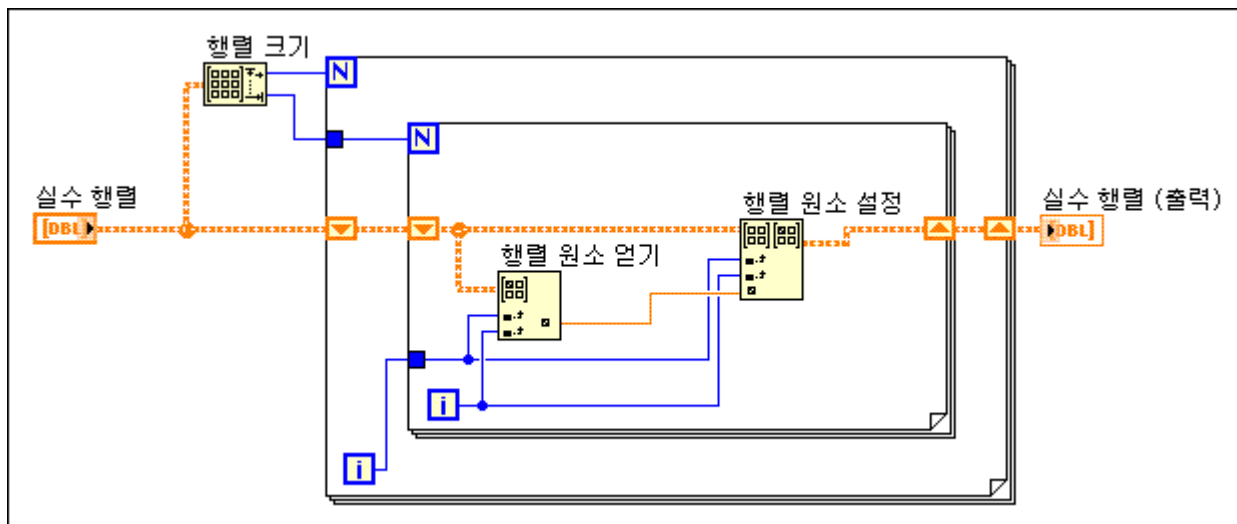
다음의 블록 다이어그램은 For 루프에서 오토인덱싱을 사용하여 행렬에서 데이터를 추출하고 행렬을 2D 배열로 다시 만듭니다.



오토인덱싱이 활성화되어 있는 경우, For 루프는 행렬을 2D 배열로 취급합니다. VI의 입정 포인트에서 행렬 데이터 타입을 입력으로 사용해야 하는 경우, 다음 블록 다이어그램에서 보이는 것처럼 [배열을 행렬로] 함수를 사용하여 2D 배열을 행렬로 변환하십시오.



데이터를 행렬 데이터 타입으로 유지하려면, 오토인덱싱을 비활성화하고 시프트 레지스터를 사용하십시오. 예를 들어, 다음의 블록 다이어그램은 시프트 레지스터를 사용하여 행렬의 2차원과 데이터 타입을 유지합니다. 시프트 레지스터는 반복 사이에 행렬 데이터 타입을 전달합니다.



행렬 데이터 타입을 받는 많은 다형성 함수는 연산 자체가 배열 기반 연산인 경우에도 행렬 데이터 타입을 반환합니다. 블록 다이어그램의 함수 또는 subVI가 행렬 데이터 타입을 2D 배열로 변환하고 데이터 흐름에서 다음의 작업이 배열 기반 작업인 경우, 배열로의 변환을 허용하고 배열 데이터로 작업한 후, 선형 대수 VI를 사용할 때와 같이 필요할 때 [배열을 행렬로] 함수를 사용하여 배열을 행렬로 변환합니다.



노트 블록 다이어그램에 행렬 데이터 타입을 받으나 2D 배열을 반환하는 subVI가 포함된 경우, 배열을 기본으로 행렬 데이터 타입을 받는 다형성 VI 또는 함수에 연결하기 전에는 결과 배열을 다시 행렬로 변환하지 않아도 됩니다. 데이터를 행렬로 저장해야 하는 경우, [행렬을 배열로] 함수를 사용하여 데이터를 다시 행렬로 변환합니다.

행렬 컨트롤, 인디케이터, 상수 생성하기

기본으로, 행렬 컨트롤과 인디케이터는 하나 이상의 원소를 보여주고 두 차원 모두의 스크롤 막대를 디스플레이합니다. 행렬 컨트롤이 타입 정의이므로, 행렬 컨트롤의 바로 가기 메뉴는 차원 추가와 차원 제거 옵션을 제외하고 타입 정의 옵션을 포함합니다. 블록 다이어그램에 행렬 상수를 생성하려면, 컨트롤 팔레트에서 행렬 컨트롤을 선택하고 블록 다이어그램에 놓거나 임의의 행렬 터미널에서 상수를 생성합니다. 배열 상수를 사용하여 상수 데이터를 저장하거나 다른 행렬과 비교하기 위한 기준으로 삼을 수 있습니다.

블록 다이어그램에서 행렬 데이터 타입은 다른 와이어 패턴을 가진 실수 2D 배열 또는 복소수 2D 배열 데이터 타입처럼 보입니다. 행렬 데이터 타입을 받는 VI 및 함수는 사용자가 행렬 데이터 타입을 입력으로 연결할 때 자동으로 행렬 특정 작업을 지원합니다.

행렬 데이터 타입을 입력으로 다음 함수 중 하나에 연결할 때, 행렬 데이터 타입을 사용하는 subVI를 포함하는 VI는 함수를 대체합니다.

- 같음?
- 같지 않음?
- 절대값
- 더하기
- 곱하기

- 제공근
- 빼기
- 지수
- 자연로그
- X의 거듭제곱
- 실수/허수를 복소수로
- 복소수를 실수/허수로
- 극좌표를 복소수로
- 복소수를 극좌표로

결과 VI는 같은 아이콘을 가지지만 행렬 특정 알고리즘을 포함합니다. 입력으로부터 행렬을 연결 해제할 경우 노드는 VI를 남깁니다. 다른 데이터 타입을 입력으로 연결하여 원래 함수를 복구합니다. 데이터 타입을 함수에 연결하고 데이터 타입이 기본 수학 연산에서 실패를 유발할 경우, 함수는 빈 행렬 또는 NaN을 반환합니다. 예를 들어, 2×3 차원의 행렬을 [더하기] 함수의 입력에 연결한 후 3×2 행렬을 다른 입력에 연결한 경우, 함수는 빈 행렬을 반환합니다.

행렬 비교하기

[같은?] 및 [같지 않음?] 함수를 사용하여 행렬을 비교합니다. 이 함수는 행렬 입력의 경우 집합 비교 모드에서 작동하며 배열을 비교하는 것과 같은 규칙을 따릅니다.

둘 또는 그 이상의 입력을 받는 숫자형 함수가 행렬과 2D 배열을 입력으로 받으면, 대부분의 경우 함수는 행렬 데이터 타입을 반환합니다. 커넥터 팬에 연결된 출력이 2D 배열이 아닌 경우 함수는 배열 연산을 수행합니다. 예를 들어, 실수 행렬과 실수 2D 배열을 [나누기] 함수에 연결하는 경우, 기본으로 함수는 함수의 결과를 포함하는 실수 행렬 출력을 반환합니다. 실수 행렬과 복소수 2D 배열을 연결하는 경우, 출력이 복소수 원소를 요구하기 때문에 함수는 복소수 2D 배열을 반환합니다. 두 번째 경우에서, 함수가 연산을 위해 실수 행렬을 복소수 2D 배열로 변환하였음을 나타내기 위해 실수 행렬이 연결된 입력에 강제 변환점이 나타납니다.

행렬의 기본 크기와 값

고정된 수의 원소로 행렬의 크기를 제한할 수 없습니다. 그러나 행렬 컨트롤의 기본값을 설정하면, 기본 크기는 설정할 수 있습니다.

필요 이상 크게 행렬의 기본 크기를 만들지 않습니다. 행렬의 기본 크기를 크게 설정한 경우, 행렬의 각 원소를 위한 기본 데이터를 VI에 저장할 때 디스크에서 VI의 크기가 증가합니다.

행렬 컨트롤을 프런트패널에 추가할 때, 컨트롤은 처음에 빈 행렬을 나타냅니다. 각 차원은 0이며 행렬 원소는 비활성화됩니다. 프런트패널 행렬 컨트롤은 두 개의 기본값을 가집니다. 즉, 부동소수 또는 복소수 기본값인 행렬 기본값과 각 셀의 기본값입니다.

행렬 기본값은 다른 모든 프런트패널 컨트롤의 기본값과 같습니다. 행렬 기본값은 VI가 로드될 때 배열의 값입니다. 또한, 호출하는 VI에 값이 연결되어 있지 않은 경우 subVI에 있는 행렬이 사용될 수 있습니다.

스칼라 기본값은 행렬이 확장될 때 행렬을 채우는데 사용되는 값입니다. 예를 들면, 행렬 인덱스를 행렬의 정의된 부분 밖의 값으로 설정하고 원소에 행렬의 정의된 부분을 지난 한 행보다 큰 값을 입력하는 경우, 행렬의 이전 끝과 추가한 원소 사이의 원소는 스칼라 기본으로 설정됩니다.

행렬 함수의 경우, 함수가 유효하지 않은 작업을 수행할 때 원래의 행렬 또는 2D 배열의 바깥 부분에 값이 나타납니다. 유효하지 않은 작업의 경우에는 LabVIEW가 바깥 부분에 반환하거나 채우는 값이 원래의 행렬 또는 2D 배열의 데이터와 원소 타입에 따라 달라집니다. 유효하지 않은 작업의 값은 행렬을 확장시킬 때 나타나는 기본 값 0과는 다릅니다.



관련 정보

[강제 변환점](#) (285 페이지 참조)

[배열과 클러스터를 이용한 데이터의 그룹화](#) (417 페이지 참조)

[루프 반복 사이에서 값 전달하기](#) (369 페이지 참조)

[컨트롤과 인디케이터의 스크롤 막대 디스플레이하기](#) (1003 페이지 참조)

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

[사용자 정의 상수 생성 및 편집하기](#) (846 페이지 참조)

[배열 비교하기](#) (1134 페이지 참조)

[행렬 컨트롤, 인디케이터, 상수 생성하기](#) (1069 페이지 참조)

VI 포팅과 현지화하기

이 섹션에서는 LabVIEW에서 플랫폼 사이의 VI 포팅을 설명합니다. 또한 LabVIEW에서 VI를 현지화하는 것에 대해 설명합니다.



관련 정보

[플랫폼 사이에서 VI 포팅하기](#) (242 페이지 참조)

[VI 현지화하기](#) (247 페이지 참조)

[VI 문자열 반출 및 반입하기](#) (916 페이지 참조)

[반출된 VI 문자열의 예](#) (917 페이지 참조)

플랫폼 사이에서 VI 포팅하기

포팅 여부는 파일 이름, 분리자 문자, 해상도와 폰트의 차이, 라벨의 겹치기 가능성, 그림 포맷의 차이와 관련이 있습니다.

포팅 가능한 VI와 포팅 불가능한 VI

LabVIEW의 버전이 같으면 VI는 LabVIEW가 실행되는 모든 플랫폼 사이에서 포팅 가능합니다. .NET나 ActiveX와 같이 플랫폼 특정 기능을 포함하는 VI는 포팅이 가능하지 않습니다. 이런 케이스에서 VI를 포팅하면 깨집니다.

LabVIEW는 모든 플랫폼에서 같은 파일 포맷을 사용합니다.

새 플랫폼에서 VI를 열 때, LabVIEW는 VI가 다른 플랫폼에서 온 것을 인식하고 VI를 리컴파일하여 현재 프로세서에 맞는 올바른 명령을 사용합니다. VI를 다른 플랫폼에 사용하도록 포맷된 디스크로 이동하는 경우, 이 디스크를 읽는 유틸리티 프로그램이 필요할 수도 있습니다.

다음 VI는 포팅할 수 없습니다:

- `vi.lib` 디렉토리에 배포된 VI — 각 LabVIEW의 배포는 자체 `vi.lib`를 포함합니다. `vi.lib`의 VI를 다른 플랫폼으로 옮기지 마십시오.
- 호환성 VI — LabVIEW의 이전 버전에서 VI를 생성하고 LabVIEW의 최신 버전에서 VI를 여는 경우, LabVIEW는 호환성 VI를 사용합니다. 호환성 VI는 LabVIEW 이전 버전의 기능과 유사하게 작동합니다. 플랫폼 특정 공유 라이브러리를 호출하기 때문에 포팅하면 깨집니다.
- 라이브러리 함수 호출 노드를 포함하는 VI — 라이브러리 호출 함수 참조와 같은 이름의 라이브러리를 찾을 수 있는 경우를 제외하고, 이 VI를 포팅하면 깨집니다.
- Windows의 .NET이나 ActiveX 함수와 macOS의 AppleEvents VI와 같은 플랫폼 특정 통신 VI.
- 하위레벨 VISA 읽기 및 하위레벨 VISA 쓰기 함수.

다른 파일 이름

플랫폼 사이에서 VI의 포팅을 쉽게 하려면, VI 파일 이름을 지을 때 주의하십시오. Windows, macOS, Linux 파일 이름은 `.vi` 확장자를 포함하여 최대 255개 문자까지 가능합니다. 다른 플랫폼에서 사용할 수 없는 문자를 파일 이름에 사용하지 마십시오. 콜론(:)은 macOS에서 사용할 수 없으며, 슬래시(/)는 Linux에서 사용할 수 없고 다음 문자들은 Windows에서 사용할 수 없습니다. \, /, :, *, ?, ", <, >, |.

혼동을 피하려면, 모든 플랫폼에 유효한 이름으로 VI를 저장하거나 VI를 LLB에 저장합니다. LLB 이름은 플랫폼 제한에 따라야 합니다. 그러나 LLB의 VI는 플랫폼에 관계없이 최대 255개 문자까지의 이름을 가질 수 있습니다. LLB가 대부분의 파일 시스템 의존성을 제거하므로, LLB는 VI를 이동하는 가장 편리한 포맷입니다.

파일 이름은 Linux에서만 대소문자를 구분합니다. 이름으로 VI를 참조하는 경우, 사용하는 곳마다 일관되게 이름을 대문자로 표시하십시오.

다른 분리자 문자

다른 플랫폼에서 사용하기 위해 VI를 개발하는 경우, 파일 이름에 \, /, :와 같은 플랫폼에 특정한 경로 구분 문자를 사용하지 마십시오. 파일 이름에 특수 문자를 사용하지 마십시오.

오. 다른 파일 시스템이 특수 문자를 다르게 해석하기 때문입니다. 예를 들어, Linux에서 숨겨진 파일은 마침표로 시작합니다.

다른 해상도와 폰트

폰트는 플랫폼에 따라 다를 수 있습니다. VI를 포팅한 후, 새 폰트를 선택하여 보기 좋은 디스플레이를 얻습니다. VI를 디자인할 때, 플랫폼 사이에서 가장 잘 맵핑되는 3가지 폰트를 고려하십시오: 어플리케이션 폰트, 시스템 폰트, 대화 상자 폰트.

어플리케이션 폰트는 컨트롤 팔레트, 함수 팔레트, 기본 도움말, 팁 상자에서 사용되는 LabVIEW 기본 폰트입니다.

시스템 폰트는 LabVIEW가 메뉴에 사용하는 기본 폰트입니다.

디지털 폰트는 대화 상자와 시스템 컨트롤에 있는 텍스트에 사용하는 LabVIEW 기본 폰트입니다.

세 가지 폰트의 기본 폰트는 플랫폼에 따라 모두 동일합니다.

- (Windows) LabVIEW는 기본적으로 Segoe UI를 사용합니다.
- (macOS) LabVIEW는 기본으로 Lucida Grande를 사용합니다.
- (Linux) LabVIEW는 기본으로 Helvetica를 사용합니다.

이 폰트 중 하나를 포함하는 VI를 다른 플랫폼으로 포팅할 때, LabVIEW는 그 폰트가 해당 플랫폼의 유사한 폰트로 맵핑되도록 합니다.

정의된 폰트를 사용하지 않는 경우, 폰트는 새 플랫폼에서 크기가 변경될 수 있습니다. 사용 가능한 폰트와 디스플레이의 해상도가 플랫폼에 따라 다르기 때문입니다. 예를 들어, macOS에서 Geneva 또는 New York을 선택하고 VI를 Windows로 포팅하는 경우, LabVIEW는 Windows에서 이 폰트 이름을 찾을 수 없으므로 폰트를 Arial로 대체합니다. 대체된 폰트는 원본과 일치하지 않을 수도 있으며, 객체가 겹칠 수 있습니다. VI를 Windows에 포팅하고 Windows가 폰트를 인식하지 못하는 경우, 새 폰트는 다르게 디스플레이될 수 있습니다.

마찬가지로, 다른 언어 시스템이 특정 폰트를 인식하지 않을 수도 있습니다. 예를 들어, 일본어 시스템은 Windows 폰트 Arial을 인식하지 않을 수도 있습니다. 어플리케이션, 대화 상자, 시스템 폰트를 사용하여 어플리케이션 현지화 후에 폰트를 다시 셋팅하지 않도록 하십시오.

텍스트 섹션에 정의된 폰트를 사용하는 경우, 해당 텍스트의 크기를 변경하지 마십시오. 폰트를 기본 크기가 아닌 다른 크기로 변경하고 VI를 다른 플랫폼으로 포팅하는 경우, LabVIEW는 폰트와 새 크기를 맞추려고 하는데, 이는 화면의 해상도에 따라 적절하지 않을 수도 있습니다.

문자 코드 유의사항

컴퓨터는 데이터를 바이트로 나타냅니다. 컴퓨터는 각 문자에 코드를 지정하여 문자를 저장합니다.

맵핑 테이블 또는 인코딩에 의하면, 가장 단순한 경우인 1 바이트는 1개의 문자에 해당합니다. 대부분의 OS는 다양한 인코딩을 지원합니다. 인코딩의 타입 중 하나인 단일 바이트 인코딩은 256개의 문자를 나타낼 수 있습니다. 또한 영어가 아닌 다른 언어의 문자를 추가로 나타내기 위해, OS는 멀티바이트 인코딩을 지원합니다. 멀티바이트 인코딩은 수 천자의 문자를 나타낼 수 있지만, 각 문자에 여러 바이트가 필요합니다. 일본어와 중국어 시스템은 멀티바이트 인코딩을 사용하여 수 천자의 문자를 나타냅니다. 또한 대부분의 모든 OS는 보다 일반적인 문자에는 더 적은 바이트를 사용하는 가변 길이 인코딩을 지원합니다. UTF-8과 같은 가변 길이 인코딩은 첫번째 128 문자에 대해서는 단일 바이트를 사용하며, 표현 가능한 문자 수를 확장하기 위해 필요한 경우 바이트를 추가합니다.



노트

멀티바이트 시스템의 어플리케이션에서, 문자열을 바이트의 배열로 변환하여 문자열을 분석하려 하지 마십시오. 두 바이트를 사용하는 문자도 있을 수 있기 때문입니다. 일부 문자열 함수는 멀티바이트 문자를 처리하지 않습니다.

미국과 서부 유럽 국가에서 가장 일반적인 8비트 인코딩은 ASCII의 확장형인 ISO Latin 1입니다.

ASCII 인코딩은 7비트 코드입니다(0부터 127까지) 여기에는 영어 대문자와 소문자, 미국 영어 문장 부호, 기본 숫자 10 개, 몇몇 컨트롤 코드가 포함됩니다. 컨트롤 코드는 텍스트 포매팅에 사용되는 디스플레이할 수 없는 문자의 세트입니다. 대부분의 ASCII 문자를 \0x21에서 \0x7F 범위에 인쇄할 수 있습니다.

여러 나라의 문자를 나타내기 위해 OS는 ASCII의 확장형인 ISO Latin 1과 같은 8비트 인코딩을 지원합니다. 하위 128 문자 코드는 ASCII와 동일하며, 상위 128 문자 코드는 각 인코딩마다 다릅니다. 다양한 국가의 문자는 상위 128 문자 코드에 속합니다. Windows, Linux용 LabVIEW는 8비트 문자 세트를 사용하지만 서로 다른 인코딩을 사용하므로 포팅 문제가 발생할 수 있습니다.

미국과 서부 유럽 국가의 경우, Windows와 Linux용 LabVIEW는 코드 페이지라고도 알려진 ISO Latin 1 인코딩을 사용합니다. 일본어, 중국어, 한국어로 현지화된 OS 시스템의 경우, LabVIEW는 서로 다른 멀티바이트 인코딩을 사용하여 이러한 각 언어의 문자를 표현합니다. LabVIEW는 OS 시스템의 로캘과 언어 설정에 기반하여 어떤 인코딩을 사용할지 결정합니다. 이러한 인코딩은 Windows, Linux에서 동일하므로 포팅이 문제가 되는 경우가 적습니다. 그러나, macOS (64비트)용 LabVIEW는 UTF-8이라는 ASCII의 가변 길이 확장형을 사용하여 모든 언어를 다 나타낼 수 있습니다.



노트 모든 OS는 UTF-8 인코딩을 지원합니다. 그러나, LabVIEW는 기존의 사용자 어플리케이션과 호환성을 유지하기 위해 Windows에서 UTF-8을 사용하지 않습니다.

VI를 macOS에서 Windows, 또는 Windows에서 macOS와 같이 한 플랫폼에서 다른 플랫폼으로 포팅할 때, 플랫폼 코드 페이지 차이 때문에 문제가 발생할 수 있습니다. 특히 상위 128 범위에서 비 ASCII 문자를 사용할 경우 문제가 발생할 수 있습니다. 예를 들어, 프랑스어에서 café라는 단어는 확장된 문자 é를 사용하며, 이 문자의 16진수 값은 Windows에서 \0xE9입니다. 6비트 macOS에서 é를 디스플레이하려면, 2 바이트의 연속인 \0xC3\0xA9를 사용합니다.

LabVIEW는 캡션, VI와 파라미터 팁 및 설명, VI 제목, 기타 프라이빗 데이터와 같은 원소들의 코드 페이지 변환을 수행합니다. 프라이빗 데이터에는 리스트박스 아이템 이름, 테이블 행과 열 헤더, 테이블 셀에 사용된 폰트, 그래프 플롯 이름과 그래프 주석 이름, 커서 이름이 포함됩니다. 그러나, LabVIEW는 라벨 또는 문자열 내용을 맵핑하지 않습니다. 라벨

과 문자열 내용은 새 플랫폼으로 포팅한 후 예상과 다르게 나타날 수 있으므로, 수동으로 수정해야 하는 경우도 있습니다. 그러므로 Windows 또는 Linux에서 macOS로, 또는 반대로 VI를 포팅하려는 경우, 라벨 또는 문자열 내용에 다른 언어 문자 사용을 삼가하십시오.

문자열은 2진 정보로 해석될 수 있으므로 LabVIEW는 문자열 내용을 맵핑하지 않습니다. GPIB 또는 시리얼 통신을 사용하는 인스트루먼트와 통신하며 인스트루먼트가 2진 문자열 \0x63\0x61\0x66\0xC3\0xA9를 기대하는 경우, 이 문자열은 64 비트 macOS 플랫폼에서 café로 나타나며, Windows 플랫폼에서는 cafÃ©로 나타납니다. 두 경우 모두 같은 2진 문자열을 인스트루먼트로 보내야 합니다. 이러한 타입의 통신 에러를 막기 위해, LabVIEW는 문자열 값에서는 문자 맵핑을 수행하지 않습니다.

라벨을 새로운 인코딩으로 다시 맵핑하면 VI가 깨지기 때문에 LabVIEW는 라벨을 맵핑하지 않습니다. 예를 들어, 이름으로 풀기 함수를 사용하는 경우, LabVIEW가 클러스터 내용의 라벨을 다시 맵핑하게 되면 컨트롤에 대한 이름 참조가 더 이상 일치하지 않기 때문에 VI가 깨지게 됩니다.

라벨 겹치기

VI를 새 플랫폼으로 이동할 때, 폰트가 새 플랫폼에서 더 크거나 작은지 여부에 따라 컨트롤과 라벨의 크기가 변경될 수 있습니다. LabVIEW는 라벨을 소속 컨트롤로부터 멀리 이동하여 겹치지 않도록 합니다. 또한 모든 라벨과 상수는 기본 텍스트 크기에 맞춤 속성을 가지고 있습니다. 라벨 또는 상수를 처음 생성할 때, 이 속성이 설정되어 객체의 경계는 포함된 모든 텍스트를 디스플레이하는데 필요한 만큼 크기 조정됩니다.

수동으로 라벨의 크기를 조정하는 경우, LabVIEW는 이 속성을 끄며 바로 가기 메뉴의 아 이템은 더 이상 확인 표시를 보이지 않습니다. 텍스트 크기에 맞춤을 꺼 놓으면, 객체의 경계는 일정하게 유지되고 LabVIEW는 포함된 텍스트를 필요한 만큼 자릅니다. 텍스트를 플랫폼 사이에서 이동할 때 LabVIEW가 텍스트를 자르지 않도록 하려면, 라벨과 상수에 이 속성을 켜두십시오. 라벨의 크기를 조정한 후, 수동으로 텍스트 크기에 맞춤을 켤 수 있습니다.

모든 컴퓨터에 고해상도 모니터가 있는 것은 아닙니다. 그러나, 고해상도 모니터를 가지고 있는 경우, 프런트패널을 올바르게 포팅하려면 프런트패널을 너무 크게 하지 마십시오. 가장 좋은 포팅 결과를 얻으려면, 컨트롤 사이에 약간의 스페이스를 두고 컨트롤이 겹치지 않도록 하십시오. 다른 라벨과 겹치는 라벨을 포팅할 때, 새 플랫폼에서 폰트를 확대하면 라벨이 전체 컨트롤과 오버랩될 수 있습니다.

그림 차이

가장 기본적인 그림 타입은 그림의 각 픽셀의 색을 지정하는 일련의 값인 비트맵을 포함합니다. 가장 복잡한 그림은 컴퓨터 시스템이 그림을 디스플레이할 때마다 실행하는 명령을 포함할 수 있습니다.

그래픽 어플리케이션의 페인트 레이어를 사용하여 비트맵 기반 그림을 생성합니다. 비트맵은 모든 플랫폼에서 그림의 일반적인 저장 포맷입니다. 프런트패널에 비트맵을 포함한 그림을 사용하는 경우, 일반적으로 다른 플랫폼에서 VI를 로드해도 그 그림은 같은 모양으로 나타납니다. 그러나 채우기와 같은 그리기 명령을 포함하는 그림은 다른 플랫폼이 지원하지 않는 명령을 포함할 수 있습니다. VI를 다른 플랫폼에서 사용하려는 경우 VI가 다른 플랫폼에서 어떻게 보이는지 확인하십시오.

그래픽 어플리케이션의 그리기 레이어를 사용하여 그리기 명령을 포함하는 그림을 생성합니다. 그림의 포팅을 쉽게 하려면, 그림을 VI로 반입하기 전에 최종 그림을 그래픽 어플리케이션의 페인트 레이어에 붙여넣습니다.

(Windows와 macOS) Windows의 일부 그래픽 어플리케이션과 macOS의 많은 그래픽 어플리케이션에서, 직사각형이 아닌 모양으로 그림을 자르거나 복사할 수 있습니다. 예를 들어, macOS에서 Lasso 도구를 사용하여 원, 삼각형, 또는 음표와 같은 더 복잡한 모양의 아웃라인을 선택할 수 있습니다.

다른 플랫폼은 직사각형 하얀 배경에 이런 불규칙한 모양을 그릴 수 있습니다. Windows에서 불규칙한 모양의 그림과 스케일이 잘 되는 그림을 그리려는 경우, 강화된 메타파일을 지원하는 어플리케이션을 사용합니다.

VI 현지화하기

태그된 파일로 **VI 문자열을 반출**한 후 파일을 번역하고 다시 VI로 반입하여 VI를 현지화할 수 있습니다. LabVIEW의 현지화된 버전은 영어 버전과 같은 실행 프로그램을 사용합니다. 따라서, 현지화된 버전은 영어 버전과 호환 가능합니다.

VI 문자열 파일로도 알려진 태그된 텍스트 파일은 VI 제목, 설명, 컨트롤 캡션, 컨트롤 설명, 기타 컨트롤의 프라이빗 데이터를 현지화하는 정보를 포함합니다. 컨트롤 라벨은 현지화할 수 없으므로, 컨트롤은 캡션도 가지고 있습니다. VI 문자열을 텍스트 파일로 반출하여, LabVIEW는 각 컨트롤에 캡션을 생성하며 프런트패널에서 라벨을 캡션으로 대체합니다.

프런트패널의 문자열을 번역하는 것 이외에, 숫자를 문자열로 변환할 때 현지화된 소수 점을 사용할 수 있습니다. [날짜/시간 문자열로 포맷] 함수를 사용하여 날짜와 시간을 디스플레이하는 방법을 지정합니다.

다음 VI 문자열을 현지화할 수 있습니다:

- VI 윈도우 제목과 설명
- 객체 캡션과 설명
- 독립 라벨
- 기본 데이터(문자열, 테이블, 경로, 배열 기본 데이터)
- 프라이빗 데이터(리스트박스 아이템 이름, 테이블 행과 열 헤더, 그래프 플롯 이름, 그래프 커서 이름, 그래프 주석 이름)
- 지정하는 경우 블록다이어그램 문자열(독립 라벨과 문자열 상수)



노트

블록다이어그램의 문자열 상수를 현지화하면 VI가 깨질 수도 있습니다. 특히 블록다이어그램 문자열 상수에 멀티바이트 문자가 있으면, VI가 깨지게 될 가능성이 큼니다.

VI 문자열 파일의 구문

VI 문자열 파일의 포맷은 HTML 파일과 비슷합니다. 시스템은 모든 원소를 시작 태그와 끝 태그로 표시합니다. 시작 태그는 <로 시작하여 >로 끝납니다. 끝 태그는 </로 시작하여 >로 끝납니다. LabVIEW는 텍스트 내에서를 제외하고 화이트 스페이스 문자를 무시합니다. <라는 문자가 태그의 시작을 나타내기 때문에, LabVIEW는 <<를 사용하여 텍스트에서 보다 작음을 표시합니다. >를 대신하여, LabVIEW는 >>를 사용하여 보다 큼을 표시합니다. LabVIEW는 큰 따옴표를 ""로 대체합니다. 또한, LabVIEW는 라인 끝 문자를 <CR>, <CRLF>, 또는 <LF>로 표시하는데, 이는 각각 캐리지 리턴, 캐리지 리턴과 그 다음의 라인 피드, 라인 피드로 취급됩니다. VI 문자열 파일의 포맷은 컴퓨터에서 읽을 수 있도록 만들어졌습니다. 따라서 읽기 어려운 경우에도 염려하지 마십시오. 반출 파일의 각 라인은 포 라이팅 혹은 기본 데이터에 해당하는 두 개의 태그 이상을 포함하지 않습니다. 또한, LabVIEW는 각 포함된 레벨의 경우, 아이템을 들여쓰기합니다. 태그를 변경하거나 삭제하는 경우, LabVIEW는 파일을 VI로 반입할 때 에러를 발생시킵니다. 반출된 VI 문자열을 포함하는 텍스트 파일의 예는 [반출된 VI 문자열의 예](#) 항목을 참조하십시오.

다음 테이블은 VI 태그 타입과 대응하는 VI 태그 구문을 나열합니다.

VI 태그 타입	VI 태그 구문
[VI 문자열 파일].	<VI [vi attributes] > [vi 정보] </VI>
[VI 속성].	syntaxVersion=5 LVversion=nnn revision=nnn name="텍스트"
[VI 정보].	[vi 제목] [설명] [내용] [배경내용]
[VI 제목].	<TITLE>텍스트</TITLE> . <TITLE><NO_TITLE></TITLE>.
[설명].	<DESC>텍스트</DESC>.
[내용].	<CONTENT>[그룹][객체]</CONTENT>
[배경 내용].	<BDCONTENT>[배경 객체] </BDCONTENT>

스페이스는 VI 속성을 구분합니다. 속성 이름과 다음 등호 사이, 등호와 속성 값 사이에는 스페이스가 허용되지 않습니다.

예를 들면:

```
<VI syntaxVersion=5 LVversion=4502007 revision=10 name="A0
Generate Waveform.vi">
```

```
<TITLE>A0 Generate Waveform.vi</TITLE>
```

```
<DESC>이 VI는 지정된 업데이트 속도로 주어진 출력 채널에 대해 단순 버퍼
Timed 웨이브폼을 생성합니다.</DESC>
```

```
<CONTENT>
```

```
.....
```

```
</CONTENT>
```

<VI>

다음 테이블은 프런트패널의 내용(독립과 고유 라벨, 캡션 라벨, 속성)을 설명하는 태그와 이에 상응하는 태그 구문을 나열합니다.

내용 태그 타입	내용 태그 구문
[내용]	<CONTENT>[그룹][객체]</CONTENT>
[그룹]	<GROUPER>[부분]</GROUPER>
[객체]	([컨트롤])[라벨] *
[컨트롤]	<CONTROL [control attributes]> [컨트롤 정보] </CONTROL>
[라벨]	<LABEL>[스타일 텍스트] </LABEL>
[스타일 텍스트]	<STEXT>폰트 정보를 가진 텍스트 </STEXT>

<STEXT>와 </STEXT> 사이에 폰트 스펙을 타이핑할 수 있습니다. LabVIEW는 다음 포맷을 사용하여 폰트 정보를 인코딩합니다: 폰트 속성은 순서와 관계 없이 나열할 수 있습니다. 폰트 스펙은 끝 태그가 없으므로 다른 원소와 다릅니다. 예를 들어, 다음과 같이 캡션을 텍스트 "굵은체 라벨"로 설명할 수 있습니다.

```
<LABEL><STEXT><FONT name="times new roman" size=12
style='B'>Bold <FONT style='I'>label</STEXT></LABEL>
```

폰트를 사전 정의로 정의하여 사전에 정의된 폰트 중 하나를 지정할 수 있습니다—어플리케이션 폰트, 시스템 폰트, 또는 대화 상자 폰트.

<GROUPER></GROUPER> 태그는 함께 그룹화한 프런트패널 객체를 둘러쌉니다. 또한, 이 태그는 분리자 막대로 설정한 각 구획을 둘러쌉니다. 프런트패널이 언제나 적어도 하나 이상의 구획을 포함하므로, VI 문자열 파일은 항상 적어도 한 세트 이상의 <GROUPER></GROUPER> 태그를 포함합니다. 또한, 클러스터도 구획을 포함합니다.

다음 테이블은 데이터를 포함하는 객체를 설명하는 태그와 대응하는 태그 구문을 나열합니다.

내용 태그 타입	내용 태그 구문
[컨트롤]	<CONTROL [control attributes]> [컨트롤 정보] </CONTROL>
[컨트롤 속성]	ID=xxx type="불리언" name="스위치"
[컨트롤 정보]	[설명] [팁 상자] [부분] [프라이빗 데이터 섹션] [기본 데이터 섹션] [내용]
[팁 상자]	<TIP>텍스트</TIP>
[부분]	<PARTS> [부분]*</PARTS>

내용 태그 타입	내용 태그 구문
[부분]	<PART [part attributes]> [부분 정보] </PART>
[부분 속성]	partID=nnn partOrder=nnn
[부분 정보]	[컨트롤] [라벨] [여러 라벨]

다음은 "링" 캡션과 Load, Unload, Open 및 Close 옵션을 가진 링 컨트롤 설명의 예제입니다.

```
<CONTROL ID=87 type="Ring" name="RINGcontrol">

<DESC>링 컨트롤</DESC>

<PARTS>

<PART ID=12 order=0

type="Ring Text"><MLABEL><STRINGS><STRING>Load<
STRING><STRING>Unload<STRING><STRING>Open<
STRING><STRING>Close<STRING></STRINGS></MLABEL></PART>

<PART ID=82 order=0 type="Caption"><LABEL><STEXT><FONT
color=FF0033 size=12>Ring<STEXT></LABEL></PART>

</PARTS>

</CONTROL>
```

LabVIEW는 위의 MLABEL (여러 라벨) 태그를 사용하여 옵션 문자열을 링 컨트롤 또는 4가지 각 상태에 대한 문자열인 불리언 버튼의 문자열에 지정합니다. 다음은 MLABEL 태그 구문의 일반적인 설명입니다:

```
[여러 라벨] <MLABEL> [여러 라벨 정보] </MLABEL>

[여러 라벨 정보] [폰트] [문자열]
```

다음 테이블은 문자열, 테이블, 배열, 경로의 기본 데이터를 설명하는 태그와 대응하는 태그 구문을 나열합니다.

내용 태그 타입	내용 태그 구문
[기본 데이터 섹션]	<DEFAULT> [기본 데이터] </DEFAULT>
[기본 데이터]	[문자열 기본][테이블 기본][배열 데이터][경로 데이터]
[문자열 기본]	[문자열]<SAME_AS_TEXT>

내용 태그 타입	내용 태그 구문
[테이블 기본]	[문자열]
[배열 데이터]	<ARRAY nElems=n> [배열 원소 데이터] </ARRAY>
[배열 원소 데이터]	[클러스터 데이터][문자열 데이터][비 문자열 데이터]
[문자열 데이터]	[문자열]
[비 문자열 데이터]	<NON_STRING>
[클러스터 데이터]	<CLUSTER nElems=n> [클러스터 원소 데이터] </CLUSTER>
[클러스터 원소 데이터]	[클러스터 데이터][문자열 데이터][비 문자열 데이터][배열 데이터][경로 데이터]
[경로 데이터]	<PATH type ="absolute"> a<SEP> 시스템 </PATH>

[배열 데이터]의 경우, n [배열 원소 데이터]는 <ARRAY> 태그 다음에 와야 합니다. 마찬가지로 [클러스터 데이터]의 경우, n [클러스터 원소 데이터]는 <CLUSTER> 태그 다음에 와야 합니다.

문자열 컨트롤 기본 데이터 [문자열 기본]의 경우, 특별한 태그인 <SAME_AS_LABEL>를 사용합니다. 이 태그는 문자열 기본 데이터가 문자열 부분 리스트의 텍스트 라벨과 같음을 나타냅니다. <SAME_AS_LABEL>을 사용할 때, 텍스트 라벨과 문자열 기본 데이터 모두에 같은 텍스트를 타이핑할 필요가 없습니다.

경로 컨트롤 기본 데이터 [경로 데이터]의 경우, <PATH> 시작 태그는 경로 타입을 지정하는 속성을 가질 수 있습니다. 가능한 속성 값은 "absolute", "relative", "not-a-path" 및 "unc"입니다. <SEP> 태그는 <PATH>와 </PATH> 태그 사이에 오는 경로 부분을 구분합니다. 예를 들면, Windows 플랫폼에서 절대 경로 c:\windows\temp\temp.txt는 다음과 같이 씁니다:

```
<PATH type="absolute">c<SEP>windows<SEP>temp<SEP>temp.txt<
PATH>
```

다음 테이블은 프라이빗 데이터를 설명하는 태그와 이에 상응하는 태그 구문을 나열합니다.

내용 태그 타입	내용 태그 구문
[프라이빗 데이터 섹션]	<PRIV> [프라이빗 데이터]* </PRIV>
[프라이빗 데이터]	([문자열][열 헤더][행 헤더][셀 폰트][폴롯][커서][경로 프라이빗 데이터][탭 컨트롤 프라이빗 데이터])
[문자열]	<STRINGS> [문자열]* </STRINGS>
[문자열]	<STRING> 텍스트 </STRING>
[열 헤더]	<COL_HEADER> [문자열]* </COL_HEADER>

내용 태그 타입	내용 태그 구문
[행 헤더]	<ROW_HEADER> [문자열]* </ROW_HEADER>
[셀 폰트]	<CELL_FONTS> [셀 폰트]* </CELL_FONTS>
[플롯]	<PLOTS> [문자열]* </PLOTS>
[커서]	<CURSORS> [문자열]* </CURSORS>
[셀 폰트]	[행 수 열 수][폰트]
[폰트]	
[경로 프라이빗 데이터]	<PROMPT>텍스트</PROMPT>
	<MTCH_PTN>텍스트</MTCH_PTN>
	<STRT_PTH>[경로 데이터]</STRT_PTH>
[탭 컨트롤 프라이빗 데이터]	<PAGE_CAPTIONS>[문자열]*</PAGE_CAPTIONS>
[탭 컨트롤 페이지]	<PAGE> [설명] [팁 상자] [객체] </PAGE>

내용 태그 타입	내용 태그 구문
[배경 내용]	<BDCONTENT>[배경 객체] </BDCONTENT>
[배경 객체]	([컨트롤])[라벨][노드])*
[노드]	<NODE [node attributes]>[노드 정보]</NODE>
[노드 속성]	ID=xxx type="시퀀스"
[노드 정보]	[설명][배경 내용]

VI 윈도우 제목 편집하기

VI 윈도우 제목을 사용자 정의하여 VI 파일 이름보다 더 상세하게 설명할 수 있으며, 이는 현지화된 VI에서 중요합니다. VI 윈도우 제목은 현지화할 수 있으며, 파일 시스템 이름짓기 제한을 받지 않으면서도 윈도우를 호출하는 VI가 인식할 수 있습니다. 그 이유는 VI 호출 시 제목이 아니라 파일 이름으로 subVI를 참조하기 때문입니다. **반출/반입 도구**를 사용하여 윈도우 제목을 현지화하거나, VI를 편집할 때 VI 프로퍼티 대화 상자의 윈도우 모양 페이지를 사용하여 수동으로 VI 윈도우 제목을 변경합니다. 또한, VI 서버를 사용하여 프로그램적으로 VI 윈도우 제목을 변경할 수 있습니다.

마침표와 쉼표 소수점

기본적으로, LabVIEW는 사용자의 OS 환경에 맞는 적당한 소수점을 찾아 사용하며, 이는 컨트롤내의 숫자 디스플레이와 숫자와 문자열 사이를 변환하는 함수의 작동에 영향을 미칩니다. 이렇게 되면 숫자를 문자열로 인스트루먼트에 전달하거나 인스트루먼트 출력 문자열을 분석할 때 문제가 발생합니다.

다음 함수는 기본으로 OS 셋팅을 사용합니다. 마침표를 사용하려면 거짓값을 시스템 소수점 사용(+) 이라고 불리는 불리언 입력에 연결해야 합니다.

- 소수점/지수 문자열을 숫자로
- 숫자를 소수점형 문자열로
- 숫자를 지수형 문자열로
- 숫자를 공학 문자열로

포맷 문자열을 받는 함수의 경우, 포맷 코드 %x는 해당 특정 호출에 대한 소수점을 지정하는데 사용될 수 있습니다. 여기서 x는 소수점 문자를 나타냅니다. 해당되는 함수는 다음과 같습니다:

- 문자열로 포맷
- 문자열로부터 스캔
- 값을 문자로 포맷
- 값 스캔
- 배열을 스프레드시트 문자열로
- 스프레드시트 문자열을 배열로
- 구분된 스프레드시트 읽기

도구 >> 옵션을 선택한 후 항목 리스트에서 프론트패널을 선택하고 현지화된 소수점 사용 확인란에 확인 표시를 하여 소수점 또는 마침표에 시스템 셋팅(로컬)을 사용할 것인지 지정합니다.

날짜와 시간 값을 문자열로 디스플레이하기

기본으로 [날짜/시간 문자열로 포맷] 함수를 사용하여 문자열의 날짜와 시간 디스플레이 포맷을 설정할 수 있습니다. LabVIEW는 OS 셋팅을 읽고 현재 로컬에 맞는 날짜/시간 형을 디스플레이합니다. LabVIEW는 중국어와 한국어와 같이 단축된 이름을 지원하지 않는 시스템을 위해 단축된 요일과 달 이름을 숫자값으로 반환합니다. 바로 가기 메뉴에서 디스플레이 포맷을 선택하여 디스플레이를 사용자 정의할 수 있습니다.

컨트롤의 시간과 날짜 포맷

정확성이 높은 절대 시간을 사용하고, 보고, 저장하려면 **타임스탬프** 컨트롤을 사용하십시오. 바로 가기 메뉴에서 디스플레이 포맷을 선택한 후 디스플레이 포맷 페이지를 설정하여 사용자 고정 포맷을 선택하거나 시스템(로컬 특정) 포맷을 사용할 수 있습니다. 또한, 숫자 컨트롤을 사용하여 타임스탬프 값을 디스플레이할 수 있지만 숫자 컨트롤은 상대적인 양을 가지고 있습니다. 타임스탬프 컨트롤은 절대적인 양을 가지고 있습니다.

디버깅 기법

VI가 깨지지 않는 이상 예상치 않은 데이터를 얻었을 경우 다음 몇 가지 기술을 사용하여 VI 또는 블록 다이어그램 데이터 흐름의 문제점을 확인하고 수정할 수 있습니다. 다음의 기술을 사용하여 VI를 디버그할 수 없다면 해당 VI에 **경합 조건**이 발생했을 수도 있습니다.



노트

독립 어플리케이션이나 공유 라이브러리를 디버그하려면, 먼저 시작 VI의 블록 다이어그램에 다음의 디버깅 기술을 적용하십시오.

에러 클러스터

대부분의 내장 VI와 함수에는 **에러 입력 및 에러 출력 파라미터**가 있습니다. 이 파라미터들은 블록 다이어그램의 각 노드에서 접할 수 있는 에러를 찾고, 에러가 있으면 어디에서 에러가 발생했는지 보여줍니다. 이 매개변수는 사용자가 만든 VI에서도 사용 가능합니다.

VI와 함수의 에러 파라미터를 연결할 때, 에러 입력 및 에러 출력 클러스터는 다음과 같은 정보 구성요소를 제공합니다:

- 상태는 에러가 발생했을 때 참을 보고하는 불리언 값입니다.
- 코드는 에러를 숫자로 표현하는 32비트 부호있는 정수입니다. 상태에서 거짓 (FALSE)이 보고되고 에러 코드가 0이 아닌 경우, 에러가 아닌 경고를 표시.
- 소스는 어디에서 에러가 발생했는지 나타내는 문자열입니다.

경고

경고가 있어도 VI를 실행할 수는 있지만, 예상하지 못한 동작을 일으킬 수 있습니다. VI를 디버깅할 때는 반드시 **경고를 디스플레이**하십시오. VI에 경고가 나타나는 경우, 에러 리스트 윈도우를 참조하여 에러의 원인을 찾아내고 VI를 수정합니다.

실행 하이라이트

다음과 같이 실행 하이라이트 버튼을 클릭하여 블록 다이어그램 실행의 애니메이션을 봅니다.



실행 하이라이트는 블록 다이어그램의 한 노드에서 다른 노드까지의 데이터 이동을 와이어를 따라 움직이는 방울을 사용하여 보여줍니다. 실행 하이라이트를 단계별 실행과 함께 사용하여 데이터 값이 VI를 통하여 어떻게 노드와 노드 사이를 움직이는지 확인합니다.

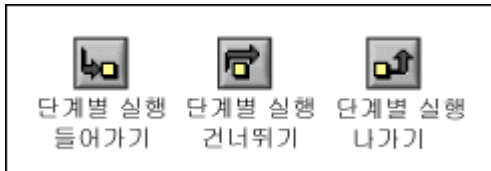


노트 실행 하이라이트는 VI의 실행 속도를 현저히 저하시킵니다. 만약 VI의 실행이 예상보다 느린 경우, SubVI의 실행 하이라이트를 끄는지 확인합니다.

에러 출력 클러스터에 에러가 발생하는 경우, 이 에러 값은 에러 출력의 옆에 빨간 경계로 나타납니다. 에러가 없다면 녹색 경계가 있는 에러 출력 옆에 [확인]이 나타납니다.

단계별 실행

VI를 단계별로 실행하여 블록다이어그램에서 VI의 각 동작을 확인할 수 있습니다. 다음과 같은 단계별 실행 버튼은 단계별 실행 모드에서의 VI 또는 subVI 실행에만 영향을 줍니다.



블록다이어그램의 도구 모음에서 단계별 실행 들어가기 또는 단계별 실행 건너뛰기 버튼을 클릭하여 단계별 실행 모드로 들어갑니다. 커서를 단계별 실행 건너뛰기, 단계별 실행 들어가기, 단계별 실행 나가기 버튼 위에 놓으면, 해당 버튼을 클릭할 때 다음 단계를 설명하는 팁 상자가 보입니다. SubVI를 단계별로 실행하거나 또는 일반적인 방식으로 실행할 수 있습니다.

VI를 단계별로 실행하면, 노드가 깜박이면서 실행할 준비가 되었음을 나타냅니다. 실행 하이라이트를 켜고 VI를 단계별로 실행하는 경우, 다음과 같은 실행 문양이 현재 실행 중인 subVI의 아이콘에 나타납니다.



프로브 도구

다음과 같이 프로브 도구를 사용하여 VI를 실행할 때 와이어에서 중간 값을 점검합니다.



정확하지 않은 데이터를 반환할 수 있는 일련의 작업을 거치는 복잡한 블록다이어그램의 경우, 프로브 도구를 사용합니다. 프로브 도구를 실행 하이라이트, 단계별 실행, 브레이크포인트와 함께 사용하여 올바르게 작동하는 데이터의 유무와 위치를 확인할 수 있습니다. 데이터가 있는 경우에는, 실행 하이라이트 또는 단계별 실행이 진행 중이거나 브레이크포인트에서 일시 정지했을 때 프로브가 즉시 업데이트되고 프로브 관찰 윈도우에 데이터를 디스플레이합니다. 또한 단계별 실행 또는 브레이크포인트로 인해 노드에서 일시 정지 상태일 때, 방금 실행된 와이어를 프로브하면 해당 와이어를 따라 진행 중인 값을 볼 수 있습니다.

프로브의 타입

일반 프로브를 사용하거나, 데이터 확인을 위한 컨트롤 팔레트의 인디케이터를 사용하거나, 제공된 프로브를 사용하거나, 사용자 정의된 제공된 프로브를 사용하거나, 새 프로브를 생성하여 VI가 실행되는 동안 와이어에서 중간값을 점검할 수 있습니다.

일반

일반 프로브를 사용하여 와이어를 통과하는 데이터를 봅니다. 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 사용자 프로브» 일반 프로브를 선택하여 일반 프로브를 사용합니다.

일반 프로브는 데이터를 디스플레이합니다. 일반 프로브를 설정하여 데이터에 응답할 수 없습니다.

이미 해당 데이터 타입에 사용자 또는 제공된 프로브를 지정한 경우를 제외하고, LabVIEW는 와이어에서 마우스 오른쪽 버튼을 클릭한 후 프로브를 선택할 때 일반 프로브를 디스플레이합니다.

VI와 유사한 방식으로 사용자 프로브를 디버깅할 수 있습니다. 하지만, 프로브는 자신의 블록다이어그램뿐만 아니라 그 SubVI의 블록다이어그램도 프로브할 수 없습니다. 프로브를 디버그할 때 일반 프로브를 사용합니다.

인디케이터를 사용하여 데이터 보기

또한 인디케이터를 사용하여 와이어를 통과하는 데이터를 볼 수 있습니다. 예를 들어 숫자 데이터를 보는 경우, 프로브 내의 차트를 사용하여 데이터를 볼 수 있습니다. 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 사용자 프로브» 컨트롤을 선택한 후, 사용하려는 인디케이터를 선택합니다. 또한 컨트롤 팔레트의 컨트롤 선택 아이콘을 클릭한 후 컴퓨터나 서버의 공유 디렉토리에 저장된 사용자 컨트롤 또는 타입 정의를 선택할 수 있습니다. 타입 정의를 사용하여 프로브된 데이터를 보는 경우, LabVIEW는 타입 정의를 사용자 컨트롤처럼 취급합니다.

선택한 인디케이터의 데이터 타입이 마우스 오른쪽 버튼으로 클릭한 와이어의 데이터 타입과 다를 경우, LabVIEW는 이 인디케이터를 와이어에 놓지 않습니다.

제공된 프로브

제공된 프로브는 와이어를 통과하는 데이터의 포괄적인 정보를 디스플레이하는 VI입니다. 예를 들어 VI 참조 번호 프로브는 VI 이름, VI 경로, 참조의 16진수 값에 대한 정보를 반환합니다. 또한 제공된 프로브를 사용하여 와이어를 통과하는 데이터에 기초하여 응답할 수 있습니다. 예를 들어, 에러 클러스터의 에러 프로브를 사용하여 에러 상태, 코드, 소스, 설명을 받으며 에러나 경고가 발생한 경우 조건적 브레이크포인트를 설정할 것인지 지정합니다.

제공된 프로브는 사용자 프로브 바로 가기 메뉴의 상단에 나타납니다. 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 사용자 프로브를 선택한 다음 제공된 프로브를 선택합니다. 마우스 오른쪽 버튼을 클릭한 와이어의 데이터 타입과 일치하는 프로브만 바로 가기 메뉴에 나타납니다.

사용자

새 프로브 생성 대화 상자를 사용하여 기존의 프로브를 기반으로 프로브를 만들거나 새 프로브를 생성할 수 있습니다. 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 사용자 정의 프로브» 새로 만들기를 선택하여 새 프로브 생성 대화 상자를 디스플레이합니다. LabVIEW가 와이어를 따라 흐르는 데이터를 프로브하는 방법을 더 많이

컨트롤하려는 경우 프로브를 생성하십시오. 새 프로브를 생성할 때, 프로브의 데이터 타입은 마우스 오른쪽 버튼으로 클릭한 와이어의 데이터 타입과 같습니다. 생성한 프로브를 편집하려는 경우, 프로브를 저장한 디렉토리에서 해당 프로브를 열어야 합니다.

사용자 프로브 바로 가기 메뉴에서 프로브를 선택한 후, 컨트롤 선택 팔레트 옵션을 사용하여 탐색하거나 새 프로브 생성 대화 상자를 사용하여 새 프로브를 생성합니다. 새 프로브는 해당 데이터 타입의 기본 프로브가 되며, LabVIEW는 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로브를 선택했을 때 이 프로브를 로드합니다. LabVIEW는 마우스 오른쪽 버튼을 클릭한 와이어의 데이터 타입과 정확히 일치하는 프로브만을 로드합니다. 즉, LabVIEW가 데이터 변환을 할 수 있지만, 배정도 부동소수 프로브가 32비트 부호없는 정수 와이어를 프로브할 수는 없습니다.



노트 사용자 프로브가 특정 데이터 타입의 기본 프로브가 되도록 하려는 경우, `user.lib\_probes\default` 디렉토리에 프로브를 저장합니다. 업그레이드를 하거나 다시 설치할 때 LabVIEW는 이 파일을 덮어쓰므로 `vi.lib\_probes` 디렉토리에 프로브를 저장하지 마십시오.

샘플링 프로브

(FPGA Module) 호스트 VI 또는 FPGA VI에서 샘플링 프로브를 사용하여 VI가 실행되는 동안 와이어의 중간값을 확인하고 Single-Cycle Timed 루프의 신호를 디버깅할 때와 같이 시간에 따른 신호 데이터의 변경 사항을 확인합니다.

호스트 VI에서 샘플링 프로브를 사용하는 경우, 먼저 프로브의 샘플링 소스를 지정해야 합니다. 샘플링 소스는 연관된 LabVIEW가 샘플이나 데이터를 읽을 시기를 결정합니다. 샘플링 프로브를 호스트 VI에서 사용하는지 또는 FPGA VI에서 사용하는지 여부에 따라 While 루프, For 루프, 또는 FPGA 시뮬레이션 시간을 샘플링 소스로 사용할 수 있습니다.

브레이크포인트

다음에서 보이는 대로 브레이크포인트 도구를 사용하여 VI, 노드, 와이어에 브레이크포인트를 놓아 해당 위치에서 실행을 일시 중지할 수 있습니다.



와이어에 브레이크포인트를 설정할 때, 실행은 데이터가 와이어를 통과한 후에 일시 정지하고 일시 정지 버튼이 빨간색으로 나타납니다. 블록다이어그램에 브레이크포인트를 놓아서 블록다이어그램의 모든 노드가 실행된 후에 실행을 일시 중지합니다. 블록다이어그램 경계가 빨간색으로 나타나고 깜빡이면서 브레이크포인트의 배치를 반영합니다.

브레이크포인트에서 VI가 일시 중지하면, LabVIEW는 블록다이어그램을 앞으로 가져와서 선택 표시를 사용하여 브레이크포인트를 포함하는 노드, 와이어, 스크립트 라인을 하이라이트합니다. 커서를 기존의 브레이크포인트 위로 이동할 때, 브레이크포인트 도구 커서의 검은 영역은 하얀색으로 나타납니다.

실행 도중 브레이크포인트에 도달하면, VI는 일시 정지하며 일시 정지 버튼은 빨간색으로 나타납니다. 또한, VI의 배경과 경계가 깜박이기 시작합니다. 이 경우 다음과 같은 작업이 가능합니다.

- 단계별 실행 버튼을 사용하여 단계별로 실행.
- VI 실행 이전에 와이어에 놓은 **프로브**에서 중간값을 확인합니다.
- **와이어 값 유지** 옵션을 활성화한 경우, VI를 실행한 후에 놓은 프로브에서 중간값 확인.
- 프런트패널 컨트롤의 값을 변경합니다.
- **호출 리스트** 폴다운 메뉴를 확인하여 브레이크포인트에서 정지한 VI를 호출하는 VI의 리스트를 볼 수 있습니다.
- 일시 정지 버튼을 클릭하여 다음 브레이크포인트나 VI가 실행을 마칠 때까지 계속 실행합니다.

LabVIEW는 VI와 함께 브레이크포인트를 저장하지만 브레이크포인트는 VI를 실행할 때만 활성화됩니다. 브레이크포인트를 제거하고 다시 생성하기보다, 브레이크포인트를 저장하여 반복적으로 사용하는 것이 유용할 수도 있습니다. 그러나, VI를 실행할 때마다 모든 브레이크포인트가 활성화되기를 원하지 않는 경우도 있습니다. 브레이크포인트를 비활성화하여 VI를 실행할 때 해당 브레이크포인트에서 실행이 중지되지 않도록 합니다. 브레이크포인트 관리자 윈도우를 사용하여 기존의 브레이크포인트를 비활성화 및 활성화하거나, 지우기, 위치 찾기를 할 수 있습니다. 보기»브레이크포인트 관리자를 선택하거나, 블록다이어그램에 있는 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 브레이크포인트»브레이크포인트 관리자를 선택하여 브레이크포인트 관리자를 엽니다.

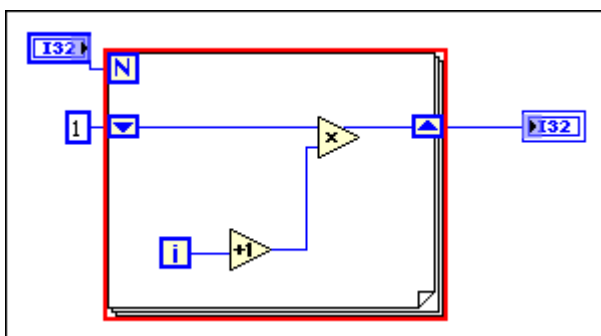
개별 혹은 전체 VI 계층구조에서 **브레이크포인트를 제거**할 수 있습니다.

LabVIEW가 브레이크포인트를 하이라이트하는 방법

사용자가 브레이크포인트를 블록다이어그램의 어느 위치에 놓았는지에 따라 LabVIEW가 브레이크포인트를 하이라이트하는 방법과 실행이 일시 정지하는 때가 다릅니다. 구조, 노드, 와이어에 브레이크포인트를 놓을 수 있습니다.

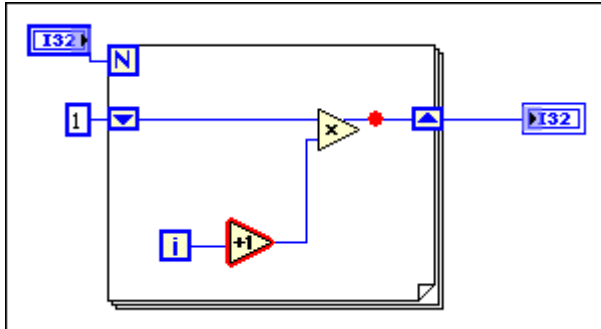
구조 브레이크포인트

다음 그림은 브레이크포인트를 가진 For 루프를 보여줍니다. 전체 For 루프가 실행을 끝내기 전에 VI를 멈춥니다. 블록다이어그램 주변의 붉은색 경계선이 브레이크포인트를 하이라이트합니다. 블록다이어그램이 어떤 구조 안에 있다면, 그 구조는 붉은색 경계선을 포함합니다. 블록다이어그램의 모든 노드가 실행을 끝마칠 때 실행 일시 정지가 일어납니다. 브레이크포인트가 루프 구조 내에 있으면, 각 루프 반복 후에 일시 정지하게 됩니다.



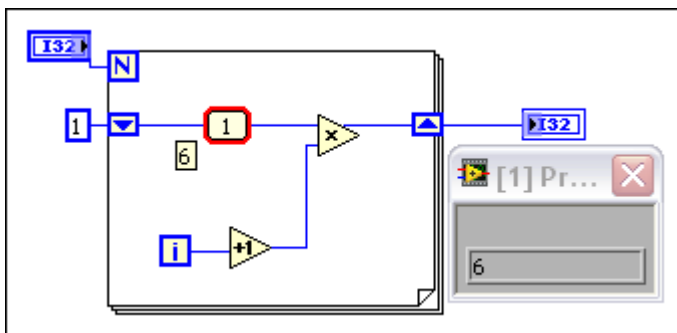
노드 브레이크포인트

다음 그림은 [증가] 노드에서 브레이크포인트를 보여줍니다. VI는 [증가] 함수가 실행하기 전에 일시 정지합니다. 또한, 한 와이어에도 브레이크포인트가 있습니다. VI는 [곱하기] 함수가 실행한 후에 다시 한번 일시 정지합니다. 노드 주변의 붉은색 경계선이 브레이크포인트를 하이라이트합니다. 실행 일시 정지는 노드가 실행하기 직전에 일어납니다. 프로브 도구를 사용하여 노드로 들어오는 모든 입력 신호를 감지할 수 있습니다.



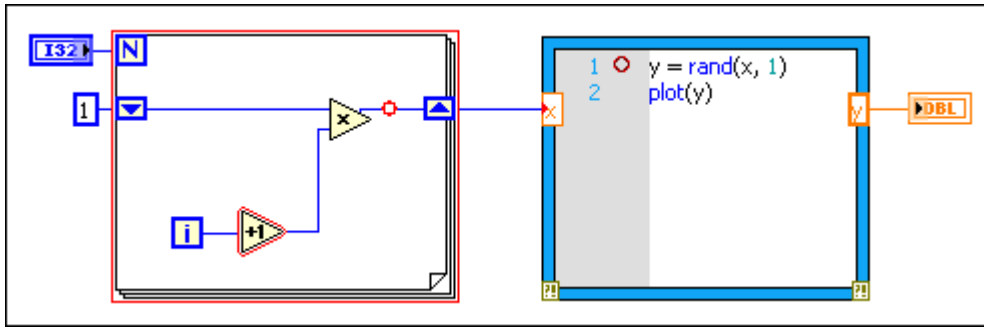
와이어 브레이크포인트

다음 그림은 프로브가 첨부된 와이어에서 브레이크포인트를 보여줍니다. **와이어 값 유지**를 선택하고 와이어 위에 커서를 놓을 때, 다음과 같이 와이어의 데이터 값을 포함한 팁 상자가 나타납니다. 데이터가 와이어를 통해 전달되고 프로브가 현재 반복의 데이터를 디스플레이한 후에 실행 일시 정지가 일어납니다. 와이어 중간의 붉은색 경계가 브레이크포인트를 하이라이트합니다. 프로브를 와이어에 붙여 놓으면, 프로브도 붉은색 경계를 가지게 됩니다.



비활성화된 브레이크포인트

다음 그림은 For 루프와 증가 노드의 브레이크포인트를 보여줍니다. 모든 브레이크포인트는 비활성화 되어있습니다. VI는 어떤 브레이크포인트 상에서도 정지하지 않습니다. 블록 다이어그램 객체를 둘러싼 빨간색 경계는 표준 경계보다 더 가늘어서, 해당 브레이크포인트가 비활성화 되어있음을 표시합니다. 브레이크포인트를 활성화시키면, 빨간색 경계가 두꺼워지며 이는 이전 설명에서 보이는 일반적인 경계와 같은 두께입니다.



실행 일시 정지하기

컨트롤과 인디케이터의 값을 바꾸거나, 호출자로 돌아가기 전에 subVI의 실행 횟수를 조절하거나, subVI 실행을 처음으로 돌리기 위해 subVI의 **실행을 일시 정지**합니다. SubVI의 모든 호출이 실행이 일시 정지된 상태로 시작하게 하거나 SubVI의 특정한 호출을 일시 정지할 수 있습니다.

SubVI의 모든 호출을 일시 정지하려면, SubVI를 열고 **수행>> 호출 시 정지**를 선택합니다. 다른 VI가 그 SubVI를 호출하면 SubVI는 자동적으로 일시 정지합니다. 단계별 실행에서 이 메뉴 아이템을 선택한 경우, SubVI는 즉시 일시 정지하지 않습니다. SubVI는 호출될 때 일시 정지합니다.

특정한 SubVI 호출을 일시 정지하려면, 블록 다이어그램의 SubVI 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 SubVI 노드 설정을 선택합니다. 호출 시 정지 확인란에 확인 표시를 하여 SubVI의 해당 인스턴스에서만 실행을 정지합니다.

보기>>VI 계층구조를 선택하여 디스플레이한 VI 계층구조 윈도우는 VI가 멈추거나 일시 정지하는지 여부를 나타냅니다. 다음과 같은 화살표 문양은 VI가 정상적으로 실행되거나 단계별로 실행되고 있음을 나타냅니다.



다음과 같은 정지 문양은 VI가 멈추거나 일시 정지 중임을 나타냅니다.



녹색의 정지 문양이나 흑백의 빈 문양은 VI가 호출될 때 멈추는 것을 나타냅니다. 빨간색 정지 문양이나 흑백의 채워진 문양은 현재 VI가 멈춰 있음을 나타냅니다. 다음과 같은 느낌표 문양은 SubVI가 일시 정지했음을 나타냅니다.



VI는 동시에 일시 정지하고 멈출 수 있습니다.

SubVI의 현재 인스턴스 결정하기

SubVI를 멈추었을 때, 도구 모음의 호출 리스트 풀다운 메뉴에 최상위 VI에서 해당 SubVI 까지의 호출자 체인이 나열됩니다. 이 리스트는 VI의 실행 여부와 상관없이 모든 호출 VI를 나열하는 탐색>>이 VI의 호출자를 선택했을 때 나타나는 리스트와 같지 않습니다. 불

록다이어그램에 한 개 이상의 인스턴스가 있을 경우 호출 리스트 메뉴를 사용하여 SubVI의 현재 인스턴스를 확인합니다. 호출 리스트 메뉴에서 한 VI를 선택하면, 이 VI의 블록다이어그램이 열리고 LabVIEW는 SubVI의 현재 인스턴스를 하이라이트합니다.

또한, [호출 체인] 함수를 사용하여 현재 VI에서부터 최상위 VI까지 호출자의 체인을 볼 수 있습니다.

블록다이어그램의 섹션 디버그하기

텍스트 기반 프로그래밍 언어에서 코드의 섹션에 주석을 다는 것과 비슷하게 블록다이어그램의 섹션을 비활성화한 VI를 실행할 수 있습니다. 블록다이어그램의 일부 섹션을 비활성화하여 이 부분이 없을 경우 VI의 성능이 향상되는지 확인합니다. 다이어그램 비활성화 구조 안에 비활성화하려는 섹션을 놓습니다.

또한 코드의 섹션을 디버그해서, 문제를 분리시킬 수 있습니다. 코드의 섹션을 디버그하려면, 조건적 비활성화 구조 안에 디버그하려는 코드를 놓습니다.

계측

프로브나, 브레이크포인트 혹은 실행 하이라이트가 VI의 코드를 디버그하는데 시간이 오래 걸리면, 인스트루멘테이션 VI를 생성하십시오. 인스트루멘테이션 VI를 사용하여 VI의 실행을 관찰하고, 문제를 디버그합니다.

디버그하려는 VI에서 데이터를 읽을 수 있고, 결과에 대한 리포트를 생성하도록 인스트루멘테이션 VI를 프로그램합니다. 인스트루멘테이션 VI에서 디버그하려는 코드의 섹션을 지정하고, 디버그하려는 VI의 경로를 지정한 다음, 시간 간격을 포함하십시오.

인스트루멘테이션 VI는 블록다이어그램에서 데이터의 실행 순서를 제공해준다는 점에서 Desktop Execution Trace Toolkit과 비슷한 작용을 합니다.

객체 동작 확인하기

디버깅 도구와 더불어, 다음 기술을 사용하여 VI 또는 블록다이어그램의 데이터 흐름에서 문제를 식별하고, 해결할 수 있습니다.

- 컨트롤과 인디케이터의 형을 확인하여, 부동소수를 정수로, 또는 정수를 더 작은 데이터 형의 정수로 변환하여 오버플로우(overflow)가 발생하고 있는 것은 아닌지 점검합니다. 예를 들어, 8비트 정수만을 받는 함수에 16비트 정수를 연결할 수도 있습니다. 이 때 함수는 16비트 정수를 8비트 정수로 형 변환하는데, 데이터의 손실이 생길 가능성이 높습니다. 추가적인 정보는 [정의되지 않은 데이터 방지하기](#)를 참조하십시오.
- 함수 또는 SubVI가 전달하는 데이터가 [정의되지 않은 데이터](#)인지 확인합니다. 주로 숫자에서 이런 경우가 발생합니다. 예를 들어, VI의 어떤 지점에서 숫자를 제로로 나누면 Inf(무한대)가 반환됩니다. 그러나 다음 함수나 SubVI에서는 숫자 값을 요구할 수 있습니다.
- For 루프가 의도하지 않게 0번 실행되어 빈 배열을 생성하지는 않았는지 확인합니다. 추가적인 정보는 [실행 하이라이트](#)와 [정의되지 않은 데이터 방지하기](#)를 참조하십시오.

- 한 번 루프를 실행한 후 그 결과로 생성된 데이터를 저장하여 다음 실행에서 사용하려고 의도한 경우가 아니라면, **시프트 레지스터**를 제대로 초기화했는지 확인합니다.
- 소스와 대상 지점의 **클러스터 원소의 순서**를 점검합니다. LabVIEW는 편집과정에서 데이터 타입이 틀린 것과 클러스터 크기가 틀린 것은 찾아냅니다. 그러나 원소가 같은 데이터 타입이면 찾아내지 못합니다.
- VI에 숨겨진 SubVI가 있는지 확인합니다. 다른 노드의 바로 위에 subVI를 놓거나 그 subVI를 보이게 놓지 않고 그 크기를 축소시켜 놓아서 subVI를 의도하지 않게 가렸을 수 있습니다. VI가 사용하는 subVI의 목록과 보기>>관계 탐색>>이 VI의 SubVI 및 보기>>관계 탐색>>열지 않은 SubVI의 결과를 비교하여 추가적인 subVI가 있는지 확인합니다.
- 보기>>VI 계층구조를 선택하여 연결되지 않은 SubVI를 찾습니다. 연결하지 않은 함수와 달리, 연결하지 않은 VI는 입력을 **필수**로 설정하지 않는 한 항상 에러를 생성하지는 않습니다. 실수로 와이어하지 않은 subVI를 블록다이어그램에 놓으면, 블록다이어그램을 실행할 때 subVI도 실행됩니다. 결과적으로, 이 VI는 필요없는 여러분의 작업을 수행하는 것입니다.
- 기본 도움말 윈도우를 사용하여 블록다이어그램의 각 함수와 subVI의 기본 설정값을 확인합니다. VI나 함수는 옵션 입력이나 권장 입력에 와이어를 연결하지 않았을 때 기본값을 전달합니다. 예를 들어, 불리언 입력을 와이어하지 않으면 기본 값인 참으로 설정됩니다.

깨진 VI 수정하기

VI가 실행되지 않는다면, VI는 깨진 VI 또는 실행 불가능한 VI입니다. 생성하거나 편집하는 VI에 에러가 있으면 다음과 같이 실행 버튼이 깨져서 나타납니다.



블록다이어그램의 연결을 마쳤는데도 버튼이 여전히 깨져있는 경우, VI는 깨진 것이며 실행할 수 없습니다.

VI가 깨진 원인 찾기

경고는 VI의 실행을 막지는 않습니다. VI의 잠재적인 문제점을 피하는데 도움을 주도록 디자인되었습니다. 그러나 에러가 발생하면 VI가 깨집니다. 반드시 모든 에러를 수정한 후에 VI를 실행해야 합니다.

깨진 실행 버튼을 클릭하거나 보기>>에러 리스트를 선택하여 VI가 깨진 이유를 찾습니다. 에러 리스트 윈도우에는 모든 에러가 나열됩니다. 에러가 있는 아이템 섹션은 VI 및 프로젝트 라이브러리 등 에러가 있는 메모리 상의 모든 아이템 이름을 나열합니다. 둘 이상의 아이템이 같은 이름을 가지고 있는 경우, 이 섹션은 각 아이템에 대한 특정한 어플리케이션 인스턴스를 보여줍니다. 에러와 경고 섹션은 에러가 있는 아이템 섹션에 선택한 VI의 에러와 경고를 나열합니다. 세부사항 섹션은 에러를 설명하고 때에 따라서 에러를 수정하는 방법을 추천합니다. 도움말 버튼을 클릭하면, 해당 에러를 상세히 설명하고 단계별 에러 수정 방법이 포함된 관련 LabVIEW 도움말 항목이 열립니다.

에러 보이기 버튼을 클릭하거나 에러 설명을 더블 클릭하여, 에러를 포함하는 블록다이어그램 또는 프런트패널 영역을 하이라이트합니다.

VI에 경고가 포함되어 있고 에러 리스트 윈도우의 경고 보이기 확인란에 확인 표시를 한 경우 다음과 같이 도구 모음에 경고 버튼이 나타납니다.



도구 >> 옵션을 선택한 후 항목 리스트에서 환경을 선택하고 디버깅 섹션에서 기본적으로 에러 리스트 대화 상자에 경고 보이기 확인란에 확인 표시를 하여, LabVIEW가 항상 에러 리스트 윈도우에 경고를 보이도록 설정합니다. 에러 리스트 윈도우가 열린 상태에서 변경할 수 있으며 변경을 즉시 확인할 수 있습니다.

VI가 깨지는 일반적 원인

아래의 리스트는 편집할 때 VI가 깨지는 일반적인 원인입니다.

- 블록 다이어그램에 데이터 타입이 맞지 않거나, 끊어지고 연결되지 않아서 깨진 와이어가 있는 경우.
- **블록 다이어그램의 필수 터미널**이 연결되어 있지 않습니다.
- **SubVI**가 깨졌거나 SubVI의 아이콘을 VI의 블록 다이어그램에 놓은 후에 커넥터 팬을 편집했습니다.

에러 핸들링

가장 안정적인 VI에서도 에러가 있을 수 있습니다. 에러 확인 메커니즘이 없는 경우에는, VI가 올바르게 작동하지 않는다는 사실 밖에 모르게 됩니다. 에러 확인은 VI에서 에러가 발생하는 원인과 위치를 확인합니다.

예를 들어, 일반적으로 입력 및 출력(I/O)을 수행하려고 할 때 에러가 발생하는 상황이 있을 수 있습니다. 거의 모든 I/O 함수는 에러 정보를 반환합니다. VI, 특히 I/O 작업(파일, 시리얼, 인스트루먼트이션, 데이터 수집, 통신)용 VI에 에러 확인을 포함시키고, 효과적으로 에러를 처리하는 메커니즘을 포함시켜야 합니다.

자동 에러 핸들링

기본적으로 LabVIEW는 VI를 실행할 때 에러가 발생하면 실행을 정지하고, 에러가 발생한 SubVI 또는 함수를 하이라이트하고, 에러 대화 상자를 열어 자동으로 에러를 처리합니다.



노트

LabVIEW 런타임 엔진은 자동 에러 핸들링을 지원하지 않습니다.

자동 에러 핸들링 비활성화하기

또한 다른 에러 핸들링 방법을 사용하기 위해 자동 에러 핸들링을 비활성화시킬 수도 있습니다. 예를 들어, 블록 다이어그램의 I/O VI가 타임아웃되는 경우, 자동 에러 핸들링이 어플리케이션을 정지시키고 에러 대화 상자를 보여주는 대신 해당 VI가 일정 주기의 시

간을 기다렸다가 다시 시도하기를 원할 수 있습니다. VI의 블록다이어그램에서 사용자 정의된 에러 핸들링을 구현할 수 있습니다.

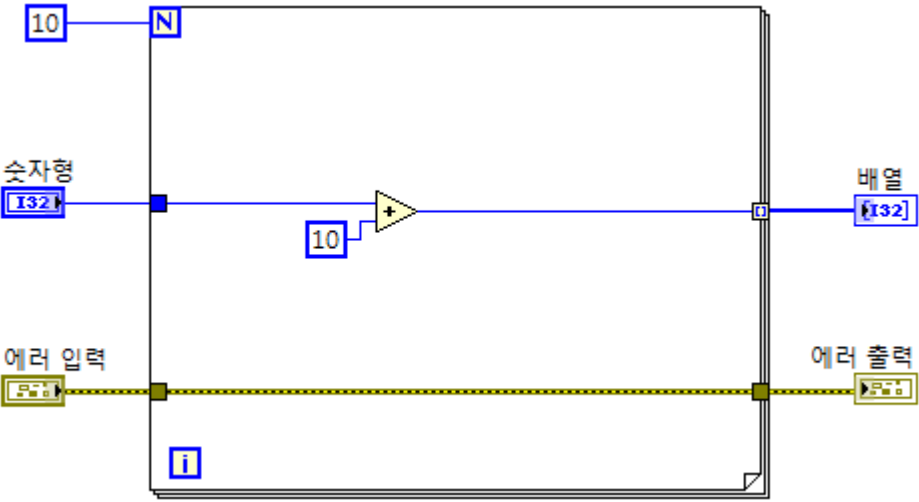
여러 종류의 VI에서 자동 에러 핸들링을 비활성화시키는 방법에 대한 정보는 다음 테이블을 참조하십시오.

VI의 타입	자동 에러 핸들링 비활성화하기
현재 VI	파일>>VI 프로퍼티를 선택합니다. 항목 풀다운 메뉴에서 실행을 선택한 후, 자동 에러 핸들링 활성화 확인란의 확인 표시를 제거합니다.
새 VI	도구>>옵션을 선택합니다. 항목 리스트에서 블록다이어그램을 선택한 후, 새 VI에서 자동 에러 핸들링 활성화 확인란의 확인 표시를 제거합니다.
VI 내부의 SubVI 또는 함수	SubVI에 에러 출력 인디케이터를 생성하거나, 해당 SubVI의 에러 출력 파라미터를 다른 SubVI나 함수의 에러 입력 입력에 연결합니다.

기타 에러 핸들링 방법

LabVIEW의 에러 핸들링은 데이터 값이 VI를 통과하는 흐름과 유사한 데이터 흐름 모델에 따릅니다. 에러 정보를 VI의 시작에서 끝까지 연결합니다. VI의 끝에 에러 핸들러 VI를 놓아 해당 VI가 에러 없이 실행되었는지 확인합니다. 각 VI의 에러 입력과 에러 출력 클러스터를 사용하여 VI를 통해 에러 정보를 전달합니다. 에러 클러스터는 일반적으로 동일한 **표준 에러 입력**과 **표준 에러 출력** 기능을 제공합니다. 에러 클러스터는 **흐름 파라미터**입니다.

예를 들어, 다음 이미지는 표준 에러 클러스터가 있는 VI를 디스플레이합니다.



VI를 실행하면 LabVIEW는 각 실행 노드에서 에러가 있는지 테스트합니다. LabVIEW가 에러를 발견하지 못하면, 노드는 정상적으로 실행됩니다. LabVIEW가 에러를 감지하면, 노드는 해당 코드 부분을 실행하지 않고 다음 노드에 에러를 전달합니다. 다음 노드도 이처럼 진행되고, 이런 방식으로 계속해서 이어집니다. 실행 흐름의 끝 부분에서 LabVIEW는 에러를 보고합니다.

에러 입력 및 에러 출력 클러스터의 원소에 대한 정보는 다음 테이블을 참조하십시오.

클러스터 원소 이름	클러스터 원소 설명
상태	에러가 발생했을 때 참을 보고하는 불리언 값입니다.
코드	에러를 숫자로 표현하는 32비트 부호있는 정수입니다. 제로가 아닌 에러 코드가 표시되고 상태에 거짓이 나타나면 에러가 아닌 경고를 나타냅니다.
소스	어디에서 에러가 발생했는지 나타내는 문자열입니다.



노트 또한 불리언 데이터를 받는 일부 VI, 함수, 구조도 에러 클러스터를 인식합니다. 예를 들어, 에러 클러스터를 불리언 함수 또는 [선택] 함수의 불리언 입력, [LabVIEW 종료] 함수, [정지] 함수 중 하나에 연결하여 논리적 연산으로 에러를 처리할 수 있습니다.

LabVIEW에서는 사용자 고유의 에러 핸들링 방법을 개발하여 구현할 수 있습니다. 대화 상자 & 사용자 인터페이스 팔레트의 LabVIEW 에러 핸들링 VI와 함수 및 대부분의 VI와 함수의 에러 입력과 에러 출력 파라미터를 사용하여 에러를 처리합니다. 예를 들어, LabVIEW가 에러를 만났을 경우 다른 종류의 대화 상자에 에러 메시지를 디스플레이할 수 있습니다.

LabVIEW 고유의 VI와 함수는 숫자형 에러 코드나 에러 클러스터로 에러를 반환합니다. 대부분의 경우 함수는 숫자 에러 코드를, VI는 에러 입력 및 출력과 함께 에러 클러스터를 사용합니다.

에러 핸들링과 함께 디버깅 도구를 사용하여 에러를 확인하고 관리할 수 있습니다.

또한 에러 클러스터를 사용하는 일반적인 방법에 대한 예는 다음 테이블을 참조하십시오.

방법	설명
에러 클러스터를 사용하여 루프 정지시키기	에러 클러스터를 While 루프 의 조건 터미널이나 조건 터미널이 있는 For 루프 에 연결하여 루프의 반복을 정지시킬 수 있습니다. 에러 클러스터를 조건 터미널에 연결하면 에러 클러스터 상태 파라미터의 참 또는 거짓값만 터미널에 전달됩니다. 에러가 발생하는 경우 루프는 정지합니다. 또한 바로 가기 메뉴 아이템도 참인 경우 정지, 참인 경우 계속해서 에러 발생 시 정지, 에러 발생 시 계속으로 변경됩니다. 조건 터미널이 있는 For 루프에서, 값을 조건 터미널에 연결하거나 입력 배열을 자동 인덱싱 하여 최대 반복 횟수를 설정해야 합니다. For 루프는 에러가 발생하거나 지정된 반복 횟수에 도달할 때까지 실행됩니다.

방법	설명
에러 상태에 따라 조건적으로 코드 실행하기	에러 클러스터를 케이스 구조 의 선택자 터미널에 연결하면, 케이스 선택자는 에러와 에러 없음이라는 2개의 케이스를 디스플레이합니다. 에러가 발생하는 경우, 케이스 구조는 에러 서브다이어그램을 실행합니다. 에러 핸들링 템플릿 VI 가 있는 SubVI를 사용하여 에러 핸들링을 위한 케이스 구조가 있는 VI를 생성합니다. 또한 케이스 구조의 경계의 색은 에러의 경우 빨간색으로, 에러 없음의 경우 녹색으로 변합니다.

에러 입력 파라미터에 대해 표준 기능 사용하기

VI 및 함수와 같은 다수의 LabVIEW 노드에는 VI의 **에러 관리**를 위해 VI에서 **에러 I/O를 설정**하는데 사용할 수 있는 에러 입력 파라미터가 포함되어 있습니다. 이 파라미터는 일반적으로 같은 표준 기능을 제공합니다. 노드가 다른 파라미터 기능을 보일 경우, 이러한 예외는 이 노드의 참조 자료에 기록됩니다.



노트

에러 핸들링 VI와 같은 일부 노드는 표준 에러 입력 기능을 제공하지 않지만, 표준 에러 입력 클러스터 원소는 포함하고 있는 에러 입력 파라미터를 가지고 있습니다.

표준 에러 입력 기능은 다음과 같습니다:

	에러 입력은 이 노드의 실행 전에 발생한 에러 조건을 설명합니다. 기본값은 에러 없음입니다. 이 노드를 실행하기 전에 에러가 발생한 경우, 노드는 에러 입력 값을 에러 출력에 전달합니다. 이 노드는 노드의 실행 전에 에러가 발생하지 않은 경우에만 정상적으로 실행됩니다. 이 노드의 실행 중 에러가 발생하면, 노드는 정상적으로 실행되고 에러 출력에서 에러 상태를 설정합니다. [단순 에러 핸들러] VI 또는 [일반 에러 핸들러] VI를 사용하여 이 에러 코드의 설명을 디스플레이합니다. 예외 컨트롤 을 사용하여 일반적인 에러를 에러 아님으로 취급하거나 경고를 에러로 취급합니다. 에러 입력과 에러 출력을 사용하여 에러를 확인하고 한 노드의 에러 출력을 다음 노드의 에러 입력에 연결하여 실행 순서를 지정하십시오.

에러 입력이 포함하는 클러스터 원소는 다음과 같습니다.

	상태는 이 노드를 실행하기 이전에 에러가 발생했을 경우에 참(X)이며, 경고를 나타내거나 이 노드를 실행하기 전에 에러가 발생하지 않았을 경우에는 거짓(확인 표시)입니다. 기본값은 거짓입니다.
	코드는 에러 또는 경고 코드입니다. 기본값은 0입니다. 상태가 참이면, 코드는 에러 코드입니다. 상태가 거짓인 경우, 코드는 0 또는 경고 코드입니다.
	소스는 에러 또는 경고의 원인을 구체적으로 나타내며, 대부분의 경우 에러 또는 경고를 발생시킨 노드의 이름입니다. 기본값은 빈 문자열입니다.

에러 출력 파라미터에 표준 기능 사용하기

VI 및 함수와 같은 다수의 LabVIEW 노드에는 VI의 **에러 관리**를 위해 VI에서 **에러 I/O를 설정**하는데 사용할 수 있는 에러 출력 파라미터가 포함되어 있습니다. 이 파라미터는 일반적으로 같은 표준 기능을 제공합니다. 노드가 다른 파라미터 기능을 보일 경우, 이러한 예외는 이 노드의 참조 자료에 기록됩니다.

표준 에러 출력 기능은 다음과 같습니다.

	에러 출력은 에러 정보를 포함합니다. 에러 입력이 이 VI 또는 함수가 실행되기 전 에러가 발생했음을 나타내면, 에러 출력도 같은 에러 정보를 포함합니다. 그렇지 않은 경우 에러 출력은 이 VI 또는 함수가 생성한 에러 상태를 나타냅니다. 에러에 대한 더 자세한 정보가 필요하면, 프런트패널의 에러 출력 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 에러 설명을 선택합니다.

에러 출력이 포함하는 클러스터 원소는 다음과 같습니다:

	상태는 에러가 발생한 경우에는 참(X)이며 에러가 발생하지 않았거나 경고를 나타낼 경우에는 거짓(확인 표시)입니다.
	코드는 에러 또는 경고 코드입니다. 상태가 참인 경우, 코드는 에러 코드입니다. 상태가 거짓인 경우, 코드는 0 또는 경고 코드입니다.
	소스는 에러 또는 경고의 원인을 구체적으로 나타내며, 대부분의 경우 에러 또는 경고를 발생시킨 노드의 이름입니다.

미리 정의된 명령 라인 작업

LabVIEW의 명령 라인 인터페이스는 다음과 같은 작업을 지원합니다.

- **MassCompile** — 지정된 디렉토리의 파일을 매스 컴파일합니다.
- **ExecuteBuildSpec** — 지정된 빌드 스펙의 셋팅을 사용하여 어플리케이션 또는 라이브러리를 빌드하고 출력 파일의 경로를 반환합니다.
- **RunVI** — 미리 정의된 커넥터 팬 인터페이스가 있는 VI를 실행하고 결과 또는 에러 정보를 반환합니다.
- **CloseLabVIEW** — 아무 입력 요청 없이 LabVIEW를 닫습니다.
- (VI Analyzer Toolkit) **RunVIAalyzer** — LabVIEW VI Analyzer Toolkit에서 지정된 VI 분석기 태스크를 실행하고 테스트 리포트를 지정된 위치에 저장합니다.
- (Unit Test Framework Toolkit) **RunUnitTests** — LabVIEW Unit Test Framework Toolkit에서 지정한 파일에 대한 테스트를 실행하고 JUnit 파일을 지정한 위치에 저장합니다.

또한 LabVIEW에서 실행되도록 **사용자 정의 작업을 생성**할 수 있습니다.

MassCompile

지정된 디렉토리의 파일을 **매스 컴파일**합니다.

다음 테이블은 이 작업에서 사용할 수 있는 인수를 보여줍니다. 필수 인수는 굵은 체로 표시됩니다.



노트

VI를 캐시하는 경우, 사용자가 최상위 VI를 컴파일할 때 LabVIEW가 캐시된 VI를 로드할 필요가 없으므로 매스 컴파일의 실행 시간을 상당히 줄일 수 있습니다. 하지만, VI 캐시는 더 많은 시스템 메모리를 소비합니다. 다른 하위 폴더에 있는 VI와 이름이 같은 VI가 포함된 폴더를 매스 컴파일하고, 이 VI가 라이브러리에 속하지 않는 경우 VI를 캐시하면 **상호 링크**가 발생할 수 있습니다.

LabVIEW의 CLI에서 이 작업을 실행하려면 다음 구문을 사용하여 명령을 실행합니다.

```
LabVIEWCLI -OperationName MassCompile -DirectoryToCompile < 파일 디렉토리 > -MassCompileLogFile < 로그 파일 경로 > -
AppendToMassCompileLog < true 또는 false > -NumOfVIsToCache <
캐시할 VI 개수 > -ReloadLVSBs < true 또는 false >
```

```
예: LabVIEWCLI -OperationName MassCompile -DirectoryToCompile
"C:\temp" -MassCompileLogFile "C:\temp\log.txt" -
AppendToMassCompileLog true -NumOfVIsToCache 0 -ReloadLVSBs
false.
```

ExecuteBuildSpec

지정된 빌드 스펙의 설정을 사용하여 어플리케이션 또는 라이브러리를 **빌드**하고 출력 파일의 경로를 반환합니다.

다음 테이블은 이 작업에서 사용할 수 있는 인수를 보여줍니다. 필수 인수는 굵은 체로 표시됩니다.

인수	설명	기본
-ProjectPath	빌드 스펙 을 포함하는 LabVIEW 프로젝트 파일 .lvproj의 경로.	—
-TargetName	빌드 스펙을 포함하는 타겟.	내 컴퓨터
-BuildSpecName	프로젝트 탐색기 윈도우에서 빌드 스펙 아래에 나타나는 빌드 스펙의 이름.	빈 문자열 — LabVIEW의 CLI는 지정된 타겟 아래에 모든 빌드 스펙을 빌드합니다.

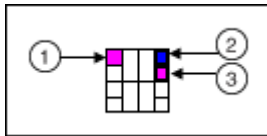
LabVIEW의 CLI에서 이 작업을 실행하려면 다음 구문을 사용하여 명령을 실행합니다.

LabVIEWCLI -OperationName ExecuteBuildSpec -ProjectPath < 프로젝트 경로 > -TargetName < 타겟 이름 > -BuildSpecName < 빌드 스펙 이름 >

예: : LabVIEWCLI -OperationName ExecuteBuildSpec -ProjectPath "C : \ temp \ test.lvproj" -TargetName "내 컴퓨터" -BuildSpecName "내 DLL".

RunVI

다음 커넥터 팬 인터페이스가 있는 VI를 실행하고 결과 또는 에러 정보를 반환합니다.



터미널	데이터 타입	설명
①		VI의 명령 라인 인수.
②		VI가 반환하는 LabVIEW의 CLI 종료 코드.
③		LabVIEW의 CLI 콘솔 또는 로그 파일에 포함시킬 출력 또는 에러 정보.

다음 테이블은 이 작업에서 사용할 수 있는 인수를 보여줍니다. 필수 인수는 굵은 체로 표시됩니다.

인수	설명	기본
-VIPath	LabVIEW에서 실행할 VI의 경로. VI의 커넥터 팬이 미리 정의된 인터페이스와 일치하지 않는 경우, LabVIEW의 CLI는 에러를 반환합니다.	—
< VI의 명령 라인 인수 >	VI의 입력 배열 요소. 원소 사이에 스페이스를 둡니다.	—

LabVIEW의 CLI에서 이 작업을 실행하려면 다음 구문을 사용하여 명령을 실행합니다.

LabVIEWCLI -OperationName RunVI -VIPath < VI 경로 > < 배열의 첫 번째 원소 > < 배열의 두 번째 원소 >...

예: LabVIEWCLI -OperationName RunVI -VIPath "C:\Users\inlabview.APAC\Documents\DemoProject\RunVI.vi" 101 202.

CloseLabVIEW

아무 입력 요청 없이 LabVIEW를 닫습니다.



노트 이 작업을 실행할 때 VI가 열려 있는 경우, 저장되지 않은 변경 사항을 잃게 됩니다. 이 작업을 실행하기 전에 모든 변경 사항을 저장하십시오.

이 작업에는 인수가 없습니다.

LabVIEW의 CLI에서 이 작업을 실행하려면 다음 명령을 실행합니다.

```
LabVIEWCLI -OperationName CloseLabVIEW
```

RunVIAalyzer

VI Analyzer Toolkit에서 지정된 VI 분석기 태스크를 실행하고 테스트 리포트를 지정한 위치에 저장합니다.

다음 테이블은 이 작업에서 사용할 수 있는 인수를 보여줍니다. 필수 인수는 굵은 체로 표시됩니다.

인수	설명	기본
-ConfigPath	분석에 사용할 VI 분석기 태스크 설정을 포함하는 설정 파일의 경로. VI 분석기 또는 VI 분석기 VI를 사용하여 저장한 설정 파일을 사용할 수 있습니다. 또는 분석할 VI, 폴더 또는 LLB를 직접 지정할 수 있습니다. 설정 파일 이외의 항목을 지정하는 경우, VI는 지정된 항목에서 모든 VI 분석기 테스트를 실행합니다.	—
-ReportPath	리포트 파일의 경로.	—
-ConfigPassword	설정 파일의 암호. (암호가 있는 경우)	빈 문자열
-ReportSaveType	리포트 또는 결과 파일의 포맷. 이 인수의 값은 다음 중 하나여야 합니다. - ASCII - HTML - RSL 파일	ASCII



노트 이 인수는 대소문자를 구분합니다.

인수	설명	기본
-ResultsPath	결과 파일의 경로. 경로가 빈 문자열인 경우, LabVIEW는 이 인수를 무시합니다.	—

LabVIEW의 CLI에서 이 작업을 실행하려면 다음 구문을 사용하여 명령을 실행합니다.

```
LabVIEWCLI -OperationName RunVIAAnalyzer -ConfigPath < 설정 파일
경로 > -ReportPath < 리포트 경로 > -ReportSaveType < 리포트 파일 타
입 > -ConfigPassword < 설정 파일 암호 >
```

```
예: LabVIEWCLI -OperationName RunVIAAnalyzer -ConfigPath
"C:\temp\test.viancfg" -ReportPath "C:\temp\output.html" -
ReportSaveType "HTML" -ConfigPassword "abc".
```

(Unit Test Framework Toolkit) RunUnitTests

LabVIEW Unit Test Framework Toolkit에서 지정한 파일에 대한 테스트를 실행하고 JUnit 파일을 지정한 위치에 저장합니다.

노트 이 작업을 실행하려면, JKI VI Package Manager (VIPM) 소프트웨어를 사용하여 UTF JUnit Report 라이브러리를 설치해야 합니다.

다음 테이블은 이 작업에서 사용할 수 있는 인수를 보여줍니다. 필수 인수는 굵은 체로 표시됩니다.

인수	설명	기본
프로젝트 파일의 경로.	—	
출력 JUnit 파일의 경로.	—	

LabVIEW의 CLI에서 이 작업을 실행하려면 다음 구문을 사용하여 명령을 실행합니다.

```
LabVIEWCLI -OperationName RunUnitTests -ProjectPath < 프로젝트
파일 경로 > -JUnitReportPath < 출력 JUnit 파일 경로 >
```

```
예: LabVIEWCLI -OperationName RunUnitTests -ProjectPath
"C:\temp\test.lvproj" -JUnitReportPath "C:\temp\test.xml".
```

RunUnitTests

LabVIEW Unit Test Framework Toolkit에서 지정한 파일에 대한 테스트를 실행하고 JUnit 파일을 지정한 위치에 저장합니다.



노트 이 작업을 실행하려면, JKI VI Package Manager (VIPM) 소프트웨어를 사용하여 UTF JUnit Report 라이브러리를 설치해야 합니다.

다음 테이블은 이 작업에서 사용할 수 있는 인수를 보여줍니다. 필수 인수는 굵은 체로 표시됩니다.

인수	설명	기본
-ProjectPath	프로젝트 파일의 경로.	—
-JUnitReportPath	출력 JUnit 파일의 경로.	—

LabVIEW의 CLI에서 이 작업을 실행하려면 다음 구문을 사용하여 명령을 실행합니다.

LabVIEWCLI -OperationName RunUnitTests -ProjectPath < 프로젝트 파일 경로 > -JUnitReportPath < 출력 JUnit 파일 경로 >

예: LabVIEWCLI -OperationName RunUnitTests -ProjectPath "C:\temp\test.lvproj" -JUnitReportPath "C:\temp\test.xml".

Windows Embedded Standard 타겟

Windows Embedded Standard 타겟 사용하기

LabVIEW 프로젝트에 타겟을 추가하여 Windows Embedded Standard 타겟에서 실행되는 LabVIEW VI를 개발할 수 있습니다. 호스트 컴퓨터 또는 호스트 VI에서 VI를 실행하는 경우, VI는 타겟에서 컴파일되고 실행됩니다. 호스트 VI는 타겟의 데이터에 접근하여 프런트패널에 데이터를 디스플레이합니다.

지원되지 않는 LabVIEW 기능

원격 개발 시, LabVIEW의 일부 기능은 Windows Embedded Standard 타겟에서 실행되는 VI에 대해 지원되지 않습니다. 지원되지 않는 기능이 있는 VI를 Windows Embedded Standard 타겟에서 실행하려고 시도하는 경우, VI가 실행될 수도 있습니다. 그러나 지원되지 않는 기능은 작동하지 않으며, 표준 LabVIEW 에러 코드를 반환할 수 있습니다.

다음은 원격 개발 시 Windows Embedded Standard 타겟에서 지원되지 않는 기능입니다.

- 서브패널 컨트롤
- 재호출 VI 디버깅
- X컨트롤

문제 해결 가이드

다음 테이블에서 LabVIEW Remote Development Target Support를 사용하고, Windows Embedded Standard 타겟에서 어플리케이션을 개발할 때 발생하는 문제의 해결책을 찾을 수 있습니다.

문제	솔루션
호스트 컴퓨터의 MAX (Measurement & Automation Explorer)에서 타겟을 찾을 수 없습니다.	호스트 컴퓨터에서, MAX는 Windows Embedded Standard 타겟을 찾는 기능을 지원하지 않습니다.
타겟 및 디바이스 추가대화 상자를 사용하여 LabVIEW Development System에서 타겟을 찾을 수 없습니다.	타겟이 원격 개발에서 디바이스를 찾는데 필요한 요구사항을 충족하도록 합니다: - 관리자 권한이 있는 사용자가 타겟에 로그인되어 있습니다. - 타겟에 LabVIEW Remote Development이 활성화되어 있습니다. (터치 패널 디바이스)타겟에서 디바이스 타입을 선택합니다.
타겟 및 디바이스대화 상자를 사용하여 타겟을 찾을 수 있지만, LabVIEW 프로젝트에서 타겟에 접속할 수 없습니다.	타겟에 있는 LabVIEW 런타임 엔진의 버전이 호스트 컴퓨터에 있는 LabVIEW Development System의 버전과 일치하도록 합니다.

문제	솔루션
호스트 컴퓨터에서 타겟에 대한 원격 개발 설정을 설정할 수 있습니까?	다음 단계에 따라, 호스트 컴퓨터에서 원격 개발 설정을 설정할 수 있습니다: 1. 호스트 컴퓨터에서 웹 브라우저를 열고 다음 URL을 입력하여 NI 웹 기반 설정 및 모니터링 어플리케이션에 접근합니다: http://localhost:3582. 2. 왼쪽에 있는 네트워크 브라우저 버튼을 클릭한 후 타겟이 포함된 행을 더블 클릭합니다. 웹 브라우저에 타겟에 특정한 NI 웹 기반 설정 및 모니터링 어플리케이션이 개별적으로 열립니다. 3. 페이지의 우측 상단에 있는 로그인버튼을 클릭합니다. 4. 사용자 이름 필드에 Admin을 입력합니다. 5. 암호는 공백으로 둡니다. 기본적으로, 암호는 설정되어 있지 않습니다. 보안을 위해 암호를 설정합니다. 6. 확인을 클릭합니다. 7. 시스템 설정페이지에서, 타겟에 대한 원격 개발 설정을 설정합니다. 8. 변경사항을 저장하려면 저장을 클릭하십시오. 9. 페이지의 우측 상단에 있는 다시 시작 버튼을 클릭하여 타겟을 다시 시작합니다. 타겟에 로그인하면 변경 사항이 나타납니다.

Windows Embedded Standard 타겟의 연결 에러 문제 해결하기

Windows Embedded Standard 타겟에 연결하는 동안 문제가 발생하는 경우, 다음과 같은 솔루션이 가능합니다. 이러한 방법을 사용하려면 타겟에 로그인하여 변경사항을 수행해야 합니다. 변경 사항을 적용하려면, 타겟을 다시 시작한 후 다시 로그인해야 합니다.

문제	솔루션
Windows 방화벽이 LabVIEW Remote Development 어플리케이션을 차단합니다.	LabVIEW Remote Development 어플리케이션이 Windows 방화벽을 통과하여 접근할 수 있도록 허용해야 합니다.
타겟에 LabVIEW Remote Development이 활성화되어 있습니다.	MAX를 시작하고 설정 트리에서 내 시스템을 선택합니다. 시스템 설정 페이지의 원격 개발 섹션에서 LabVIEW 원격 개발에 확인 표시가 되어 있는지 확인합니다.

문제	솔루션
타겟에 있는 원격 개발용 LabVIEW 런타임 엔진의 버전과 비트는 호스트 컴퓨터에 있는 LabVIEW Development System의 버전과 비트와 일치하지 않습니다.	MAX를 시작하고 설정 트리에서 내 시스템을 선택합니다. 시스템 설정 페이지의 원격 개발 섹션에서 런타임 엔진 버전을 확인합니다.
LabVIEW Remote Development 어플리케이션이 타겟에서 실행 중입니다.	Windows 작업 관리자를 시작합니다. 프로세스 페이지에서, 프로세스의 리스트에 niLVRemoteDev.exe가 포함되어 있는지 확인합니다. 리스트에 niLVRemoteDev.exe가 포함되어 있지 않은 경우, LabVIEW Remote Development을 다시 시작해야 합니다.

관련 정보

LabVIEW 프로젝트에 타겟 추가하기

Windows Embedded Standard 타겟에서 VI 실행하기

Windows Embedded Standard 타겟에 설치 프로그램 배포하기

Windows Embedded Standard 타겟에서 VI 실행하기

프로젝트 탐색기 윈도우의 Windows Embedded Standard 타겟 아래에 VI와 의존성 파일을 추가할 수 있습니다. 타겟 아래에 있는 VI를 실행하면, LabVIEW는 해당 VI와 타겟 설정을 Windows Embedded Standard 타겟에 배포합니다.

타겟을 모니터에 연결하면, VI의 프런트패널이 Windows Embedded Standard 타겟에서 실행되는 것을 볼 수 있습니다. 타겟의 프런트패널은 호스트 컴퓨터의 프런트패널을 보여줍니다. 그러나 호스트와 타겟에서 프런트패널의 컨트롤 및 모양은 다음과 같은 유의 점을 갖습니다.

- 호스트가 타겟에 접속되면, 타겟은 프런트패널을 제어할 수 없습니다. 예를 들어, 타겟에서 프런트패널의 크기를 조정하거나, 프런트패널을 최소화하거나 최대화하거나 이동할 수 없습니다. 리모트 프런트패널과 같은 다른 프로토콜을 사용하여 사용자 인터페이스를 제어하려고 해도, 시도가 거부됩니다. 타겟에 연결할 때 프런트패널의 위치로 인해 프런트패널이 잘라지지 않고 유지되도록 해야 합니다.
- 호스트가 타겟에 연결되어 있을 때에는 타겟의 프런트패널 윈도우 제목에는 외부에서 제어됨이라고 디스플레이되고, 오른쪽 아래 코너에도 워터마크로 디스플레이 됩니다.
- 호스트가 타겟과의 접속이 끊어지면, 타겟은 다시 사용자 인터페이스를 제어할 수 있게 되고 VI는 로컬 VI와 같이 작동합니다. 호스트가 다시 타겟에 접속하게 되면, 호스트는 다시 사용자 인터페이스를 제어하게 되고 컨트롤과 인디케이터를 타겟에서 생성된 값으로 업데이트합니다. 호스트는 프런트패널 윈도우 크기 및

위치와 같은 사용자 인터페이스를 타겟에서 생성된 변경 사항에 따라 업데이트하지 않습니다.

- 호스트가 타겟에 연결되고 VI가 실행을 중지하게 되면, 타겟 상의 프론트패널은 열린 상태로 있게 됩니다. 호스트 컴퓨터의 프론트패널이 닫히면, 타겟의 프론트패널도 또한 닫힙니다. VI 실행 중에 호스트 컴퓨터가 타겟과의 연결이 끊어지면, VI는 타겟에서 계속 실행됩니다.
- 몇몇 예외 사항을 제외하고, 모든 대화 상자는 어느 디바이스가 사용자 인터페이스의 컨트롤을 갖는지에 따라 호스트 컴퓨터나 타겟 중 하나에만 나타납니다. VI 기반 대화 상자의 경우, 대화 상자는 호스트 컴퓨터 및 타겟에 모두 나타납니다. 호스트 컴퓨터와 타겟은 모두 대화 상자를 디스플레이할 수 있지만, 오직 호스트 컴퓨터만 사용자 인터페이스를 제어할 수 있습니다.
- 파일 대화 상자 및 Windows나 사용자 DLL과 같이 호출 라이브러리 노드를 실행하여 나타나는 대화 상자의 경우, 대화 상자는 항상 타겟에 나타납니다. 이러한 대화 상자에 대해, 호스트 컴퓨터는 타겟에 있는 대화 상자의 작업을 알려주는 메시지를 디스플레이합니다. 호스트 컴퓨터에서의 사용자 인터페이스는 타겟 상의 대화상자와 상호 작용할 때까지 잠금 상태가 됩니다.

관련 정보

Windows Embedded Standard 타겟 사용하기

Windows Embedded Standard 타겟에 설치 프로그램 배포하기

Windows Embedded Standard 타겟에 설치 프로그램 배포하기

호스트 컴퓨터에서 VI를 개발한 후, VI를 실행 가능한 어플리케이션으로 빌드할 수 있습니다. 또한 실행 가능한 어플리케이션이나 기타 파일을 포함하는 설치 프로그램을 빌드하여, 해당 설치 프로그램을 Windows Embedded Standard 타겟에 배포 하거나 실행할 수 있습니다. 설치 프로그램을 빌드할 때에는, 어플리케이션 실행할 때 필요할 모든 구성요소를 설치 프로그램에 포함시키십시오.



노트

독립 어플리케이션 또는 설치 프로그램을 빌드하려면, 어플리케이션 빌더가 설치되어 있어야 합니다. LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. 다른 LabVIEW 개발 시스템을 사용하는 경우, National Instruments 웹사이트를 방문하여 어플리케이션 빌더를 별도로 구매할 수 있습니다.

어플리케이션이나 설치 프로그램을 빌드하려면, 반드시 먼저 빌드 스펙을 생성해야 합니다. 빌드 스펙은 포함할 파일, 생성할 디렉토리, VI의 셋팅 등 빌드에 대한 모든 셋팅을 포함합니다. 빌드 스펙은 **프로젝트 탐색기** 윈도우의 Windows Embedded Standard 타겟 아래에 있는 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 새로 만들기 >> 어플리케이션(EXE) 또는 새로 만들기 >> 설치 프로그램을 선택하여 언제든지 생성할 수 있습니다.

빌드 스펙을 생성한 후 **프로젝트 탐색기** 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 옵션을 선택합니다.

- 빌드—어플리케이션이나 설치 프로그램과 같은 빌드 스펙을 빌드합니다. 이 옵션은 설치 프로그램을 타겟에 자동으로 배포하지 않습니다.
- 배포—설치 프로그램을 빌드하고 타겟에 배포합니다. 이 옵션은 자동으로 설치 프로그램을 타겟에 실행하지 않습니다.
- 설치—설치 프로그램을 타겟에서 빌드 스펙에 지정된 위치로 빌드, 배포 및 실행합니다.

관련 정보

어플리케이션 개발하기 및 배포하기

빌드 스펙 생성하기

빌드 스펙 사용하기

LabVIEW의 블록다이어그램

블록다이어그램 디자인하기

다음 지침에 따라 블록다이어그램을 디자인합니다.

- 왼쪽에서 오른쪽, 위에서 아래의 방향으로 객체를 배치합니다. 블록다이어그램 원소의 위치가 실행 순서를 결정하지는 않지만, 블록다이어그램을 알아보기 쉽게 구성하려면 오른쪽에서 왼쪽으로 연결하는 것을 피합니다. 실행 순서를 결정하는 것은 와이어와 구조뿐입니다.
- 두 개 이상의 스크린을 넘어서는 블록다이어그램을 생성하는 것을 피합니다. 블록다이어그램이 커지고 복잡해지면, 블록다이어그램을 이해하고 디버깅하기가 힘들어질 수 있습니다.
- 일부 블록다이어그램 구성요소를 다른 VI에서 재사용할 수 있는지, 또는 블록다이어그램의 한 섹션이 같은 논리 구조로 되어있는지를 확인합니다. 그런 경우, 블록다이어그램을 특정 작업을 수행하는 SubVI로 나눕니다. SubVI를 사용하면 블록다이어그램을 수정하고 디버깅하기가 쉽습니다.
- 에러 핸들링 VI, 함수, 파라미터를 사용하여 블록다이어그램의 에러를 관리합니다.
- 효율적으로 연결하여 블록다이어그램의 모양을 향상시킵니다. 복잡한 연결 구조는 에러를 일으키지는 않으나, 블록다이어그램을 읽고 디버깅하는 것을 어렵게 하거나 VI의 작동을 잘못 표현할 수 있습니다.
- LabVIEW가 결과 와이어의 일부 선분을 숨길 수 있으므로, 구조 경계 아래 또는 겹쳐진 객체들 사이로 와이어를 연결하는 것을 피합니다.
- 와이어 위에 객체를 놓지 않습니다. 터미널 또는 아이콘을 와이어 위에 놓으면 연결이 없는 경우에도 연결이 존재하는 것처럼 보입니다.
- 라벨을 사용하여 블록다이어그램의 코드를 설명합니다.
- 조밀하게 모아서 있는 객체 사이의 공간을 늘리려면, <Ctrl>을 누른 상태에서 공간을 추가하려는 방향으로 마우스를 끌어 이동합니다. (macOS) <Option>을 누르니

다. 흩어져 있는 객체 사이의 공간을 줄이려면, <Ctrl-Alt>를 누른 상태에서 공간을 축소하려는 방향으로 마우스를 끌어 이동합니다. (macOS) <Option-Ctrl>을 누릅니다. 마우스를 끄는 동안 그와 동시에 객체가 이동합니다. 마우스를 끄는 방향이 주로 수직 또는 수평이면, 자동으로 주가 되는 방향으로 맞추어집니다.

블록다이어그램을 손쉽게 알아보고 관리하기 용이하게 만드는 방법에 대한 추가적인 팁은 LabVIEW Style Checklist를 참조하십시오.



노트

블록다이어그램에서 자동으로 기존의 와이어를 모두 재정리하고 기존의 객체를 모두 재정렬할 수 있습니다.

관련 정보

블록다이어그램 정리하기

모듈러 코드 생성하기

에러 핸들링

라벨 붙이기

LabVIEW Style Checklist

와이어를 사용하여 블록다이어그램 객체에 연결하기

와이어를 통해 블록다이어그램 객체 간에 데이터를 전달합니다. 각 와이어는 단일 데이터 소스를 갖지만, 데이터를 읽을 수 있는 여러 VI와 함수에 연결할 수 있습니다. 이는 텍스트 기반 프로그래밍 언어에서 필수 파라미터를 전달하는 것과 비슷합니다. 반드시 필수 블록다이어그램 터미널을 모두 연결해야만 합니다. 그렇지 않은 경우, VI는 **깨지고** 실행되지 않을 것입니다. 기본 도움말 윈도우를 디스플레이하여 블록다이어그램 노드에 필요한 터미널을 확인합니다. **필수 터미널**의 라벨은 기본 도움말 윈도우에 굵은체로 나타납니다.

와이어 모양과 구조

와이어는 데이터 타입에 따라 색, 스타일, 굵기가 다릅니다. 이는 터미널의 색과 기호가 대응하는 컨트롤 또는 인디케이터의 데이터 타입을 나타내는 것과 비슷합니다. 깨진 와이어는 중앙에 붉은 색의 X 표시가 있는 검정색 점선으로 나타납니다. 깨진 와이어는 호환되지 않는 두 데이터 타입을 연결하려고 할 때와 같은 다양한 이유로 발생합니다. 깨진 와이어의 빨간색 X 옆의 화살표는 데이터 흐름 방향을 나타내며, 화살표의 색은 와이어를 따라 흐르는 데이터의 데이터 타입을 나타냅니다.

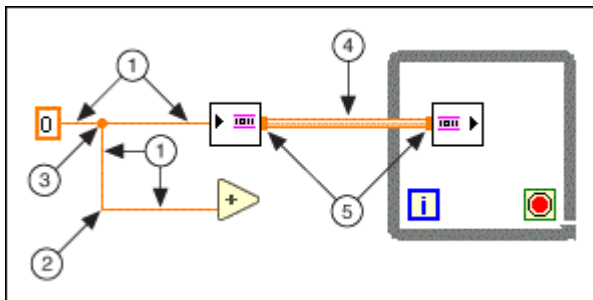
데이터 타입에 대한 추가적인 정보는 **컨트롤과 인디케이터 데이터 타입** 테이블을 참조하십시오.

와이어 표시자는 와이어링 도구를 VI 또는 함수 위로 움직일 때 연결되지 않은 터미널 옆에 나타나는 와이어의 절단부입니다. 와이어 표시자는 각 터미널의 데이터 타입을 나타냅니다. 터미널의 이름을 나열하는 팁 상자도 나타납니다. 터미널을 연결한 후에는 와이어링 도구를 노드 위로 움직여도 터미널의 와이어 표시자가 나타나지 않습니다.

와이어 선분은 와이어의 단일 수평 또는 수직 조각입니다. 와이어의 꺾임은 두 선분이 만나는 위치입니다. 둘 또는 그 이상의 와이어 선분이 만나는 지점은 접합입니다. 와이어 절가지는 접합하는 곳에서 접합하는 곳, 터미널에서 접합하는 곳, 또는 접합하는 곳이 없을 때 터미널에서 터미널까지의 모든 와이어 선분을 포함합니다.

대부분의 와이어는 한 노드에서 다른 노드로 전달되는 동기식 통신을 나타냅니다. 채널 와이어는 일종의 비동기식 통신을 나타내는 와이어입니다. 채널 와이어에는 해당 데이터 색으로 된 줄무늬가 있는데, 이 줄무늬는 가운데에서 벗어나 한쪽으로 치우쳐 있고 색이 약간 희미합니다. 채널 와이어가 구조 안으로 들어가고 나갈 때 터미널이 생기지 않습니다. 채널 와이어가 노드 터미널에 연결되는 경우 터미널의 가장자리 지점에 플랜지라고 하는 작은 블록이 생깁니다.

다음 그림은 와이어 선분, 꺾임, 접합, 채널 와이어, 플랜지를 보여줍니다.



1	선분
2	꺾임
3	접합
4	채널 와이어
5	플랜지

객체 연결하기

와이어 연결 도구를 사용하여 블록 다이어그램 노드의 터미널을 다른 블록 다이어그램 노드의 터미널에 수동으로 연결합니다. 와이어 연결 도구를 터미널 위로 가져가면, 터미널이 깜박입니다. 와이어 연결 도구를 VI 또는 함수 터미널 위로 가져갈 때, 터미널의 이름을 나열하는 팁 상자도 나타납니다. 터미널에 와이어를 연결하면 깨진 와이어가 생성될 수 있습니다. VI를 실행하기 전에 반드시 **깨진 와이어**를 수정해야만 합니다.

기본 도움말 윈도우를 사용하여 정확히 어디에 와이어를 연결할지를 결정합니다. 커서를 VI 또는 함수 위로 움직일 때, 기본 도움말 윈도우는 VI 또는 함수의 각 터미널을 보여줍니다.



노트

기본 도움말 윈도우의 옵션 터미널과 전체 경로 보이기 버튼을 클릭하여 VI 또는 함수의 옵션 터미널을 디스플레이합니다.

또한 기본 도움말 윈도우를 사용하여 객체 터미널에 어떤 데이터 타입을 연결해야 할지 알 수 있습니다. VI 또는 함수의 터미널 위로 마우스를 움직일 때, 기본 도움말 윈도우의 터미널 데이터 타입 필드는 터미널이 어떤 데이터 타입을 받는지 보여줍니다. 터미널이 여러 개의 데이터 타입을 받는 경우, 터미널 데이터 타입 필드는 해당 터미널의 기본 데이터 타입에 대해 설명합니다.



노트 데이터 타입이 서로 다른 두 객체를 연결할 때 해당 데이터 타입이 서로 유사하여 LabVIEW가 강제 변환할 수 있다면, LabVIEW는 블록 다이어그램의 노드에 **강제 변환점**을 표시하여 LabVIEW가 해당 노드에 전달된 값을 다른 형의 데이터 타입으로 강제 변환하였음을 사용자에게 알립니다. **일부 강제 변환의 경우**, 강제 변환점은 메모리 사용 및 실행 시간의 증가를 가져오고 결과의 정확도를 떨어뜨릴 수 있습니다.

와이어 꺾기

터미널을 연결하는 동안, 커서를 수평 또는 수직 방향으로 움직여서 와이어를 90도로 꺾습니다. 여러 방향으로 와이어를 꺾으려면, 마우스 버튼을 클릭하여 와이어를 고정한 다음 커서를 새로운 방향으로 움직입니다. 반복적으로 와이어를 고정한 후 새 방향으로 움직일 수 있습니다.

와이어링 취소하기

와이어를 고정한 마지막 지점에서 실행 취소를 하려면, <Shift> 키를 누른 채로 블록 다이어그램의 아무 곳이나 클릭합니다. (macOS) <Option> 키를 누르고 클릭합니다. {Linux} 마우스 가운데 버튼을 클릭합니다. 전체 와이어링 작업을 강제 종료하려면, 블록 다이어그램의 아무 곳에서 마우스 오른쪽 버튼을 클릭하거나 또는 <Esc> 키를 누릅니다.

입력이 두 개인 함수에서 수동으로 와이어를 지우고 대체하지 않고 와이어 위치를 바꾸려면 <Ctrl> 키를 누른 채로 입력 중 하나를 클릭합니다.



주의 겹치는 **와이어**는 블록 다이어그램을 혼란스럽게 하며 디버깅을 어렵게 만듭니다.

자동으로 객체 연결하기

선택한 객체를 블록 다이어그램의 다른 객체 가까이로 움직일 때, LabVIEW는 임시 와이어를 그려서 유효한 연결을 보여줍니다. 이때 마우스 버튼에서 손가락을 떼고 객체를 블록 다이어그램에 놓으면 와이어가 자동으로 연결됩니다. 또한 이미 블록 다이어그램에 있는 객체를 연결할 수도 있습니다. LabVIEW는 가장 잘 일치하는 터미널을 연결하고 일치하지 않는 터미널은 연결하지 않습니다.

위치 도구를 사용하여 객체를 움직이는 동안 스페이스 바를 눌러서 자동 와이어링을 활성화 또는 비활성화할 수 있습니다.

기본적으로, 함수 팔레트에서 객체를 선택할 때 또는 <Ctrl> 키를 누른 후 객체를 끌어서 블록다이어그램에 이미 있는 객체를 복사할 때 자동 와이어링은 활성화됩니다. 위치 도구를 사용하여 이미 블록다이어그램에 위치한 객체를 움직일 때 기본적으로 자동 와이어링은 비활성화됩니다.

(macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

도구 >> 옵션을 선택한 후 항목 리스트에서 블록다이어그램을 선택하여 자동 와이어링을 비활성화할 수 있습니다.

와이어 경로 설정하기

LabVIEW는 와이어링할 때, 와이어 경로를 자동으로 지정합니다. LabVIEW는 루프 및 구조와 같은 블록다이어그램에 있는 기존 객체의 주변으로 와이어의 경로를 정합니다. 또한, LabVIEW는 와이어의 굽어진 곳의 수를 줄이도록 경로를 정합니다. 가능하다면, 컨트롤 터미널에서 자동적으로 경로가 정해진 와이어는 터미널의 오른쪽 편으로 나오고 인디케이터 터미널에서 자동적으로 경로가 정해진 와이어는 터미널의 왼쪽으로 들어갑니다.

기존의 와이어의 경로를 자동으로 설정하려면, 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 와이어 정리를 선택합니다. 블록다이어그램에서 자동으로 **기존의 와이어를 모두 재정리하고 기존의 객체를 모두 재정렬**할 수 있습니다.

와이어링을 시작한 다음, <A> 키를 누르면 잠시 자동 와이어 경로 설정 기능이 비활성화되고 수동으로 와이어의 경로를 지정할 수 있습니다. <A> 키를 다시 누르면 와이어의 자동 와이어 경로 설정을 활성화할 수 있습니다. 와이어 연결을 끝낸 후, LabVIEW는 자동 와이어 경로 설정을 다시 활성화합니다. 또한, 와이어를 시작하거나 고정시킨 후 다른 터미널에 연결하거나 포인트를 고정하는 동안 마우스 버튼을 누른 후 마우스 버튼을 놓아서 자동 경로 설정을 임시로 비활성화할 수 있습니다. 마우스 버튼을 놓은 후, LabVIEW는 자동 경로 설정을 다시 활성화합니다.

도구 >> 옵션을 선택한 후 항목 리스트에서 블록다이어그램을 선택하고 자동 와이어 경로 설정 활성화 확인란의 확인 표시를 제거하여 모든 새 와이어의 자동 와이어 경로를 설정을 비활성화할 수 있습니다.

자동 와이어 경로 설정을 비활성화하는 경우, 와이어링 도구를 처음 움직이는 방향에 따라서 수직 또는 수평으로 터미널을 연결할 수 있습니다. 터미널의 어느 부분을 클릭하든 와이어는 터미널의 중앙으로 연결됩니다. 터미널을 클릭한 후, 스페이스바를 눌러서 수직과 수평 방향을 바꿉니다.

또한 자동 와이어 경로 설정이 활성화된 경우에도 스페이스바를 눌러서 수평과 수직 방향을 바꿀 수 있습니다. LabVIEW가 새 방향에서 와이어에 대한 경로를 찾은 경우, 와이어는 그 방향으로 바뀝니다.

와이어 선택하기

위치 도구를 사용하여 한번 클릭, 더블 클릭, 또는 트리플 클릭으로 와이어를 선택합니다. 와이어를 한번 클릭하면 와이어의 한 선분이 선택됩니다. 와이어를 더블 클릭하면 와이어 끝가지가 선택됩니다. 와이어를 트리플 클릭하면 전체 와이어가 선택됩니다.

와이어에 라벨 붙이기

고유 라벨을 사용해서 긴 와이어에 라벨을 붙이고, 용도를 설명할 수 있습니다. 와이어에 고유 라벨을 만드려면, 와이어에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 보이는 **아이템 > 라벨**을 선택합니다. 와이어에 라벨 붙이기는 특히 시프트 레지스터에서 오는 와이어나, 블록다이어그램 전면에서 걸쳐 있는 와이어에 사용하면 유용합니다. 고유 라벨은 와이어를 따라 아무 위치로 옮길 수 있습니다. 와이어에 고유 라벨을 **고정**할 수는 없습니다.

깨진 와이어 수정하기

깨진 와이어는 중앙에  붉은 색의 X 표시가 있는 검정색 점선으로 나타납니다. 깨진 와이어는 호환되지 않는 두 데이터 타입을 연결하려고 할 때와 같은 다양한 이유로 발생합니다. 와이어링 도구를 깨진 와이어로 위로 이동하여 왜 와이어가 깨졌는지를 설명하는 팁 상자를 디스플레이합니다. 이 정보는 또한 와이어링 도구를 깨진 와이어 위로 이동할 때 기본 도움말 윈도우에도 나타납니다. 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **에러 열거**를 선택하여 에러 리스트 윈도우를 디스플레이합니다. 와이어가 깨진 이유에 대한 추가적인 정보를 보려면 도움말 버튼을 클릭합니다.

위치 도구로 와이어를 트리플 클릭한 후 <Delete> 키를 눌러서 깨진 와이어를 제거합니다. 또한, 와이어에서 마우스 오른쪽 버튼을 클릭한 후 와이어 끝가지 삭제, 와이어 끝가지 생성, 연결되지 않은 끝 제거, 와이어 정리, 컨트롤로 변경, 인디케이터로 변경, 소스에서 인덱싱 활성화, 소스에서 인덱싱 비활성화와 같은 바로 가기 메뉴 옵션을 선택할 수 있습니다. 이 옵션은 깨진 와이어의 이유에 따라서 변합니다.

편집 > 깨진 와이어 제거를 선택하거나 <Ctrl-B> 키를 눌러서 모든 깨진 와이어를 제거합니다. (macOS) <Command-B> 키를 누릅니다. (Linux) <Meta-B> 키를 누르십시오. 또한 깨진 와이어가 있는 구조 또는 블록다이어그램의 한 영역을 선택하고, <Ctrl-B> 키를 눌러 선택된 영역에서만 깨진 와이어를 제거할 수도 있습니다.



주의 모든 깨진 와이어를 제거할 때에는 주의하십시오. 때때로 블록다이어그램의 연결을 끝내지 않았기 때문에 깨진 와이어가 나타나기도 합니다.

와이어의 일부를 삭제하면 와이어가 깨질 수 있습니다. 와이어 일부를 연결시켜 깨진 와이어를 수정하십시오.



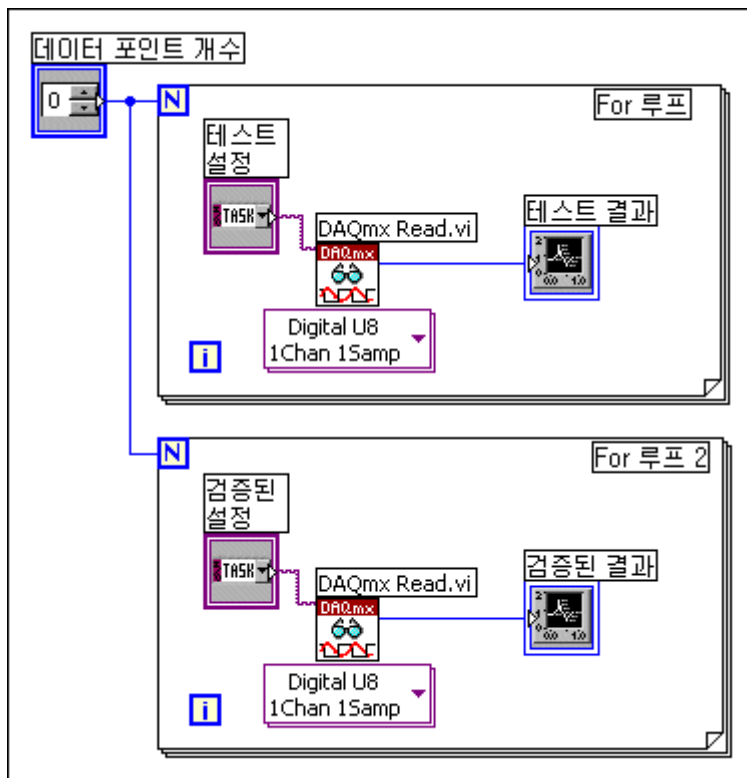
노트 **고유 라벨**을 가진 와이어의 일부를 삭제하면 라벨은 삭제되지 않고 깨진 와이어에 그대로 남아있습니다. 각각 고유 라벨이 있는 두 개의 깨진 와이어를 연결하면, 소스 터미널에 연결된 와이어의 라벨이 새 와이어의 라벨이 됩니다.

블록다이어그램의 데이터 흐름

LabVIEW는 데이터 흐름 모델에 따라 vi를 실행합니다. 블록다이어그램은 필요한 모든 입력을 받은 후에 실행됩니다. 노드가 실행되면, 출력 데이터를 생성한 후 이를 데이터 흐름 경로에 따라 다음 노드로 전달합니다. 노드를 통과하는 데이터의 흐름은 블록다이어그램에 있는 vi 및 함수의 실행 순서를 결정합니다.

Visual Basic, C++, JAVA 및 대부분의 다른 텍스트 기반 프로그래밍 언어는 프로그램 실행에 흐름 제어 모델을 따릅니다. 흐름 제어에서, 프로그램 원소의 순서는 프로그램의 실행 순서를 결정합니다.

LabVIEW에서, 명령의 순서가 아닌 데이터의 흐름이 블록다이어그램 원소의 실행 순서를 결정합니다. 그러므로, 동시 작업을 할 수 있는 블록다이어그램을 생성할 수 있습니다. 예를 들어, 다음 블록다이어그램에서와 같이 두 개의 For 루프를 동시에 실행하고 프런트패널에 결과를 디스플레이할 수 있습니다.



LabVIEW는 여러개의 실행 스레드와 vi를 동시에 실행하는 **멀티태스킹 및 멀티스레드** 시스템입니다.

데이터 의존성과 인위적인 데이터 의존성

흐름 제어 모델의 실행은 명령에 따라서 수행됩니다. 데이터 흐름은 데이터에 의해 실행되거나 데이터 의존적으로 실행됩니다. 다른 노드에서 데이터를 받는 노드는 항상 다른 노드가 실행을 완전히 끝낸 이후에 실행됩니다.

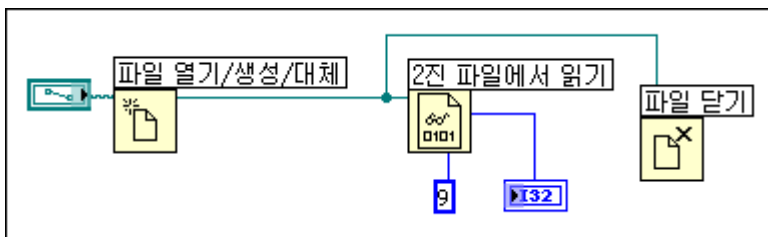
와이어가 연결되지 않은 블록다이어그램 노드는 임의의 순서로 실행될 수 있습니다. 데이터 의존성이 없을 때 **흐름 파라미터**를 사용하여 실행 순서를 컨트롤할 수 있습니다. 흐

를 파라미터를 사용할 수 없을 때 **시퀀스 구조**를 사용하여 실행 순서를 컨트롤할 수 있습니다.

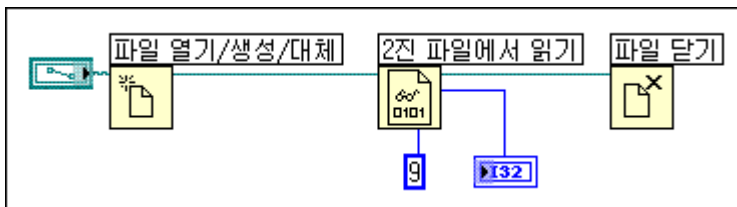
데이터 의존성 잃기

데이터 의존성이 없을 때 실행이 왼쪽에서 오른쪽, 위에서 아래쪽으로 진행된다고 가정하지 마십시오. 필요한 경우 데이터 흐름을 연결해서 이벤트의 순서를 명백히 정의합니다.

다음 블록 다이어그램에서, [2진 파일로부터 읽기] 함수가 [파일 닫기] 함수에 연결되지 않았기 때문에 [2진 파일로부터 읽기] 함수와 [파일 닫기] 함수 사이에 의존성이 없습니다. 어떤 함수가 먼저 실행될지 결정할 방법이 없기 때문에 이 예제는 의도한대로 실행되지 않을 것입니다. [파일 닫기] 함수가 먼저 실행되면, [2진 파일로부터 읽기] 함수는 작동할 수 없습니다.



다음 블록 다이어그램은 [2진 파일로부터 읽기] 함수의 출력을 [파일 닫기] 함수에 연결하여 의존성을 만듭니다. [파일 닫기] 함수는 [2진 파일로부터 읽기] 함수의 출력을 받기 전까지는 실행되지 않습니다.



흐름 파라미터

일반적으로 참조 번호 또는 **에러 클러스터**인 흐름 파라미터는 대응하는 입력 파라미터와 같은 값을 반환합니다. 이 파라미터를 사용하여 데이터 의존성이 없을 때 실행 순서를 컨트롤할 수 있습니다. 실행하려는 첫번째 노드의 흐름 출력을 실행하려는 다음으로 실행하려는 노드의 대응하는 입력에 연결하여 인위적인 데이터 의존성을 생성합니다. 이러한 흐름 파라미터 없이 원하는 순서대로 데이터 연산을 수행하려면 반드시 시퀀스 구조를 사용해야 합니다.

데이터 흐름과 메모리 관리

데이터 흐름 실행은 실행 흐름 제어 모델보다 메모리 관리가 쉽습니다. LabVIEW에서는 변수에 메모리를 할당하거나 변수에 값을 지정하지 않습니다. 대신에, 데이터 전달을 나타내는 와이어를 가진 블록 다이어그램을 생성합니다.

데이터를 생성하는 VI와 함수는 자동적으로 해당 데이터에 대한 메모리를 할당합니다. VI 또는 함수가 더 이상 데이터를 사용하지 않을 때, LabVIEW는 할당된 메모리를 해제합니다. 새 데이터를 배열 또는 문자열에 추가할 때, LabVIEW는 새 데이터를 관리할 수 있는 충분한 메모리를 할당합니다.

LabVIEW가 자동적으로 메모리를 관리하기 때문에, 언제 메모리를 할당하고 해제할지를 조절하기 어렵습니다. VI가 큰 데이터 세트를 다루는 경우, **메모리 할당**이 언제 이루어지는지 알 필요가 있습니다. 관련된 원리를 이해하는 것은 메모리의 사용이 극히 적은 VI를 만드는데 도움이 될 수 있습니다. **메모리 사용을 최소화**하는 것은 VI의 실행 속도를 높이는 데 도움이 될 수 있습니다.

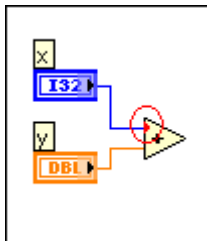
강제 변환점

강제 변환점은 블록 다이어그램 노드에 나타나며, 노드에 전달된 값을 LabVIEW가 다른 형으로 변환시켰음을 나타냅니다. LabVIEW는 변환이 일어나는 곳의 터미널 경계에 강제 변환점을 놓습니다. VI와 함수에는 많은 터미널이 있을 수 있으므로, 한 터미널에서 다른 터미널로 연결할 경우 아이콘 내에 강제 변환점이 나타날 수 있습니다.

빨간색 강제 변환점

한 숫자형 타입의 데이터를 그와 다른 숫자형 타입의 터미널에 와이어로 연결하거나, 배열이 아닌 데이터를 **배리언트** 터미널에 연결할 때 LabVIEW는 블록 다이어그램 노드에 빨간색 강제 변환점을 나타냅니다.

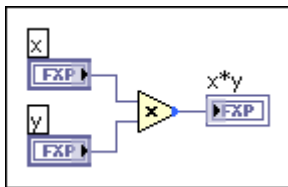
예를 들어 LabVIEW는 다음 그림에서 [더하기] 함수에 빨간색 강제 변환점을 놓는데, 이는 해당 함수는 서로 일치하는 숫자형 데이터 타입을 갖는 두 개의 입력을 요구하지만 이 함수가 배정도 부동소수와 정수라는 서로 다른 데이터 타입의 입력들을 받았기 때문입니다. 이 경우 데이터 타입이 유사하므로 LabVIEW는 정수를 강제 변환하고 더하기 작업을 수행할 수 있습니다.



파란색 강제 변환점

함수에 대한 **고정 소수점 출력 설정을 설정**할 때 LabVIEW는 파란색 강제 변환점을 숫자형 함수의 출력 터미널에 나타냅니다. 이 점은 LabVIEW가 **오버플로우 및 반올림** 에러 방지 목적으로 출력 데이터 타입을 자동 조정하지 않는다는 것을 나타냅니다.

다음 블록 다이어그램에서 파란색 강제 변환점을 볼 수 있습니다.



정확성 유지 및 메모리 사용 최적화를 위해 강제 변환점 제거하기

강제 변환점은 다음과 같은 방법으로 **메모리 사용** 또는 VI의 전체 성능에 영향을 줄 수 있습니다:

- 타입 정의 강제 변환—타입 정의의 인스턴스를 같은 타입 정의에 링크되어 있지 않은 객체에 연결하는 경우, VI는 부정확한 결과를 생성할 수 있습니다.
- 스칼라 숫자형 강제 변환—서로 다른 숫자형 타입을 조작하는 경우, LabVIEW는 데이터 타입이 일치하도록 데이터를 강제 변환합니다. 이 강제 변환이 발생할 때 VI는 정확하지 않은 결과를 생성합니다.
- 큰 배열 강제 변환—데이터 타입을 매우 큰 배열로 강제 변환하는 경우, VI는 대량의 메모리를 사용하게 되고 실행에 시간이 더 걸리게 됩니다.

데이터 타입을 일치시켜서 불필요한 강제 변환을 방지할 수 있습니다.



팁

커서를 강제 변환점이 있는 터미널 위로 이동하면, 기본 도움말 윈도우의 연결된 와이어 데이터 타입 필드는 VI 또는 함수에 연결된 와이어의 데이터 타입을 나타냅니다. 강제 변환점을 제거하려면, 연결된 와이어 데이터 타입과 터미널 데이터 타입을 일치시킵니다.

타입 정의 강제 변환

타입 정의의 인스턴스를 동일한 타입 정의에 링크되어 있지 않은 객체에 연결하는 경우, LabVIEW는 그 데이터를 받는 터미널에 강제 변환점을 놓아서 해당 타입 정의를 편집하면 VI가 깨지거나 정확하지 않은 데이터를 생성하게 된다는 경고를 나타냅니다.

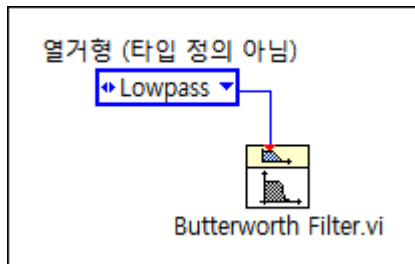


노트

타입 정의를 같은 데이터 타입의 객체에 연결하는 경우, LabVIEW는 해당 객체에 동일한 타입 정의가 필요하지 않는 한 해당 타입 정의를 강제 변환합니다. 타입 정의를 삭제하고 타입 정의가 없는 상태에서 동일한 파라미터를 객체에 연결하면 강제 변환을 피할 수 있습니다.

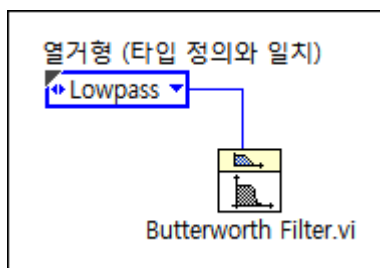
클러스터나 열거형에 연결되어 있는 타입 정의를 편집하는 경우, 클러스터나 열거형을 연결하는 와이어는 깨어지게 됩니다.

다음 블록 다이어그램은 타입 정의를 편집할 경우 결과의 정확성에 영향을 주거나 와이어를 깨뜨릴 수 있는 타입 정의 강제 변환점을 보여줍니다. 이 예에서 SubVI의 입력은 필터 타입을 정의하는 타입 정의에 링크되어 있습니다.



LabVIEW는 SubVI에서 와이어로 연결된 터미널에 강제 변환점을 놓아, 열거형 상수가 동일한 타입 정의에 링크되어 있지 않음을 나타냅니다.

다음 블록 다이어그램은 정확한 결과를 유지하고 타입 정의를 편집해도 와이어가 깨지지 않는 타입 정의의 사용을 보여줍니다.

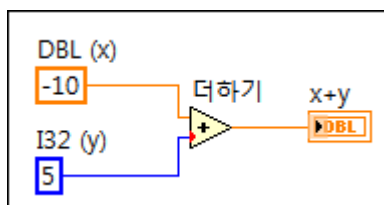


타입 정의에 링크된 객체를 **복사**하거나 타입 정의에 링크되지 않은 객체를 타입 정의의 인스턴스로 **대체**하여 SubVI의 입력의 타입 정의를 일치시킬 수 있습니다. 또한 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 **생성»상수**를 선택하여 SubVI의 입력의 타입 정의를 일치시킬 수 있습니다. 상수는 자동으로 타입 정의에 링크됩니다.

스칼라 숫자형 강제 변환

단순 스칼라 **숫자형 데이터 타입**을 조작할 경우, LabVIEW는 소수 부분의 데이터 손실이 일어나지 않도록 데이터 타입을 강제 변환합니다. 두 데이터 타입이 모두 정수인 경우, LabVIEW는 더 낮은 최대값을 갖는 데이터 타입을 더 높은 최대값을 갖는 데이터 타입으로 강제 변환합니다.

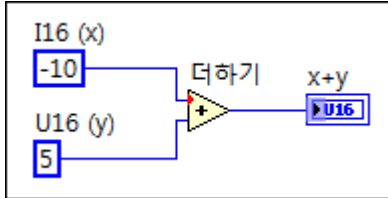
다음 블록 다이어그램은 결과의 정확성에 영향을 주지 않는 강제 변환을 보여줍니다.



이 예에서 배정도 부동소수 숫자형을 롱 정수에 더하면, LabVIEW는 롱 정수를 배정도 부동소수 숫자형으로 강제 변환하고 두 값을 더합니다. LabVIEW가 [더하기] 함수에 놓는 강제 변환점은 더 낮은 최대값을 갖는 데이터 타입을 더 정확한 데이터 타입으로 강제 변환하기 때문에 작업의 결과는 변하지 않습니다. 그러나 가능하면 데이터 타입을 일치시켜야 합니다.

LabVIEW에서 데이터를 사용자가 원하는 타입으로 강제 변환시키려면, 컨트롤, 상수 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 형을 선택하여 데이터 타입을 변경할 수 있습니다.

다음 블록다이어그램은 정확하지 않은 결과를 발생시키는 강제 변환을 보여줍니다.



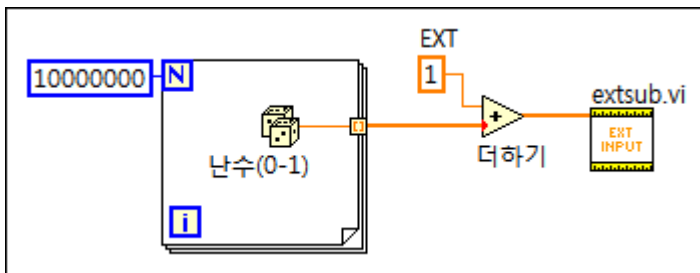
이 예에서 LabVIEW는 부호 있는 정수를 부호 없는 정수로 강제 변환합니다. LabVIEW가 음의 값을 지원하지 않는 데이터 타입으로 음수를 강제 변환하였기 때문에, LabVIEW는 음수 -5가 아닌 65531이라는 값을 생성합니다. 정확하지 않은 데이터를 생성하는 것을 방지하려면, 블록다이어그램 객체의 **형을 변경**하여 데이터 타입을 일치시킵니다.



팁 출력 인디케이터의 소스에 적용 프로퍼티를 활성화하여 인디케이터 데이터 타입이 자동으로 숫자형 입력 값의 데이터 타입으로 변경되도록 할 수 있습니다. 소스에 적용 프로퍼티를 활성화하려면, 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 소스에 적용을 선택합니다.

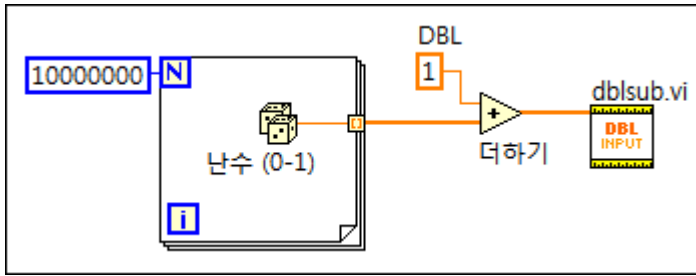
큰 배열 강제 변환

LabVIEW가 매우 큰 배열에 저장된 데이터를 강제 변환하는 경우, VI는 많은 양의 메모리를 사용하게 되고 실행에 오랜 시간이 걸리게 됩니다. 다음 블록다이어그램은 LabVIEW가 큰 배열의 데이터 타입을 SubVI의 입력의 데이터 타입으로 강제 변환하며 많은 양의 메모리를 사용하는 VI를 보여줍니다.



이 예에서 LabVIEW는 매우 큰 배열을 SubVI의 데이터 타입과 일치시키기 위해 확장형 정밀도 부동소수로 강제 변환합니다. 메모리 사용을 최소화하려면, 블록다이어그램의 컨트롤, 상수 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 형을 선택하여 데이터 타입을 일치시킵니다. 이전 블록다이어그램의 경우에서 SubVI 블록다이어그램의 상수 및 컨트롤의 데이터 타입을 바꾸면 LabVIEW가 데이터를 강제 변환해야 할 필요가 없어집니다.

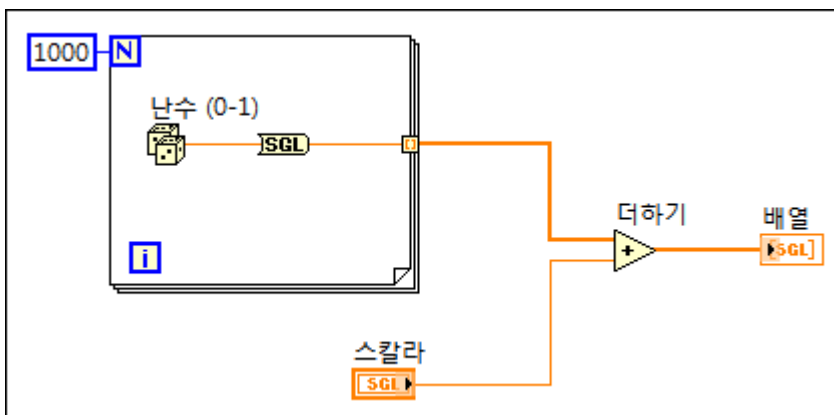
다음 블록다이어그램은 메모리를 보다 효율적으로 사용하는 VI를 보여줍니다.



팁 VI의 실행 시간과 메모리 사용에 대한 데이터를 수집하여 디스플레이하는 성능과 메모리 프로파일 윈도우를 사용하여 VI가 얼마만큼의 메모리를 사용하는지 추적 및 모니터링할 수 있습니다.

변환 함수

변환 함수를 사용하여 강제 변환점을 제거할 수 있습니다. 일반적으로 변환 함수는 강제 변환점과 동일한 효과가 있지만, 블록 다이어그램 상에서 공간을 더 많이 차지합니다. 그러나 다음 블록 다이어그램에서 보이는 것과 같이, 배열을 생성하는 루프 내부에 변환 함수를 사용하면 LabVIEW가 배열을 생성하기 전에 변환이 일어나므로 큰 데이터 버퍼의 변환을 피할 수 있습니다.



강제 변환점 색 변경하기

빨간색 강제 변환점의 색을 바꾸려면, 옵션 대화 상자의 환경 페이지를 사용합니다. 색 섹션에 있는 기본 색 사용 확인란에서 확인 표시를 제거하고 강제 변환점 색 상자를 클릭하여 다른 색을 선택합니다.

파란색 강제 변환점의 색을 변경할 수는 없습니다.

LabVIEW에서 프로젝트 생성하기

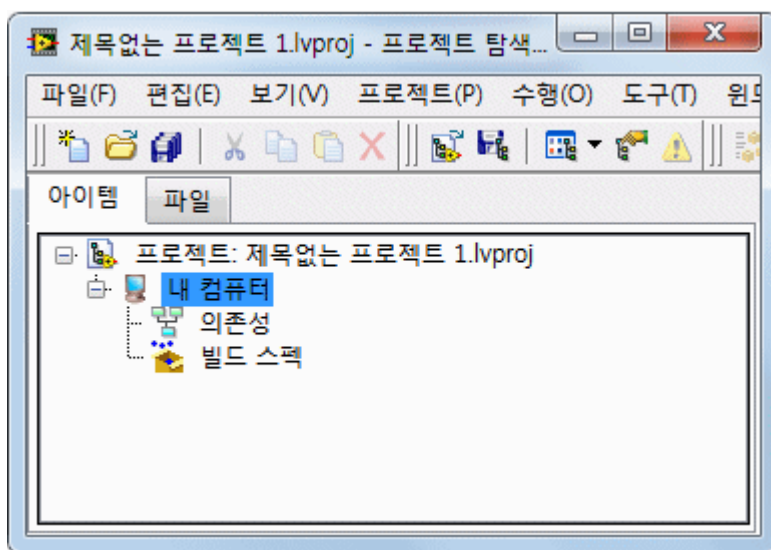
LabVIEW에서 프로젝트 관리하기

LabVIEW의 프로젝트는 VI 및 이 VI가 제대로 실행되기 위해 필요한 파일, 문서, 관련 링크와 같은 보조 파일들로 구성됩니다. 프로젝트는 프로젝트 탐색기 윈도우를 사용하여 관리할 수 있습니다. 프로젝트 탐색기 윈도우에서, **폴더**와 **라이브러리**를 사용하여 여러 아이템을 하나로 그룹화하고, **의존성**이라고 부르는 VI 계층구조 리스트를 사용하여 VI가 의존하는 아이템을 추적할 수 있습니다.

프로젝트 탐색기 윈도우 사용하기

프로젝트 탐색기 윈도우를 사용하여 프로젝트를 편집합니다. 프로젝트 탐색기 윈도우를 보려면 **새 프로젝트를 생성**해야 합니다. 또한 **파일 >> 프로젝트 열기**를 선택하여 기존의 프로젝트를 열 수 있습니다.

다음에 보이는 것처럼, 프로젝트 탐색기 윈도우에는 아이템 및 파일이라는 2개의 페이지가 있습니다.



아이템 페이지는 프로젝트 아이템을 프로젝트 트리의 형태로 보여줍니다. 파일 페이지는 디스크에 상응하는 파일이 있는 프로젝트 아이템을 디스플레이합니다. 이 페이지에서 파일 이름과 폴더를 정리할 수 있습니다. 파일 페이지에서 프로젝트를 작업하면 실제 디스크 상의 파일에 내용이 반영되고 업데이트됩니다. 타겟 아래의 폴더나 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이템 보기에서 보이기 또는 파일 보기에서 보이기를 선택하여 다른 페이지로 전환할 수 있습니다.

프로젝트 탐색기 윈도우에는 기본 설정으로 다음의 아이템이 들어 있습니다:

- 프로젝트 루트—프로젝트 탐색기 윈도우의 모든 아이템을 포함합니다. 프로젝트 루트의 이 라벨은 프로젝트의 파일 이름을 포함합니다.
 - 내 컴퓨터—로컬 컴퓨터가 프로젝트의 타겟으로 나타납니다.

- 의존성 – 다른 VI, 공유 라이브러리, LabVIEW 프로젝트 라이브러리와 같이 **타겟 아래의 VI가 필요로 하는** 아이템을 포함합니다.
- 빌드 스펙 – 소스 배포용 빌드 및 LabVIEW 툴킷과 모듈에서 사용할 수 있는 기타 타입의 빌드에 대한 설정 정보가 들어있습니다. LabVIEW Professional Development System 또는 어플리케이션 빌더를 설치한 경우, 빌드 스펙을 사용하여 **다음을 설정**할 수 있습니다.
 - 독립 어플리케이션
 - 설치 프로그램
 - .NET Interop 어셈블리
 - 묶음 라이브러리
 - 공유 라이브러리
 - 소스 배포
 - 웹 서비스
 - 압축(Zip) 파일

프로젝트 탐색기 윈도우에서 의존성 및 빌드 스펙 아이템의 **숨기기**가 가능합니다. 두 아이템 중 하나를 숨긴 후 어플리케이션이나 공유 라이브러리 빌드에서 처럼 숨긴 아이템에 다시 접근하려면 아이템이 다시 나타나도록 해야합니다.

프로젝트에 **다른 타겟을 추가**하면, LabVIEW는 프로젝트 탐색기 윈도우에 아이템을 추가로 생성하여 그 타겟을 나타냅니다. 또한, 각 타겟 노드는 의존성과 빌드 스펙을 포함합니다. 각 타겟 아래에 **파일을 추가**할 수 있습니다.

다른 열린 VI의 블록 다이어그램에 프로젝트 탐색기 윈도우의 VI를 놓을 수 있습니다. SubVI로 사용하려는 VI를 프로젝트 탐색기 윈도우에서 선택하여 다른 VI의 블록 다이어그램으로 끌어 이동합니다.

프로젝트 프로퍼티와 메소드를 사용하여 프로젝트와 프로젝트 탐색기 윈도우를 프로그램적으로 설정하고 수정할 수 있습니다.

프로젝트에서 아이템 구성하기

다음 리스트는 프로젝트에서 아이템을 구성할 때 알아야 할 유의점과 권장사항을 설명합니다.

- **정렬 옵션**을 사용해서 한 프로젝트 내에서 여러 아이템을 정렬할 수 있습니다. 정렬 옵션은 프로젝트 안의 아이템에 자동적으로 적용되며, 디스크에 있는 프로젝트의 구성은 바뀌지 않습니다. 정렬 옵션을 사용해서 프로젝트 안에서 아이템을 더 효율적으로 구성하고 관리할 수 있습니다.
- 생성한 프로젝트마다 별도의 파일 디렉토리를 생성합니다. 프로젝트 파일을 각각 별도의 디렉토리로 구성하면 디스크에서 특정한 프로젝트에 연관된 파일을 식별하기 쉽습니다.
- 디스크의 디렉토리가 **버추얼 폴더**의 프로젝트 구조와 일치하지 않습니다. 디스크의 디렉토리를 버추얼 폴더의 프로젝트에 추가한 후 디스크의 디렉토리를 변경하는 경우, LabVIEW는 프로젝트의 폴더를 업데이트하지 않습니다. 디스크의 디렉토리를 **자동 적용한 폴더**의 프로젝트에 추가하여 프로젝트의 디스크에서 변경사항을 모니터하고 업데이트합니다.

- (Windows) **설치 프로그램을 만드는 경우**, 프로젝트에 있는 파일을 .lvproj 프로젝트 파일과 같은 드라이브에 저장하도록 하십시오. 일부 파일이 네트워크 드라이브 등의 다른 드라이브에 있는 경우, 프로젝트를 설치 프로그램에 포함시키면 링크가 깨질 수 있습니다.
- 소스 배포의 파일 구조는 프로젝트 탐색기 윈도우의 구조와 일치하지 않아도 됩니다. **소스 배포를 만들 때** 다른 구조를 지정할 수 있습니다.
- **의존성**은 프로젝트에 아이템을 추가, 제거 또는 저장하면 자동으로 업데이트됩니다. 파일을 직접 의존성에 추가하거나 제거할 수 없습니다. LabVIEW는 자동으로 사용자가 프로젝트에 추가한 모든 아이템의 구조를 추적하여 아이템이 의존하는 파일이 반드시 의존성에 포함되도록 합니다.
- 프로젝트를 열면, VI가 다이나믹하게 호출하는 아이템은 의존성 아래에 나타나지 않습니다. 호출자를 실행하면, 다이나믹하게 로드된 아이템은 의존성 아래의 메모리의 아이템에 나타납니다. 이 아이템을 타겟 아래에 수동으로 추가하면 **프로젝트에서 아이템을 관리**할 수 있습니다.
- 어플리케이션을 만들 때, 셋팅을 전체 폴더에 적용할 수 있습니다. 모든 다이나믹 아이템을 타겟 아래의 폴더 안에 그룹화하는 것을 고려하십시오.
- 경로는 다르지만 정규화된 이름이 같은 아이템이 두 개 이상 프로젝트에 존재하면 프로젝트에 충돌이 있을 수 있습니다. 노란색 경고 삼각형이 충돌하는 아이템에 나타납니다. 충돌 해결 버튼을 클릭하여 프로젝트 충돌 해결 대화 상자에서 프로젝트 충돌 요약 정보를 확인하고 **프로젝트 충돌을 해결**합니다.

메모리에 프로젝트 아이템 로드하기

프로젝트를 열면, LabVIEW가 디스크상의 프로젝트 위치를 검색하여 프로젝트 트리를 채웁니다. 그런 다음 LabVIEW는 의존성 아래에 위치한 라이브러리를 포함하여 다음의 라이브러리를 메모리에 로드합니다.

- **프로젝트 라이브러리** .lvlib
- **묶음 프로젝트 라이브러리** .lvlibp
- **클래스 라이브러리** .lvclass
- **X컨트롤 라이브러리** .xctl
- Statechart 라이브러리 .lvsc

라이브러리를 로드할 때, LabVIEW는 다음과 같이 라이브러리의 멤버를 로드합니다.



노트

멤버 라이브러리를 로드할 때, LabVIEW는 멤버 라이브러리의 모든 멤버 VI를 로드합니다.

VI 계층구조 윈도우를 사용하여 메모리상의 VI를 볼 수 있습니다. 라이브러리가 VI 또는 타입 정의를 메모리에 로드하는 경우, 라이브러리는 또한 전체 VI 계층구조를 로드합니다. 다른 경우라면 포함되지 않을 VI까지 여기에 포함될 수 있습니다.

프로젝트 및 타겟 사용하기

프로젝트를 사용하여 LabVIEW 파일과 LabVIEW 전용이 아닌 파일을 하나로 그룹화하고, 파일을 타겟에 배포하거나 다운로드하며, 또한 빌드 스펙을 생성하고, 큰 어플리케이션을 관리합니다. 프로젝트를 생성하고 저장할 때, LabVIEW는 프로젝트 파일 .lvproj를 생성합니다. 이 파일에는 프로젝트의 파일 참조, 설정 정보, 배포 정보, 빌드 정보 등이 포함됩니다. LabVIEW의 소스 컨트롤을 사용하여 여러 명의 사용자 간에 파일과 프로젝트를 공유합니다.

독립 어플리케이션과 공유 라이브러리를 만들려면 반드시 프로젝트를 사용해야 합니다. 또한 Windows Embedded Standard, RT, FPGA 또는 Touch Panel 타겟으로 작업하려면 프로젝트를 사용해야 합니다. 타겟과 함께 프로젝트를 사용하는 것에 대한 추가적인 정보는 특정한 모듈 문서를 참조하십시오.

NI 디바이스 드라이버와 함께 프로젝트를 사용하는 경우, 드라이버와 함께 프로젝트를 사용하는데 대한 추가적인 정보는 특정 드라이버 문서를 참조하십시오.

LabVIEW 프로젝트 관리 관련 예제는 labview\examples\project 디렉토리를 참조하십시오.

관련 예제 찾기



관련 정보

- [LabVIEW에서 프로젝트 관리하기](#) (290 페이지 참조)
- [어플리케이션 만들기 및 배포하기](#) (341 페이지 참조)
- [프로젝트 저장하기](#) (776 페이지 참조)
- [LabVIEW에서 소스 컨트롤 사용하기](#) (336 페이지 참조)
- [LabVIEW 프로젝트의 폴더 사용하기](#) (295 페이지 참조)
- [LabVIEW 프로젝트에서 라이브러리 사용하기](#) (296 페이지 참조)
- [LabVIEW 프로젝트에서 묶음 프로젝트 라이브러리 사용하기](#) (300 페이지 참조)
- [LabVIEW 프로젝트에서 의존성 관리하기](#) (302 페이지 참조)
- [프로젝트 충돌 해결하기](#) (304 페이지 참조)
- [프로젝트에 폴더 추가하기](#) (767 페이지 참조)
- [프로젝트에 아이템 추가하기](#) (770 페이지 참조)
- [프로젝트에 타겟 추가하기\(Windows\)](#) (772 페이지 참조)
- [프로젝트에서 아이템 보이기와 숨기기](#) (777 페이지 참조)
- [프로젝트에서 파일 보기](#) (777 페이지 참조)

예제, VI 템플릿, 프로젝트 템플릿, 샘플 프로젝트

예제, VI 템플릿, 프로젝트 템플릿 및 샘플 프로젝트를 사용하여 VI를 만들고 어플리케이션을 설계할 수 있습니다.

예제

새 VI를 만들기 전에, 도움말>>예제 찾기를 선택하여 NI 예제 탐색기를 열고 필요 사항에 맞는 예제 VI가 있는지 검색해 봅니다. LabVIEW는 바로 사용하거나 사용자가 생성하는 VI에 붙여서 사용할 수 있는 수백 개의 예제 VI를 검색합니다. 어플리케이션에 맞도록 예제를 수정하거나, 하나 또는 여러 예제를 복사하여 생성한 VI에 붙여넣을 수 있습니다. 도움말>>예제 찾기를 선택하여 NI 예제 탐색기로 예제 VI를 탐색 또는 검색합니다.

또한, LabVIEW 도움말에서 특정 VI와 함수 참조 토픽의 아래에 위치한 예제 열기와 관련 예제 찾기 버튼을 사용하여 예제에 접근할 수 있습니다. 주제와 관련된 예제 VI를 열기 위해서 예제 열기 버튼을 클릭합니다. NI 예제 탐색기를 열고 관련된 예제 VI들을 보려면 관련 예제 찾기 버튼을 클릭합니다.

또한, 블록다이어그램 또는 고정된 팔레트의 VI나 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 예제를 선택하여 도움말 토픽과 해당 VI나 함수 예제의 링크를 디스플레이할 수 있습니다.

추가적인 예제 VI는 ni.com/support를 참조하십시오. (Windows) 또한 ni.com에서 예제 VI를 [LabVIEW에서 검색](#)할 수도 있습니다.

LabVIEW VI 템플릿

적당한 예제 VI를 찾을 수 없는 경우, 새로 만들기 대화 상자에서 템플릿 VI를 열고 함수 팔레트의 내장 VI와 함수로 템플릿을 채우십시오. 또한 파일>>프로젝트 생성을 선택하여 프로젝트 생성 대화 상자를 열어 사용자 정의할 수 있는 템플릿과 샘플 프로젝트를 탐색할 수 있습니다.

내장된 VI 템플릿은 일반 측정 어플리케이션을 시작할 때, 필요한 SubVI, 함수, 구조, 그리고 프론트패널 객체들을 가지고 있습니다. VI 템플릿은 [템플릿 이름] #.vi로 열립니다. 이 때, 템플릿 이름은 템플릿의 이름이고 #는 고유의 십진수 번호입니다. 파일>>새로 만들기를 선택하여 새로 만들기 대화 상자를 디스플레이합니다. 이 대화 상자에는 내장 VI 템플릿이 나열됩니다.

어플리케이션 디자인 패턴은 VI>>템플릿으로부터>>프레임워크 항목 아래의 디자인 패턴에서 사용 가능합니다. 어플리케이션 디자인 패턴은 소프트웨어 디자인 내의 특정한 문제의 해결책인 LabVIEW의 구조와 기술을 제시합니다.

사용자 VI 템플릿 생성하기

비슷한 작업을 하기 위해서 매번 프론트패널이나 블록다이어그램을 동일한 요소를 추가하는 일이 없도록 사용자 템플릿 VI를 만들 수 있습니다. VI를 만든 후 템플릿으로 저장하여 사용자 VI 템플릿을 생성합니다.

또한 템플릿을 템플릿 VI에 SubVI로 추가할 수 있습니다. 템플릿을 SubVI로 추가하려면, VI 템플릿을 열고 프론트패널 또는 블록다이어그램 윈도우의 오른쪽 위 코너에 있는 VI 아이콘을 최상위 템플릿 VI의 블록다이어그램으로 끕니다. 템플릿 SubVI를 가지는 템플릿으로부터 VI를 생성하고 이 VI를 저장하면, LabVIEW는 각 템플릿 SubVI를 VI로 저장하도록 입력요청합니다.

LabVIEW 프로젝트 템플릿과 샘플 프로젝트

LabVIEW에는 상태 머신 및 큐 메시지 핸들러와 같이 유용한 디자인 패턴을 만들 때 사용할 수 있는 여러 기본 템플릿이 있습니다. LabVIEW에는 또한 이러한 템플릿에 기반하여 만든 어플리케이션의 작동을 보여주는 샘플 프로젝트가 여럿 포함되어 있습니다. 이러한 템플릿 및 샘플 프로젝트를 어플리케이션의 목적에 따라 사용자 정의할 수 있습니다.

파일 » 프로젝트 생성을 선택하여 템플릿과 샘플 프로젝트가 나열되어 있는 프로젝트 생성 대화 상자를 디스플레이합니다.

기타 파일

새로 만들기 대화 상자의 새로 생성 리스트에서 기타 파일 항목의 아이টে를 선택하여 런타임 메뉴, 사용자 컨트롤, 글로벌 변수, 프로젝트 라이브러리를 생성합니다.

LabVIEW 프로젝트의 폴더 사용하기

LabVIEW 프로젝트에는 버추얼 폴더와 자동 적용 폴더라는 두 종류의 폴더를 추가할 수 있습니다. 버추얼 폴더란 프로젝트 아이টে를 구성하는 폴더입니다. 타겟에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 추가 » 폴더(스냅샷)를 선택하여 **버추얼 폴더를 프로젝트에 추가**합니다. 자동 적용 폴더란 실제 디스크 상에 있는 폴더의 내용을 반영하도록 실시간으로 업데이트되는 폴더입니다. 프로젝트 라이브러리의 경우처럼, 자동 적용 폴더의 내용이 디스크 내용과 정확히 일치하지 않을 수도 있습니다. 이 프로젝트의 아이টে 페이지는 디스크 구성이 아닌 라이브러리 계층구조 기준으로 프로젝트 라이브러리 .lvlib 내용을 디스플레이합니다. 예를 들어, VI가 프로젝트 라이브러리 파일에 있으면, VI가 라이브러리 파일 아래의 프로젝트에서 나타납니다. 이 VI는 디스크의 위치에 있는 프로젝트에서는 나타나지 않습니다.

자동 적용 폴더는 프로젝트 탐색기 윈도우의 아이টে 페이지에서만 볼 수 있습니다. 자동 적용 폴더의 디스크 내용은 볼 수 있지만 프로젝트 아이টে의 이름 변경, 재구성 및 제거와 같은 디스크 작업은 수행할 수 없습니다. 자동 적용 폴더에 있는 아이টে에 대한 디스크 작업을 수행하려면, 프로젝트 탐색기의 파일 페이지를 사용해야 합니다. 파일 페이지는 디스크에서의 프로젝트 폴더 위치를 디스플레이합니다. 파일 페이지에서 프로젝트 관련 작업을 수행하면, 실제 디스크 상의 폴더의 내용이 반영되고 업데이트됩니다. 마찬가지로, LabVIEW 밖에 있는 디스크의 폴더를 변경하는 경우, LabVIEW가 자동으로 프로젝트의 자동 적용 폴더를 업데이트합니다.

LabVIEW 밖에 있는 파일 시스템을 사용하지 않고 파일 페이지에서 파일 작업을 수행하면 LabVIEW가 참조하는 파일을 업데이트할 수 있는 장점이 있습니다. 파일 페이지의 아이টে를 이동, 제거 또는 이름 변경하면 LabVIEW가 참조하는 아이টে를 업데이트하여 이 같은 변경사항을 반영합니다. 또한, 프로젝트가 열려 있고 자동 적용 폴더에도 있는 파일 이름을 디스크에서 변경하면, LabVIEW가 이 변경을 감지하고 파일 이름 바꾸기 실행 취소? 대화 상자가 나타납니다. 파일 이름 변경을 취소하거나 변경을 계속할 수 있습니다. 그러나 이름 변경이 LabVIEW 밖에서 발생했기 때문에 변경하는 파일을 참조하는 프로젝트의 아이টে가 새로운 이름을 참조하지 않으므로 충돌이 발생할 수 있습니다.



노트

(macOS and Linux) LabVIEW는 macOS나 Linux OS에서 자동 적용 폴더에 있는 파일에는 기호형 링크의 사용을 지원하지 않습니다.

자동 적용 폴더는 버추얼 폴더가 라이브러리 계층구조내에 있어야만 버추얼 폴더를 포함할 수 있습니다. 버추얼 폴더는 디스크의 파일을 나타내지 않습니다. 아이템 페이지의 버추얼 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 자동 적용 폴더로 변경을 선택하여 기존의 버추얼 폴더를 자동 적용 폴더로 **변환**할 수 있습니다. 디스크에 있는 폴더 선택을 요청하는 파일 대화 상자가 나타납니다. LabVIEW가 자동으로 버추얼 폴더의 이름을 디스크에 있는 폴더와 같게 변경하고 디스크에 있는 폴더의 모든 내용을 프로젝트에 추가합니다. 디스크 상의 디렉토리에 존재하지 않는 없는 버추얼 폴더의 아이템은 타겟으로 이동합니다.

자동 적용을 비활성화하려면, 자동 적용 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 자동 적용 정지를 선택하십시오. 자동 적용 폴더는 버추얼 폴더가 됩니다.



노트 자동 적용 폴더는 디스크에 해당하는 아이템이 없으므로 변수 및 하이퍼링크와 같은 가상 아이템을 포함할 수 있습니다. 자동 폴더에서 변수와 하이퍼링크와 같은 가상 아이템을 재배치할 수 있습니다. 재배치해도 디스크에 영향을 주지 않기 때문입니다. 또한 가상 아이템을 자동 적용 폴더로 또는 폴더 밖으로 이동할 수도 있습니다.

LabVIEW 프로젝트에서 라이브러리 사용하기

LabVIEW 프로젝트 라이브러리는 VI, 타입 정의, **공유 변수**, 팔레트 파일 및 다른 프로젝트 라이브러리를 포함한 기타 파일의 집합입니다. **새 프로젝트 라이브러리를 생성하고 저장**할 때 LabVIEW는 프로젝트 라이브러리 파일 .lvlib를 생성합니다. 이 파일에는 프로젝트 라이브러리의 프로퍼티와 프로젝트 라이브러리가 소유하는 파일에 대한 참조가 포함됩니다.



노트

묶음 프로젝트 라이브러리는 어플리케이션 빌더를 사용해서 여러 개의 프로젝트 라이브러리를 한 개의 파일로 만든 것입니다. 묶음라이브러리에서 파일을 변경하거나 **묶음 라이브러리를 새로 빌드**하지 않는 한 파일을 변경할 수 없습니다.

프로젝트 라이브러리는 사용자가 파일을 하나의 아이템 계층구조로 구성하려 할 때, 잠재적인 VI 이름 중복을 피하려 할 때, 특정 파일에 대한 퍼블릭 접근을 제한하려 할 때, 파일 집합에 대한 편집 권한을 제한하려 할 때, VI의 그룹에 대해 **기본 팔레트 파일을 설정**하려 할 때 유용합니다. 프로젝트 라이브러리가 소유하고 있는 아이템을 **프로젝트 탐색기** 윈도우에서 블록 다이어그램이나 프런트패널로 끌어올 수 있습니다.

프로젝트 탐색기 윈도우 또는 독립 프로젝트 라이브러리 윈도우에서 프로젝트 라이브러리의 구조를 볼 수 있습니다. 프로젝트 탐색기 윈도우가 열려있지 않은 경우, 프로젝트 라이브러리 파일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 열기를 선택하면 프로젝트 라이브러리 윈도우에서 열 수 있습니다.



노트 선택한 프로젝트 라이브러리 파일이 최상위 레벨 프로젝트 라이브러리 파일이 아닌 경우, 열리는 독립 윈도우는 최상위 레벨 프로젝트 라이브러리의 프로젝트 라이브러리 윈도우입니다. 선택한 프로젝트 라이브러리 파일은 최상위 레벨 프로젝트 라이브러리 윈도우의 목차 트리에 있습니다.

프로젝트 라이브러리를 사용하여 아이템의 논리적 가상 계층구조를 구성할 수 있습니다. 프로젝트 라이브러리 파일은 LLB와는 달리 소유하는 실제 파일을 포함하지 않습니다.

다. LLB는 VI를 포함하는 물리적인 디렉토리입니다. 프로젝트 라이브러리가 소유하는 파일들은 여전히 여러분이 저장한 디스크의 디렉토리에 개별적으로 나타납니다.

프로젝트 라이브러리를 사용하여 VI와 다른 LabVIEW 파일의 이름을 유효화합니다. LabVIEW는 VI를 파일 이름으로 식별하므로, LabVIEW는 의도치 않게 VI를 로드하고 참조할 수도 있습니다. VI가 다른 VI와 같은 파일 이름을 가지고 있는 것은 상호 링크라고 알려진 문제입니다. VI가 프로젝트 라이브러리의 일부인 경우, LabVIEW는 상호 링크를 피하기 위해 프로젝트 라이브러리 이름으로 VI 이름을 식별합니다. 정규화된 이름은 파일 이름과 소유하는 프로젝트 라이브러리의 정규화된 이름을 포함합니다.

예를 들어 library1.lvlib가 소유하는 init.vi라는 이름의 SubVI를 포함하는 caller.vi라는 이름의 VI를 만드는 경우, library2.lvlib가 소유하는 init.vi라는 이름의 다른 SubVI를 포함시켜 상호 링크 문제를 피할 수 있습니다. caller.vi를 저장할 때 LabVIEW가 기록하는 정규화된 이름은 각각 library1.lvlib:init.vi와 library2.lvlib:init.vi입니다.

	노트
	- 하나의 프로젝트 라이브러리만이 특정 VI를 소유할 수 있습니다. 그러나 텍스트 파일이나 HTML 파일과 같은 LabVIEW 이외의 파일은 여러 프로젝트 라이브러리와 연관시킬 수 있습니다. - 파일이 이미 한 라이브러리에 속해 있는 경우, 이 파일을 새 라이브러리에 추가하지 마십시오. 한 라이브러리와 이미 연관되어 있는 파일을 새 라이브러리에 추가해야 할 경우, 해당 파일의 원본 라이브러리도 새 라이브러리에 추가해야 합니다. 그렇지 않으면, 빌드 출력 시 이 파일을 새 라이브러리에 추가하지 않고 대상 폴더에 놓게 됩니다.

프로젝트 라이브러리의 버전 번호를 지정하여 파일 집합에 실시한 변경을 시간에 따라 구별할 수 있습니다. 프로젝트 라이브러리 프로퍼티 대화 상자의 일반 셋팅 페이지에서 버전 번호를 설정하고 정기적으로 번호를 업데이트합니다. 버전 번호는 프로젝트 라이브러리 이름에 영향을 주지 않습니다.

	주의 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다른 이름으로 저장 또는 이름 바꾸기를 선택하여 다른 이름으로 저장 대화 상자를 디스플레이하고 프로젝트 라이브러리의 이름을 변경합니다. LabVIEW 외에서 프로젝트 라이브러리 이름을 변경하는 경우, 프로젝트 라이브러리를 갯 수도 있습니다.

특정 타입의 파일에 대한 퍼블릭 접근을 제한하기 위해 프로젝트 라이브러리를 사용하십시오. 퍼블릭 또는 프라이빗으로 프로젝트 라이브러리의 아이템 및 폴더에 대한 **접근을 설정**하여 사용자가 특정 아이템에 접근하는 것을 막을 수 있습니다. 폴더에 대한 접근을 프라이빗으로 설정하면 해당 폴더의 모든 VI에 프라이빗 접근이 적용됩니다.

프로젝트 라이브러리를 잠그거나 암호로 보호하여 **편집 권한을 제한**할 수 있습니다. 프로젝트 라이브러리를 잠그면, 사용자가 아이템을 추가 또는 제거하지 못하며 **프라이빗으로 설정**한 아이템을 볼 수 없습니다. 프로젝트 라이브러리에 암호를 설정하면, 사용자가 암호 없이 프로젝트 라이브러리 프로퍼티에서 아이템을 추가 또는 제거하거나 편집할 수 없습니다. 사용자는 프로젝트 라이브러리 프로퍼티 대화 상자를 열 수 있으나, 보호 옵션을 제외한 모든 대화 상자 구성요소가 비활성화됩니다. 사용자는 프로젝트 라이브러리를 잠금해제하거나 암호를 입력하여 대화 상자 구성요소를 활성화해야 합니다.

프로젝트 라이브러리를 잠금 해제하려면 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 암호 입력을 선택합니다.



노트 구매한 LabVIEW 버전에 특정한 유효한 라이선스가 없는 LabVIEW 프로젝트 내의 라이브러리에 암호를 입력하면 아이템을 라이브러리 안이나 밖으로 끌어올 수 없습니다.



노트 프로젝트 라이브러리에 **암호 보호를 추가**해도 소유하는 VI에 암호 보호가 추가되지는 않습니다. 블록 다이어그램과 프런트패널의 편집을 제한하고자 하는 경우, **개별 VI에 암호 보호를 설정**해야 합니다. 혼동을 피하기 위해 프로젝트 라이브러리와 프로젝트 라이브러리가 소유하는 VI에 같은 암호를 사용하는 것을 고려하십시오.

프로젝트 폴더에서 프로젝트 라이브러리를 생성할 수 있습니다. 또한, **LLB를 프로젝트 라이브러리로 변환**할 수도 있습니다. LLB는 프로젝트 라이브러리와는 다른 특징과 장점을 가지고 있으므로, 프로젝트 라이브러리로 변환할지 여부를 결정하기 전에 LLB를 사용할 수 있는 방법을 고려하십시오. LLB에 프로젝트에서 라이브러리 파일을 포함할 수 있습니다.

프로젝트 라이브러리에 팔레트 파일 .mnu를 포함하는 경우, 이를 프로젝트 라이브러리가 소유하는 모든 VI의 **기본 팔레트 파일로 설정**할 수 있습니다. 함수 팔레트에서 블록 다이어그램에 끌어다 놓는 여러가지 VI 및 함수의 바로 가기 메뉴에서 소스 팔레트에 접근할 수 있듯이, 프로젝트 라이브러리의 기본 팔레트 파일은 프로젝트 라이브러리가 소유하고 있는 VI의 SubVI 호출에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 선택할 수 있는 팔레트입니다. 그러나 소스 팔레트와는 달리, 프로젝트 라이브러리의 기본 팔레트 파일은 프로젝트 라이브러리가 소유하고 있는 VI를 포함하지 않아도 됩니다. 프로젝트 라이브러리 프로퍼티 대화 상자의 일반 셋팅 페이지에서 기본 팔레트 링 컨트롤에 있는 팔레트 파일을 선택합니다. 또한 아이템 셋팅 페이지에서 기본 팔레트 파일을 설정할 수도 있습니다. 목차 트리에서 .mnu 파일을 선택하고 기본 팔레트 확인란에 확인 표시를 합니다.

LabVIEW가 **공유 변수 엔진(SVE)**에 배포하는 프로젝트 라이브러리를 일컬어 '프로세스'라 합니다. 대응하는 라이브러리에 영향을 주지 않고 프로세스 또는 프로세스내 공유 변수를 변경하는 것을 온라인 설정이라 합니다. 대응하는 프로세스에 영향을 주지 않고 라이브러리 또는 라이브러리내 공유 변수를 변경하는 것을 오프라인 설정이라 합니다. 오프라인 설정을 사용하여 공유 변수를 변경하는 경우, 라이브러리내의 VI를 실행하기 전 반드시 라이브러리를 SVE에 다시 배포해야 SVE가 프로세스에서 공유 변수를 업데이트할 수 있습니다.

프로젝트 라이브러리 구성하기

LabVIEW 프로젝트 라이브러리가 소유하는 파일에 대한 체계적 구조를 생성할 수 있습니다. 프로젝트 라이브러리 아이템의 구조가 잘 구성되면 사용자가 쉽게 **소스 컨트롤**을 사용하고, 파일 이름 충돌을 피하고, 프로젝트 라이브러리를 퍼블릭과 프라이빗 접근 영역으로 분리할 수 있습니다.

다음 리스트는 프로젝트 라이브러리와 프로젝트 라이브러리가 소유하는 파일을 구성할 때 고려해야 할 유의점과 권장사항을 설명합니다:

- 프로젝트 라이브러리에 관련된 파일만을 포함한 별도의 LabVIEW 프로젝트 내에 각 **프로젝트 라이브러리를 생성**합니다. 이 파일에는 예제 파일과 프로젝트 라이브러리를 생성하고 테스트하기 위해 사용한 파일이 포함됩니다. 프로젝트와 프로젝트 라이브러리의 파일 이름을 비슷하게 정합니다. 프로젝트 라이브러리가 별도의 여러 기능 영역을 포함하는 경우, 각 영역에 **프로젝트 서브라이브러리**를 사용할 것을 고려하십시오.
- 생성한 각 프로젝트 라이브러리에 별도의 파일 디렉토리를 생성합니다. 디렉토리에 프로젝트 라이브러리가 소유하는 파일을 포함시킬 수 있습니다. 같은 디렉토리에 하나 이상의 프로젝트 라이브러리 파일을 포함시키는 경우, 같은 이름의 VI를 다른 라이브러리에 넣으려 할 때 충돌이 발생할 수 있습니다. 프로젝트 라이브러리 파일을 별도의 디렉토리로 구성하면 디스크에서 특정한 프로젝트 라이브러리에 연관된 파일을 식별하기 쉽습니다.
- **정렬 옵션**을 사용하여 잡기지 않은 프로젝트 라이브러리 내의 아이템을 구성할 수 있습니다.
- 프로젝트 라이브러리가 소유하고 있는 디스크의 파일을 이동하는 경우, 프로젝트 라이브러리를 다시 열고 저장하여 프로젝트 라이브러리가 이동된 아이템에 정확히 링크되어 있는지 확인합니다.
- (Windows) 프로젝트 라이브러리가 포함된 **설치 프로그램을 만드는** 경우, 프로젝트 라이브러리가 소유하는 파일을 프로젝트 라이브러리와 같은 드라이브에 저장하십시오. 일부 파일이 네트워크 드라이브와 같은 다른 드라이브에 있는 경우, 설치 프로그램에 프로젝트 라이브러리를 포함시키면 프로젝트 라이브러리 링크가 깨질 수도 있습니다.
- 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 라이브러리가 잘못 요구한 아이템 찾기를 선택하여 라이브러리와 라이브러리가 요구한 아이템 간의 **충돌을 해결**할 수 있습니다. 이 기능을 사용해서 라이브러리의 요구를 인식하지 않는 아이템을 찾습니다.
- 프로젝트 라이브러리의 어떤 아이템을 **퍼블릭과 프라이빗으로 설정**할지 결정합니다. 사용자는 프라이빗 VI를 다른 VI나 어플리케이션에서 SubVI로 사용할 수 없습니다. 퍼블릭 아이템은 프로젝트 라이브러리 기능의 인터페이스를 제공하며 사용자가 찾고 사용할 수 있는 팔레트 VI, X컨트롤, 인스트루먼트 드라이버, 도구가 포함됩니다. 프라이빗 아이템에는 지원 VI, 저작권이 있는 파일, 또는 사용자 코드를 깨지 않고 나중에 편집하려는 아이템이 포함됩니다. 다음의 권장 사항을 고려하십시오.
 - 프라이빗이라는 이름의 프로젝트 라이브러리에 폴더를 생성합니다. 프로젝트 라이브러리 프로퍼티 대화 상자의 아이템 셋팅 페이지에서 폴더의 접근 셋팅을 프라이빗으로 설정합니다. LabVIEW는 자동으로 프라이빗 폴더에 추가한 모든 프로젝트 파일을 프라이빗으로 설정하므로 개별 VI에 대한 접근 셋팅을 설정할 필요가 없습니다.
 - 프라이빗 폴더 외에 있는 모든 프로젝트 라이브러리 파일은 퍼블릭이라 가정합니다. 퍼블릭 파일을 위한 폴더를 생성할 필요는 없습니다.
 - 또한 프로젝트 라이브러리 내에서 각 기능 그룹에 대한 폴더를 만들고 각 기능 그룹 폴더에 프라이빗 서브폴더를 추가하여, 프로젝트 라이브러리를 퍼블릭 아이템과 프라이빗 아이템으로 정리할 수 있습니다.

프로젝트 서브라이브러리

프로젝트 서브라이브러리는 다른 프로젝트 라이브러리가 소유하는 프로젝트 라이브러리입니다. 소유하는 프로젝트 라이브러리의 셋팅은 프로젝트 서브라이브러리 내의 아이템에 대한 접근 셋팅 및 편집 권한에 영향을 미치지 않습니다. 프로젝트 서브라이브러

리 파일 .lvlib의 접근을 소유하는 프로젝트 라이브러리 내에서 프라이빗으로 설정할 수 있습니다. 그러나 프로젝트 라이브러리 자체를 편집하는 경우, 서브라이브러리가 소유하는 아이템은 퍼블릭 또는 프라이빗 접근 설정을 유지합니다.

프로젝트 서브라이브러리는 별도의 기능 영역을 포함하는 프로젝트 라이브러리를 생성하려 할 때 유용합니다. 예를 들어, 그래픽 도구의 프로젝트 라이브러리를 생성하는 경우, 2차원과 3차원 그리기 도구를 별도의 서브라이브러리로 나눌 수 있습니다.

LabVIEW 프로젝트에서 묶음 프로젝트 라이브러리 사용하기

LabVIEW 묶음 프로젝트 라이브러리는 여러 개의 파일을 .lvlibp 파일 확장자를 가진 한 개의 파일로 묶는 **프로젝트 라이브러리**입니다. 묶음 라이브러리의 최상위 레벨 파일은 프로젝트 라이브러리입니다. 묶음 라이브러리는 기본으로 최상위 프로젝트 라이브러리와 같은 이름을 가집니다.

LabVIEW 프로젝트에서 여러 VI를 묶음 라이브러리로 만드는 이유는 다음과 같습니다.

- 독립 어플리케이션을 만들 때, 독립 어플리케이션의 일부가 묶음 라이브러리 안에 있을 경우, 만드는데 걸리는 시간이 줄어듭니다. 묶음 라이브러리는 미리 컴파일된 파일이기 때문에, 독립 어플리케이션을 만들 때 따로 컴파일을 하지 않아도 되므로 만드는 시간이 짧습니다.
- 묶음 라이브러리는 여러 개의 파일을 한 개의 파일로 묶기 때문에, VI를 묶음 라이브러리에 배포할 때 배포되는 파일의 수가 더 적습니다.
- 묶음 라이브러리에서 반출된 VI를 호출하는 VI는 **메모리 할당 변경에 적응**할 수 있어, 호출자 VI를 다시 컴파일하지 않아도 됩니다.

프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 묶음 라이브러리를 선택하여 묶음 라이브러리 프로퍼티 대화 상자를 디스플레이하고 설정을 설정해서 **묶음 라이브러리를 만듭니다**.

묶음 라이브러리는 LabVIEW 파일만 포함합니다. LabVIEW는 기본으로 LabVIEW가 아닌 파일을 묶음 라이브러리와 같은 대상 디렉토리에 저장합니다. 대상 리스트에서 지원 디렉토리를 선택하고, 대상 경로 텍스트 박스에서 경로를 변경하여 LabVIEW가 아닌 파일을 저장하는 장소를 변경합니다.

프로젝트에서 묶음 라이브러리로 **프로젝트 라이브러리를 대체**하려면 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택합니다. 대체를 선택하면, LabVIEW 프로젝트에서 프로젝트 라이브러리의 모든 호출자 VI는 묶음 라이브러리의 정규화된 이름으로 업데이트됩니다.

묶음 프로젝트 라이브러리의 반출된 VI

묶음 라이브러리를 열면, 반출된 LabVIEW VI만 보입니다. 반출된 VI는 프로젝트 라이브러리 안에 있는 퍼블릭 접근 영역을 가진 VI거나, LabVIEW 클래스 안에 있으면서 퍼블릭 또는 보호된 접근 영역을 가진 VI입니다.



노트

LabVIEW 클래스로부터 묶음 라이브러리를 생성하려면, LabVIEW 클래스를 프로젝트 라이브러리에 추가한 후, 프로젝트 라이브러리에서 **묶음 라이브러리를 생성**합니다.

묶음 라이브러리에서 파일의 접근 영역을 알려면, 프로젝트 라이브러리의 파일에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 접근 영역을 선택합니다. 묶음 라이브러리에서는 파일의 접근 영역을 변경할 수 없으며, 변경하려면 LabVIEW 프로젝트에서 변경하고, 묶음 라이브러리를 다시 만들어야 합니다.

묶음 프로젝트 라이브러리 빌드 타입

묶음 라이브러리를 만들 때, 릴리즈 빌드나 디버그 빌드를 생성할 수 있습니다.

묶음 라이브러리의 기본 빌드는 릴리즈 빌드입니다. 라이브러리에 VI의 블록 다이어그램을 포함하지 않으려면, 묶음 라이브러리를 릴리즈 빌드로 생성하십시오. 또한 VI가 대화 상자가 아닐 경우, 묶음 라이브러리 프로퍼티 대화 상자의 소스 파일 설정 페이지에서 프런트패널을 제거할 수 있습니다.



노트 묶음 라이브러리 릴리즈 빌드에서는 블록 다이어그램이 없기 때문에 VI를 디버그할 수 없습니다.

VI의 블록 다이어그램을 포함하려면, 묶음 라이브러리를 디버그 빌드로 생성하십시오. 디버그 빌드를 사용하면 묶음 라이브러리에서 문제점을 디버그할 수 있습니다. 기존의 묶음 라이브러리에 있는 VI에는 변경된 사항을 저장할 수 없으므로, 변경된 사항을 원래의 VI에 저장한 후, 묶음 라이브러리를 다시 만듭니다.

디버그 빌드를 생성하려면, 묶음 라이브러리 프로퍼티 대화 상자의 고급 페이지에서 디버깅 활성화 확인란에 확인 표시를 하십시오.

묶음 프로젝트 라이브러리와 호출 VI의 호환성

묶음 라이브러리의 VI를 다음과 같이 변경할 경우, 호출 VI는 다시 컴파일해야 합니다.

- 커넥터 팬 패턴 변경하기.
- 커넥터 팬에서 다이나믹 디스패치 터미널 변경하기.
- 커넥터 팬에서 입력 터미널을 출력 터미널로, 또는 출력 터미널을 입력 터미널로 변경하기.
- 커넥터 팬에서 옵션 입력을 필수 입력으로 변경하기.
- 커넥터 팬에서 입력과 출력의 데이터 타입 변경하기.

- 커넥터 팬에 새로운 터미널 추가하기.

VI가 한 타겟에 대해 컴파일된 묶음 프로젝트 라이브러리를 호출하고, 이 VI를 다른 OS가 설치된 다른 타겟에서 여는 경우, 묶음 프로젝트 라이브러리는 로드되지 않습니다.

독립 어플리케이션, 공유 라이브러리 또는 묶음 라이브러리가 묶음 라이브러리에 있는 VI를 호출할 경우, 호출되는 묶음 라이브러리를 다른 **빌드 스펙**과 같은 디렉토리에 포함 시키십시오. 빌드 스펙이 묶음 라이브러리의 VI를 호출할 때, 라이브러리를 업데이트된 버전으로 대체하려면, 업데이트된 묶음 라이브러리의 커넥터 팬이 빌드 스펙과 호환 가능해야 합니다.

묶음 라이브러리의 반출된 VI의 커넥터 팬이 빌드 스펙의 VI와 호환되지 않는 경우, 라이브러리를 다시 만든 후, 호출 VI로 빌드 스펙을 다시 생성합니다.

묶음 라이브러리에 있는 VI의 변경이 커넥터 팬의 호환성에 영향을 미칠 경우, 호출 VI는 기본으로 변경 사항에 적응하기 때문에 라이브러리를 호출하는 VI를 사용자가 다시 컴파일할 필요가 없습니다. 호출자 VI가 다시 컴파일되도록 하려면 묶음 라이브러리 프로퍼티 대화 상자의 커넥터 팬 상태 페이지에서 호출자는 실행 시 반출 VI 커넥터 팬 상태에 맞게 적응 확인란의 확인 표시를 제거합니다.

(Real-Time Module) 묶음 라이브러리에 있는 VI의 변경으로 커넥터 팬의 호환성에 영향이 미치는 경우, 기본적으로 라이브러리를 호출하는 VI를 다시 컴파일해야 합니다. 묶음 라이브러리에서 VI를 변경한 후, 호출 VI가 다시 컴파일하지 않고 실행되게 하려면, 묶음 라이브러리 프로퍼티 대화 상자의 커넥터 팬 상태 페이지에서 호출자는 실행 시 반출 VI 커넥터 팬 상태에 맞게 적응 확인란에 확인 표시를 합니다.

묶음 프로젝트 라이브러리 프로퍼티

묶음 라이브러리는 최상위 프로젝트 라이브러리와 같은 프로퍼티를 가집니다. 묶음 라이브러리는 다시 만들지 않고서는 변경하는 것이 불가능하기 때문에 프로젝트 라이브러리 프로퍼티 대화 상자에서 프로퍼티를 변경할 수 없습니다. 프로젝트 라이브러리 프로퍼티 대화 상자를 디스플레이하려면 묶음 라이브러리를 열고, .lvlibp 파일에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택합니다.

프로젝트 라이브러리 프로퍼티 대화 상자를 업데이트하려면, 최상위 프로젝트 라이브러리의 프로퍼티를 변경하고 묶음 라이브러리를 다시 빌드합니다.



노트 프로젝트 라이브러리 프로퍼티 대화 상자의 프로퍼티는 버전 번호를 제외하면 묶음 라이브러리 프로퍼티 대화 상자의 프로퍼티와 다릅니다. 묶음 라이브러리를 생성할 때, 묶음 라이브러리 프로퍼티 대화 상자에서 버전 정보 페이지의 버전 번호는 프로젝트 라이브러리 프로퍼티에서 대화 상자 일반 셋팅 페이지의 버전 번호를 덮어씁니다.

LabVIEW 프로젝트에서 의존성 관리하기

의존성을 사용하여 다음 아이템과 같이 타겟 아래의 VI가 요구하는 아이템을 볼 수 있습니다:

- 다른 VI
- 공유 라이브러리(DLL)
- **LabVIEW 프로젝트 라이브러리**

LabVIEW 프로젝트의 각 타겟은 의존성을 포함합니다. LabVIEW는 프로젝트의 각 아이템이 필요로 하는 파일을 자동으로 식별하고, 이러한 파일을 의존성에 포함시킵니다. LabVIEW는 의존성을 다음 세 개의 폴더에 넣습니다: `vi.lib`, `user.lib`, 메모리의 아

의존성 리스트 사용하기

아이템을 직접 의존성에 추가할 수 없습니다. 의존성은 프로젝트에 아이템을 추가, 제거 또는 저장하면 자동적으로 업데이트됩니다. 예를 들어, SubVI를 포함하는 VI를 타겟에 추가하는 경우, LabVIEW는 SubVI를 의존성에 추가합니다. 그러나, 타겟 아래에 SubVI를 수동으로 추가하는 경우, 해당 아이템은 의존성에 나타나지 않습니다. 의존 파일을 이동하거나 이름을 만드는 경우, **잘못된 의존성을 방지**하도록 주의하십시오.

프로젝트의 현재 의존성을 보려면, 의존성에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 고침을 선택합니다. 의존성 새로 고침 메소드를 사용하여 프로그램적으로 의존성을 새로 고칠 수도 있습니다.



노트 프로젝트 탐색기 윈도우에서 의존성을 숨긴 경우, 프로젝트 탐색기 윈도우에서 이에 접근하려면 다시 **아이템을 디스플레이**해야 합니다.

LabVIEW는 SubVI를 반복적으로 추적합니다. LabVIEW는 공유 라이브러리(DLL)를 반복적으로 추적하지 않습니다. 예를 들어, `a.vi`가 `b.dll`을 정적으로 호출하고 `b.dll`이 `c.dll`을 정적으로 호출하는 경우, LabVIEW는 `b.dll`만을 의존적인 아이템으로 고려합니다. 프로젝트에서 `c.dll`을 관리하려면, 반드시 `c.dll`을 타겟 아래에 추가해야 합니다.

의존성이 프로젝트 라이브러리의 일부인 경우, LabVIEW는 의존성아래의 전체 프로젝트 라이브러리를 포함합니다.

새 아이템을 의존성 아래에 생성할 수 없습니다. 프로젝트 탐색기 윈도우의 다른 위치에서 아이템을 끌어서 의존성 노드에 놓을 수 없습니다.

다이나믹 의존성 포함하기

프로젝트를 열면, VI가 다이나믹하게 호출하는 아이템은 의존성 아래에 나타나지 않습니다. 호출자를 실행하면, 다이나믹하게 로드된 아이템은 의존성 아래의 메모리의 아이템에 나타납니다. 이 아이템을 타겟 아래에 수동으로 추가하면 **프로젝트에서 아이템을 관리**할 수 있습니다.

다이나믹하게 로드된 파일은 프로젝트 내에서 호출자에 의해 정적으로 링크되지 않기 때문에, 다이나믹하게 로드된 파일에의 경로를 변경하면 프로젝트가 파일을 로드하지 않을 수도 있습니다. 다이나믹하게 로드된 파일이 올바른 위치에 있도록 하려면 다음 가이드라인을 따르십시오:

- 파일을 별도의 폴더에 모아 놓습니다.
- 상대 경로를 사용하여 파일을 참조합니다.
- 프로젝트를 이동해야 하거나 어플리케이션을 배포해야 하는 경우, 다이나믹한 의존성을 모아놓은 폴더를 포함하도록 합니다.

공유 의존성 관리하기

코드를 공유하는 어플리케이션을 만드는 경우, 이 코드를 변경하면 다른 호출자에 의도하지 않은 영향을 미칠 수 있습니다. 공유 코드가 있는 의존성을 관리할 때는 다음 가이드라인을 따르십시오:

- 여러 어플리케이션이 공유하는 코드의 변경을 최소화합니다.
- 공유 코드를 변경해야 할 경우, 테스트와 통합 목적으로 로컬 복사본을 만듭니다.
- 공유 코드 변경 사항을 관리하고 통합하도록 합니다.
- **소스 코드 컨트롤**을 사용합니다.

프로젝트 아이템 대체하기

LabVIEW 프로젝트의 VI, 클래스, 라이브러리를 같은 타입의 다른 아이템으로 대체할 수 있습니다. 프로젝트 탐색기 윈도우의 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택하여, 원래 아이템을 대체할 파일을 선택합니다. 이와 같이 프로젝트 아이템을 대체할 때 LabVIEW는 라이브러리와 묶음 프로젝트 라이브러리를 동일한 타입으로 처리합니다.

프로젝트 아이템을 대체하면 프로젝트의 호출자에 영향이 미칠 수 있습니다. 프로젝트 아이템을 대체한 후, LabVIEW는 호출자가 원래 프로젝트 아이템 대신 새로운 프로젝트 아이템을 참조하도록 업데이트합니다. LabVIEW는 호출자를 업데이트하고, 원래 프로젝트 아이템이나 대체 프로젝트 아이템은 변경하지 않습니다.

라이브러리 대체하기

최소 한 개 멤버가 원래 라이브러리의 호출자에 의해 참조되고 참조되는 멤버를 대체 라이브러리에서 찾을 수 없는 라이브러리를 대체하면 LabVIEW는 경고를 반환합니다. 경고 대화 상자에서 동의를 클릭하면 계속해서 라이브러리를 대체할 수 있습니다. 하지만, 참조되는 아이템이 포함된 원래 라이브러리를 LabVIEW가 프로젝트에서 제거하기 때문에, 라이브러리 대체 후 호출자가 깨집니다.

대체 작업은 자체 라이브러리의 멤버를 참조하는 호출자에 대해 영향을 미치지 않습니다.

프로젝트 충돌 해결하기

프로젝트는 프로젝트에서의 다른 아이템과 충돌하는 아이템을 포함할 수 있습니다. 충돌이란 LabVIEW 프로젝트에 동일한 타겟 아래 같은 유효한 이름을 가진 있는 아이템이 두 개 혹은 이상 있을 때 발생하는 잠재적 상호 링크입니다. 예를 들어, VI가 다른 경로의

프로젝트에 있는 아이템과 유효한 이름이 같은 SubVI를 호출하면, 상호 링크 때문에 충돌이 발생합니다. 대부분의 충돌은 다른 프로젝트 아이템이 **충돌하는 아이템을 참조**하기 때문에 발생합니다.

충돌하는 아이템 찾기

상호 링크가 존재하는지 찾는 가장 좋은 방법은 해당 아이템에 대한 전체 경로를 보는 것입니다. 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보기>> 전체 경로를 선택하여, 경로 열을 디스플레이하고 해당 프로젝트 아이템에 해당하는 파일 경로를 확인합니다. 유효한 이름이 같은 아이템외의 모든 아이템의 이름을 변경하거나 프로젝트에서 제거해야 합니다.

기존의 충돌에 대한 자세한 정보를 확인하려면, 다음에 보이는 것처럼 충돌 해결 버튼을 클릭하여 프로젝트 충돌 해결 대화 상자를 디스플레이합니다.



또한 프로젝트 메뉴에서 프로젝트>> 충돌 해결을 선택하거나 충돌하는 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 충돌 해결을 선택하여 이 대화 상자를 열 수도 있습니다.

아이템 페이지에 있는 특정 충돌 아이템에서 마우스 오른쪽 버튼을 클릭하거나 찾기>> 충돌을 선택하여 충돌 요소 찾기 대화 상자에서 모든 충돌하는 아이템을 볼 수 있습니다. 아이템이 다른 한 아이템과만 충돌하는 경우, LabVIEW는 프로젝트 탐색기 윈도우에서 해당 아이템을 하이라이트합니다. 또는 찾기: 충돌 프로퍼티를 사용하여 프로그램적으로 프로젝트에서의 충돌을 찾을 수 있습니다.



노트

노란색 경고 삼각형이 프로젝트 탐색기 윈도우에서 모든 충돌하는 아이템에 나타납니다.

이미 LabVIEW 프로젝트에 있는 아이템과 충돌하는 파일을 로드하려 하면 프로젝트에서 계층구조 충돌 대화 상자가 나타납니다. 계층구조의 아이템이 프로젝트 또는 의존성의 아이템과 정규화된 이름이 같기 때문에 LabVIEW가 파일을 열 수 없습니다. 의존하는 아이템이 다른 프로젝트 아이템과 충돌하는 경우, 의존하는 아이템을 참조하는 아이템을 제거하면 아이템이 의존성에서 제거됩니다.

사용자가 VI의 블록 다이어그램을 열고 새 SubVI를 추가하면, LabVIEW는 VI 계층구조를 스캔합니다. 계층구조의 멤버가 이미 프로젝트에 있는 아이템과 비교해 정규화된 이름이 같고 경로가 다르면 SubVI는 로드되지 않습니다. 프로젝트에 추가 및 의존성 업데이트 대화 상자가 나타납니다. 계층구조를 추가하거나 로드를 취소하고 다른 SubVI를 선택할 수 있습니다.

충돌하는 아이템 제거하기

VI의 계층구조가 프로젝트에 있는 다른 내용의 계층구조와 충돌하는 경우, 호출자 VI를 프로젝트에서 제거하여 충돌을 해결할 수 있습니다. VI 또는 타입 정의에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 찾기>> 호출자 또는 찾기>> SubVI를 선택하여 프로젝트 탐색기 윈도우에서 아이템이 참조하는 호출자 또는 SubVI를 하이라이트합니다. 프로젝트에 하나 이상의 호출자 또는 SubVI가 있는 경우, 호출자 찾기 또는 SubVI 찾기 대

화 상자가 나타납니다. 프로젝트 루트 또는 타겟에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 호출자가 없는 아이템 찾기를 선택하여 호출자가 없는 아이템 찾기 대화 상자를 열고 모든 최상위 레벨 아이템을 찾습니다. 충돌하는 SubVI를 참조하는 호출자가 없으면, 충돌하는 SubVI를 프로젝트에서 제거합니다. 또한 찾기: 호출자, 찾기: SubVI, 또는 찾기: 호출자가 없는 아이템 프로퍼티를 사용하여 프로그램적으로 호출자, SubVI 및 호출자가 없는 아이템을 찾습니다.



노트 충돌하는 SubVI를 프로젝트에서 제거해도 충돌이 해결되지 않을 수 있습니다. 프로젝트의 다른 VI가 여전히 충돌하는 SubVI를 참조할 수 있기 때문입니다. 충돌하는 아이템을 호출하는 모든 호출자를 프로젝트에서 제거할 때까지 이 아이템은 충돌하는 아이템으로 남아있게 됩니다. 호출자가 있는 아이템을 프로젝트에서 삭제하면 아이템은 의존성으로 이동합니다.

충돌하는 아이템의 이름 변경하기

충돌하는 아이템을 삭제하고 싶지 않고 프로젝트에 정규화된 이름이 같은 다른 아이템이 있는 경우, 해당 아이템의 이름을 변경하거나 아이템을 프로젝트 라이브러리에 추가하면 됩니다.

아이템 이름을 변경하면 잘못된 아이템 경로를 참조하는 호출자를 로드하고, 아이템의 이름을 변경하고 아이템과 호출자를 저장합니다. 이렇게 하면 호출자가 새 이름을 참조하게 됩니다. 호출자를 저장하지 않으면, 호출자는 여전히 원래의 아이템을 참조하기 때문에 원래의 아이템이 의존성 아래에 나타납니다.

VI가 **프로젝트 라이브러리**의 일부인 경우, LabVIEW는 상호 링크를 피하기 위해 프로젝트 라이브러리의 이름으로 VI 이름을 식별합니다. 정규화된 이름은 파일 이름과 소유하는 프로젝트 라이브러리 파일 이름의 정규화된 이름을 포함합니다. 정규화된 이름은 경로나 파일 이름을 변경하지 않고 변경됩니다.

메모리에 있는 LabVIEW 프로젝트 라이브러리가 다른 프로젝트 아이템과 충돌하는 경우, 로드하기 전에 적어도 하나의 충돌하는 라이브러리의 이름을 변경해야 합니다. 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 로드 해제를 선택합니다. LabVIEW가 라이브러리를 로드 해제한 후에는 다시 올바른 경로에서 라이브러리와 VI를 로드할 수 있습니다. 라이브러리가 다른 프로젝트 아이템과 충돌하고 메모리에 존재하지 않는 경우, 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 로드를 선택합니다. LabVIEW가 라이브러리를 로드한 후에는 라이브러리와 내용을 편집하고 올바른 경로에서 VI를 로드할 수 있습니다.

충돌하는 아이템 수정하기

두 개 이상의 아이템이 같은 정규화된 이름을 가지고 있고 그 중 하나의 아이템만 디스크에 존재하는 경우, 충돌하는 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로젝트에서 발견한 아이템으로 대체를 선택합니다. LabVIEW는 잘못된 아이템의 호출자를 업데이트하여 디스크에서 발견한 아이템을 참조하도록 합니다.

하나 이상의 VI가 잘못된 SubVI를 참조하는 경우, 다른 경로의 SubVI를 참조하도록 호출자의 방향을 변경합니다. 프로젝트 탐색기 윈도우의 충돌하는 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택하여 디스크에서 올바른 SubVI를 선택합니다. 파일 대화 상자가 열리면 대체할 파일을 선택합니다. LabVIEW는 잘못된 경로를 참조하는 모든 아이템을 자동으로 업데이트하여 대체된 파일을 참조하도록 합니다. 또한

충돌하는 아이템을 참조하는 각 VI를 로드할 수 있습니다. 로드 충돌 해결 대화 상자가 나타납니다. 로드할 특정 호출자 VI를 선택할 수 있습니다.



노트 LabVIEW는 아이템이 프로젝트 라이브러리 또는 프로젝트 라이브러리의 멤버인 경우, 프로젝트에서 발견한 아이템으로 대체 및 대체를 비활성화합니다.

잃어버린 아이템 찾기

LabVIEW가 찾을 수 없는 아이템을 잘못 참조하는 VI가 1 개 이상이라고 판단되는 경우, 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **잃어버린 아이템 찾기**를 선택합니다. 잃어버린 아이템 찾기 대화 상자가 나타납니다. 이 대화 상자는 디스크에 있는 아이템을 참조하는 프로젝트의 아이템 중에서 LabVIEW가 찾지 못하는 모든 아이템을 나열합니다. 아이템이 프로젝트에 없고 프로젝트의 다른 아이템이 없는 아이템에 의존하는 경우, 없는 아이템이 의존성에 나타납니다.

잘못 요구한 아이템 찾기

LabVIEW에서 라이브러리와 라이브러리가 요구한 아이템 간의 충돌을 찾아내고 해결할 수 있습니다. 아이템이 라이브러리의 요구에 응답하는지 아닌지 알려면 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 라이브러리가 잘못 요구한 아이템 찾기를 선택하여 프로젝트 아이템 찾기 대화 상자에서 잘못 요구된 아이템 리스트를 디스플레이합니다. 이 대화 상자를 사용해서 특정 아이템을 탐색하여 라이브러리에서 제거하거나 추가하십시오.



팁 폴더나 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 라이브러리가 잘못 요구한 아이템을 선택하여 이 기능을 사용할 수도 있습니다.

LabVIEW 프로젝트에 타겟 추가하기(Windows)

타겟 및 디바이스 추가 대화 상자를 사용하여 LabVIEW 프로젝트에 **타겟이나 디바이스를 추가**할 수 있습니다. Windows Embedded Standard, RT, FPGA 또는 터치 패널 타겟으로 작업하려면 프로젝트를 사용해야 합니다. 타겟과 함께 프로젝트를 사용하는 것에 대한 추가적인 정보는 특정한 모듈 문서를 참조하십시오.



노트

타겟 및 디바이스 추가 대화 상자를 디스플레이하기 위해서는 설치된 타겟을 지원하는 모듈 또는 드라이버가 있어야 합니다.

프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 타겟과 디바이스를 선택하여 타겟과 디바이스 추가 대화 상자를 디스플레이합니다. 프로젝트의 타겟이 다른 타겟을 지원하는 경우, 타겟에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 타겟과 디바이스를 선택하여 기존 타겟 아래에 타겟을 추가할 수도 있습니다. 예를 들어, 컴퓨터에 NI PCI 디바이스가 설치되어 있는 경우, 디바이스를 내 컴퓨터 아래에 추가할 수 있습니다.

타겟 아래에 **아이템을 추가**할 수 있습니다. 또한 타겟 폴더를 사용하여 LabVIEW 프로젝트에서 한 개 또는 이상의 타겟을 구성할 수 있습니다. 프로젝트에 타겟 폴더를 추가하려

면, 프로젝트 탐색기 윈도우의 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기»타겟 폴더를 선택합니다. 타겟 폴더가 프로젝트 탐색기 윈도우에 나타납니다.

파일을 타겟에 배포하기

프로젝트 탐색기 윈도우에서, 타겟 아래의 VI를 열고 실행하여 타겟에 파일을 배포합니다. LabVIEW는 소스 파일, 빌드 파일, 객체 코드, 하드웨어 설정 파일을 포함하여, 타겟에 적합한 파일 타입을 생성하고 배포합니다.



노트 **Windows Embedded Standard 타겟**을 사용하여 작업하는 경우, 오직 빌드된 설치 프로그램만 타겟에 배포할 수 있습니다.

여러 타겟을 가진 팔레트 사용하기

LabVIEW는 각각의 타겟에 대하여 서로 다른 팔레트 세트를 로드합니다. LabVIEW 프로젝트에서 여러 타겟에 VI를 연 경우, 여러 팔레트 세트가 메모리에 존재합니다. 타겟 사이를 스위치할 때, LabVIEW는 각 고정된 팔레트를 활성화 타겟의 대응 팔레트에 매핑하려 합니다. 고정된 팔레트에 대응 팔레트가 없는 경우, LabVIEW는 활성화 타겟의 최상위 레벨 팔레트를 디스플레이합니다.

특정한 타겟에 대한 팔레트 세트를 편집하려면, 수정하려는 팔레트 세트 타겟의 VI에서 도구»고급»팔레트 세트 편집을 선택합니다.

어플리케이션 인스턴스로 작업하기

LabVIEW는 **LabVIEW 프로젝트**에 각 타겟에 대한 어플리케이션 인스턴스를 생성합니다. 프로젝트 탐색기 윈도우에서 VI를 열 때, VI는 타겟의 어플리케이션 인스턴스에서 열립니다. 또한, LabVIEW는 주요 어플리케이션 인스턴스를 생성합니다. 여기에는 프로젝트의 일부가 아닌 열린 VI와 프로젝트에서 열지 않은 VI가 포함됩니다. 추가적으로 LabVIEW는 고유한 어플리케이션 인스턴스에서 **공유 라이브러리**를 로드합니다. 이렇듯 유일한 어플리케이션 인스턴스에서 공유 라이브러리를 열게 되면, 공유 라이브러리 내의 VI와 공유 라이브러리 밖에서의 VI가 이름이 같아 충돌하게 되는 경우를 방지할 수 있습니다.

프런트패널과 블록다이어그램의 왼쪽 아래 코너에 나타나는 어플리케이션 인스턴스 바로 가기 메뉴에 나타나는 어플리케이션 인스턴스 이름을 사용하여 VI가 어떤 어플리케이션 인스턴스에 속하는지 식별합니다.

VI가 열린 상태로 프로젝트 탐색기 윈도우에서 **VI를 제거**하는 경우, 열린 VI는 같은 어플리케이션 인스턴스 안에 남아있습니다. 프런트패널과 블록다이어그램의 왼쪽 아래 코너의 **어플리케이션 인스턴스 선택** 바로 가기 메뉴를 사용하여 VI의 어플리케이션 인스턴스를 선택할 수 있습니다.

프런트패널이나 블록다이어그램 오른쪽 위 코너의 VI 아이콘을 선택하고 아이콘을 프로젝트 탐색기 윈도우의 타겟에 끌어 열린 VI를 프로젝트에 추가하는 경우, 또는 프로젝트 탐색기 윈도우에서 하나의 타겟에서 다른 타겟으로 아이템을 드래그하는 경우, 열린 VI는 어플리케이션 인스턴스를 선택할 때까지 해당 타겟의 어플리케이션 인스턴스의 일부가 되지 않습니다.

또한 어플리케이션:기본:어플리케이션 프로퍼티를 사용하여 기본 어플리케이션 참조를 프로그램적으로 반환할 수 있습니다. 어플리케이션 프로퍼티를 사용하여 타겟 어플리케이션 인스턴스를 프로그램적으로 엽니다.

여러 어플리케이션 인스턴스의 VI 편집하기

동시에 여러 어플리케이션 인스턴스에서 디스크의 같은 VI를 열 수 있습니다. 예를 들어, 같은 VI를 두 개의 다른 프로젝트에서 열거나 프로젝트 내의 두 개의 다른 타겟으로부터 열 수 있습니다.

한 어플리케이션 인스턴스에서만 열린 VI를 편집한 후 그 VI를 다른 어플리케이션 인스턴스에서 열었을 경우, 방금 연 VI는 VI의 편집된 인스턴스의 최신 변경사항을 포함하고 있습니다. 그러나 이미 하나 이상의 어플리케이션 인스턴스에서 열린 VI를 편집하는 경우, LabVIEW는 자동으로 VI의 변경사항을 다른 어플리케이션 인스턴스에 적용하지 않습니다. VI가 모든 어플리케이션 인스턴스에서 같을 때까지 다른 어플리케이션 인스턴스의 VI를 편집, 실행, 또는 저장할 수 없습니다.

다음 방법 중 하나를 사용하여 VI가 모든 어플리케이션 인스턴스에서 같도록 만듭니다.

- VI 도구 모음의 다른 어플리케이션 인스턴스와 동기화 버튼을 클릭하여 변경 사항을 모든 어플리케이션 인스턴스의 VI에 적용합니다.
- 변경을 포함하는 VI를 저장합니다. VI를 저장할 때, LabVIEW는 자동으로 변경을 모든 다른 어플리케이션 인스턴스의 VI에 적용합니다.
- 변경을 포함하는 어플리케이션 인스턴스의 VI 편집을 실행 취소합니다.



노트

VI를 동기화한 후에는 VI의 편집을 실행 취소할 수 없습니다.

하나의 어플리케이션 인스턴스에서 VI 서버 프로퍼티와 메소드를 사용하면서 다른 어플리케이션 인스턴스와 상호 작용을 하고자 할 때에는 어플리케이션 참조를 사용해야 합니다. 여러 어플리케이션 인스턴스에서 VI가 실행되는지 분명하지 않을 수도 있습니다. 예를 들어, 각 LabVIEW 프로젝트는 자체적인 어플리케이션 인스턴스를 가집니다.

project1.lvproj에서 실행되는 VI는 project2.lvproj에서 실행되는 VI와는 다른 어플리케이션 인스턴스에서 실행 중입니다. 프로젝트에서 실행되지 않는 VI는 주요 어플리케이션 인스턴스에서 실행됩니다. 뿐만 아니라, LabVIEW는 labview\help, labview\project, labview\wizard 디렉토리에 있는 사용자 VI를 프라이빗 어플리케이션 인스턴스에서 엽니다. 예를 들어, VI를 labview\project 디렉토리에 놓은 후 도구 메뉴에서 이 VI를 실행하면, LabVIEW는 주요 어플리케이션 인스턴스와는 별개의 프라이빗 어플리케이션 인스턴스(NI.LV.Dialog)에서 VI를 실행합니다.

VI가 실행 중이거나 다른 어플리케이션 인스턴스에서 실행되기 위해 보존되고 있는 동안에는 VI를 편집할 수 없습니다. VI가 **프로젝트 라이브러리**의 일부인 경우, VI를 실행할 때 LabVIEW는 모든 어플리케이션 인스턴스의 프로젝트 라이브러리를 일시적으로 잠급니다. VI가 실행 중인 경우에는 프로젝트 라이브러리를 편집할 수 없습니다. VI가 정지한 후, LabVIEW는 프로젝트 라이브러리를 잠금해제합니다.

프로젝트 계획하고 설계하기

프로젝트를 설계하기 전에, 일반적인 개발 애로사항과 소프트웨어 엔지니어링 주기 모델을 참고하시면 유용합니다.

VI를 개발하기 전에, 사용자가 실행해야 하는 작업의 리스트를 생성합니다. 데이터 분석과 분석 결과의 디스플레이 등에 필요한 사용자 인터페이스의 구성요소, 컨트롤과 인디케이터의 개수와 타입을 결정합니다. 미래의 사용자 또는 다른 팀 멤버와 함께 사용자가 함수와 기능에 어떻게 그리고 언제 접근하게 될 것인지 생각하고 토론합니다. 샘플 프론트패널을 만들어서 미래의 사용자 또는 팀 멤버에게 보여주어 해당 프론트패널이 사용자가 작업을 수행하는데 도움이 되는지 결정합니다. 이러한 대화식 과정을 통하여 사용자 인터페이스를 편리하게 만듭니다.

어플리케이션을 처리하기 쉬운 크기의 논리적인 단위로 나눕니다. 어플리케이션의 주요 구성요소를 포함하는 상위 레벨 블록다이어그램부터 시작합니다. 예를 들어, 블록다이어그램은 설정, 수집, 분석, 데이터 디스플레이, 데이터 로깅, 에러 핸들링에 대한 다른 섹션을 포함할 수 있습니다.

상위 레벨 블록다이어그램을 설계한 다음, 입력과 출력을 정의합니다. 그 다음, 상위 레벨 블록다이어그램의 주된 구성요소를 구성하는 **SubVI**를 설계합니다. SubVI를 만들고 테스트합니다. 상위 레벨의 테스트 루틴도 생성할 수 있습니다. 그러나 여러 VI를 가지고 있는 계층구조에서 테스트를 하는 것보다 작은 모듈에서 에러를 찾는 것이 더 쉽습니다. 상위 레벨 블록다이어그램의 초기 설계가 완벽하지 않을 수도 있습니다. SubVI를 사용하여 하위 레벨의 작업을 처리하면 사용자 어플리케이션을 수정하고 재구성하기가 쉬워집니다.

도움말 >> 예제 찾기를 선택하여 NI 예제 탐색기를 시작합니다. 예제 VI를 검색하면 VI의 블록다이어그램을 확인하고 SubVI의 예제를 볼 수 있습니다.

파일 >> 프로젝트 생성을 선택하여 사용자 정의할 수 있는 템플릿 및 일반 어플리케이션에서 사용할 수 있는 샘플 프로젝트의 리스트가 있는 프로젝트 생성 대화 상자를 시작합니다.

프로젝트 탐색기윈도우를 사용하여 **LabVIEW 프로젝트**를 관리할 수 있습니다.

모듈러 구성요소 디자인하기

모듈러 프로그래밍을 사용하면 블록다이어그램을 빠르게 변경하거나 디버깅할 수 있습니다. SubVI를 생성하여 일반적이거나 반복되는 작업에서 다시 사용할 수 있습니다. SubVI를 사용하여 쉽게 상위 레벨 블록다이어그램을 읽거나, 디버깅하고, 이해하고, 관리할 수 있습니다. 프로젝트를 진행할 때 설계 테크닉을 확인하는 것도 유용할 수 있습니다.

여러 개발자와 함께 프로젝트 설계하기

여러 개발자가 같은 프로젝트를 수행할 경우, 개발 과정과 어플리케이션이 서로 잘 진행되기 위해서는 시작 단계에서 프로그래밍의 책임, 인터페이스, 코딩 기준 등을 정의해야 합니다. 코딩 기준에 대한 추가적인 정보는 **LabVIEW Style Checklist**를 참고하십시오.

하나의 컴퓨터에 VI의 마스터 복사본을 보관하고 소스 컨트롤 정책을 지정합니다. 타사 소스 컨트롤 제공자와 함께 LabVIEW Professional Development System를 사용하여 LabVIEW 내에서 파일을 검토하고, 변경된 사항을 추적하고, **변경 사항을 병합**할 수 있습니다.

VI를 편집하기 전에 다른 플랫폼에서 마지막으로 편집된 모든 VI를 다시 컴파일하여 편집 세션 중의 로드 타임을 개선합니다. VI를 연 후, 실행 버튼을 클릭하는 동안 <Ctrl-Shift> 키를 눌러서 메모리의 모든 VI를 다시 컴파일합니다. (macOS) 실행 버튼을 클릭하는 동안 <Option-Shift> 키를 누르십시오. (Linux) 실행 버튼을 클릭하는 동안 <Meta-Shift> 키를 누르십시오. 그 후 파일 >> 모두 저장을 선택하여 메모리의 모든 VI를 저장합니다.

LabVIEW Style Checklist

Follow these recommendations to ensure consistent user interfaces, behavior, and quality among your LabVIEW projects. You can customize these checklists to fit the specifications of your application.



노트

The LabVIEW VI Analyzer Toolkit provides tests that check VIs interactively or programmatically for style, efficiency, and other aspects of LabVIEW programming. Refer to ni.com to learn more about this LabVIEW toolkit.

Front Panel Checklist

Set default values for controls and indicators. Default values enable a VI to execute as expected without requiring additional interaction.

Details



노트

Avoid setting default values for indicators such as graphs, arrays, and strings. Setting default values for these indicators uses unnecessary disk space.

Run the VI to verify that controls and indicators with default values do not cause the VI to fail.

If you set default values, you enable a function or VI to handle more options internally without requiring as much user interaction. For example, if you set a default value of 0 for the Write Delimited Spreadsheet VI, LabVIEW uses that default value to automatically present an empty file dialog and prompt for an input.

Use Size to Text for labels so the text rectangle automatically resizes to accommodate the text you enter. By default, Size to Text is enabled for one line of text.

Details

The three standard fonts, **application, system, and dialog**, vary depending on the platform you use. For example, preferences and video driver settings in Windows affect the size of the fonts. Text may appear larger or smaller on different systems. To compensate for font variations, allow extra space for larger fonts and enable the Size to Text option on the shortcut menu.

(VI Analyzer) Avoid overlapping the labels, digital displays, or other parts of controls and indicators. Allow for extra space between controls and indicators to account for font changes on different platforms. Overlapped controls and indicators slow down screen updates and can make the control or indicator flicker.

User Interface Front Panel Checklist

Design **user interface front panels** so that operations and results are clear and enhance the usability of the VI or application.

Avoid excessive use of color. **Use color logically**, sparingly, and consistently, if at all.

Disable the Abort button. Use the system X button or File»Exit option to safely shutdown resources instead of adding an additional exit button.

Details

The Abort Execution button stops the VI immediately and prevents the VI from finishing the current iteration. If you abort a VI that uses external resources, such as external hardware, resources may remain in an unknown state because they were not reset or released properly.

Hide the toolbar.

Use one consistent style for all controls. Use system or silver style controls when possible.

Use custom controls or strict type definition controls to customize the appearance of a control. Use strict type definition controls only to preserve the exact appearance of a control in each location of use.

(VI Analyzer) Use the standard application, system, and dialog fonts for all user interfaces in the application.

Details

Use the standard font unless you have a specific reason to use a different font. For example, you can use monospace fonts because they are proportionally spaced for string controls and indicators that constrict character use.

Configure numeric inputs to have data ranges that account for potential invalid values that a user may enter.

Arrange controls from top left to bottom right.

Details

Place controls in the order of typical use. For top-level VIs visible to users, place the most important controls in the most prominent positions. For subVI front panels, place the controls and indicators of the subVI so that they correspond to the connector pane pattern.

Organize UI elements by proximity, alignment, decorations, and UI features such as tab controls and subpanels.

Use an array style consistent with similar data structures.

Details

For example, remove the index control and add a scrollbar to make an array style consistent with listbox style.

Configure the front panel to [fit the monitor resolution](#) used in the application.

Use labels to describe the function of controls and indicators. Limit caption use to controls and indicators that require more information than the label conveys or when you want the ability to change the display text of a control programmatically.

Details

Here are some guidelines and examples of when to use captions.

Situation: If the control is visible to the user, use captions to display a long description and add a short label to prevent using valuable space on the block diagram.

Example: When you label a ring or slide control with options for volts, ohms, or amperes, selecting a caption such as "select units for display" is more usable than the label V/O/A. [Use Property Nodes](#) to change captions programmatically.

Situation: If the default value or unit is unusual, include default values and unit information in label names.

Example: Include the default value of no error in parentheses for the error in parameter.

Situation: If you use the VI with the control on multiple platforms, avoid using special characters in control names.

Example: Use degF instead of ° F because the ° symbol might not display correctly on all platforms.

Situation: If you use Boolean controls, use a name to give an indication of which state, TRUE or FALSE, corresponds to the function, such as "ON." Indicate the default state in parenthesis. For checkboxes and radio buttons, the user can click the Boolean text of the control, and the value of the Boolean control changes. Free labels next to a Boolean control can help clarify the meaning of each position on a switch.

Example: Use free labels such as Cancel, Reset, and Initialize that describe the user action.

Front Panel Dialog Checklist

(VI Analyzer) Use the system control style when possible. Using the system control style ensures that the dialogs you create look similar to the dialogs provided by the operating system you use.

Avoid using scroll bars on front panels that look like dialog windows. Design the dialog box to display all information. Use scroll bars on individual controls and indicators as appropriate.

If you use the title bar, provide a meaningful window title that explains the purpose of a dialog. Avoid using the default VI name as the window title because the VI does not effectively explain the purpose of the dialog.

If you use menus, only use menus that are specific to the dialog window. Avoid using default menus.

Details

Default menus provide more options than are necessary and make sense for the particular application. Customize the menu options for an application to enhance its professional appearance.

Limit dialog box use to critical user interactions.

Details

Use the system controls in dialog boxes to prompt and gather settings and configuration information from the user. You also can use tab controls to group front panel controls and indicators in a smaller area and reduce clutter in a dialog box.

Many programs also use dialog boxes to announce messages to the user. However, avoid overusing this type of dialog box. Use a status text window to display less serious warnings.

Make sure that all dialogs are either modal to the main application or floating. You can prevent a dialog from staying hidden behind other windows if the dialog is modal or floating.

Use the system panel background color for dialog user interfaces to maintain consistency.

Conditional Checklist

Write descriptions and create tip strips for UI front panels and controls to describe the purpose of an object and give users instructions for using the object.

Details

When you copy an object to another VI, you also copy the description. You may need to change the description for the new object.

Set up intuitive keyboard shortcuts and tabbing order for controls.

Details

Some users prefer to use the keyboard instead of a mouse. In some environments, such as a manufacturing plant, only a keyboard is available. Consider including **keyboard shortcuts** for VIs even if the use of a mouse is available because keyboard shortcuts add convenience to a VI.

Pay attention to the key navigation options for objects on the front panel and set the tabbing order for the objects to read left to right and top to bottom. Set the <Enter> key as the keyboard shortcut for the front panel default control, which is usually the OK button. However, if you have a multiline string control on the front panel, you might not want to use the <Enter> key as a shortcut.

If the front panel has a Cancel button, set the <Esc> key to be the keyboard shortcut. You also can use function keys as navigation buttons to move from screen to screen. If you do this, be sure to use the shortcuts consistently. Select Edit»Set Tabbing Order to arrange controls in a logical sequence when the user needs to tab between the controls. For controls that are offscreen, use the Key Navigation tab of the Properties dialog box to skip over the controls when tabbing or to hide the controls.

Also consider using the key focus property to set the focus programmatically to a specific control when the front panel opens.

Non-User Interface Front Panels

(VI Analyzer) Use the default front panel background color on all VIs.

(VI Analyzer) Use transparent front panel labels to align with current style. Transparent front labels also indicate that users should not interact with that front panel.

(VI Analyzer) Arrange controls and indicators relative to corresponding connector pane terminals.

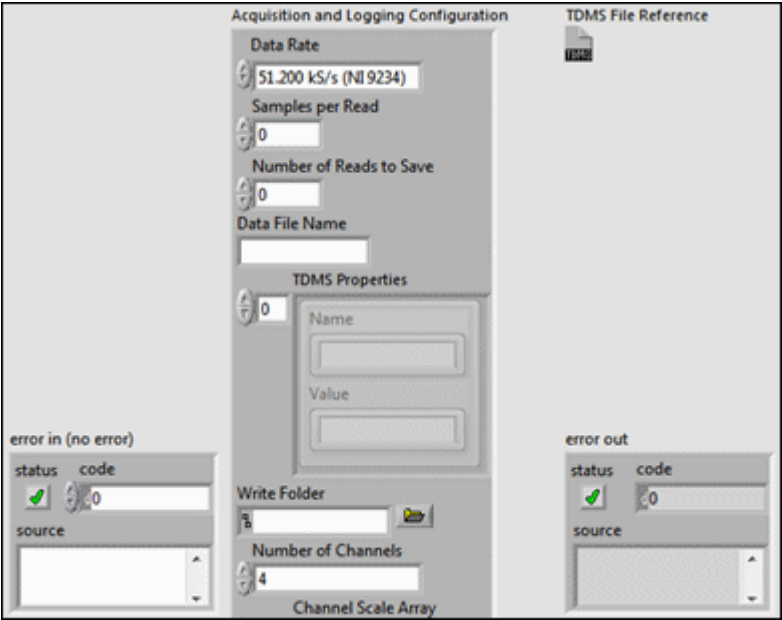
Use default colors.

(VI Analyzer) Size the front panel to fit the entire toolbar. Display the search bar for ease of use.

Group related input and output data into clusters. Make the cluster a type definition. If multiple VIs interact with the same clusters, consider using LabVIEW classes instead of clusters.

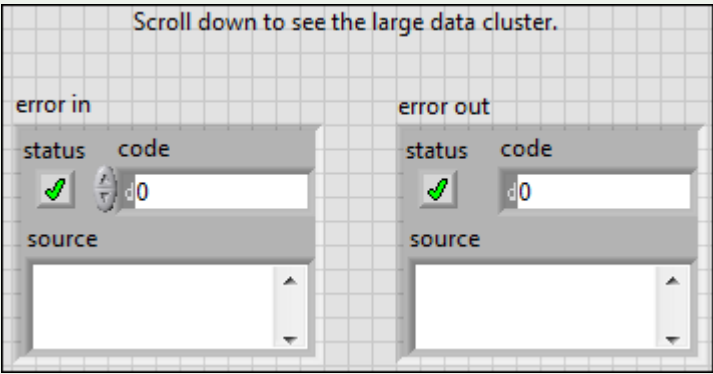
If a front panel contains a large and skinny type definition cluster, size the panel to fit existing regular controls and include part of the type definition. With a large and skinny type definition cluster, only the width fits in the Minimum Screen Resolution, not the height.

Details



If a front panel contains a large and fat type definition cluster, include the cluster below the visible screen and a note indicating that the user can scroll down to see it. With a large and fat type definition, neither width nor height fits in the Minimum Screen Resolution.

Details



Block Diagram Checklist

(VI Analyzer) Verify that data flows from left to right, unless you use feedback nodes, and that wires enter from the left and exit to the right.

Details

If you have a reason not to follow this guideline, provide a wire label or a floating comment to explain any exceptions.

Use **owned labels** to label long wires. Labeling wires is useful for wires coming from shift registers and for long wires that span the entire block diagram.

Organize the block diagram and draw wires according to **suggested design guidelines**. You also can **automatically clean up the wires** on the block diagram.

(VI Analyzer) Arrange objects symmetrically on the block diagram by aligning and distributing functions, terminals, and constants evenly.

Details

Use the Align Objects and Distribute Objects pull-down menus on the toolbar to arrange objects the block diagram. When objects are aligned and distributed evenly, you can use straight wires to connect objects together. Using straight wires makes the block diagram easier to read.

(VI Analyzer) When possible, avoid leaving front panel terminals unwired on the block diagram

Avoid coloring diagrams. Color can distract users.

Use default colors for structures, such as Case or Event structures.

Avoid using Flat Sequence structures unless you must use them to define execution order.

If possible, arrange parallel loops from top to bottom to represent logical data flow.

Place control and indicator terminals outside of any loops or structures unless values need to be read or updated continuously. Align control terminals to the far left and indicator terminals to the far right of the block diagram.

Make sure the subVI icon, rather than the connector pane, is visible on the block diagram.

Details

The subVI icon is more helpful than the connector pane on the block diagram. The VI icon reveals the purpose of the VI. A connector pane only displays how inputs and outputs are connected. Right-click the subVI and select Visible Items. Verify that the Terminals option does not contain a checkmark.

(VI Analyzer) Display the terminal for front panel objects on the diagram, not the icon, to conserve space and align with current LabVIEW style. Navigate to Tools»Options to change the default for this option. You can make exceptions for type definitions. Type definition icons are unique and can display useful information about the VI. To make an exception for a type definition icon, right click the icon and select Show as Icon.

(VI Analyzer) Avoid creating a string constant that contains a single, difficult to read value, such as a space, if the value within that constant is meaningful.

Details

Use the built-in LabVIEW string constants. For example, you can use an empty string as a constant to specify the data type of the elements the queue holds.

Use the Bundle by Name and Unbundle by Name functions when you access clusters. Avoid using the unnamed Bundle and Unbundle functions.

Details

The Bundle by Name and Unbundle by Name functions reference elements of a cluster by name instead of cluster order. You can eliminate the challenge of deciphering the purpose of each data element by referencing elements of a cluster by name. You also can reorder elements or add new elements to a cluster without potentially causing a VI to break.

If the value of a large type definition constant is the same as the default values of the elements within the type definition cluster, display the type definition cluster constant as an icon on the block diagram.

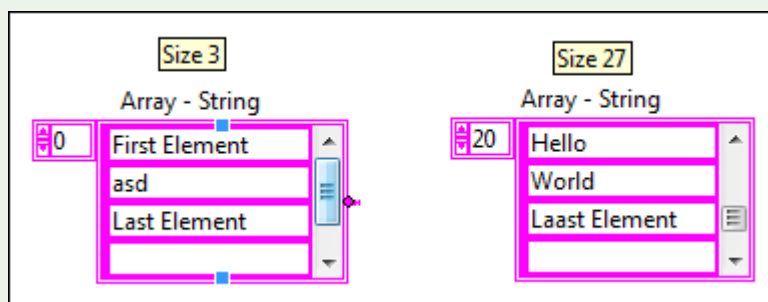
Details

Otherwise, use the Bundle by Name and Unbundle by Name functions to specify the values of the cluster. If many elements differ from the default value, wrap the Unbundle by Name or Bundle by Name functions and associated values in a subVI and inline the subVI.

To save space on the block diagram, avoid displaying more than three or four array elements.

Details

For large arrays that contain many elements, show only three or four elements and update the index so that the last element and first empty element are visible. Also, display the scrollbar for large arrays.



Set conditional structures such as Case structures and Event structures wide enough to view the full condition text. You can make an exception for multiple conditions in the same frame. If you have multiple conditions in the same frame, you can opt to not increase the display size of the structure to show all conditions.

Right click a Property Node or Invoke Node and select Name Format»Short Names to make the block diagram more readable. Short names use less space.

Details

The Long Names option may be appropriate in certain instances. For example, use the Long Names option for an element if that element shares the same short name with another element. Avoid using the No Names option. The No Names option does not provide enough information for the user to understand elements or how the code functions. Apply the same reasoning for similar nodes, such as the Invoke Library Node and IP Integration Node.

Save each VI with the most important frame of multi-frame structures displayed. For example, Case and Event structures contain multiple frames.

(VI Analyzer) Remove unused code.

LabVIEW Classes

Name all classes as nouns. For example, name an example class Waveform Reader.lvclass rather than Reads Waveforms.lvclass.

When **designing classes**, follow good software engineering practices from other programming languages.

Details

For example, avoid inheriting one concrete class from an existing concrete class. Instead, create an abstract class as the common parent to both of the concrete classes.

Name all member VIs of a LabVIEW class as verbs. For example, name an example member VI Read Value VI rather than Value VI.

Avoid including the name of the class in VI names. For example, if you name a class Special Waveform.lvclass and have a VI that adds two such waveforms together, name that VI Add VI rather than Add Special Waveforms VI.

Use the word "Read" to begin the name of any VI that strictly reads data from the private data of a class. Avoid using the word "Get" or any synonym of "Get."

Use the word "Write" to begin the name of any VI that strictly bundles data into the private data of a class. Avoid using the word "Set" or any synonym of "Set."

Consider supporting the syntax of Property Nodes for **data accessor** VIs. However, accessors that use the syntax of Property Nodes require error terminals and cause significantly more performance burden at run time than data accessor VIs without error terminals. Evaluate and choose appropriately for each class.

Mark all VIs as private unless you have a specific reason to mark them as public, protected, or community.

Avoid creating excessive wire patterns and colors if you change the wire appearance of a LabVIEW class wire

Details

The number of colors and patterns in the block diagram window can affect the readability of LabVIEW code. Change the wire appearance of LabVIEW class wires only if necessary to differentiate between other class wires in the block diagram window.

Use dark colors on the exterior of class wires to provide contrast against white background.

If the private data of a class contains only refnum types, use the standard refnum color for the class wire. Locate the color named Refnum Wire in the User Colors dialog box for the LabVIEW color picker.

Details

Best practice requires that any class containing one refnum type should only contain refnum types as elements. Although you can include a mixture of data value reference types and by-value types in a class, the mixture can demonstrate ambiguous behavior when you fork the wire on the diagram. Data value reference types refer to the original element. By-value types create copies of the original element instead of referring directly to the original. Mixing the two types can cause confusing behavior.

Labels and Comments

(VI Analyzer) Use visible and meaningful labels for all terminals.

Use a specific and detailed label or **attached comment** to document constants that contain non-obvious values

(VI Analyzer) Use transparent labels.

Use consistent capitalization for labels.

Use spaces between words in a label.

VI Analyzer) Avoid using duplicate labels.

If an example subVI contains an input and output pair, add a suffix of in to the control and a suffix of out to the indicator. For example, if a VI includes an "input" called object, name the corresponding output "object out."

Always include the word "in" when creating an error in input.

Use parenthesis to document the default value and units of a control or indicator.

Details

Avoid using parenthesis for any other purposes. In most cases, you can exclude the default value if it is the same as the default value for the given data type.

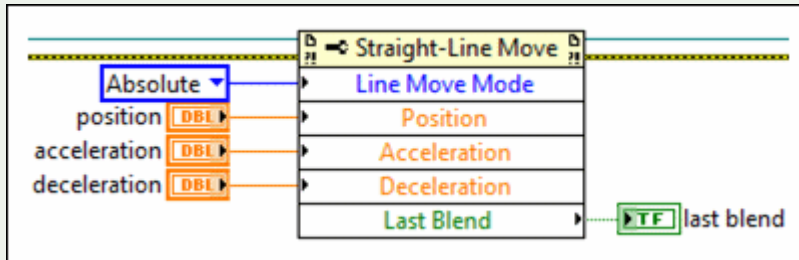
Attach free labels on the block diagram to explain code. Use free labels to explain why the code executes the way you configured it to execute, not how the code executes.

Label shift registers and wires when the content is potentially unclear or confusing.

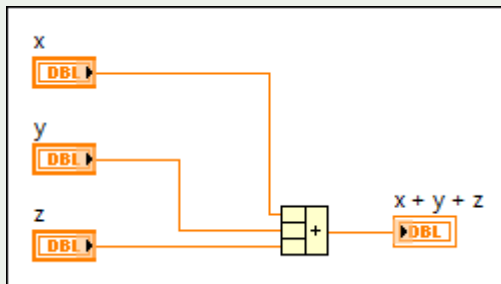
Use one of two styles consistently for the placement of terminal labels.

Details

For the first style, place terminal labels to the left of the terminal for controls and to the right of the terminal for indicators.



For the second style, left-align terminal labels above the terminal. Use the style you choose consistently within a VI, a library, or a project.



(VI Analyzer) Use carriage returns in free labels only to separate paragraphs.

(VI Analyzer) Unless a free label occupies a single line, avoid using the Size to Text method or property for these labels. Instead, size free labels so that all the text is visible. Also, provide extra space at the bottom of a free label to account for font size changes across platforms.

Use the standard application font for free labels on all block diagrams.

Details

Use the application font on the block diagram unless you have a specific reason to use a different font.

Use the default free label color for all comments. You can make exceptions under special circumstances, such as sample projects in LabVIEW 2012 and later. LabVIEW 2012 and later use a new style precedent for comments that display different types of information.

For subdiagram labels, use the default appearance for all structures, such as Case and Event structures.

For each VI, such as API VIs and examples, designed for third-party use, describe what the VI does in the VI Description text box. Consider describing all VIs in the VI Description text box.

Details

To test VI the documentation you create for a VI, write and revise a VI description and confirm that the description displays correctly in the Context Help window. Describe how to use the VI and any required parameter settings in the VI Description text box on the Documentation page of the VI Properties dialog box.

Check the spelling of all words in the VI Properties dialog box and on the front panel and block diagram.

Size and Location

To create a visually appealing block diagram, avoid both unnecessary white space and overcrowding of the block diagram. Provide just enough white space for visual guidance.

Avoid creating large block diagrams.

Details

The size of the block diagram window can affect the readability of LabVIEW code. Size the block diagram window no larger than the screen size. Code larger than the window is difficult to read because you must scroll through the window. If the code is too large to fit on one screen, configure the screen to scroll only in one direction to display the rest of the code. If the block diagram requires scrolling, consider using subVIs.

Size the block diagram large enough to fit the entire toolbar. For example, size the block diagram large enough to show the search bar.

Programming Checklist

Use type definitions, not strict type definitions, to preserve the data type of controls used in more than one place, especially for enumerated type controls and data structures.

Details

Use a type definition when you use the same unique control in more than one location or when you have a very large data structure passing between several subVIs.

By using a **type definition** control, LabVIEW automatically propagates changes to the control or the data structure throughout all the VIs and subVIs.

Use enumerated type controls if possible.

Details

If you use a Boolean control for two options, consider using an enumerated type control instead to potentially expand options in the future.

Use an **enumerated type** over a ring control.

If you must use a ring control, configure it as a strict type definition so that all menu items update for each use. If you do create a strict type definition, extend the size beyond the longest words to allow for easier expansion of future options. Ring controls allow you to easily interact with front panels if you want to programmatically change the string labels. You also may want to use a ring control instead of a Boolean control. If you change the control to include more than two options, you can easily add options to a ring control.

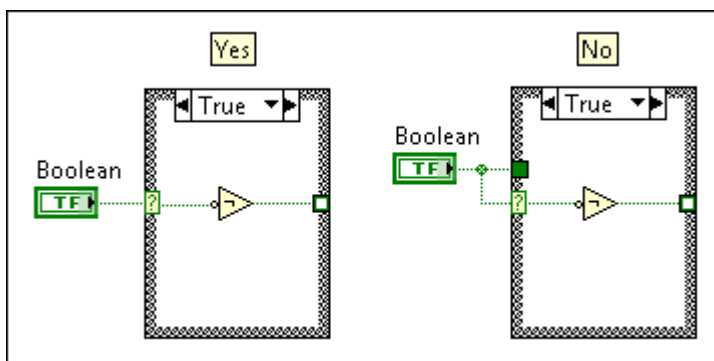
You cannot programmatically change the string labels in an enumerated type control at run time because the string labels are a part of the data type.

When you use enumerated type controls, always make a type definition of the control. Creating type definitions prevents you from needing to rewrite the code each time you add or remove an item from an enumerated type control.

Enumerated type controls also help make block diagram code easier to read because when you wire an enumerated type control to a Case structure, the string labels appear in the selector label of the Case structure.

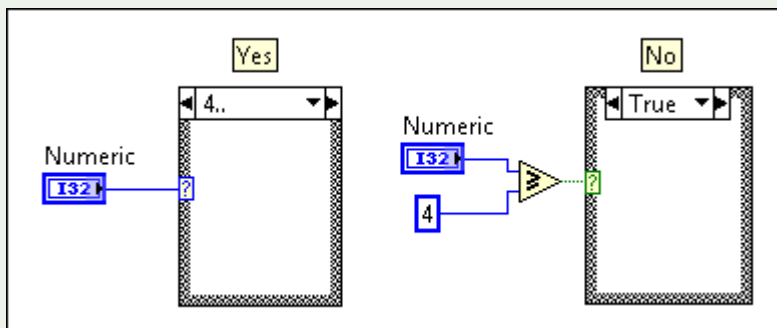
When you use a Case structure, avoid creating a separate tunnel for the wire connected to the selector terminal. Instead, wire the selector terminal inside the structure.

Details



Avoid placing Boolean outputs prior to a Case structure. Use non-Boolean selectors instead. If you place Boolean logic inside the structure, you are less likely to encounter a mistake.

Details



Eliminate the default case for Case structures wired to enums.

Details

If you add a new element to an enum wired to the default case of a Case structure, the code executes the default case and can cause an error.

Avoid using named queues and named notifiers.

Details

If possible, pass the reference wire in place of using named queues and notifiers. If you must use named queues and named notifiers, avoid name collisions.

If you open a reference, such as an application, control, VI, or file, close the reference by using the Close Reference function. Close the reference in the same VI in which you opened the reference.

Details

You can use subVIs for opening and closing references. If you use subVIs to open or close a reference, pair the subVIs up. Use VI names and icons that reflect these pairings. For example, use pairs of words such as "Open" and "Close" or "Obtain" and "Release."

Use path controls or constants instead of string controls or constants to specify the location of files or directories.

Details

Use a path control or indicator to specify the location of a file or directory. Unlike strings, path controls and indicators format paths using the standard syntax for the platform you use.

Set the browse options appropriately for the Browse button of path controls. For example, if you need to select a directory, select the Folders only and Existing only options on the Browse Options page of the Properties dialog box.

Use a path constant and the path data type to supply a constant path value to the block diagram. The path constant and data type use the platform-specific notation for paths, unlike the string constant and data type.

Avoid using absolute paths in VIs.

Details

When possible, either use relative paths and define a root path in your code or determine the root path dynamically at run-time based on system settings or the application location. Ensure that relative file paths work for the operating system on which the code is intended to run.

Absolute paths can cause problems when you build an application or run the VI on a different computer. If you must use an absolute path, ensure that you include code to both test that the path exists and to create a path if one does not exist.

Avoid using a variable when you can use a wire to transfer data. Every local variable that reads the data makes a copy of the data. Use **global and local variables** only when you cannot use a wire. More specifically, use wires, data value references, or classes instead of functional global variables because you can follow and debug wires more easily than functional global variables.

When applicable, use the In Place Element structure to clearly convey the movement of data.

Avoid duplicating code. Use subVIs to encapsulate and modularize different parts of your application. Create each subVI to have a single clear purpose so that the subVI is easy to reuse and test.

Create test VIs that check error conditions, invalid values, and cancel buttons. Save test VIs in a separate directory so you can reuse them. You also can use the Unit Test Framework Toolkit to perform unit testing on VIs.

Review the VI for efficiency, data copying, and accuracy, especially the parts of the VI without data dependency. Use the Show Buffer Allocations window to determine where LabVIEW allocates memory.

Corner Cases Checklist

Avoid race conditions by using appropriate **synchronization** techniques.

Details

You can use synchronization techniques such as queues, data value references, and semaphores to prevent race conditions. You also can use functional global variables to prevent race conditions if the functional global variable protects a critical section containing actions that modify data. If a functional global variable protects sections containing actions that only read and write data, you cannot use the functional global variable to prevent race conditions.

Handle all unsupported values appropriately.

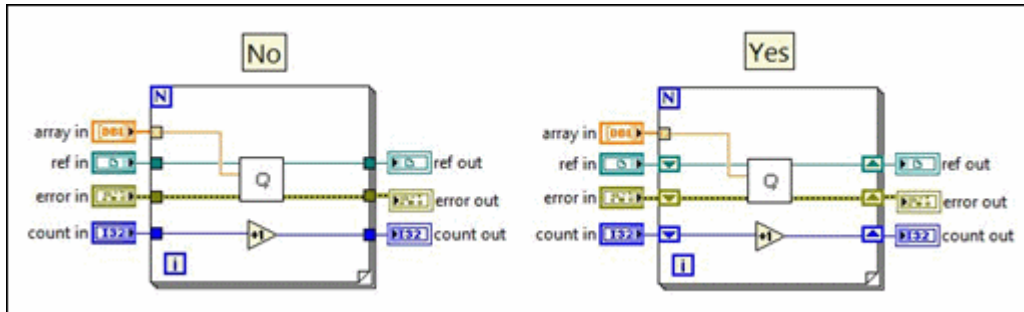
Details

For example, if you create an API, consider returning an error for unsupported inputs. If you create a user interface, configure the input ranges of controls.

Ensure that For Loops behave as expected if they execute zero times.

Details

Use a shift register if necessary to pass a value through the loop in this situation. If you wire a value or a reference to pass through a For Loop, **use shift registers instead of tunnels**.



Only select Use Default if Unwired on tunnels of Case and Event structures when confident that adding new cases or events in the future will not introduce a bug in the program. When in doubt, avoid using this feature

(VI Analyzer) Verify that you can easily identify the number of times a For Loop executes. Avoid using both Auto-indexing and the N terminal to govern iterations of the same For Loop. If you use more than one of these options, you can cause confusion about how many times the loop executes. Also, be cautious when auto-indexing on multiple arrays of different sizes.

(VI Analyzer) Avoid creating clusters that contain multiple elements of the same name.

Take special care when creating VIs to execute on multiple platforms.

Details

If possible, avoid using VIs that only execute properly on a specific platform. Instead, place platform specific functionality inside of a conditional disable structure specific to a target.

VIs that contain Call Library Function Nodes are platform specific. VIs that call shared libraries using a Call Library Function Node are platform dependent and only work on different platforms if you have a corresponding library for each platform. If you move a VI that contains a Call Library Function Node to another platform, you must update the node to specify the shared library compiled for the new platform.

You also can use the Conditional Disable structure to configure platform-specific code. Configure these VIs as source only to increase compatibility with source control.

If you use an infinite timeout, provide a mechanism to programmatically abort the wait.

Details

Avoid using arbitrary timing delays and assuming that important events execute during those delays. Also, avoid proceeding without verifying that a relevant event occurred. If you must use a delay, provide a comment about how you determined the duration of the delay.

Error Handling Checklist

Handle errors **appropriately**.

Details

Carefully consider every place you report a potential error inside the code. Add comments to explain any potentially confusing decisions about error placement.

Use a consistent error handling strategy. Make sure the program can deal with all error conditions and invalid values. Determine the actions for errors, such as log, terminate, and notify user.

(VI Analyzer) Disable automatic error handling for all VIs, including new VIs. Automatic error handling stops execution in the event of an error and waits on an input before continuing. This automatic stopping can cause problems in a deployed system or application.

Give priority to errors that occurred first. These errors likely caused any subsequent errors.

Details

When passing errors through loops, use shift registers instead of tunnels to maintain warnings. Give the first errors priority when merging errors unless there is a specific reason for not doing so.

Use a Case structure for VIs that include error in and error out parameters to distinguish between the error and no error conditions.

Details

Make exceptions for code that should execute even upon encountering an incoming error or for VIs that need performance optimization in the No Error case. Use VI documentation and/or comments to document exceptions.

Consider creating a custom error file to contain all the error descriptions for an application.

Performance Checklist

Always keep performance in mind when designing a VI. When you must decide between additional performance, ease of development, or code maintenance, choose the tradeoff most appropriate for the specific project.

(VI Analyzer) Unless necessary, avoid including code that reallocates memory inside of loops, such as code that contains the Build Array or Concatenate Strings functions. Reallocating memory inside of loops can cause slow performance.

(VI Analyzer) Avoid unnecessary type coercions. Type coercions can allocate memory and cause slow performance.

Avoid unnecessary data copies, especially if the data is large.

Consider placing code that opens and closes resources, such as DAQ or File I/O code, outside of loops. Repeatedly opening and closing references inside of a loop is inefficient.

(VI Analyzer) Include code to limit the execution speed of continuously running loops.

Details

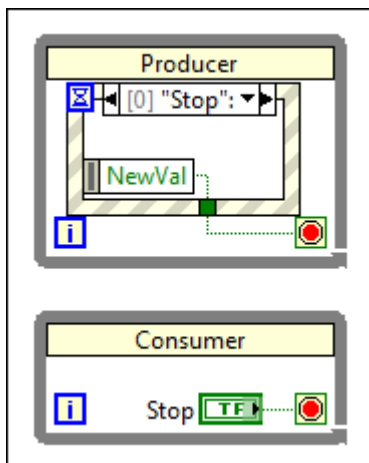
By default, While Loops execute as quickly as possible and use all available CPU resources during execution. You can use a blocking mechanism, such as a Wait function or timeout value, to make sure a While Loop executes only as often as needed to preserve system resources.

(VI Analyzer) Use an Event Structure to monitor UI interaction where appropriate.

Details

Avoid polling front panel controls when you can use a Value Change event in an Event structure more appropriately. Avoid using time consuming code inside of the Event structure. Consider using something similar to producer-consumer pattern in the application. Place the control terminal for a Value Change event configured for a control in the Value Change event case.

The following illustration shows a situation in which you can acceptably place the control terminal as a stop button outside the event structure.



(VI Analyzer) Resize Event Data Nodes to contain only data being used. If the node is not used, resize the node to one element and move it to a corner of the frame.

Conditional Checklist

(VI Analyzer) Use inlining and subroutine priority strategically to improve **execution speeds**.

(VI Analyzer) Consider configuring **parallel For Loop iterations** to make minor adjustments to performance.

(VI Analyzer) Use a local variable instead of the Value property to update a front panel object.

(VI Analyzer) If you perform many updates to the front panel, use the Get Control Values by Index function and Set Control Values by Index functions.

(VI Analyzer) Use the **In Place Element Structure** for Array Index / Replace Elements operations and for cluster read, modify, and write operations.

SubVIs Checklist

Design each subVI with low coupling and high cohesion. This helps ensure each subVI is easy to understand, maintain, and test. Organize related groups of subVIs into libraries. Avoid using shared libraries (DLLs).

(VI Analyzer) Create a meaningful icon for subVIs.

Details

Avoid using the default icon or a blank icon. Design each new icon as distinct and different from each existing icon. The icon represents the VI on a palette and a block diagram. Well-designed icons give you a better understanding of the subVI without requiring excessive documentation.

(VI Analyzer) Make good use of reusable, testable subVIs.

(VI Analyzer) Use error in and error out clusters in all subVIs.

Details

Include error in and error out clusters in all subVIs, even if the subVI does not process errors, because error in and error out clusters help control execution flow. If a subVI has an incoming error, you can use a Case structure to send the error through the VI without executing any of the subVI code.

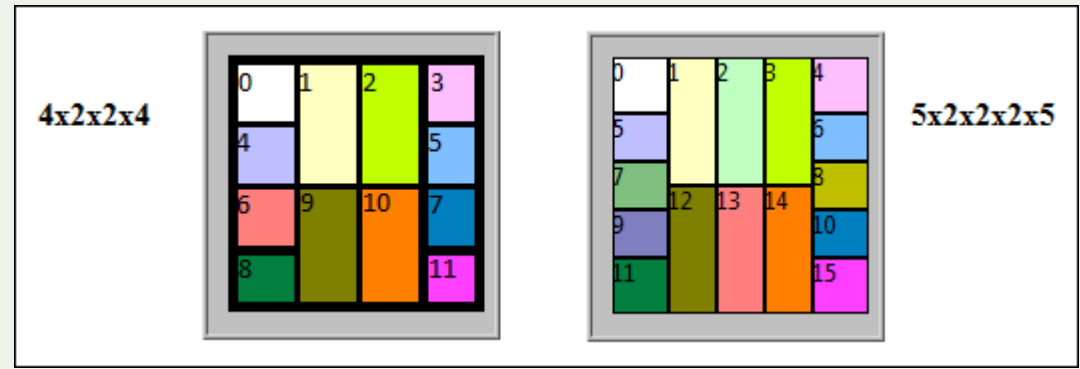
API Design Checklist

Use the same connector pane for related VIs. Wire the connector pane **according to best practices** for LabVIEW programming.

(VI Analyzer) Use either the 5x2x2x2x5 (5x3x3x5) or the 4x2x2x4 connector pane pattern.

Details

Maintain a consistent connector pane pattern with a library, project, or development team. Avoid using connector panes with more than 16 terminals. Use the 4x2x2x4 pattern if you work frequently with core LabVIEW VIs. Use the 5x2x2x2x5 pattern if you work frequently with libraries that use this style, such as NI-DAQmx or LabVIEW FPGA VIs.



Use the left terminals for controls and right terminals for indicators. You can also use the top and bottom terminals as long as you place all controls to the left of indicators.

Place related controls and indicators at the same relative position on the connector pane. For example, if you assign a data-in control to the middle left terminal, assign the data-out indicator to the middle right terminal.

If a control or indicator serves the same function throughout an API, name the control or indicator consistently and place it in the same position throughout all API connector panes.

(VI Analyzer) Place paths, references, and LabVIEW Classes in the top corners.

(VI Analyzer) Assign error in and error out terminals to the corresponding bottom left and right terminals (terminals 8 and 11 on the 4x2x2x4 connector and terminals 11 and 15 on the 5x2x2x2x5 connector).

(VI Analyzer) Preserve empty terminals on the connector pane to allow for future expansion.

Set inputs terminals for parameters that must be wired for the VI to function as required.

Set input terminals with good default values as recommended.

Use optional terminals only when needed.

Use Required, Recommended, and Optional settings on the connector pane.

Details

The Required, Recommended, Optional setting for connector pane terminals affects the appearance of the inputs and outputs in the Context Help window and prevents users from neglecting to wire subVI connections. Use the Required setting for inputs that users must wire for the subVI to run properly. Use the Optional setting for inputs that have default values that work for the subVI most of the time. The VI stills functions you leave these inputs unwired.

Use a consistent banner or glyph for API icons. An icon template will help with this. More details on icon style can be found [here](#).

Avoid using text in VI icons, especially for LabVIEW class public and dynamic dispatch VIs. Users who speak other languages may have difficulties understanding the text.

Place APIs in a library to improve organization.

Mark any VIs in a library as private if they are not part of the API.

Use the following suggestions to create icons.

Details

- Use the icons in the LabVIEW libraries as prototypes. When you do not include a picture for an icon, you can use text. If you localize the application, also localize the text on the icon. Select a user friendly font and font size, such as 8 point Small Font in all caps.
- Create a unified icon style for related VIs to help users visually understand which subVIs are associated with the top-level VI.
- Always create standard size (32 × 32 pixel) icons. VIs with smaller icons can be awkward to select and wire because of the variation in size.
- Avoid using colloquialisms in icons. Colloquialisms are difficult to translate. Also, consider cultural and language difference when you create icons. Certain pictures may not translate well. For example, avoid representing a data logging VI with a picture of a tree branch or a lumberjack.

Organization Checklist

Organize all files **logically**. Give VIs meaningful names without special characters, such as backslash (\), slash (/), colon (:), and tilde (~). Select a VI name that clearly and accurately represents the VI function.

Details

When naming VIs, LLBs, and directories, avoid using characters not accepted by all file systems, such as backslash (\), slash (/), colon (:), and tilde (~). Most operating systems accept long descriptive filenames up to 255 characters.

If you make backup copies of files, **save them** in a directory outside of where LabVIEW typically searches to prevent LabVIEW from loading these file copies into memory when you open development VIs.

Use full words with spaces for the names of all files for clarity. Use "Title" case for each word in file names.

(VI Analyzer) Use virtual folders to delineate between public and private members of a library.

Save custom controls with the same name as the label. For example, Alarm Boolean.ctl has the default label Alarm Boolean.

Work within the maximum path length restrictions on Windows, which is 260 characters.

Project Organization Checklist

Place the top-level VIs for the project directly under appropriate targets. Because you cannot flag a VI as the top-level VI, placing the top-level VIs under the corresponding target helps clarify which VIs are top-level.

Organize VIs in a hierarchical directory with easily accessible top-level VIs and subVIs in subdirectories. Limit the number and level of directories in a project.

Place all VIs in libraries, classes, or virtual folders. Avoid using auto-populating folders unless created code for a plug-in architecture that allows you to call other code. Auto-populating folders display code from the directory where plug-ins are installed.

Remove all unused files.

Details

Right-click the project root in the Project Explorer window and select Find Items with No Callers from the shortcut menu to display the Find Items with No Callers dialog box. This dialog box identifies all the items you can remove.

Distributing Source Code Checklist

Save all VIs and projects in the upper-left region of the primary monitor.

Save all VIs with the block diagram slightly lower than the front panel window bar and slightly to the right of the front panel scrollbar so that both windows are accessible from the window bar when open.

Avoid saving a VI with a maximized front panel or block diagram to make sure the VI loads in a predictable place.

Save each VI with the most important frame of multi-frame structures, such as Case or Event structures, displayed.

(VI Analyzer) Remove all breakpoints before distributing VIs to prevent unintended and unwanted halts in execution.

(VI Analyzer) Remove the revision history of a VI to remove comments.

(VI Analyzer) Disable debugging to improve performance.

RT-Specific Checklist

Whenever possible, use while loops, not timed loops, for non-deterministic tasks.

In your documentation, use the term "deterministic" instead of "time critical" to describe code running in timed loops. The term "time critical" is commonly recognized as a VI priority you can use to schedule parallel tasks.

Whenever possible, use timed structures instead of a deterministic **priority** setting on SubVIs.

Serialize code within timed structures so that it executes in a defined, sequential order.

To minimize subVI overhead, use the inline property for VIs in timed structures.

When you use subVIs in timed loops that access a shared resource, consider using subroutine priority and right-clicking a subVI to select the Skip Subroutine Call If Busy option.

Avoid placing controls or indicators within timed loops because they execute slowly and non-deterministically.

With timed loops, merge inner left ear error with error input to catch any configuration errors. Also merge outer right ear error with error outputs.

Make sure to properly handle errors in time-critical code.

If possible, use the Select function instead of a Case structure in deterministic code. The Select function provides greater determinism.

Check all time-critical loops for [lateness](#).

Handling Data Checklist- RT

If file IO performance matters, establish the best chunk size for writing and reading. Note that this may vary by target.

Use RT FIFO functions instead of RT FIFO enabled [Shared Variables](#) for streaming data. However, you can use RT FIFO enabled shared variables for current value communication.

For critical data, use the Flush File function to flush the file frequently. Reliance and FAT only flush once per second.

Limit the size and quantity of log files to improve system reliability. A directory with a large and growing number of files can grow slow and demonstrate changes in system behavior. As a result, you may encounter unexpected behavior.

Initialization and Shutdown Checklist- RT

Implement shutdown code to stop anything initialized, even if the next step calls for a reboot.

Avoid writing to configuration files from within the application. Writing to configuration files can corrupt the file and prevent you from properly initializing the next reboot. If you must write to a configuration file, develop a strategy to recover from or limit the damage of corruption.

Place code that reboots a controller in a conditional disable structure so you can easily disable the reboot at the end of a program. If you place reboot code in a conditional disable structure, you can disable the code when debugging and enable the code when deploying the application.

Miscellaneous-RT

Use [Timing VIs](#) on the RT palette.

Monitor the processor and memory usage for the system from development through deployment.

Details

Ensure processor usage does not stay high long enough to cause unacceptable thread starvation when a thread does not get time to execute. Verify that memory usage does not grow continually and that a plan exists to recover in case memory usage does grow continuously.

FPGA-Specific Checklist

When you plan how to use FPGA resources, keep in mind that top-level VI controls and indicators are resource intensive.

If you use a node that has an error output, include code to handle the error immediately.

Details

Localized error handling in FPGAs typically results in lower FPGA resource utilization because the error cluster in FPGA consists of 33 bits, one bit for status and 32 for the error code. You can often reduce 33 bits down to one.

Avoid writing to hardware outputs inside of a Case structure, especially when inside of a [single-cycle Timed Loop](#). Explicitly define hardware outputs for each case and write to hardware outputs outside of the Case structure.

Monitor the timeout output of DMA FIFO reads and writes and respond accordingly.

Details

Also monitor the timeout output of target-scoped FIFOs unless you either do not care if a timeout occurs or are certain a timeout will never occur. To handle the length of timeouts, either latch the output and notify the host that the FPGA should be restarted to ensure data integrity or use a handshaking protocol to handle resending data.

Use 4-Wire handshaking anywhere appropriate, and be sure to label the wires, such as "input valid," "ready for output," "output valid," and "ready for input".

Consider converting fractional data types to the single data type before passing them up to the host.

FPGA Host Interface

Consider calling the Reset and Run methods specifically instead of automatically running the VI from the [Open FPGA Reference](#) function. This ensures the FPGA is in a known state before it executes.

When reading from **DMA FIFOs** do not reallocate memory or poll the CPU.

Details

- Consider using the Configure method to set an appropriate host-side buffer. Avoid using large FPGA size buffers when possible to preserve block RAM.
- Set the DMA Read timeout to 0.
- Read a fixed-size number of elements (a multiple of the number of data channels).
- Wait until the buffer is full before reading elements.

Use the **Dynamic reference** type when creating an API that communicates with an FPGA.

LabVIEW의 소스 컨트롤

LabVIEW에서 소스 컨트롤 사용하기

소스 컨트롤 제공자를 사용하여 여러 개발자 사이에서 파일을 공유하고 보안과 품질을 향상시키며, 공유된 프로젝트의 변경 사항을 추적할 수 있습니다. 타사 소스 컨트롤 제공자와 함께 LabVIEW를 사용하여 LabVIEW 내에서 파일을 검토하고 변경된 사항을 추적할 수 있습니다. 도구 » 소스 컨트롤을 선택하여 LabVIEW에서 일반 소스 컨트롤 기능에 접근합니다. LabVIEW 프로젝트의 파일이나 개별적인 VI에서 소스 컨트롤을 사용할 수 있습니다.



노트

소스 컨트롤은 Professional Development System에서만 사용 가능합니다.

소스 컨트롤은 소프트웨어 프로젝트 관리를 도와주고 여러 개발자와 여러 소프트웨어 프로젝트 사이에서 파일 공유를 용이하게 합니다. 소스 컨트롤 제공자는 프로젝트 파일에 대해 집중된 마스터 복사본을 유지합니다. 변경할 것이 있으면, 마스터 복사본을 업데이트하여 그 변환된 내용을 반영합니다. 이것은 모든 개발자가 가장 최근 버전의 프로젝트 파일에 접근하기 쉽도록 해줍니다. 또한, 소스 컨트롤 제공자는 모든 코드를 쉽게 접근할 수 있기 때문에 코드 재사용을 가능케 합니다.

소스 코드 컨트롤로 알려진 소스 컨트롤은 보안과 품질을 향상시키는데도 도움을 줍니다. 파일을 변경하기로 결정하였다면 그것을 편집하기 위해 **파일을 체크아웃**합니다. 파일의 변경을 끝냈으면, **새로운 버전의 파일을 체크인**하여 이것을 소스 컨트롤 프로젝트의 마스터 복사본의 일부가 되도록 합니다. 올바르게 않은 변경을 체크인하였다면, 대부분의 소스 컨트롤 제공자는 이전 버전의 파일 접근을 허용하여 개정을 취소할 수 있습니다.

소스 컨트롤 제공자는 소프트웨어 프로젝트에 대한 변경 사항을 추적하는데 도움을 줍니다. 파일을 체크인할 때, 제공자는 사용자에게 변경 사항을 기술하라고 입력 요청합니다. 소스 컨트롤 제공자는 이 정보를 유지하여 프로젝트의 향상을 명확하게 문서화합니다. 소스 컨트롤에서 각 파일에 대한 개정 히스토리를 볼 수 있습니다.



팁 **정렬 옵션**을 사용하여 프로젝트 내의 아이템을 구성할 수 있습니다. 선택한 정렬 옵션은 디스크 상에 있는 프로젝트의 구성을 변경하지 않습니다. 따라서 프로젝트를 소스 컨트롤에 제출할 때 통합하고 비교해야 하는 변경 내역이 더 적습니다.

소스 코드를 유지하는 것에 덧붙여서 소스 컨트롤 제공자는 소프트웨어 프로젝트의 다른 측면을 관리할 수 있습니다. 예를 들면, 소스 컨트롤 제공자를 사용하여 스펙과 다른 문서에서 만들어진 변경을 추적할 수 있습니다.

LabVIEW와 함께 사용하려는 **타사의 소스 컨트롤 제공자를 선택**한 후에 그 제공자에 대한 **LabVIEW 셋팅을 설정**할 수 있습니다.

LabVIEW에서 이용할 수 없는 소스 컨트롤 작업을 수행하려면 그 특정한 함수를 위해 소스 컨트롤 제공자로부터 직접 작동시킵니다.

소스 코드를 관리 및 컨트롤하는 방법에 관한 더 자세한 정보는 ni.com의 Software Configuration Management and LabVIEW 문서를 참조하십시오.

소스 컨트롤 제공자 선택하기

LabVIEW는 몇가지 타사 소스 컨트롤 제공자를 지원합니다. LabVIEW에서 이용 가능한 소스 컨트롤 동작은 어느 타사 제공자를 선택하든 같습니다. 각 동작에 대한 특정한 지원이나 기능은 제공자에 따라 다릅니다. 더 이상 LabVIEW 내에서 Perforce 설정 파일을 생성할 수 없습니다.

어떤 경우에 사용자의 회사가 특정 어플리케이션에 표준화되어 있기 때문에 특정한 소스 컨트롤 제공자를 사용하기로 결정할 수 있습니다. 그렇지 않은 경우, 사용자는 파일의 관리를 위해 어느 제공자를 사용할 것인지 결정해야 합니다. 특정한 제공자를 사용해야 하는지 알려면 사용자 회사의 소스 컨트롤 관리자에게 문의하십시오.

소스 컨트롤 제공자를 선택하고 설치한 후에 그 제공자와 작동하는 **LabVIEW를 설정**해야 합니다. LabVIEW를 설정하여 한 번에 단지 하나의 소스 컨트롤 제공자와 작동하도록 할 수 있습니다.

LabVIEW에는 두 개의 소스 컨트롤 통합 인터페이스 타입이 있습니다. Windows에서 LabVIEW는 Microsoft Source Code Control Interface를 지원하는 모든 소스 컨트롤 제공자와 통합됩니다. Windows 플랫폼이 아닌 경우, LabVIEW는 명령 라인 인터페이스를 사용하는 Perforce와 통합됩니다.

LabVIEW와 소스 코드 컨트롤 통합에 대한 더 자세한 정보는 ni.com에서 Software Configuration Management and LabVIEW 문서를 참조하십시오.

LabVIEW에서 소스 컨트롤 작업

타사 소스 컨트롤 제공자를 사용하도록 **LabVIEW를 설정**한 후에는, LabVIEW 프로젝트나 개별 VI의 모든 파일이나 아이템 폴더에서 소스 컨트롤 작업을 수행할 수 있습니다. 폴더에서 소스 컨트롤 작업을 수행하는 경우, 계층구조내의 모든 해당 아이템이 영향을 받습

니다. 예를 들어 파일을 소스 컨트롤에 추가하는 경우, LabVIEW는 폴더 내의 파일 중 사용자가 소스 컨트롤에 추가하지 않은 파일들만 추가합니다. 도구 » 소스 컨트롤을 선택한 후 사용 가능한 옵션에서 선택하여 다음 작업을 접근합니다. LabVIEW 프로젝트 내에서 소스 컨트롤 도구 모음 버튼을 사용하거나 **프로젝트 탐색기** 윈도우의 파일이나 폴더에서 마우스 오른쪽 버튼을 클릭하여 바로 가기 메뉴에서 이 옵션을 선택할 수 있습니다.

또한, 프로젝트 프로퍼티 대화 상자의 소스 컨트롤 페이지에서 **개별 LabVIEW 프로젝트의 소스 컨트롤을 설정**할 수도 있습니다. LabVIEW 환경에 지정한 것 이외의 다른 소스 컨트롤 프로젝트를 사용하려는 경우 또는 LabVIEW 프로젝트에서 소스 컨트롤을 사용하지 않으려는 경우, 개별 LabVIEW 프로젝트의 소스 컨트롤을 설정합니다. LabVIEW 프로젝트는 기본적으로 LabVIEW 환경에서 지정한 소스 컨트롤 설정을 사용합니다. 프로젝트 » 프로퍼티를 선택하거나 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 프로젝트 프로퍼티 대화 상자를 디스플레이합니다. 항목 리스트에서 소스 컨트롤을 선택하여 소스 컨트롤 페이지를 디스플레이합니다.

소스 컨트롤을 사용하기 전에 다음 유의점을 참조하십시오:

- 소스 컨트롤에 저장하는 VI 및 다른 파일 타입에서 **컴파일된 코드를 분리**할 수 있습니다. VI를 편집하면, 기본적으로 LabVIEW는 변경된 VI를 호출하는 모든 VI를 자동으로 다시 컴파일하기 때문에, 저장되지 않은 변경사항이 호출 VI에 생깁니다. 호출 VI가 소스 컨트롤에 저장된 경우, 이를 체크아웃하고 나서 다시 체크인해야 합니다. 그러나 **VI 계층구조** 내에 있는 VI에서 컴파일된 코드를 분리하는 경우, VI 하나가 변경되었을 때 VI 계층구조의 다른 곳에 저장되지 않은 변경이 생기지 않습니다. 이러한 동작으로 수정하려는 VI만 체크아웃할 수 있습니다.
- LabVIEW 프로젝트 밖에서 VI와 함께 소스 컨트롤을 사용하면, 프로젝트 라이브러리 .lvlib나 프로젝트 .lvproj와 같은 프로젝트에 특정한 아이템에서 소스 컨트롤 작업을 수행할 수 없습니다.
- LLB의 한 VI에서 소스 컨트롤 작업을 수행하려고 하는 경우, LabVIEW는 VI 자신이 아닌 VI를 포함하고 있는 LLB에서 그 작업을 수행합니다. LLB에서 오직 하나의 VI에서 소스 컨트롤 작업을 수행할 수 없습니다.
- VI가 **재호출**이면 소스 VI의 복제에서 소스 컨트롤 작업을 수행할 수 없습니다. LabVIEW는 복제 VI의 도구 » 소스 컨트롤 메뉴에서 소스 컨트롤 작업 아이템을 비활성화시킵니다.

다음 방법을 따라 LabVIEW에서 소스 컨트롤 작업을 수행할 수 있습니다:

- 최신 버전 얻기 – 소스 컨트롤에서 로컬 디렉토리로 **선택한 파일의 가장 최신 버전을 복사**하여 두 개의 버전을 동기화합니다. 소스 컨트롤에서 파일의 가장 최신 버전이 로컬 디렉토리에 있는 버전을 덮어씁니다.
- 체크인 – **선택한 파일을 소스 컨트롤에 체크인**합니다. 적용된 변경사항을 반영하도록 소스 컨트롤의 파일 버전이 업데이트됩니다. 선택한 파일에 저장되지 않은 변경사항이 있는 경우, 저장되지 않은 파일 대화 상자가 나타납니다. 소스 컨트롤에 체크인하기 위해서 파일을 저장할 필요는 없습니다.
- 체크아웃 – **선택한 파일을 소스 컨트롤에서 체크아웃**합니다. 체크아웃하지 않은 소스 컨트롤의 파일을 편집하려고 하면, 입력요청을 활성화하도록 소스 컨트롤을 설정한 경우에는 LabVIEW가 파일을 체크아웃하도록 입력요청합니다.
- 체크아웃 취소 – **이전 체크아웃 작업을 취소**하고 선택한 파일의 내용을 이전 버전으로 재저장합니다. 파일에서 변경한 모든 부분을 잃게 됩니다.

- 소스 컨트롤에 추가 — **선택한 파일을 소스 컨트롤에 추가**합니다. 입력 요청을 활성화하기 위해 소스 컨트롤을 설정한 경우 LabVIEW는 SubVI와 같은 모든 의존 파일을 소스 파일에 추가하기 위해 입력 요청합니다.



노트

파일을 저장한 이후에 이 작업을 수행할 수 있습니다.

- 소스 컨트롤에서 제거 — **선택한 파일을 소스 컨트롤에서 제거**합니다.



주의 소스 컨트롤에서 파일을 지울 때 주의하십시오. 일부 소스 컨트롤 제공자는 파일의 로컬 디렉토리 복사본과 함께, 제공자가 보관하는 파일의 이전 버전 및 해당 파일에 대한 히스토리 로그를 삭제합니다.

- 히스토리 보기—선택된 파일의 소스 컨트롤 히스토리를 디스플레이합니다. 히스토리에는 소스 컨트롤에 추가된 후에 파일에 변경사항이 기록되어 있습니다. 히스토리는 파일 체크인 날짜와 사용자 동작에 관한 파일의 이전 버전에 대한 정보를 제공합니다. 폴더에 이 작업을 수행할 수 없습니다.
- 차이 보기 — 선택한 파일의 로컬 복사본과 소스 컨트롤 버전의 차이를 디스플레이합니다. 텍스트 파일의 경우, LabVIEW는 소스 컨트롤 제공자의 기본적인 비교 도구를 사용합니다. 비교할 VI를 선택했을 경우, LabVIEW는 차이 대화 상자에서 결과를 디스플레이합니다. 폴더에 이 작업을 수행할 수 없습니다.

노트 LabVIEW와 타사 소스 컨트롤 제공자가 호환되지 않는 경우, LabVIEW는 타사 소스 컨트롤 제공자의 기본 비교 도구를 실행합니다. Perforce SCM 및 Rational Clearcase는 LabVIEW가 VI의 그래픽 차이를 비교할 때 사용하는 절차와 호환되지 않습니다.

- 프로퍼티—선택된 파일의 소스 컨트롤 프로퍼티를 체크아웃 상태 및 변경 날짜와 함께 디스플레이합니다. 폴더에 이 작업을 수행할 수 없습니다.
- 상태 새로 고침—LabVIEW 프로젝트에서 파일의 소스 컨트롤 상태를 업데이트하거나 LabVIEW 프로젝트 밖에서 작업 중이면 VI의 소스 컨트롤 상태를 업데이트합니다.
- 소스 컨트롤 클라이언트 실행—소스 컨트롤 제공자의 파일 관리 클라이언트를 시작합니다.



노트 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭하면 다음 소스 컨트롤 작업이 바로 가기 메뉴에 나타납니다.

- 모든 파일의 최신 버전 얻기 — 이전 버전이거나 프로젝트에서 찾을 수 없는 파일의 **가장 최신 버전을 찾고 가져옵니다**. LabVIEW는 모든 찾을 수 없거나 이전 버전 파일의 가장 최신 버전을 소스 컨트롤에서 가져옵니다. LabVIEW가 가져올 수 없는 찾을 수 없는 파일에 대한 정보는 찾을 수 없는 파일 정보 대화 상자에 나타납니다.

VI 및 다른 파일 타입에서 컴파일된 코드 분리하여 소스 컨트롤하기

VI는 사용자가 편집하는 그래픽 소스 코드, 이 코드의 컴파일된 버전으로 LabVIEW가 VI를 실행하는데 사용하는 코드, 이렇게 두 가지 코드를 포함하도록 기본설정되어 있습니다.

VI의 소스 코드를 편집하면, LabVIEW는 자동으로 VI를 다시 컴파일해서 컴파일된 버전의 코드에 변경사항을 반영합니다. 또한 LabVIEW는 변경된 VI를 호출하는 모든 VI를 자동으로 다시 컴파일하기 때문에, 저장되지 않은 변경사항이 호출 VI에 생깁니다. 호출하는 VI가 **소스 컨트롤에 저장되어 있는 경우**, 저장되지 않은 대기 중인 변경사항 때문에 이 VI의 SubVI를 업데이트할 때 VI를 체크아웃해야만 합니다.

이 문제를 피하기 위해 LabVIEW에서는 VI의 그래픽 소스 코드로부터 컴파일된 코드를 분리할 수 있도록 합니다. VI에서 컴파일된 코드를 분리한 후, LabVIEW는 SubVI를 수정할 때 다시 이 VI를 컴파일합니다. 그러나 LabVIEW는 다시 컴파일된 버전의 소스를 소스 컨트롤의 VI에 저장하는 대신, **컴파일된 객체 캐시**에 저장합니다. 소스 컨트롤의 VI는 변경되지 않기 때문에, SubVI를 변경할 때 소스 컨트롤에서 이 VI를 체크아웃하지 않아도 됩니다.

컴파일된 코드를 VI에서 분리할지 결정하기

다음과 같은 이유로 VI에서 **컴파일된 코드를 분리**합니다:

- 소스 컨트롤 단순화—**VI 계층구조** 또는 LabVIEW 프로젝트 안의 모든 파일에서 컴파일된 코드를 분리할 경우, VI 하나가 변경되었을 때 VI 계층구조의 다른 곳에서 저장되지 않은 변경사항이 생기지 않습니다. 그렇기 때문에 변경하려는 VI만 체크아웃하고 이 VI를 호출하는 모든 VI를 체크아웃할 필요가 없습니다.
- 소스 컨트롤에 있는 VI를 LabVIEW 새 버전으로 업그레이드하기 위해 준비—소스 컨트롤에서 VI 체크아웃하지 않고 소스만 있는 VI를 새 LabVIEW 버전으로 업그레이드할 수 있습니다.
- VI 로드 시간 향상—LabVIEW는 일반 VI보다 소스만 있는 VI를 더 빨리 로드할 수 있습니다. 이러한 장점을 최대화하려면 VI 계층구조 또는 프로젝트의 모든 파일에서 컴파일된 코드를 분리하십시오..

LabVIEW 런타임 엔진 컴파일된 객체 캐시 소스 배포 빌드 아님

다른 파일 타입에서 소스 컨트롤하기

LabVIEW에서는 소스만 있는 프로젝트 라이브러리, LabVIEW 클래스 및 X컨트롤을 생성할 수 있어, 소스 컨트롤의 멤버 파일 버전에 저장되지 않은 변경사항이 생기는 것을 예방합니다. 소스만 있는 VI와는 달리, LabVIEW는 이러한 파일 타입에서는 컴파일된 코드를 분리하지 않으며, 일부 파일 타입에는 컴파일된 코드가 없는 경우도 있습니다. 대신 이러한 파일 타입에서 소스 파일에서 컴파일된 코드 분리 옵션을 선택하는 경우, LabVIEW는 각 파일의 최신 버전을 객체 캐시에 저장합니다. 라이브러리, 클래스 또는 X컨트롤의 멤버 VI를 변경하는 경우, LabVIEW는 소스 컨트롤에 있는 버전을 저장하도록 요청하는 대신, 객체 캐시의 파일을 업데이트합니다. 그러므로, 소스만 있는 라이브러리, 클래스 및 X컨트롤 역시 소스만 있는 VI의 장점을 가집니다.

빌드 스펙

어플리케이션 만들기 및 배포하기

빌드 스펙을 사용해서 사용자 정의된 LabVIEW 어플리케이션을 생성하고 배포할 수 있습니다.



노트

독립 어플리케이션, 설치 프로그램, .NET Interop 어셈블리, 묶음 프로젝트 라이브러리, 공유 라이브러리 또는 압축(Zip) 파일을 만들려면 어플리케이션 빌더가 설치되어 있어야 합니다. LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. 다른 LabVIEW 개발 시스템을 사용하는 경우, National Instruments 웹사이트를 방문하여 어플리케이션 빌더를 별도로 구매할 수 있습니다.

다음 토픽을 사용해서 LabVIEW로 만든 어플리케이션을 시작하거나 LabVIEW로 만든 어플리케이션에서 발생한 문제를 해결합니다.

시작하기

- 어플리케이션 개발하기 및 배포하기

디버깅

- 어플리케이션과 공유 라이브러리 디버그하기
- VI 디버그하기

어플리케이션 빌더 VI

- 빌드 VI
- 배포 VI
- 정리 VI

시작하기

- 어플리케이션 개발하기 및 배포하기

디버깅

- 어플리케이션과 공유 라이브러리 디버그하기
- VI 디버그하기

어플리케이션 빌더 VI

- 빌드 VI
- 배포 VI
- 정리 VI



관련 정보

- [VI 서버 어플리케이션 만들기](#) (185 페이지 참조)
- [어플리케이션 개발하기 및 배포하기](#) (1448 페이지 참조)
- [어플리케이션과 공유 라이브러리 디버그하기](#) (1389 페이지 참조)
- [디버깅 기법](#) (254 페이지 참조)
- [어플리케이션 개발하기 및 배포하기](#) (1448 페이지 참조)
- [어플리케이션과 공유 라이브러리 디버그하기](#) (1389 페이지 참조)
- [디버깅 기법](#) (254 페이지 참조)

빌드 스펙 사용하기

프로젝트 탐색기 윈도우에 있는 빌드 스펙을 사용하여, LabVIEW 빌드의 빌드 스펙을 **생성**하고 **설정**할 수 있습니다. 빌드 스펙은 포함할 파일, 생성할 디렉토리, VI의 셋팅 등 빌드에 대한 모든 셋팅을 포함합니다. 다음 테이블은 각 빌드 스펙에 대한 LabVIEW 요구사항을 보여줍니다.

빌드 스펙	요구사항
독립 어플리케이션	어플리케이션 빌더 또는 Professional Development System
설치 프로그램	어플리케이션 빌더 또는 Professional Development System
.NET Interop 어셈블리	어플리케이션 빌더 또는 Professional Development System
패키지	어플리케이션 빌더 또는 Professional Development System
묶음 라이브러리	어플리케이션 빌더 또는 Professional Development System
공유 라이브러리	어플리케이션 빌더 또는 Professional Development System
소스 배포	Base Development System 또는 Full Development System
웹 서비스	Base Development System 또는 Full Development System

빌드 스펙	요구사항
압축(Zip) 파일	어플리케이션 빌더 또는 Professional Development System



노트

어플리케이션 빌더가 설치되어 있어야 독립 어플리케이션, 공유 라이브러리, 설치 프로그램, 압축(Zip) 파일을 만들 수 있습니다. LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. LabVIEW Base Development System 또는 Full Development System을 사용하는 경우, National Instruments 웹사이트를 방문하여 어플리케이션 빌더를 별도로 구매할 수 있습니다. 어플리케이션 빌더를 이미 구입하신 경우, 도움말 >> LabVIEW 제품 정품인증을 선택하여 이 제품의 정품인증을 받으십시오.

빌드 스펙 인터페이스는 사용자가 빌드 스펙 타입을 사용자 정의할 수 있는 여러 대화 상자의 세트입니다. 각각의 대화 상자는 각 빌드 스펙 타입에 대해 고유한 페이지를 가지고 있습니다. 예를 들어, 독립 어플리케이션을 만들 때 어플리케이션 프로퍼티 대화 상자를 사용하여, 어플리케이션을 시작할 때 실행할 VI와 항상 포함할 VI를 선택할 수 있습니다.



노트 이전에 프로젝트 탐색기 윈도우에서 빌드 스펙을 숨긴 경우, 프로젝트 탐색기 윈도우에서 이에 접근하려면 다시 **아이템을 디스플레이**해야 합니다.

다음 타입의 **빌드 스펙**을 생성하고 설정할 수 있습니다:

- 독립 어플리케이션— 독립 어플리케이션을 사용하여 다른 사용자에게 VI의 실행 가능한 버전을 제공합니다. 어플리케이션은 사용자가 LabVIEW 개발 시스템을 설치하지 않고 VI를 실행할 수 있도록 하려는 경우 유용합니다. 독립 어플리케이션을 사용하려면 **LabVIEW 런타임 엔진**이 필요합니다. (Windows) 어플리케이션의 확장자는 .exe입니다. (macOS) 어플리케이션의 확장자는 .app입니다.
- 설치 프로그램—(Windows) 설치 프로그램을 사용하여 어플리케이션 빌더로 생성하는 독립 어플리케이션, 공유 라이브러리, 소스 배포를 배포할 수 있습니다. LabVIEW 런타임 엔진을 포함한 설치 프로그램은 사용자가 LabVIEW를 설치하지 않고도 어플리케이션을 실행하거나 공유 라이브러리를 사용할 수 있도록 하려는 경우 유용합니다.
- .NET interop 어셈블리 — (Windows) .NET Interop 어셈블리를 사용하여 Microsoft .NET Framework용 VI 패키지를 만들 수 있습니다. 어플리케이션 빌더를 사용하여 .NET interop 어셈블리를 생성하려면 .NET Framework 4.0을 설치해야 합니다.
- 패키지 - (Windows) **패키지**를 사용하여 소스 배포, 묶음 프로젝트 라이브러리, 공유 라이브러리, .NET 어셈블리, 실행 파일을 클라이언트에 배포합니다. NI 패키지 관리자 또는 SystemLink를 통해 클라이언트에 패키지를 배포할 수 있습니다. 클라이언트는 패키지 관리자 또는 SystemLink를 통해 피드를 구독하여 패키지를 찾고 설치할 수 있습니다. 패키지의 확장자는 .nipkg입니다. (NI Linux Real-Time) LabVIEW Real-Time Module을 설치하면 NI Linux Real-Time 타겟에서도 opkg 패키지 .ipk를 생성할 수 있습니다. 클라이언트는 SystemLink 또는 NI Linux Real-Time 타겟의 명령 라인을 사용하여 패키지를 설치할 수 있습니다. 패키지 관리자는 .ipk 파일을 지원하지 않습니다.
- 묶음 프로젝트 라이브러리-묶음 프로젝트 라이브러리를 사용해서 여러 LabVIEW 파일을 하나의 파일로 묶습니다. 묶음 라이브러리로 VI를 배포하면, 묶음 라이브

러리가 한 개 파일이기 때문에 배포되는 파일의 수가 줄어듭니다. 묶음 라이브러리의 최상위 레벨 파일은 프로젝트 라이브러리입니다. 묶음 라이브러리는 특정 OS에 맞게 컴파일되는 VI 계층구조를 한 개 이상 포함합니다. 묶음 라이브러리의 확장자는 .lvlibp입니다.

- 공유 라이브러리—LabWindows™/CVI™, Microsoft Visual C++, Microsoft Visual Basic과 같은 텍스트 기반의 프로그래밍 언어를 사용하여 VI를 호출하려 할 때 공유 라이브러리를 사용합니다. 공유 라이브러리를 사용하면 LabVIEW가 아닌 다른 프로그래밍 언어도 LabVIEW로 개발된 코드에 접근할 수 있습니다. 공유 라이브러리는 본인이 만든 VI의 기능을 다른 개발자들과 공유하려는 경우에 유용합니다. 다른 개발자들은 공유 라이브러리를 사용할 수 있지만, **디버깅을 활성화**하지 않은 이상 블록다이어그램을 편집하거나 볼 수는 없습니다. (Windows) 공유 라이브러리의 확장자는 .dll입니다. (macOS) 공유 라이브러리의 확장자는 .framework입니다. (Linux) 공유 라이브러리의 확장자는 .so입니다. .so를 사용하거나 lib로 시작해서 .so로 끝내고, 옵션으로 버전 번호를 붙일 수 있습니다. 이렇게하면 다른 어플리케이션이 이 라이브러리를 사용할 수 있습니다.



노트 (Linux) LabVIEW와 함께 설치되는 libstdc++.so 라이브러리에 대한 더 자세한 정보는 ni.com/info를 방문하여 정보 코드 exjze4를 입력하여 참조하십시오.

- 소스 배포—소스 배포를 사용하여 여러 소스 파일의 모음을 묶습니다. 소스 배포는 LabVIEW에서 사용하도록 코드를 다른 개발자들에게 보낼 때 유용합니다. 선택한 VI에 암호를 추가하거나, 블록다이어그램을 제거하도록 설정하고, 이외에도 다른 셋팅을 적용할 수 있습니다. 또한 VI와 SubVI 사이의 링크를 깨지 않으면서 소스 배포의 VI들에 대해 다른 대상 디렉토리를 선택할 수 있습니다.
- 압축(Zip) 파일—여러 파일이나 전체 LabVIEW 프로젝트를 1 개의 이동 가능한 단일 파일로 배포하려 할 때, Zip 파일을 사용합니다. Zip 파일은 압축된 파일을 포함하며 사용자에게 직접 보낼 수 있습니다. Zip 파일은 소스 파일을 일부 선별한 후 다른 LabVIEW 사용자에게 보내려는 경우 유용합니다. 또한 압축(Zip) VI를 사용하여 프로그램적으로 Zip 파일을 생성할 수 있습니다.

LabVIEW 개발 시스템이 없어도 이 파일을 배포할 수 있습니다. 그러나 독립 어플리케이션이나 공유 라이브러리를 사용하려면 사용자 컴퓨터에 **LabVIEW 런타임 엔진**이 설치되어 있어야 합니다.

어플리케이션 만들기 및 배포하기 홈

빌드 스펙 생성하기

프로젝트 탐색기 윈도우에서 내 컴퓨터를 확장하고 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기>>어플리케이션 타입을 선택합니다. 다음 대화 상자를 사용하여 빌드 스펙의 셋팅을 설정합니다:

- 어플리케이션 프로퍼티
- 묶음 라이브러리 프로퍼티
- 공유 라이브러리 프로퍼티

- 소스 배포 프로퍼티 대화 상자
- 압축(Zip) 파일 프로퍼티
- (Windows) .NET Interop 어셈블리 프로퍼티
- (Windows) 설치 프로그램 프로퍼티
- (Windows) 웹 서비스 프로퍼티
- (Windows) 패키지 프로퍼티

이 설정을 정의한 후, 확인 버튼을 클릭하여 대화 상자를 닫고 프로젝트의 빌드 스펙 설정을 업데이트합니다. 이 빌드 스펙은 빌드 스펙 아래에 나타납니다.

스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 빌드를 선택하여 빌드를 완료합니다. 또한 [빌드] VI 또는 **ExecuteBuildSpec**을 사용하여 어플리케이션을 프로그램 방식으로 빌드할 수 있습니다.

또한 바로 가기 메뉴에서 모두 빌드를 선택하여 빌드 스펙 아래에 모든 스펙을 만들 수도 있습니다.

주어진 스펙을 다시 빌드하는 경우, LabVIEW는 이전 빌드의 파일 중 현재 빌드에 속하는 파일을 덮어씁니다. 그러나 버전 정보 페이지의 자동 증가 확인란 또는 설치 프로그램의 제품 버전 자동 증가 확인란을 활성화한 후 적절한 빌드 대상 필드에 [버전 번호] 또는 [제품 버전] 태그를 사용하는 경우, LabVIEW는 디스크에 있는 이전 빌드를 덮어쓰는 대신 해당 버전 번호를 포함하는 새 디렉토리를 생성합니다.



노트

프로젝트를 저장해야, 빌드 스펙을 생성할 수 있습니다. 저장하지 않은 프로젝트를 사용하는 경우, LabVIEW는 각 빌드 스펙을 설정하는 대화 상자를 열기 전에 사용자에게 프로젝트를 저장하도록 요청합니다.

명령 라인으로 프로젝트 빌드 스펙을 만드는 예제는 `labview\examples\Application Control\Application Builder API` 디렉토리의 `Generate Programmatic Build Command Line String VI`를 참조하십시오.

관련 정보

어플리케이션 개발하기 및 배포하기

빌드 스펙을 위해 파일 준비하기

어플리케이션을 위한 빌드 스펙을 생성하기 전에, 배포할 파일을 준비해야 합니다. 예를 들어, LabVIEW 개발 시스템이 설치되지 않은 시스템에 독립 어플리케이션을 배포할 계획이라면, 사용 중인 프로퍼티 및 메소드가 모두 LabVIEW 런타임 엔진에 의해 지원되는지 반드시 확인해야 합니다.

관련 토픽

다음 토픽은 빌드 스펙을 위해 파일을 준비할 때 완료해야 하는 여러 단계에 대해 설명합니다.

- LabVIEW 런타임 엔진에서 지원되지 않는 VI 서버 프로퍼티와 메소드

빌드 스펙 설정하기

빌드 스펙을 생성할 때, 각 빌드 스펙 타입에 대한 고급 기능을 설정하여 빌드를 사용자 정의할 수 있습니다. 기존 빌드 스펙의 프로퍼티를 변경하려면, 프로젝트 탐색기에 있는 해당 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택합니다. 빌드 스펙 프로퍼티 대화 상자의 페이지를 사용하여, 어플리케이션에 대한 고급 셋팅을 설정할 수 있습니다.

관련 토픽

다음 토픽은 빌드 스펙을 설정할 때 유용한 단계에 대해 설명합니다.

- 어플리케이션 및 공유 라이브러리에서 런타임 언어 설정하기
- LabVIEW 작업 환경 옵션 복사하기
- 빌드 전 또는 후에 VI 실행하기



관련 정보

[빌드 스펙 생성하기](#) (344 페이지 참조)

[어플리케이션 및 공유 라이브러리에서 런타임 언어 설정하기](#) (1456 페이지 참조)

[LabVIEW 작업 환경 옵션 복사하기](#) (1457 페이지 참조)

[VI 실행하기](#) (151 페이지 참조)

독립 어플리케이션과 공유 라이브러리에서 소스 파일 참조하기

어플리케이션 빌더는 디스크 상에서의 소스 파일 디렉토리 구조를 기반으로 어플리케이션이나 공유 라이브러리에 소스 파일을 저장합니다. 이러한 내부 파일 레이아웃은 어플리케이션 내에 소스 파일 계층구조를 보존합니다.

다이나믹하게 VI를 호출하는 경우, **상대 경로**를 사용하여 어플리케이션이 런타임에서 정확하게 VI를 로드하도록 보장합니다. 어플리케이션이 외부에 있는 데이터 파일을 참조하는 경우, [어플리케이션 디렉토리] VI를 사용하여 파일의 경로를 확인할 수 있습니다.

다이나믹하게 labview\vi.lib 디렉토리에서 VI를 호출하려는 경우, 새 VI의 블록 다이어그램에 VI를 추가함으로써 로드하려는 파일에 대한 래퍼 VI를 생성합니다. 래퍼 VI를

SubVI로 준비하고, 래퍼VI의 커넥터 팬이 원래 vi.lib VI의 커넥터 팬과 일치하는지 확인합니다.

LabVIEW 8 사용하기 파일 레이아웃

어플리케이션 프로퍼티 또는 공유 라이브러리 프로퍼티 대화 상자의 고급 페이지에서 LabVIEW 8.x 파일 레이아웃 사용 옵션을 활성화하면, 어플리케이션 빌더는 소스 파일을 어플리케이션 안에 플랫폼 리스트로 저장합니다. 이러한 파일 구조 때문에, LabVIEW는 파일 이름이 충돌하는 파일을 모두 서로 다른 폴더로 옮깁니다. 사용자가 만든 어플리케이션이 이전의 동작에 의존하여 어플리케이션 안에 있는 플랫폼 리스트에 파일을 저장하는 경우, National Instruments는 이 옵션을 활성화할 것을 권장합니다.

labview\vi.lib\appbuilder 디렉토리에 있는 Built App File Layout VI를 사용하여, 어플리케이션이 이전 파일 레이아웃을 사용하여 VI를 독립 어플리케이션이나 공유 라이브러리 안에 저장하는지 구분할 수 있습니다.

소스 파일 경로	어플리케이션에 있는 파일 경로	8.x 레이아웃 어플리케이션에 경로
C:\Source\foo.vi	C:\..\Application.exe\foo.vi	C:\..\Application.exe
C:\Source\xxx\a.vi	C:\..\Application.exe\xxx\a.vi	C:\..\Application.exe
C:\Source\yyy\b.vi	C:\..\Application.exe\yyy\b.vi	C:\..\Application.exe

빌드 스펙 관리하기

프로젝트 탐색기 윈도우에서 빌드 스펙을 설정한 이후에, LabVIEW는 사용자의 빌드 스펙을 관리하고 추적할 수 있는 방법을 제공합니다. 개발 과정 전반에서 사용자 빌드 스펙을 추적함으로써 개발 작업을 향상시킬 수 있습니다.

또한, LabVIEW는 프로젝트 내의 빌드 스펙 작업을 수행할 때 이용할 수 있는 기능을 제공합니다. 프로젝트 탐색기 윈도우에서 빌드 스펙을 복제할 수 있습니다. 복제할 빌드 스펙 아이템에서 마우스 오른쪽 버튼을 클릭하여 바로 가기 메뉴에서 복제를 선택하여 빌드 스펙 아이템의 복사본을 생성합니다.

또한 빌드 스펙 아이템을 끌어서 놓아 같은 빌드 스펙 내의 빌드 순서를 다시 정렬할 수 있습니다.

가장 최근 빌드 과정 중에 LabVIEW가 생성한 빌드 스펙 파일을 삭제하려면, 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 정리를 선택합니다. 정리를 선택하더라도 LabVIEW가 가장 최근 빌드 전에 생성한 파일은 삭제되지 않습니다. 정리 VI를 사용하여 가장 최근 빌드에서 파일을 삭제할 수도 있습니다.

독립 어플리케이션을 만드는 경우, 사용자의 LabVIEW 작업 환경 옵션을 빌드 안에 복사할 수 있습니다.

LabVIEW의 어플리케이션

어플리케이션 관리하기

LabVIEW에서 빌드를 추적하여, 어플리케이션에 대한 여러 버전을 관리할 때 활용할 수 있습니다. 특정 빌드에 대한 여러 버전을 배포해야 할 경우, 빌드 추적 기능은 매우 유용합니다. 예를 들어, 최종 사용자에게 여러 버전의 빌드를 배포할 때 주요 버전과 보조 버전의 빌드를 구분하려고 할 수도 있습니다.

프로퍼티 대화 상자의 버전 정보 페이지에 있는 버전 번호 기능을 사용하여 빌드를 추적할 수 있습니다. 자동 증가 확인란에 확인 표시를 한 경우, LabVIEW는 자동적으로 빌드 구성요소의 버전 번호를 업데이트합니다. 주요, 보조, 또는 수정 구성 요소를 편집하려면, 이 확인 표시를 제거하고 수동으로 번호를 증가시켜야 합니다. 또한, 버전 정보 페이지를 통해 빌드의 내부 이름을 정하거나, 최종 사용자가 볼 수 있는 빌드에 대한 설명을 입력할 수 있습니다. 예를 들어, 최종 사용자에게 제공할 어플리케이션에 대한 주요 변경 사항에 대한 설명을 설명 필드에 입력할 수 있습니다.

관련 토픽

다음 토픽은 빌드 스펙을 관리할 때 유용한 여러 단계에 대해 설명합니다.

- 독립 어플리케이션의 정보 대화 상자 생성하기
- 어플리케이션과 공유 라이브러리 디버그하기
- 독립 어플리케이션에서 웹 서버 활성화하기
- 프로그램적으로 어플리케이션 및 공유 라이브러리 버전 검색하기
- 독립 어플리케이션에서 설정 파일 사용하기

빌드 어플리케이션의 기능

LabVIEW로 만든 어플리케이션을 다른 컴퓨터 및 다른 프로그래밍 언어에서 사용할 수 있습니다. LabVIEW로 만든 어플리케이션에는 특정한 기능이 있습니다. 예를 들어, .NET interop 어셈블리의 데이터 타입 특성에 특정한 기능이 있습니다.

관련 토픽

다음 토픽에서는 LabVIEW로 만든 여러 타입의 어플리케이션이 가지는 특징에 대해 설명합니다.

- LabVIEW로 만든 .NET Interop 어셈블리의 특징

- LabVIEW로 만든 공유 라이브러리의 특징



관련 정보

[VI 서버 어플리케이션 만들기](#) (185 페이지 참조)

[LabVIEW로 만든 .NET Interop 어셈블리의 특징](#) (1461 페이지 참조)

[LabVIEW로 만든 공유 라이브러리의 특징](#) (1463 페이지 참조)

독립 어플리케이션 및 공유 라이브러리 설치하기

Windows에서 독립 어플리케이션 또는 공유 라이브러리를 만든 경우, 빌드를 배포할 때 설치 프로그램을 만드는 것을 고려해볼 수 있습니다. 설치 프로그램을 만들면, 어플리케이션을 실행할 때 사용자에게 필요한 구성 요소를 모두 설치 프로그램에 포함시킬 수 있습니다. 예를 들어, LabVIEW 개발 시스템이 설치되지 않은 컴퓨터에서 어플리케이션을 실행하게 될 경우에는 LabVIEW 런타임 엔진을 포함시킵니다.

관련 토픽

다음 토픽은 독립 어플리케이션 및 공유 라이브러리를 설치할 때 고려할 사항에 대해 설명합니다.

- 설치 프로그램 배포 구성요소 캐시하기
- 설치 프로그램 만들기에 대한 유의점과 권장사항
- 설치 프로그램 빌드에 추가적인 National Instrument 설치 프로그램 포함하기
- LabVIEW로 만든 설치 프로그램 내부적으로 설치하기
- LabVIEW로 만든 설치 프로그램 설치 제거하기
- LabVIEW 런타임 엔진 사용하기



관련 정보

[어플리케이션 개발하기 및 배포하기](#) (1448 페이지 참조)

[설치 프로그램 배포 구성요소 캐시하기\(Windows\)](#) (1464 페이지 참조)

[설치 프로그램 만들기에 대한 유의점과 권장사항](#) (1464 페이지 참조)

[설치 프로그램 빌드에 추가적인 National Instrument 설치 프로그램 포함하기](#) (1466 페이지 참조)

[LabVIEW로 만든 설치 프로그램 내부적으로 설치하기](#) (1468 페이지 참조)

[LabVIEW로 만든 설치 프로그램 설치 제거하기](#) (1468 페이지 참조)

[LabVIEW 런타임 엔진 사용하기](#) (350 페이지 참조)

LabVIEW 런타임 엔진 사용하기

LabVIEW로 생성한 모든 타입의 어플리케이션을 실행하려는 컴퓨터에는 LabVIEW 런타임 엔진을 설치해야 합니다. LabVIEW 런타임 엔진에는 LabVIEW로 생성한 기본 타입의 어플리케이션을 실행하는데 필요한 라이브러리 및 그 외 파일이 포함되어 있습니다.



노트

LabVIEW 런타임 엔진은 여러 언어를 지원합니다.

설치 프로그램에 대한 빌드 스펙을 생성할 때, 어플리케이션 빌더에는 기본으로 LabVIEW 런타임 엔진이 포함됩니다.

LabVIEW의 특정 버전으로 구축된 모든 타입의 어플리케이션은 하나의 LabVIEW 런타임 엔진을 공유하므로, 해당 버전에는 LabVIEW 런타임 엔진을 한번만 설치하면 됩니다. 그러나, 같은 컴퓨터에서 서로 다른 버전의 LabVIEW를 사용하여 생성한 여러 개의 어플리케이션을 실행하려면, 어플리케이션 타입을 생성에 사용된 LabVIEW의 각 버전에 대응하는 버전의 LabVIEW 런타임 엔진을 설치해야 합니다. 예를 들어, LabVIEW 7.0에서 생성된 어플리케이션과 LabVIEW 8.0에서 생성된 어플리케이션을 같은 컴퓨터에서 실행하고자 하는 경우, 컴퓨터에는 LabVIEW 런타임 엔진의 7.0과 8.0 버전이 있어야 합니다.

LabVIEW 런타임 엔진에 대한 고려사항

- (Windows) 어플리케이션 빌더로 구축한 설치 프로그램에 LabVIEW 런타임 엔진을 포함시키려면, 반드시 관리자 또는 관리자 권한이 있는 사용자로 로그인하여 설치 프로그램을 실행해야 합니다. (Linux) 수학 VI 또는 신호 처리 VI를 사용하는 어플리케이션을 실행하려는 타겟 컴퓨터에 LabVIEW 런타임 엔진을 설치할 때, LabVIEW 런타임 고급 분석 지원 패키지 `labview-rte-aal-1.1-1.i386.rpm`을 설치할지 물어보면 예를 입력합니다.
- 몇몇 VI 서버 프로퍼티와 메소드는 LabVIEW 런타임 엔진에서 지원하지 않습니다. 어플리케이션 또는 공유 라이브러리에 포함시킨 VI에 이러한 프로퍼티와 메소드를 사용하는 것을 피하십시오.
- LabVIEW는 LabVIEW 런타임 엔진에 자동 에러 핸들링 대화 상자를 디스플레이하지 않기 때문에 어플리케이션의 VI에 에러 핸들링 기능을 포함하도록 합니다.
- VI가 사용자 런타임 메뉴를 사용하는 경우, VI가 사용하는 어플리케이션 메뉴 아이템이 LabVIEW 런타임 엔진에서 사용 가능한지 확인하십시오.
- 어플리케이션의 모든 프런트패널 윈도우를 닫을 때, 어플리케이션은 정지합니다. 어플리케이션으로 만드는 VI가 마지막 프런트패널 윈도우가 닫힌 후에 실행되는 코드를 포함하는 경우, 이 코드는 어플리케이션에서 실행되지 않습니다. 마지막 프런트패널 윈도우가 닫힌 후에 실행되는 코드는 블록 다이어그램에 쓰는 것을 피하십시오.
- 참조에 의한 호출 노드를 사용하여 어플리케이션에서 VI를 참조하는 경우, VI가 프로퍼티 노드를 사용하여 프런트패널 프로퍼티를 설정하는 경우 또는 프런트패널 윈도우가 사용자에게 나타나는 경우, 어플리케이션 프로퍼티 대화 상자의 소스 파일 셋팅 페이지에 있는 기본 저장 셋팅 사용 섹션에서 해당 VI에 대한 프런트패널 제거 확인란의 확인 표시를 제거하십시오. 프런트패널 윈도우를 제거하는 경우, 프런트패널 윈도우를 참조하는 참조에 의한 호출 노드와 프로퍼티 노드는 에러를 반환하며, 이러한 에러는 어플리케이션의 동작에 영향을 미칠 수 있습니다.

LabVIEW에서 ActiveX 사용하기

LabVIEW를 ActiveX 클라이언트로 사용하여 다른 ActiveX를 지원하는 어플리케이션과 연관된 객체, 프로퍼티, 메소드, 이벤트에 접근할 수 있습니다. 또한 LabVIEW를 ActiveX Server로 사용하여 다른 어플리케이션이 LabVIEW 객체, 프로퍼티 및 메소드에 접근하는 것을 허용할 수 있습니다. LabVIEW에서 ActiveX를 사용하는 것에 대한 추가적인 정보는 ni.com의 ActiveX and LabVIEW를 참조하십시오.

ActiveX 객체, 프로퍼티 및 메소드

ActiveX 활성화 어플리케이션은 다른 어플리케이션에서 접근할 수 있는 노출된 프로퍼티와 메소드를 갖는 객체를 포함합니다. 객체는 버튼, 윈도우, 그림, 문서, 대화 상자처럼 사용자에게 보일 수도 있고, 어플리케이션 객체처럼 보이지 않을 수도 있습니다. 어플리케이션과 관련된 객체에 접근하고 해당 객체의 프로퍼티를 설정하거나 메소드를 실행하여 어플리케이션에 접근할 수 있습니다.

ActiveX 이벤트

이벤트는 마우스를 클릭하거나 키를 누르는 것과 같이 객체에 행해지는 작업입니다. 객체에 이러한 동작이 행해지면, 객체는 이를 알리기 위해 이벤트와 이벤트 특성의 데이터를 ActiveX 컨테이너로 보냅니다. ActiveX 객체는 객체에 사용 가능한 이벤트를 정의합니다.

ActiveX 이벤트를 어플리케이션에서 사용하려면, 반드시 이벤트를 등록하고 이벤트가 발생하면 이를 처리해야 합니다. ActiveX 이벤트 등록은 **다이나믹 이벤트 등록**과 유사합니다. 다이나믹 이벤트에서는, While 루프 내부에 이벤트를 놓고 객체 참조를 이벤트 구조에 연결한 후, LabVIEW가 생성하기를 원하는 이벤트를 선택합니다. 예를 들어, ActiveX 컨테이너에서 Windows 기반의 트리 컨트롤을 호출하고 이 트리 컨트롤에 디스플레이되는 항목에 더블 클릭 이벤트를 생성하도록 지정할 수 있습니다. 그러나 ActiveX 이벤트 VI의 구조는 이벤트 핸들링 VI의 구조와 다릅니다. 이벤트를 지정하고 등록하려면 [이벤트 등록] 함수 대신 [이벤트 콜백 등록] 함수를 사용해야 합니다.



노트

National Instruments는 이벤트를 처리할 필요가 없어지면 이벤트 등록 해제 함수를 사용하여 해당 이벤트의 등록을 해제할 것을 권장합니다. 이벤트의 등록을 해제하지 않는 경우, 해당 이벤트를 다른 이벤트 구조가 없어도 LabVIEW는 VI가 실행되는 한 계속 이벤트를 생성하고 큐에 넣습니다. 따라서 이벤트를 위한 프런트패널 잠금을 활성화시키는 경우 메모리가 소모되고 VI가 멈출 수 있습니다.

LabVIEW에서 ActiveX 이벤트를 생성 및 등록하는 구성요소에 대한 정보는 다음 테이블을 참조하십시오.

구성요소	설명	용도
ActiveX 객체	이것은 이벤트를 생성하고자 하는 ActiveX 객체입니다.	이벤트를 생성하려는 ActiveX 객체를 선택한 후, 오토메이션 참조 번호를 사용하여 이벤트를 지정할 수 있습니다.

구성요소	설명	용도
[이벤트 콜백 등록] 함수	이 함수는 [이벤트 등록] 함수와 유사하게 여러 이벤트를 핸들링할 수 있는 확장 가능한 노드입니다.	ActiveX 객체 참조를 [이벤트 콜백 등록] 함수에 연결한 후, 객체에 생성하려는 이벤트를 지정할 수 있습니다. 객체에 생성하려는 이벤트를 지정할 때에는, 해당 이벤트에 대해 ActiveX 객체를 등록해야 합니다.
콜백 VI	이 VI에는 이벤트를 처리하기 위해 작성한 코드가 포함됩니다.	[이벤트 콜백 등록] 함수를 사용하여 이벤트를 등록한 후 이 VI를 생성할 수 있습니다. 서로 다른 이벤트는 다른 이벤트 데이터 포맷을 가질 수 있습니다. 따라서, 콜백 VI를 생성한 후 이벤트를 변경하면 블록 다이어그램의 와이어가 깨질 수 있습니다. 콜백 VI를 생성하기 전에 이벤트를 선택하십시오.

ActiveX VI, 함수, 컨트롤, 인디케이터

어떤 VI, 함수, 컨트롤 및 인디케이터를 사용하여 기타 ActiveX 활성화 어플리케이션과 관련된 객체, 프로퍼티, 메소드, 이벤트에 접근할 지에 대한 정보는 다음 테이블을 참조하십시오.

VI, 함수, 컨트롤 또는 인디케이터 이름	설명	예	사용 방법
오토메이션 참조 번호 컨트롤	이 컨트롤은 ActiveX 객체에 대한 참조를 생성합니다.	VI의 프런트패널에서 ActiveX 컨테이너에 포함된 워드 문서에 대한 참조를 생성할 수 있습니다.	프런트패널의 이 컨트롤에서 마우스 오른쪽 버튼을 클릭하여 접근하려는 타입 라이브러리에서 객체를 선택합니다.
오토메이션 열기 함수	이 함수는 ActiveX 객체를 엽니다.	이 함수를 사용하여 Microsoft Excel 문서를 열 수 있습니다.	함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기에서 ActiveX 클래스 선택을 선택하여 해당 객체의 클래스를 선택합니다. 참조 번호를 얻 후에는 참조 번호를 다른 ActiveX 함수에 전달할 수 있습니다.
ActiveX 컨테이너	이 컨테이너는 프런트패널의 ActiveX 객체에 접근하고 디스플레이합니다.	ActiveX 컨테이너를 사용하여 웹 브라우저와 같이 Active-X가 활성화된 어플리케이션의 버튼을 디스플레이할 수 있습니다.	컨테이너에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 ActiveX 객체 삽입을 선택합니다. 그 후 접근하려는 객체를 선택합니다.

VI, 함수, 컨트롤 또는 인디케이터 이름	설명	예	사용 방법
프로퍼티 노드	이 함수는 ActiveX 객체와 연관된 프로퍼티를 읽고(읽고) 설정(작성)합니 다.	이 함수를 사용하여 프론트패널의 ActiveX 컨테이너에 포함시킨 객체의 프로퍼티를 설정할 수 있습니다.	프론트패널이나 블록다이어그램의 ActiveX 객체에서 마우스 오른쪽 버튼을 클릭한 후, 생성>>프로퍼티 노드를 선택합니다. 그 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티를 설정합니다.
인보크 노드	이 함수는 ActiveX 객체와 연관된 메소드를 실행합니다.	프로퍼티 노드를 사용하여 프로퍼티를 설정한 방법과 같이, 이 함수를 사용하여 프론트패널의 ActiveX 컨테이너에 포함시킨 객체의 메소드를 인보크할 수 있습니다.	프론트패널이나 블록다이어그램의 ActiveX 객체에서 마우스 오른쪽 버튼을 클릭한 후, 생성>>인보크 노드를 선택합니다. 그 후 바로 가기 메뉴에서 메소드를 선택하여 해당 객체에 대한 메소드를 인보크합니다.
이벤트 콜백 등록	이 함수는 ActiveX 객체에서 발생하는 이벤트를 처리합니다.	이 함수를 사용하여 데이터를 Microsoft Excel 스프레드시트에 추가하는 것과 같은 이벤트를 생성하고 등록할 수 있습니다.	
배리언트 컨트롤과 인디케이 터	이 컨트롤과 인디케이터를 함께 사용하여 ActiveX 컨트롤과 데이터를 주고 받습니다.	이 컨트롤과 인디케이터를 사용하여 Microsoft Excel이나 Microsoft Word와 같이 ActiveX가 활성화된 어플리케이션과 데이터를 주고 받습니다.	배리언트 & 클래스 팔레트를 탐색하여 이 컨트롤과 인디케이터를 찾은 후 프론트패널이나 블록다이어그램에 놓습니다.

데이터 타입 매핑하기

LabVIEW는 일부 ActiveX 프로퍼티와 메소드의 데이터 타입을 LabVIEW 데이터 타입으로 변환하여 데이터를 읽고 해석합니다. 다음 테이블은 ActiveX 데이터 타입과 해당 변환된 LabVIEW 데이터 타입을 나열합니다.

ActiveX 데이터 타입	LabVIEW 데이터 타입
char	 , 
short	 , 

ActiveX 데이터 타입	LabVIEW 데이터 타입
long	 , 
hyper	 , 
float	 , 
double	
BSTR	 , 

LabVIEW를 ActiveX 클라이언트로 사용하기

LabVIEW는 다른 ActiveX 활성화 어플리케이션과 관련된 객체에 접근할 때 ActiveX 클라이언트로 동작합니다. 다음의 방법으로 LabVIEW를 ActiveX 클라이언트로 사용할 수 있습니다:

- Microsoft Excel과 같은 어플리케이션을 열고, 문서를 생성하고, 문서에 데이터를 추가합니다.
- 컨테이너의 프런트패널에 Microsoft Word 문서 또는 Excel 스프레드시트와 같은 문서를 포함 시킵니다.
- 다른 어플리케이션의 버튼이나 객체(예를 들어, 다른 어플리케이션의 도움말 파일을 호출하는 도움말 버튼)를 프런트패널에 놓습니다.
- 생성한 ActiveX 컨트롤을 다른 어플리케이션에 링크합니다.

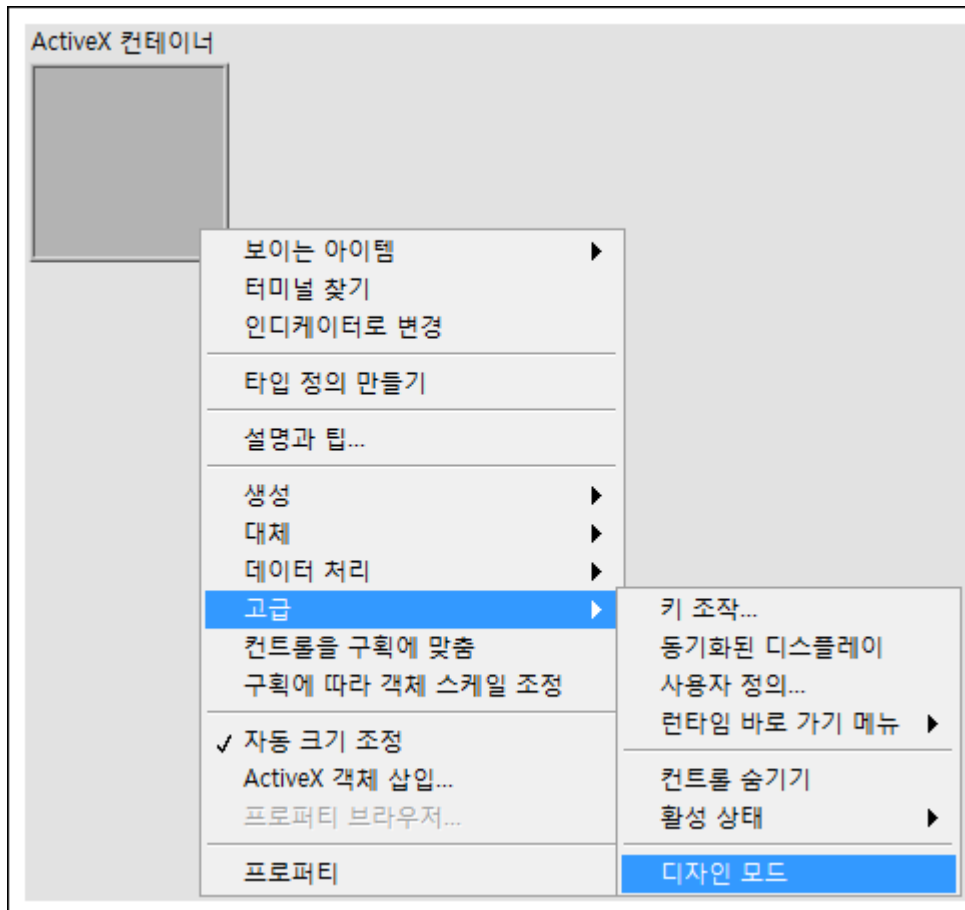
LabVIEW는 프런트패널 객체인 오토메이션 참조 번호 컨트롤 또는 ActiveX 컨테이너로 ActiveX 객체에 접근합니다. 오토메이션 참조 번호 컨트롤을 사용하여 ActiveX 객체를 선택합니다. ActiveX 컨테이너를 사용하여 버튼 또는 문서와 같이 디스플레이 가능한 ActiveX 객체를 선택하고, 해당 객체를 프런트패널에 놓습니다. 블록 다이어그램에서는 두 객체 모두 오토메이션 참조 번호 컨트롤로 나타냅니다.

ActiveX 객체에 대한 디자인 모드

디자인 모드 옵션을 선택하면, ActiveX 컨테이너는 컨트롤 참조를 소유하여 새 컨트롤을 생성하고 해제합니다. 컨트롤은 서로 다른 디자인과 실행 모드를 갖습니다. VI를 실행할 때마다 LabVIEW는 디자인 모드 컨트롤을 제거하고 실행 모드에서 이를 새 컨트롤로 대체합니다. VI가 정지하면 LabVIEW는 실행 모드 컨트롤을 제거하고 새로운 디자인 모드 컨트롤을 생성합니다. ActiveX와 LabVIEW를 사용할 때 디자인 모드와 실행 모드의 관계는 VI 스크립팅이나 VI 서버와 같이 다른 영역의 LabVIEW 프로그래밍에서의 편집 모드와 실행 모드의 관계와 유사합니다.

빌드된 어플리케이션에는 디자인 모드 컨트롤이 포함되어 있지 않습니다. LabVIEW는 VI가 실행을 시작할 때 실행 모드 컨트롤을 생성하고 VI가 정지할 때 컨트롤을 해제합니다. LabVIEW는 디자인 모드가 아닌 실행 모드에서만 이벤트를 발생합니다. 기본 모드는 실행 모드입니다.

디자인 모드 옵션을 설정하는 방법에 대한 정보는 다음 그림을 참조하십시오.



ActiveX 프로퍼티를 설정하는 방법

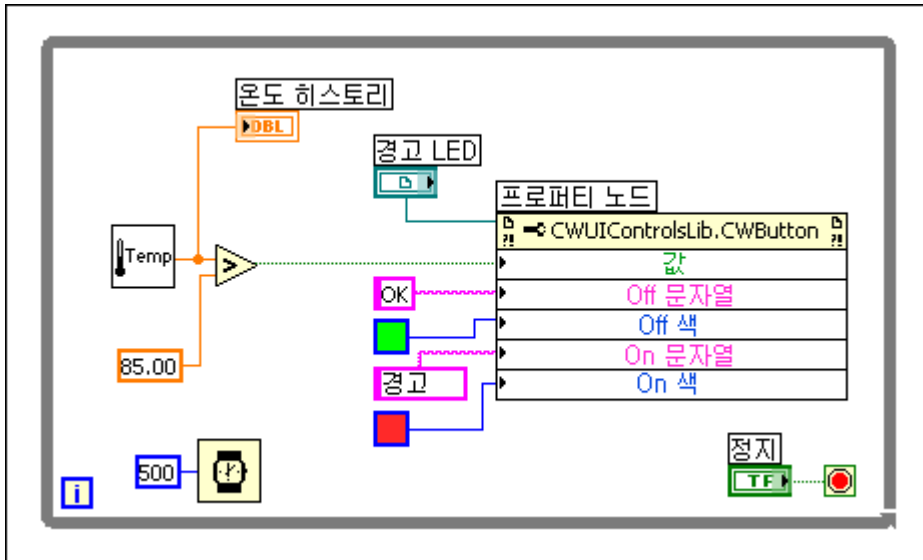
ActiveX 서버를 열거나 ActiveX 컨트롤 또는 문서를 삽입한 후, ActiveX 컨트롤 프로퍼티 브라우저, 프로퍼티 페이지, 그리고 프로퍼티 노드를 이용하여 컨트롤이나 문서와 관련된 프로퍼티를 설정할 수 있습니다.

이름	설명	접근 방법
ActiveX 컨트롤 프로퍼티 브라우저	이 브라우저를 사용하여 ActiveX 컨테이너 안의 ActiveX 컨트롤 또는 문서와 관련된 모든 프로퍼티를 확인하고 설정합니다. ActiveX 컨트롤 프로퍼티 브라우저는 대화식으로 ActiveX 객체의 프로퍼티를 설정하는 쉬운 방법입니다.	다음 방법 중의 하나로 ActiveX 컨트롤 프로퍼티 브라우저에 접근할 수 있습니다: 1. 프론트패널의 컨트롤 또는 문서에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티 브라우저를 선택합니다. 2. 보기>>ActiveX 컨트롤 프로퍼티 브라우저를 선택합니다.
	노트 이 브라우저는 편집 모드에서 컨테이너 내부의 ActiveX 객체에 프로퍼티를 설정할 때에만 사용할 수 있습니다.	

이름	설명	접근 방법
ActiveX 프로퍼티 페이지	많은 ActiveX 객체에는 각 개별 탭에 객체와 관련된 프로퍼티를 정리해 놓은 프로퍼티 페이지가 있습니다. ActiveX 컨트롤 프로퍼티 브라우저와 같이, ActiveX 프로퍼티 페이지를 사용하여 ActiveX 객체의 프로퍼티를 대화식으로 간편하게 설정할 수 있습니다. 또한 ActiveX 컨트롤 프로퍼티 브라우저와 같이, 프로퍼티 페이지를 사용하여 오직 편집 모드에서 컨테이너 내부의 ActiveX 객체에 대해서만 프로퍼티를 설정할 수 있습니다.	ActiveX 프로퍼티 페이지에 접근하기 위하여 프런트패널의 컨테이너에 있는 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 그 객체의 이름을 선택합니다.
	노트 모든 ActiveX 객체가 프로퍼티 페이지를 제공하지는 않습니다.	
프로퍼티 노드	[프로퍼티 노드]를 사용하여 ActiveX의 프로퍼티를 프로그램적으로 설정하십시오.	다음 단계를 완료하여 프로퍼티 노드를 사용하여 ActiveX 프로퍼티를 프로그램적으로 설정합니다: 1. ActiveX 프로퍼티를 생성하려는 ActiveX 객체에 대한 참조를 얻습니다. 2. 블록다이어그램에 프로퍼티 노드를 추가합니다. 3. 객체의 참조를 노드의 참조 입력에 연결합니다.

다음 블록다이어그램의 예에서는 프로퍼티 노드를 사용하여 프로그램적으로 ActiveX 컨트롤의 값 프로퍼티를 변경하여, 온도가 일정 한계를 초과할 때 경고를 알리는 방법에 대해 보여줍니다. National Instruments Measurement Studio User Interface ActiveX Library의 일부인 CWButton ActiveX 컨트롤의 값 프로퍼티는 온도가 화씨 85도 이상에 이르면 변경됩니다.

이 예에서 CWButton 컨트롤은 CWButton 컨트롤의 "on" 상태인 한계 온도에 이르면 LED 처럼 작동하여 색이 변하고, 경고를 디스플레이합니다.



노트 CWButton 컨트롤 프로퍼티에 대해 프로그램적으로 OffText, OffColor, OnText, OnColor 프로퍼티를 설정하는 대신, ActiveX 컨트롤 프로퍼티 브라우저나 프로퍼티 페이지를 사용하여 이러한 컨트롤에 대한 프로퍼티를 설정할 수 있습니다.

LabVIEW를 ActiveX 서버로 사용

다른 어플리케이션의 ActiveX 호출을 통해 LabVIEW 어플리케이션, VI 및 컨트롤의 프로퍼티와 메소드에 접근할 수 있습니다. Microsoft Excel과 같은 다른 ActiveX 활성화 어플리케이션은 LabVIEW의 프로퍼티, 메소드, 개별 VI를 요청할 수 있습니다. 이 때 LabVIEW는 ActiveX 서버로 동작합니다.

예를 들어, Excel 스프레드시트에 VI 그래프를 임베드할 수 있고, 스프레드시트에서 VI 입력으로 데이터를 입력하고 VI를 실행시킬 수 있습니다. VI를 실행하면 데이터가 그래프에 플롯됩니다.

사용자 ActiveX 오토메이션 인터페이스에 대한 지원

LabVIEW를 사용하는 타사 ActiveX 클라이언트에 데이터를 작성하여 LabVIEW ActiveX 서버의 프로퍼티와 메소드에 접근하면, 서버가 공개한 사용자 정의 인터페이스에 접근할 수 있습니다. 이 작업은 IDispatch가 없이 수행할 수 있습니다.

그러나 ActiveX 서버의 개발자는 사용자 인터페이스의 프로퍼티와 메소드의 파라미터가 반드시 오토메이션(IDispatch) 데이터 타입을 갖도록 해야 합니다. 여러 객체가 아닌 하나의 객체에서 여러 개의 인터페이스가 공개되도록 하려면, 서버의 개발자는 반드시 파라미터에 오토메이션(IDispatch) 데이터 타입이 있는지 확인해야 합니다. 오토메이션 데이터 타입이 없는 경우, 단 하나의 객체에만 연결하는 것이 아니라 연결하려는 각 객체에 연결을 확립해야 합니다. LabVIEW 환경에서 이 인터페이스를 사용할 수 있습니다.

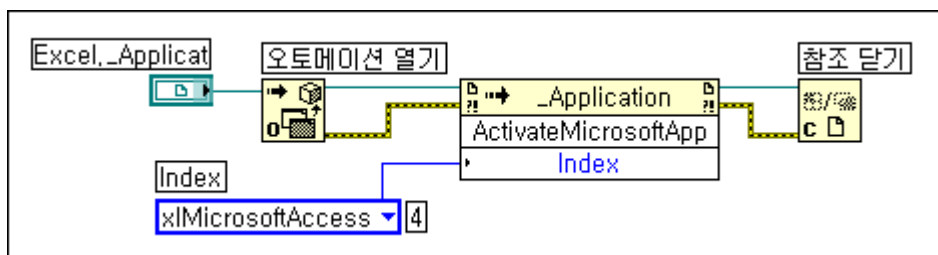
상수를 사용하여 ActiveX VI의 파라미터 설정하기

ActiveX 노드의 몇몇 파라미터는 유효한 값의 비연속적인 리스트를 허용합니다. 링 상수에서 설명적인 이름을 선택하여 파라미터 값을 설정합니다.

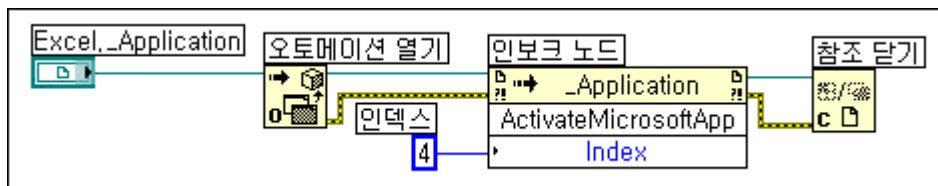
ActiveX VI를 만들 때 링 상수에 접근하려면, 데이터 값을 받는 노드의 파라미터에서 마우스 오른쪽 버튼을 클릭한 후 상수를 생성합니다. 링 상수에서 선택 가능한 목록은 노드에 전달된 참조 번호에 따라 달라집니다. 해당하는 숫자 데이터 값을 보려면, 링 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 디지털 디스플레이를 선택합니다.

다음 블록 다이어그램은 링과 숫자형 상수를 사용하여 파라미터 값을 설정하는 방법을 설명합니다. 이 블록 다이어그램은 Microsoft Excel 어플리케이션에 접근하여 메소드를 실행하고, 인덱스 파라미터에는 여러 개의 옵션이 포함되어 있습니다. 어떤 옵션을 선택하면, LabVIEW는 해당 옵션에 대한 숫자값을 디스플레이합니다.

다음 블록 다이어그램의 디스플레이와 같이, LabVIEW는 현재 선택한 옵션인 MicrosoftAccess의 숫자값을 링 상수 옆의 박스에 디스플레이하도록 기본 설정되어 있습니다.



그러나, 옵션의 숫자값을 숫자형 상수로 디스플레이하도록 선택할 수도 있습니다. 다음 블록 다이어그램에서 보이는 것과 같이, LabVIEW는 옵션의 숫자값을 인덱스 입력에 연결된 숫자형 상수로 디스플레이합니다.



VI에 업데이트된 ActiveX 타입 라이브러리 로드하기

.NET ActiveX 타입 라이브러리의 주요 버전이 변경된 VI를 로드할 때, LabVIEW는 경고 대화 상자를 실행시켜 변경사항에 대해 알려줍니다. ActiveX 타입 라이브러리의 보조 버전 또는 타임스탬프가 변경된 VI를 로드하면, 일단 로드된 VI는 제목 표시줄과 윈도우 메뉴에 디스플레이된 열린 VI 리스트에 별표를 포함합니다. VI를 저장하면, 새로 변경할 때까지 별표는 사라집니다.

Windows Vista 및 이후 버전 OS에서 LabVIEW 어플리케이션 배포하기

Windows Vista의 보안 변경 사항 때문에, Windows Vista 또는 이후 버전에서 LabVIEW 어플리케이션 빌더를 사용할 때에는 다음의 두 가지 문제를 고려해야 합니다:

- LabVIEW는 모든 독립 어플리케이션과 설치 프로그램에 **어플리케이션 매니페스트**를 내장합니다.
- 독립 어플리케이션과 독립 어플리케이션용 설치 프로그램은 **서명**되어 있지 않습니다.

Windows Vista 및 이후 버전의 LabVIEW 어플리케이션 빌더에 대한 최신 정보는 이 도움말 토픽 외에 User Account Control, Digital Signing, and Manifests in LabVIEW Built Applications를 참조할 수 있습니다.

어플리케이션 매니페스트

LabVIEW는 모든 독립 어플리케이션과 설치 프로그램에 어플리케이션 매니페스트를 내장합니다. 매니페스트가 어플리케이션에 임베드되어 있으므로, 매니페스트가 어플리케이션의 디렉토리에서 나타나려면 먼저 **매니페스트를 검색**해야 합니다. 매니페스트는 확장명이 .manifest인 XML 파일입니다. 매니페스트는 Windows에 대한 어플리케이션의 프로퍼티 정보를 포함합니다. 기존의 Windows에도 매니페스트는 있었지만 매니페스트의 보안 섹션은 새로이 추가된 기능입니다. 이제 Windows Vista 또는 이후 버전에서 배포한 어플리케이션에 대해 필요한 실행 레벨을 매니페스트에서 지정할 수 있습니다.

어플리케이션의 실행 레벨을 지정하면 어플리케이션을 시작할 때 필요한 보안 권한이 결정됩니다. 지정한 실행 레벨에 따라 실행에 앞서 어플리케이션이 관리자에게 동의할 것을 요청하는 대화 상자를 디스플레이하거나 관리자 암호 입력을 요청하는 대화 상자를 디스플레이합니다. 관리자 권한이 필요한 어플리케이션은 어플리케이션 아이콘에 보안 쉴드 오버레이가 있습니다. 어플리케이션이 관리자로 실행되는지 또는 표준 사용자로 실행되는지 여부에 따라 어플리케이션이 쓰일 디렉토리나 같은 여러가지 어플리케이션의 프로퍼티가 결정됩니다. 표준 사용자로 어플리케이션을 시작할 경우, 어플리케이션은 보호된 디렉토리나 레지스터에 쓸 수 없습니다. Windows Vista 및 이후 버전에서 보호된 디렉토리의 예로는 프로그램 파일 디렉토리가 있습니다.

Microsoft는 어플리케이션을 표준 사용자 권한에서 실행하기를 권장합니다. 이런 경우, 어플리케이션이 OS의 보호된 영역을 사용하지 않도록 설계해야 합니다. 관리자 권한이 아닌 상태에서 실행하면 모든 사용자가 원하는 대로 어플리케이션을 실행할 수 있습니다.

다음 코드는 독립 어플리케이션에 대한 예제 매니페스트입니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
manifestVersion="1.0">

<assemblyIdentity version="1.0.0.0"
```

```

processorArchitecture="X86"

name="National Instruments.LabVIEW.LabVIEW "

type="win32"↵

<description>LabVIEW Development System</description>

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">

<security>

<requestedPrivileges>

<requestedExecutionLevel

level="asInvoker"

uiAccess="false"↵

</requestedPrivileges>

</security>

</trustInfo>

</assembly>

```

독립 어플리케이션에서 요청되는 실행 레벨은 다음 중에서 어떤 값도 될 수 있습니다:

- **asInvoker**—(기본) 독립 어플리케이션이 사용자(표준 또는 관리자)와 동일한 접근 권한으로 실행됩니다. 어플리케이션은 실행전 사용자에게 동의를 요청하는 대화 상자를 디스플레이하지 않습니다. 어플리케이션이 표준 사용자로 실행되는 경우, 어플리케이션은 보호된 디렉토리에 쓸 수 없습니다. 어플리케이션이 관리자로 실행되는 경우, 어플리케이션은 보호된 디렉토리에 쓸 수 있습니다.
- **highestAvailable**—독립 어플리케이션이 가장 높은 접근 권한에서 실행됩니다. 사용자에게 표준 사용자 권한만이 있는 경우, 어플리케이션은 사용자에게 동의를 요청하는 대화 상자를 디스플레이하지 않지만 보호된 디렉토리에 쓸 수 없습니다. 사용자에게 관리자 권한이 있는 경우, 어플리케이션은 사용자에게 동의를 요청하거나 또는 관리자 암호를 입력하라는 대화 상자를 디스플레이하며 보호된 디렉토리에 쓸 수 있습니다.
- **requireAdministrator**—독립 어플리케이션이 관리자 레벨에서 실행됩니다. 사용자에게 표준 사용자 특권만이 있는 경우, 어플리케이션은 관리자 암호 입력을 요청하는 대화 상자를 디스플레이하며 보호된 디렉토리에 쓸 수 있습니다. 사용자에게 관리자 권한이 있는 경우, 어플리케이션은 실행전 사용자에게 동의를 요청하는 대화 상자를 디스플레이합니다. 어플리케이션은 보호된 디렉토리에 쓸 수 있습니다.

LabVIEW 8.2.1 또는 이후 버전에서 독립 어플리케이션을 개발하여 Windows XP 또는 이전 버전에 배포하는 경우, 매니페스트를 포함하고 요청된 실행 레벨이 있는 독립 어플리케이션은 Windows Vista 및 이후 버전에서와는 다르게 동작합니다. Windows XP는 매니페스트의 보안 섹션을 무시합니다.

LabVIEW로 만든 설치 프로그램의 `install.exe`도 매니페스트를 포함합니다. 필요한 실행 레벨은 기본적으로 `requireAdministrator`입니다. `install.exe`가 시스템 레벨 파일, 셋팅, 설정을 업데이트하기 때문에, `install.exe`의 요청 실행 레벨을 변경해서는 안 됩니다. `install.exe`의 요청 실행 레벨을 변경하면 설치 프로그램이 제대로 작동하지 않을 수 있습니다.

디지털 서명

디지털 서명을 사용하면 어플리케이션 또는 시작 프로그램 생성자를 확인하고, 어플리케이션이나 설치 프로그램의 설치와 실행이 안전한지 결정할 수 있습니다. Windows Vista 및 이후 버전은 추가적인 레벨의 보안을 제공하므로, 서명되지 않은 어플리케이션이나 설치 프로그램을 설치하거나, 실행할 때 위험성에 대해 경고하는 대화 상자가 나타날 수 있습니다. 기본으로, LabVIEW는 독립 어플리케이션이나 설치 프로그램을 서명하지 않기 때문에 디지털 서명을 사용자 정의할 수 있습니다. 불필요한 대화 상자가 나타나지 않도록, 어플리케이션 프로퍼티 대화 상자의 Windows 보안 페이지나 설치 프로그램 프로퍼티 대화 상자를 Windows 보안 페이지를 사용하여, 빌드 시 독립 어플리케이션과 설치 프로그램을 서명하십시오.

어플리케이션이나 설치 프로그램에 서명하려면 다음의 도구가 필요합니다:

- 서명 확인서—Verisign과 같은 여러 회사에서 서명 확인서(Signing Certificate)를 구매할 수 있습니다. 서명 확인서는 회사나 기관을 위해서만 구입할 수 있습니다. 개인은 서명 확인서를 구매할 수 없습니다. 서명 인증은 반드시 개인 정보 교환(PFX) 파일이어야 합니다.
- 서명 도구—Windows Driver Kit은 서명 도구(Signing Tool)를 포함합니다. 서명 도구를 사용하여 디지털 서명을 적용하십시오.

독립 어플리케이션을 서명하려면, `.exe` 파일에만 서명하면 됩니다. LabVIEW로 만든 설치 프로그램을 서명하려면, `install.exe` 파일에만 서명하면 됩니다. 독립 어플리케이션과 설치 프로그램의 경우 다른 파일에 서명할 필요는 없습니다. 디지털 서명을 위해서는 타사(Third-party Provider)에서 디지털 확인을 위한 제품을 구입해야 하지만 코드 서명에 대해서는 구입할 필요가 없습니다.



관련 정보

[User Account Control, Digital Signing, and Manifests in LabVIEW Built Applications](#)

Windows Vista 및 이후 버전 OS에 빌드 배포하기

Microsoft의 Windows Vista 및 이후 버전 OS는 사용자 계정 컨트롤과 같은 추가적인 레벨의 보안을 포함합니다. 어플리케이션을 설치하거나 다른 작업을 완료하기 전 사용자 암호를 입력해야 하므로, 사용자 계정 컨트롤을 사용하면 바이러스가 시스템에 유입되거나 시스템을 손상할 수 있는 변경사항이 적용되는 것을 방지할 수 있습니다. 또한

Windows Vista 및 이후 버전OS에 표준 사용자로 로그인하는 사용자는 디스크의 특정한 위치에서만 쓸 수 있습니다. 표준 사용자는 이러한 OS에서의 기본 로그인 값입니다.

관련 토픽

다음 토픽은 Windows Vista 및 이후 버전에서 빌드 스펙을 배포할 때 고려할 사항에 대해 설명합니다.

- [Windows Vista 및 이후 버전 OS에서 LabVIEW 어플리케이션 배포하기](#)
- [어플리케이션 매니페스트 작업하기](#)

Windows Embedded Standard 타겟에 설치 프로그램 배포하기

호스트 컴퓨터에서 VI를 개발한 후, VI를 실행 가능한 어플리케이션으로 빌드할 수 있습니다. 또한 실행 가능한 어플리케이션이나 기타 파일을 포함하는 설치 프로그램을 빌드하여, 해당 설치 프로그램을 Windows Embedded Standard 타겟에 배포하거나 실행할 수 있습니다. 설치 프로그램을 빌드할 때에는, 어플리케이션 실행할 때 필요할 모든 구성요소를 설치 프로그램에 포함시키십시오.



노트

독립 어플리케이션 또는 설치 프로그램을 빌드하려면, 어플리케이션 빌더가 설치되어 있어야 합니다. LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. 다른 LabVIEW 개발 시스템을 사용하는 경우, National Instruments 웹사이트를 방문하여 어플리케이션 빌더를 별도로 구매할 수 있습니다.

어플리케이션이나 설치 프로그램을 빌드하려면, 반드시 먼저 빌드 스펙을 생성해야 합니다. 빌드 스펙은 포함할 파일, 생성할 디렉토리, VI의 셋팅 등 빌드에 대한 모든 셋팅을 포함합니다. 빌드 스펙은 [프로젝트 탐색기](#) 윈도우의 Windows Embedded Standard 타겟 아래에 있는 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 새로 만들기 » 어플리케이션(EXE) 또는 새로 만들기 » 설치 프로그램을 선택하여 언제든지 생성할 수 있습니다.

빌드 스펙을 생성한 후 [프로젝트 탐색기](#) 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 옵션을 선택합니다.

- 빌드—어플리케이션이나 설치 프로그램과 같은 빌드 스펙을 빌드합니다. 이 옵션은 설치 프로그램을 타겟에 자동으로 배포하지 않습니다.
- 배포—설치 프로그램을 빌드하고 타겟에 배포합니다. 이 옵션은 자동으로 설치 프로그램을 타겟에 실행하지 않습니다.
- 설치—설치 프로그램을 타겟에서 빌드 스펙에 지정된 위치로 빌드, 배포 및 실행합니다.

관련 정보

어플리케이션 개발하기 및 배포하기

빌드 스펙 생성하기

빌드 스펙 사용하기

Windows 연결

LabVIEW에서는 .NET 또는 ActiveX 기술을 사용하여 다른 Windows 기반의 어플리케이션에 접속할 수 있습니다.

.NET은 Microsoft의 .NET 기술을 의미합니다. .NET Framework는 웹 기반 어플리케이션, Smart Client 어플리케이션 및 XML 웹 서비스를 만들고 배포하고 실행하는데 사용하는 .NET 환경의 프로그래밍 기반입니다. .NET Framework를 설치해야 합니다. .NET과 framework 설치에 관한 추가적인 정보는 MSDN(Microsoft Developer Network) 웹 사이트를 참조하십시오.



노트

LabVIEW에서 .NET 객체를 생성하고 이 객체와 통신하려면 LabVIEW와 함께 설치되는 .NET CLR 4.0이 필요합니다. .NET 2.0 혼합 모드 어셈블리를 로드하려면 .NET 2.0 설정 파일을 사용해야 합니다. LabVIEW에서 .NET 제한에 대한 더 자세한 정보는 LabVIEW에서 .NET를 사용할 때의 요구조건을 참조하십시오. National Instruments는 LabVIEW 프로젝트 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

ActiveX는 Microsoft의 ActiveX 기술과 OLE 기술을 의미합니다. ActiveX 오토메이션을 사용하는 경우, LabVIEW와 같은 Windows 기반의 어플리케이션은 다른 Windows 기반 어플리케이션에서도 접근할 수 있도록 공개된 객체, 명령, 그리고 함수를 제공합니다. ActiveX에 대한 추가 정보는 MSDN 문서를 참조하십시오.

- Inside OLE, Kraig Brockschmidt 저, second edition
- Essential COM, Don Box 저

LabVIEW에서 ActiveX를 사용하는 것에 대한 추가적인 정보는 ni.com의 ActiveX and LabVIEW를 참조하십시오.

관련 예제 찾기



관련 정보

- [LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건](#) (745 페이지 참조)
- [LabVIEW에서 .NET 2.0, 3.0, 3.5 어셈블리 로드하기](#) (1435 페이지 참조)
- [.NET 객체 생성하기](#) (746 페이지 참조)
- [LabVIEW 프로젝트에서 라이브러리 사용하기](#) (296 페이지 참조)
- [콜백 VI](#) (174 페이지 참조)
- [LabVIEW에서 .NET 사용하기](#) (743 페이지 참조)
- [LabVIEW에서 ActiveX 사용하기](#) (351 페이지 참조)
- [웹 서비스 반입하기](#) (681 페이지 참조)
- [프로그램적으로 모달이 아닌 윈도우 호출하기](#) (988 페이지 참조)
- [웹 서비스를 VI의 라이브러리로 반입하기](#) (988 페이지 참조)
- [.NET와 ActiveX 이벤트 등록 및 핸들링하기](#) (1430 페이지 참조)

LabVIEW의 루프와 구조

루프와 기타 구조

구조는 텍스트 기반 프로그래밍 언어의 루프 및 케이스 명령문을 그래픽하게 표현한 것입니다.

다음 토픽은 구조 팔레트에 위치한 구조 사용에 대해 정보를 제공합니다.

구조 용도	구조
코드의 블록 반복하기	
지정된 횟수만큼 또는 에러가 발생할 때까지	For 루프
조건이나 에러가 발생할 때까지	While 루프
시간 제한과 지연을 수반하여 정지 조건이 발생할 때까지	Timed 루프
시간 제한과 지연을 순서적으로 수반하여 정지 조건이 발생할 때까지	프레임이 있는 Timed 루프
순서대로 모든 코드 실행하기	
모든 서브다이어그램이 보임	플랫 시퀀스 구조
하나의 서브다이어그램만 보임	다층 시퀀스 구조

구조 용도	구조
시간 제한과 지연 수반	Timed 시퀀스 구조
조건적으로 코드 실행하기	
값에 기반	케이스 구조
이벤트에 기반	이벤트 구조
코드 비활성화하기	
조건에 기반	조건적 비활성화 구조
특정 블록인 경우	다이어그램 비활성화 구조
컴파일 결과에 기반	타입 특수화 구조
메모리 효율성 높이기	In Place 원소 구조



관련 정보

[For 루프: 지정된 횟수만큼 작업 반복하기](#) (367 페이지 참조)

[While 루프: 조건이 발생할 때까지 작업 반복하기](#) (368 페이지 참조)

[Timed 루프 및 프레임이 있는 Timed 루프 설정하기 \(Real-Time, Windows\)](#) (385 페이지 참조)

[Timed 루프 및 프레임이 있는 Timed 루프 설정하기 \(Real-Time, Windows\)](#) (385 페이지 참조)

[시퀀스 구조: 순차적으로 코드 섹션 실행하기](#) (375 페이지 참조)

[시퀀스 구조: 순차적으로 코드 섹션 실행하기](#) (375 페이지 참조)

[Timed 시퀀스 구조 설정하기\(Real-Time, Windows\)](#) (389 페이지 참조)

[케이스 구조: 입력값에 따라 코드 섹션 실행하기](#) (374 페이지 참조)

[비활성화 구조: 코드 실행 막기](#) (376 페이지 참조)

[비활성화 구조: 코드 실행 막기](#) (376 페이지 참조)

[비활성화 구조: 코드 실행 막기](#) (376 페이지 참조)

[In Place 원소 구조: 메모리 효율성 높이기](#) (379 페이지 참조)

[루프와 기타 구조](#) (364 페이지 참조)

[구조를 사용하여 하나 이상의 코드 섹션 실행하기](#) (373 페이지 참조)

[In Place 원소 구조: 메모리 효율성 높이기](#) (379 페이지 참조)

[Timed 구조의 셋팅 설정하기 \(Real-Time, Windows\)](#) (390 페이지 참조)

[케이스 구조: 입력값에 따라 코드 섹션 실행하기](#) (374 페이지 참조)

[블록다이어그램에 구조 추가하기](#) (1215 페이지 참조)

[구조 크기 조정하기](#) (1216 페이지 참조)

[구조 안에 객체 추가하기](#) (1216 페이지 참조)

[구조를 다른 구조로 대체하기](#) (1217 페이지 참조)

[구조에서 객체를 삭제하지 않고 구조를 삭제하기](#) (1217 페이지 참조)

루프를 사용하여 반복적인 작업 컨트롤하기

For 루프와 While 루프를 사용하여 블록다이어그램에서 코드의 블록이나 작업을 반복합니다.



노트

For 루프 또는 While 루프 내에서 참조 번호를 여는 경우, 루프의 각 반복에서 해당 참조 번호를 닫아야 합니다. 그렇지 않으면 LabVIEW가 그 참조 번호에 중복으로 메모리를 할당하며 VI 실행이 끝나기 전까지 메모리를 해제하지 않습니다.

관련 토픽

- For 루프 실행의 횟수 지정하기
- While 루프의 조건 지정하기
- 루프를 사용하여 배열이나 집합의 개별 원소 처리하기
- 루프의 조건 터미널로 에러 핸들링하기
- 루프 반복 사이에서 값 전달하기
- Timed 루프 설정하기



관련 정보

[For 루프: 지정된 횟수만큼 작업 반복하기](#) (367 페이지 참조)

[While 루프: 조건이 발생할 때까지 작업 반복하기](#) (368 페이지 참조)

[참조 번호 컨트롤과 인디케이터](#) (87 페이지 참조)

[For 루프의 작업 실행 횟수 지정하기](#) (1225 페이지 참조)

[While 루프의 조건 지정하기](#) (1231 페이지 참조)

[루프를 사용하여 배열이나 집합의 개별 원소 처리하기](#) (1218 페이지 참조)

[루프의 조건 터미널로 에러 핸들링하기](#) (1225 페이지 참조)

[루프 반복 사이에서 값 전달하기](#) (369 페이지 참조)

[Timed 구조의 셋팅 설정하기 \(Real-Time, Windows\)](#) (390 페이지 참조)

For 루프: 지정된 횟수만큼 작업 반복하기

For 루프는 서브다이어그램이라고 불리는 루프 안의 코드 한 섹션을 정해진 횟수만큼 실행합니다. 카운트 N 터미널이 서브다이어그램을 몇 번 실행할지 지정합니다. 서브다이어그램을 한번 실행하는 것을 반복이라고 지칭합니다.

어떤 작업을 정해진 횟수만큼 실행하려고 할 때 For 루프를 사용합니다. 예를 들어, 측정값을 정해진 횟수만큼 파일에 쓰려면, 측정값을 쓰는 코드를 For 루프 안에 놓습니다. 카운터 터미널에 연결하는 숫자값이 쓰기 작업의 발생 횟수를 지정합니다. For 루프는 지정한 횟수만큼 서브다이어그램을 반복한 후, 정지합니다.

특정 조건이 발생할 때 For 루프가 정지하도록 하려면, 조건 터미널을 추가합니다. 조건 터미널을 추가하면 For 루프는 해당 조건이 발생하거나 지정된 횟수만큼 반복 실행을 완료할 때 정지합니다.

For 루프를 통해 데이터 전달하기

For 루프는 이전 반복에서 다음 반복으로 값을 전달하지 않도록 기본 설정되어 있습니다. 이전 반복에서의 값을 다음 반복에서 사용하려면 For 루프에 시프트 레지스터를 추가합니다. 예를 들어, 이전 반복에서 수행된 수학 연산의 결과를 다음 반복에서 수학 연산의 초기값으로 사용하려면 시프트 레지스터를 사용합니다.

배열 또는 집합 데이터 타입을 For 루프의 입력 터널에 연결하면, For 루프는 기본적으로 오토인덱싱을 활성화합니다. 오토인덱싱이 활성화되면, For 루프는 루프가 반복될 때마다 배열 또는 집합 전체를 읽고 처리하기보다 배열 또는 집합의 원소를 한 개씩 읽고 처리합니다. 오토인덱싱을 활성화 또는 비활성화하여 For 루프가 데이터를 처리하는 방법을 변경할 수 있습니다.

For 루프 실행 속도 향상하기

For 루프의 실행 속도가 VI 실행에 결정적인 영향을 미치고 VI가 실행되는 컴퓨터에 여러 개의 프로세서가 있는 경우, For 루프에서 병렬 반복을 활성화합니다. 이 때, 여러 또는 경 고가 발생하면 병렬 반복을 비활성화하십시오. 병렬 반복을 활성화하면, 프로세서는 동시에 반복을 실행하여 속도를 향상시킵니다.

While 루프: 조건이 발생할 때까지 작업 반복하기

텍스트 기반 프로그래밍 언어의 Do 루프 또는 Repeat-Until 루프와 유사하게, 다음과 같이 While 루프는 조건이 발생할 때까지 서브다이어그램을 실행합니다.



While 루프는 입력 터미널인 조건 터미널이 특정 불리언 값을 받을 때까지 서브다이어그램을 실행합니다. While 루프의 조건 터미널은 **조건 터미널이 있는 For 루프**와 마찬가지로 작동합니다. 그러나 While 루프에는 지정된 반복 카운트가 없기 때문에 조건이 발생하지 않는 한 무한히 실행됩니다.

다음과 같이 반복 터미널은 출력 터미널이며 완료된 반복 횟수가 포함되어 있습니다.



While 루프의 반복 카운트는 항상 0에서 시작합니다.

While 루프를 통해 데이터 전달하기

While 루프는 이전 반복에서 다음 반복으로 값을 전달하지 않도록 기본 설정되어 있습니다. 이전 반복에서의 값을 다음 반복에서 사용하려면 While 루프에 시프트 레지스터를 추가합니다. 예를 들어, 이전 반복에서 수행된 수학 연산의 결과를 다음 반복에서 수학 연산의 초기값으로 사용하려면 시프트 레지스터를 사용합니다.

배열 또는 집합 데이터 타입을 While 루프의 입력 터널에 연결하면, While 루프는 기본적으로 오토인덱싱을 비활성화합니다. 루프 경계의 터널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 인덱싱 활성화를 선택하면 오토인덱싱을 활성화할 수 있습니다. 오토인덱싱이 활성화되면, While 루프는 루프가 반복될 때마다 배열 또는 집합의 원소를 한 개씩 처리합니다. 루프의 반복 횟수가 배열 또는 집합의 크기보다 크면, While 루프는 배열 또는 집합의 원소를 모두 처리한 후, 다음 반복에서 배열 또는 집합 원소 타입의 기본값을 사용합니다. 오토인덱싱을 활성화 또는 비활성화하여 While 루프가 데이터를 처리하는 방법을 변경할 수 있습니다.

루프 반복 사이에서 값 전달하기

다음 객체를 사용하여 For 루프, While 루프, 또는 (Real-Time, Windows) Timed 루프에서 값을 전송합니다.

- 시프트 레지스터—다음 루프 반복으로 하나의 값을 전달하거나 여러 값을 전달합니다.
- 에러 레지스터—병렬 For 루프에서 에러 클러스터를 전달합니다.
- 피드백 노드—노드가 실행될 때마다 이전 실행의 값을 전달하거나 값을 새로 설정합니다.



관련 정보

[For 루프: 지정된 횟수만큼 작업 반복하기](#) (367 페이지 참조)

[While 루프: 조건이 발생할 때까지 작업 반복하기](#) (368 페이지 참조)

[Timed 구조의 특성\(Real-Time, Windows\)](#) (381 페이지 참조)

[시프트 레지스터: 루프 반복 사이에 값 전달하기](#) (370 페이지 참조)

[다음 루프 반복으로 값 전달하기](#) (1233 페이지 참조)

[다음 루프 반복으로 여러 값 전달하기](#) (1234 페이지 참조)

[에러 레지스터: 병렬 반복에서 에러 핸들링하기](#) (369 페이지 참조)

[병렬 반복: For 루프 실행 속도 향상](#) (1227 페이지 참조)

[피드백 노드: 루프 반복 사이에 데이터 저장하기](#) (371 페이지 참조)

[최종 값을 이전 실행에서 피드백 노드로 전달하기](#) (1237 페이지 참조)

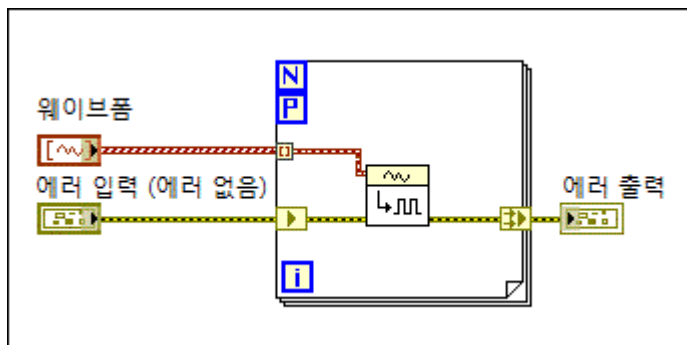
[루프 실행 시마다 피드백 노드의 입력값 리셋하기](#) (1238 페이지 참조)

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

에러 레지스터: 병렬 반복에서 에러 핸들링하기

병렬 반복이 활성화된 For 루프에서 LabVIEW는 에러 레지스터를 사용하여 루프를 통해 에러 클러스터를 전달합니다. 에러 레지스터는 루프가 반복될 때마다 각 반복에서 발생하는 에러 핸들링을 지원합니다.

에러 레지스터는 다음 블록 다이어그램과 같이 병렬 For 루프의 양쪽 옆에 한 쌍의 터미널로 나타납니다.



에러 레지스터의 왼쪽 터미널은 인덱싱하지 않는 입력 터널처럼 동작하며 모든 반복에서 동일한 값을 생성합니다. 에러 레지스터의 오른쪽 터미널은 각 반복의 값을 합쳐서 인덱스에 따른 가장 초기 반복의 에러 또는 경고 값이 에러 레지스터의 출력 값이 되도록 합니다. For 루프가 0번 반복되면 왼쪽 터널에 연결한 값이 오른쪽 터널의 출력으로 전달됩니다.

사용자가 For 루프에서 병렬 반복을 설정할 때 LabVIEW는 자동으로 **시프트 레지스터**를 에러 레지스터로 변환하여 시프트 레지스터를 통해 에러가 전달되도록 유지합니다. 또한 터널을 마우스 오른쪽 버튼으로 클릭하고 생성할 터널 타입을 선택하여 터널 타입을 변경할 수 있습니다.

시프트 레지스터: 루프 반복 사이에 값 전달하기

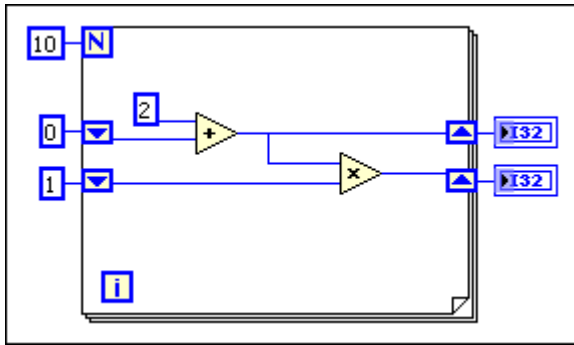
다음 반복으로 **이전 반복의 값을 루프를 통해 전달**하고자 할 때 시프트 레지스터를 사용합니다. 다음과 같이 시프트 레지스터는 루프 경계의 양 옆에 서로 반대인 터미널 쌍으로 나타납니다.



루프 오른쪽의 터미널에는 윗방향 화살표가 있고 반복이 끝날 때마다 데이터를 저장합니다. LabVIEW는 시프트 레지스터의 오른쪽 터미널에 저장된 데이터를 왼쪽 터미널로 전송합니다. 그 후에는 루프가 왼쪽 터미널에서 받은 데이터를 다음 반복의 초기값으로 사용합니다. 이 과정은 루프의 모든 반복이 실행될 때까지 계속됩니다. 루프가 실행된 후에 루프의 오른쪽의 터미널은 시프트 레지스터에 저장된 마지막 값을 반환합니다.

시프트 레지스터는 모든 데이터 타입을 전달할 수 있으며, 시프트 레지스터에 연결된 첫 번째 객체와 일치하도록 자동으로 데이터의 타입을 변환시킵니다. 각 시프트 레지스터의 터미널에 연결하는 데이터는 동일 타입이어야 합니다. 에러 클러스터가 **병렬 For 루프**의 시프트 레지스터에 연결되면 시프트 레지스터가 **에러 레지스터**로 변환됩니다.

한 루프에 여러 개의 시프트 레지스터를 놓을 수 있습니다. 루프 속에서 여러 개의 실행이 이전 루프 값을 사용해야 될 경우, 다음 블록 다이어그램과 같이 여러 시프트 레지스터를 사용하여 구조 내 각기 다른 프로세스의 데이터 값을 저장합니다.



이전 블록다이어그램에서 오른쪽 위 시프트 레지스터 터미널은 첫번째 반복에서 2, 0과 2의 합을 추가적인 작업의 두번째 반복을 위해 왼쪽 위 터미널에 초기값으로 전달합니다. 오른쪽 아래 터미널은 첫번째 반복에서 1과 2의 곱인 2를 곱셈 작업의 두번째 반복을 위해 왼쪽 아래 터미널에 초기값으로 전달합니다.

두번째 반복은 2와 2를 더하고 세번째 반복을 위해 왼쪽 위 터미널에 결과인 4를 전달합니다. 또한 두번째 반복도 2와 2를 곱하고 세번째 반복을 위해 왼쪽 아래 터미널에 결과인 8을 전달합니다.

열번 반복 후 오른쪽 위 터미널은 더하기 작업의 마지막 결과를 위쪽 인디케이터에 전달하고 오른쪽 아래 터미널은 곱하기 작업의 마지막 결과를 아래쪽 인디케이터에 전달합니다.

데이터 통신 방법

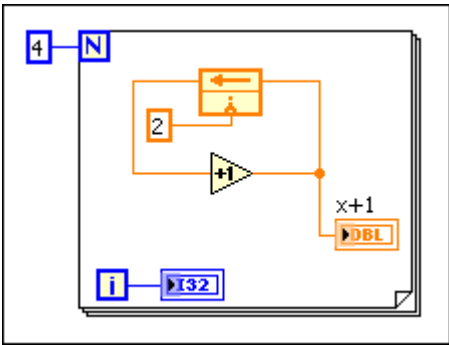
피드백 노드: 루프 반복 사이에 데이터 저장하기

다음과 같이 피드백 노드는 루프의 **시프트 레지스터**와 비슷하게 작동합니다.



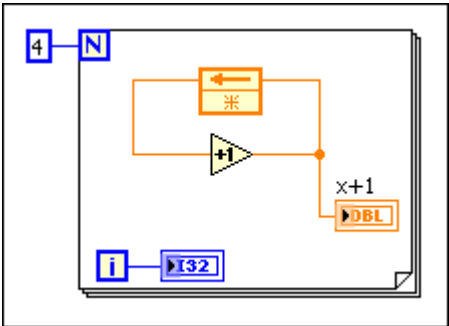
시프트 레지스터처럼, 피드백 노드는 루프가 반복을 완료하면 데이터를 저장한 이후에 다음 루프 반복으로 저장된 데이터를 보냅니다. 다음 반복은 데이터를 읽고 새 값을 데이터에 쓴 후 다시 루프를 통해 보냅니다. 이 과정은 루프가 실행을 마칠 때까지 계속됩니다.

피드백 노드가 첫번째 실행 후 다시 초기화될 때, 한 실행에서 다음 실행으로 전달되는 각 루프 반복의 값은 변하지 않습니다. 이런 경우, 다음 블록다이어그램에서 보이는 것처럼, 초기화 터미널을 피드백 노드로 연결합니다.



첫번째 실행		두번째 실행	
반복 횟수	마지막 값	반복 횟수	마지막 값
0	3	0	3
1	4	1	4
2	5	2	5
3	6	3	6

다음과 같이 초기값을 초기화 터미널에 연결하지 않는 경우 피드백 노드는 다르게 작동합니다.



첫번째 실행		두번째 실행	
반복 횟수	마지막 값	반복 횟수	마지막 값
0	1	0	5
1	2	1	6
2	3	2	7
3	4	3	8

이전 블록 다이어그램에서 초기화 터미널은 데이터 타입의 기본값인 0을 첫번째 실행의 초기값으로 사용하고, 이후의 실행에서는 피드백 노드를 다시 초기화하지 않습니다.

루프와 하위 루프에서 노드와 초기화 터미널을 분리할 수는 있지만 초기화 터미널을 노드가 들어있는 하위 구조 세트 밖으로 이동시킬 수는 없습니다. 또한 SubVI를 생성할 때에도 노드와 초기화 터미널을 분리할 수 없습니다. 노드와 초기화 터미널 중 어느 하나만 들어있는 블록 다이어그램의 섹션에서 SubVI를 생성하려는 경우, LabVIEW는 에러를 반환합니다.



노트

Properties 대화상자의 모양페이지를 이용해 Z 변환 보기를 변경할 수 있는데, 이는 피드백 컨트롤 이론 및 신호 처리에서의 az^{-1} 블록과 비슷하게 보입니다. 피드백 노드를 Z 변환 보기로 변경하면, 피드백 노드의 외형적 모양만 변하며 노드의 기능은 변하지 않습니다.



데이터 통신 방법 홈

구조를 사용하여 하나 이상의 코드 섹션 실행하기

다음 구조는 여러 개의 서브다이어그램을 포함합니다. 서브다이어그램은 구조 내의 한 코드 섹션입니다.

- 케이스 구조
- 비활성화 구조
- 이벤트 구조
- In Place 원소 구조
- 시퀀스 구조

관련 토픽

- 서브다이어그램 추가하기, 복제하기, 사이에서 이동하기, 삭제하기, 재배포하기
- 케이스 구조에서 케이스 서브다이어그램 값 지정하기
- 플랫폼 시퀀스 구조의 프레임을 통해 데이터 전달하기
- 다중 시퀀스 구조의 프레임을 통해 데이터 전달하기

- 조건적 비활성화 구조의 기호 및 값 선택하기



관련 정보

[케이스 구조: 입력값에 따라 코드 섹션 실행하기](#) (374 페이지 참조)

[비활성화 구조: 코드 실행 막기](#) (376 페이지 참조)

[In Place 원소 구조: 메모리 효율성 높이기](#) (379 페이지 참조)

[시퀀스 구조: 순차적으로 코드 섹션 실행하기](#) (375 페이지 참조)

[서브다이어그램 추가 및 복제하기](#) (1241 페이지 참조)

[서브다이어그램 사이에서 움직이기](#) (1242 페이지 참조)

[서브다이어그램 삭제하기](#) (1243 페이지 참조)

[서브다이어그램 재배치하기](#) (1243 페이지 참조)

[조건 코드 실행을 위한 케이스 구조 생성하기](#) (1244 페이지 참조)

[플랫 시퀀스 구조의 프레임을 통해 데이터 전달하기](#) (1250 페이지 참조)

[다층 시퀀스 구조의 프레임을 통해 데이터 전달하기](#) (1250 페이지 참조)

[조건적 비활성화 구조의 조건 설정하기](#) (1252 페이지 참조)

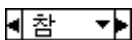
케이스 구조: 입력값에 따라 코드 섹션 실행하기

다음과 같이 케이스 구조에는 2개 이상의 [서브다이어그램](#)이 있습니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.



한 번에 하나의 케이스 서브다이어그램만 볼 수 있으며, 구조는 한 번에 하나의 서브다이어그램만 실행합니다. 입력값은 어느 서브다이어그램을 실행할 것인지 결정합니다. 케이스 구조는 텍스트 기반 프로그래밍 언어의 switch 문이나 if...then...else 문과 유사합니다.

다음과 같이 케이스 구조 위의 케이스 선택자 라벨에는 가운데에 그 케이스에 대응하는 선택자 값의 이름이 있고 양쪽에 증가 감소 화살표가 있습니다.



시퀀스 구조: 순차적으로 코드 섹션 실행하기

시퀀스 구조에는 순차적인 순서로 실행되는 하나 이상의 **서브다이어그램** 또는 프레임이 포함됩니다. 시퀀스 구조의 각 프레임 안에서 블록다이어그램의 다른 부분처럼, 데이터 의존성이 노드의 실행 순서를 결정합니다.

시퀀스 구조에는 **플랫 시퀀스** 구조와 **다층 시퀀스** 구조의 두 가지 타입이 있습니다. 시퀀스 구조는 코드를 숨기므로 꼭 필요한 곳에만 사용하십시오. 시퀀스 구조보다는, 데이터 흐름에 기반하여 실행 순서를 제어하는 것이 더욱 좋습니다. 시퀀스 구조의 경우, 시퀀스로컬 변수를 사용할 때마다 왼쪽에서 오른쪽으로 향하는 데이터 흐름 패러다임을 깨게 됩니다.



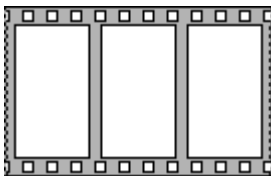
팁

데이터 흐름 컨트롤을 원활하게 하려면 대신 에러 클러스터를 사용하십시오. **흐름 파라미터**를 사용할 수 없고 VI에서 시퀀스 구조를 꼭 사용해야 하는 경우, 플랫 시퀀스 구조 사용을 고려하십시오.

케이스 구조와 달리 시퀀스 구조의 출력 터널에는 데이터 소스가 하나만 있을 수 있습니다. 출력은 모든 프레임으로부터 나올 수 있습니다. 케이스 구조에서처럼 입력 터널의 데이터는 플랫 시퀀스나 다층 시퀀스 구조의 모든 프레임에서 이용할 수 있습니다.

플랫 시퀀스 구조

다음과 같이 플랫 시퀀스 구조는 프레임에 연결된 모든 데이터 값이 사용 가능할 때 왼쪽에서 오른쪽으로 프레임을 실행합니다. 프레임이 실행을 종료할 때 데이터는 각 프레임을 떠납니다. 이는 한 프레임의 입력값이 다른 프레임의 출력값에 따라 달라질 수도 있다는 의미입니다.

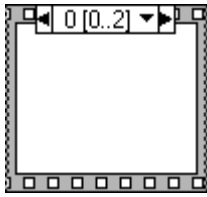


플랫 시퀀스 구조에 **프레임을 추가하거나 삭제**할 때, 구조는 자동적으로 크기 조정됩니다.

다층 시퀀스로 **플랫 시퀀스를 변경**한 다음 다시 플랫 시퀀스로 변경하는 경우, LabVIEW는 모든 입력 터미널을 시퀀스의 첫번째 프레임으로 이동합니다. 마지막 플랫 시퀀스는 다층 시퀀스처럼 작동합니다. 다층 시퀀스를 첫번째 프레임에 모든 입력 터미널을 가진 플랫 시퀀스로 변경한 후, 와이어를 원래 플랫 시퀀스에 위치해 있었던 곳으로 이동시킬 수 있습니다.

다층 시퀀스 구조

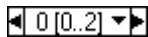
다음과 같이 다층 시퀀스 구조는 각 프레임을 쌓아서 한 번에 하나의 프레임을 보여주며, 0번 프레임, 1번 프레임의 순으로 마지막 프레임까지 실행됩니다.



다층 시퀀스 구조는 마지막 프레임까지 실행을 마친 후에만 데이터를 반환합니다. 블록 다이어그램의 공간을 절약하려면 다층 시퀀스 구조를 사용하십시오.

플랫 시퀀스 구조의 프레임 사이에서 데이터를 전달할 때와 달리, 다층 시퀀스 구조에서는 한 프레임에서 다른 프레임으로 **데이터를 전달하는데 시퀀스 로컬을 사용**해야 합니다.

다음과 같이 시퀀스 선택 식별자는 다층 시퀀스 구조의 맨 위쪽에 있으며, 현재 프레임 번호와 전체 프레임의 크기를 보여줍니다.



시퀀스 선택 식별자를 사용하여 **사용 가능한 프레임을 탐색**하고, **프레임의 순서를 조정**합니다. 다층 시퀀스 구조의 프레임 라벨은 케이스 구조의 케이스 선택자 라벨과 유사합니다. 프레임 라벨은 프레임 번호가 가운데 있고, 양 옆으로 증가와 감소 화살표가 있습니다.

프레임 라벨에는 값을 입력할 수 없습니다. 다층 시퀀스 구조에서 프레임을 더하고, 삭제하고, 순서를 재조정하면, LabVIEW는 자동으로 프레임 라벨의 번호를 조정합니다.

비활성화 구조: 코드 실행 막기

비활성화 구조는 정확히 하나가 컴파일되고 실행되는 여러 **서브다이어그램**을 포함합니다. 실행할 서브다이어그램은 컴파일 시에 결정됩니다. 활성화되어 있지 않은 서브다이어그램 내의 코드는 실행 시나 컴파일 시에 실행되지 않습니다. 비활성화 구조를 사용하여 블록다이어그램의 코드 섹션을 비활성화시킬 수 있습니다.

조건적 비활성화 구조

다음과 같이 조건적 비활성화 구조에는 하나 이상의 서브다이어그램이나 케이스가 있으며, LabVIEW는 서브다이어그램의 조건 설정에 따라 정확히 이 중 하나를 실행 기간 동안 사용합니다. 컴파일할 때, LabVIEW는 조건적 비활성화 구조의 비활성화 서브다이어그램의 어떤 코드도 포함시키지 않습니다.



어떤 특정 코드가 컴파일되고 실행될지의 조건을 정의하려면 조건적 비활성화 구조를 사용합니다. 예를 들어, VI의 하나의 섹션이 타겟 지정인 경우, 이 코드를 조건적 비활성화 구조에 놓고 설정하여 특정 타겟에서 실행되도록 할 수 있습니다. Windows, Mac, Unix 시스템 및 FPGA 타겟에서 조건적 비활성화 구조를 설정할 수 있습니다.

다이어그램 비활성화 또는 타입 특수화 구조로 **조건적 비활성화 구조를 변환**하는 경우, 조건적 비활성화 구조의 조건은 다이어그램 비활성화 또는 타입 특수화 구조로 전달되지 않습니다.

조건적 비활성화 구조 사용에 대한 예제는 labview\examples\Structures\Disable Structures 디렉토리에 있는 Conditional Disable Structure VI를 참조하십시오. 관련 예제 찾기

다이어그램 비활성화 구조

다음의 다이어그램 비활성화 구조에서 보이는 것처럼, LabVIEW는 컴파일을 할 때 비활성화 서브다이어그램에 어떤 코드도 포함시키지 않습니다.



실행 시 컴파일되지 않도록 블록다이어그램의 특정 코드를 비활성화하려면, 다이어그램 비활성화 구조를 사용하십시오. 예를 들어, 다이어그램 비활성화 구조를 **디버깅 도구**로 사용하여, 코드를 주석 처리하고, 코드를 대체한 후, 구조의 비활성화 서브다이어그램에서 코드를 삭제하지 않고 VI를 컴파일할 수 있습니다.



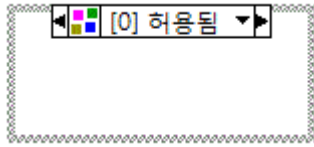
노트

특정 조건이 충족된 경우에만 코드 섹션을 활성화하려면, 조건적 비활성화 구조를 사용하십시오.

다이어그램 비활성화 구조 사용에 대한 예제는 labview\examples\Structures\Disable Structures 디렉토리의 [다이어그램 비활성화 구조] VI를 참조하십시오. 관련 예제 찾기

타입 특수화 구조

타입 특수화 구조는 아래와 같이 하나 또는 그 이상의 서브다이어그램을 가지며 서브다이어그램의 순서와 컴파일 결과에 따라서 LabVIEW는 하나를 정확하게 컴파일하고 실행합니다. 컴파일 및 실행 시, LabVIEW는 타입 특수화 구조의 비활성화 서브다이어그램의 코드는 전혀 포함시키지 않습니다.



타입 특수화 구조를 사용하여 특정 데이터 타입에 대해 [가변 vi.vim](#)의 코드 일부를 사용자 정의하거나 가변 vi가 사용 가능한 데이터 타입만 받도록 합니다. 예를 들어, 타입 특수화 구조를 사용하여 가변 vi가 입력 데이터가 정수일 때와 정수 배열일 때 각각 다르게 동작하도록 설정할 수 있습니다. 이렇게 하면 가변 vi는 두 데이터 타입에 대해 모두 예상대로 동작하게 됩니다.

타입 특수화 구조를 사용하는 예제는 `labview\examples\Malleable VIs\Type Specialization Structure\Malleable VIs - Type Specialization Structure.lvproj`를 참조하십시오.

비활성화 구조의 사용자 정의 객체

LabVIEW가 다이어그램 비활성화 구조의 비활성화 서브다이어그램이나 조건적 비활성화 구조의 비활성화 서브다이어그램, 또는 타입 특수화 구조에서 SubVI와 타입 정의 같은 사용자 정의 객체들이 있는 VI를 로드하는 경우, LabVIEW는 이 객체들을 메모리에 로드하지 않습니다. 그러나, VI의 블록다이어그램을 디스플레이할 때 LabVIEW가 객체를 찾지 못하는 경우, 물음표 아이콘이 찾을 수 없는 객체에 나타납니다. LabVIEW가 VI를 컴파일하고 실행할 때 코드를 포함하지 않기 때문에 VI는 깨지지 않습니다. 또한, subVI를 포함하는 서브다이어그램을 활성화하면, LabVIEW는 이와 같이 찾을 수 없는 객체를 자동으로 검색하지 않습니다.

찾을 수 없는 subVI의 위치를 다이어그램 비활성화 구조에서 LabVIEW로 하여금 강제로 찾게 하려면, 찾을 수 없는 subVI를 포함하는 서브다이어그램을 활성화하고 subVI를 호출하는 VI를 저장합니다. 다음에 subVI를 호출하는 VI를 열면, LabVIEW가 subVI를 검색합니다. 그러면 subVI를 포함하는 서브다이어그램을 다시 비활성화할 수 있습니다.



노트 LabVIEW는 비활성화 서브다이어그램의 코드 구문을 확인합니다. 그러나 비활성화 서브다이어그램내에 깨진 코드가 있어도 VI는 컴파일되고 실행됩니다.



노트 케이스 구조와 같이, 비활성화 구조는 터널을 제공합니다. 하지만, 기본적으로 모든 케이스에서 비활성화 구조의 출력 터널을 연결할 필요는 없습니다. 와이어로 연결되지 않은 모든 터널은 해당 터널 데이터 타입에 대한 [기본값](#)을 사용합니다. [터널을 설정](#)하여 연결되지 않은 케이스에 자동으로 입력 및 출력 터널을 연결할 수 있습니다.

빌드된 어플리케이션에서 비활성화 구조의 동작

비활성화 구조는 코드의 어떤 부분이 빌드된 어플리케이션에 포함될지를 결정합니다. 어떤 VI가 비활성화 다이어그램에서만 호출되는 경우에는 빌드에서 제외됩니다. 어플리케이션이 빌드된 이후에는 LabVIEW는 조건적 비활성화 구조의 조건 또는 타입 특수화 구조의 컴파일 결과를 다시 평가하지 않습니다.

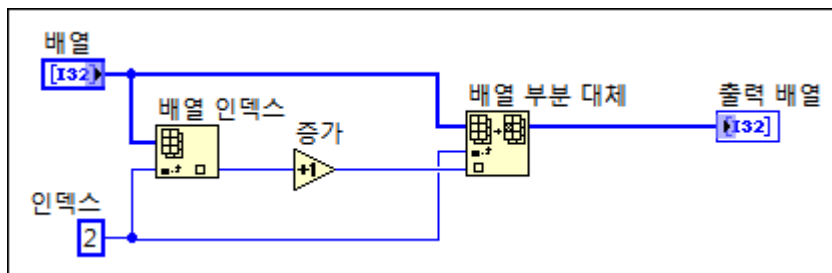
In Place 원소 구조: 메모리 효율성 높이기

In Place 원소 구조를 사용하여 LabVIEW 컴파일러가 기존의 LabVIEW 프로그래밍 방법보다 더욱 효율적으로 일반적인 작업을 수행하도록 컨트롤하고 메모리와 VI 효율성을 높입니다. 많은 LabVIEW 작업의 경우, LabVIEW가 메모리에 데이터를 복사하고 저장해야 합니다. 따라서 실행 속도가 떨어지고 메모리 사용이 늘어납니다.

In Place 구조는 데이터에 작업을 수행하기 위해 경계 노드 또는 In Place 구조의 경계에 붙어있는 노드를 사용합니다. 경계 노드를 이동하거나 삭제하면 LabVIEW는 자동으로 구조의 반대쪽의 해당 노드를 이동하거나 삭제합니다.

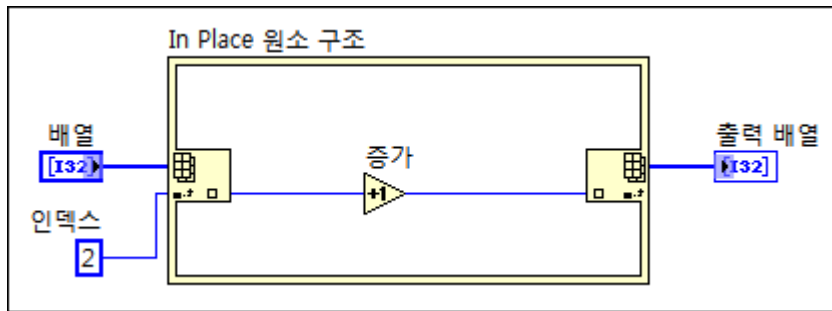
In Place 원소 구조는 기존의 LabVIEW 프로그래밍 방법보다 더욱 효율적으로 일반적인 LabVIEW 작업을 수행할 수 있습니다.

다음 블록다이어그램은 In Place 원소 구조를 사용하여 VI의 실행과 메모리 효율성을 개선하는 상황을 보여줍니다.



이전 블록다이어그램에서, LabVIEW는 32비트 부호없는 정수의 배열을 인덱스하고, 배열의 세번째 원소를 증가시키고, 해당 원소를 배열의 같은 위치에서 대체합니다. 같은 배열에서 새 값을 대체하려면 [배열 부분 대체] 함수는 배열과 배열 인덱스 값을 함수의 배열과 인덱스 입력에 연결하도록 요구합니다. VI를 실행하면 LabVIEW는 배열 원소의 새 복사본을 만들고 해당 복사본을 메모리에 저장합니다.

In Place 원소 구조를 사용하여 다음 블록다이어그램과 같이 배열 원소의 추가적인 복사본 생성을 방지합니다.



이전 블록다이어그램에서, 첫번째 블록다이어그램 예제와 유사하게 In Place 구조의 배열 인덱스 / 원소 대체 경계 노드가 32비트 부호없는 정수의 배열을 인덱스하고, 배열의 세번째 원소를 증가시키고, 해당 원소를 배열의 같은 위치에서 대체합니다. 그러나 이 인스턴스에서, LabVIEW는 메모리에 원소의 추가적인 복사본을 생성하거나 저장할 필요가 없습니다.

In Place 원소 구조는 배열의 각 인덱스된 배열 원소를 구조 오른쪽의 노드와 대체하도록 요구합니다. 이 구조가 배열 원소를 대체하도록 요구하고 LabVIEW 컴파일러가 어떤 원소를 대체할지 알고 있기 때문에 LabVIEW는 메모리에 배열 원소의 추가적인 복사본을 만들거나 저장할 필요가 없습니다.

LabVIEW 어플리케이션의 타이밍 컨트롤하기 (Real-Time, Windows)

Timed 구조를 사용하여 사용자 코드의 타이밍을 모니터하고 컨트롤합니다. Timed 구조는 리얼타임 어플리케이션과 고급 Windows 어플리케이션에서 최상으로 작동됩니다. 다른 어플리케이션의 경우, Timed 구조 대신 다음 객체를 사용할 수 있습니다. 이 객체를 사용하면 Time 구조에 비해 사용자 코드의 타이밍에 대한 컨트롤 능력이 약간 떨어지지만 필요한 설정은 보다 적습니다.

- While 루프
- [기다림(ms)] 함수
- [다음 ms 배수까지 기다림] 함수
- **LabVIEW의 기타 루프 및 구조**

사용할 Timed 구조의 종류 결정하기

어플리케이션에 Timed 구조가 적합하다고 판단되는 경우, 사용할 Timed 구조를 결정하는데 다음 정보가 도움이 될 수 있습니다.

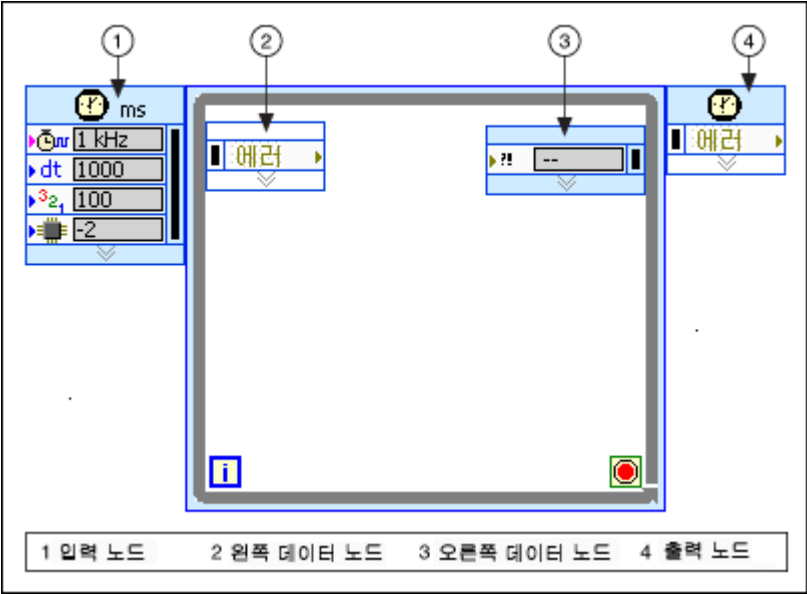
Timed 구조	구조의 용도	실행
Timed 루프	Timed 루프가 유용한 대상: - 특정 타이밍 특성에 따라 반복하려는 코드 - 반복이 필요하거나 정지 조건이 있는 코드	정지 조건이 충족될 때까지 코드를 반복해서 실행합니다.
프레임이 있는 Timed 루프	프레임이 있는 Timed 루프가 유용한 대상: - 시퀀스로 실행하려는 코드 - 큰 어플리케이션—각 개별 프레임에 대한 타이밍 정보를 받기 때문에, 복잡한 코드의 타이밍 프로퍼티를 보다 쉽게 모니터링할 수 있습니다.	정지 조건이 충족될 때까지 코드를 반복해서 시퀀스로 실행합니다. 프레임이 있는 Timed 루프는 시퀀스 구조가 임베이드된 Timed 루프처럼 동작합니다.
Timed 시퀀스	Timed 시퀀스 구조가 유용한 대상: 정확한 타이밍, 실행 피드백,ダイナ믹하게 변하는 타이밍 특성 또는 여러 레벨의 실행 우선순위를 통해 한 번만 실행하려는 코드	코드를 시퀀스로 한 번만 실행합니다. Timed 루프와 달리, Timed 시퀀스 구조는 반복되지 않습니다.

Timed 구조의 특성(Real-Time, Windows)

모든 Timed 구조에는 다음의 공통된 특성이 있습니다:

- 블록다이어그램의 코드 섹션(서브다이어그램이라고도 부름)을 둘러싸는 크기 조정 가능한 경계
- 설정 데이터를 제공하고 에러와 타이밍 정보를 반환하는 입력 및 출력 노드
- 구조 서브다이어그램 안팎으로 데이터를 주고받는 구조 경계의 터널

다음 이미지는 블록다이어그램의 Timed 루프를 나타냅니다.



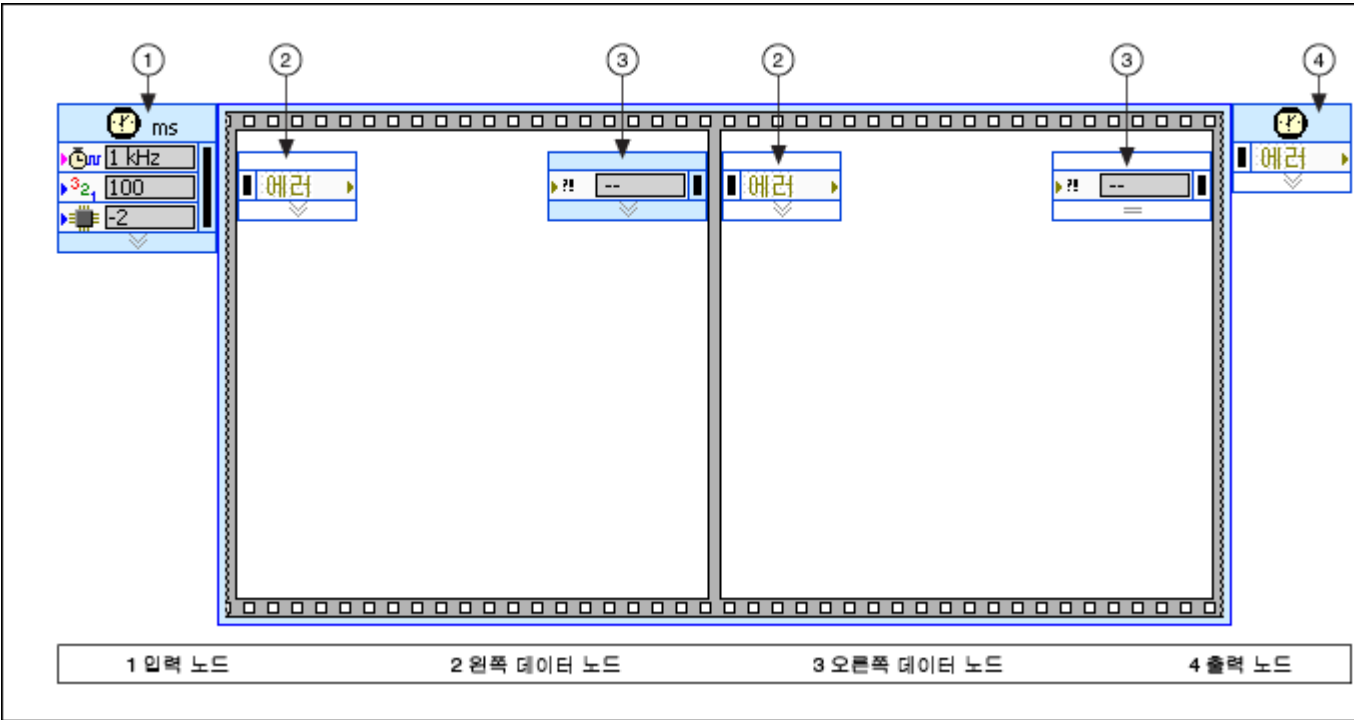
다음에 보이는 것처럼, 반복 터미널은 완료된 반복 횟수를 반환하는 출력 터미널입니다.



다음과 같이 조건 터미널은 입력 터미널입니다. 조건 터미널로 불리언 값을 전달하여 Timed 구조의 실행을 정지할 수 있습니다.



다음 이미지는 블록다이어그램의 Timed 시퀀스 구조를 나타냅니다.



이미지에 나타난 것처럼, Timed 루프와 Timed 시퀀스 구조는 유사한 노드를 포함합니다. 다음 정보는 Timed 구조의 다양한 노드에 대한 설명입니다.



노트

노드를 사용하여 **Timed 구조를 설정**할 수 있습니다. 또한 다양한 설정 대화 상자로 Timed 구조를 설정할 수 있습니다. 하지만, 설정 대화 상자를 사용하면 Timed 구조의 설정 옵션에 대해 아주 정밀한 컨트롤이 불가능합니다. 고급 어플리케이션에는 설정 노드를 사용할 수 있으며 이는 추가적인 설정 옵션을 제공합니다.

1. 데이터를 입력 노드에 연결하여 Timed 구조를 설정합니다. 또한 입력 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 Timed 루프 설정 대화 상자를 디스플레이합니다.
2. 왼쪽 데이터 노드는 설정 옵션 값을 반환하고 현재 또는 이전 루프 반복이나 프레임에 대한 타이밍 정보를 제공합니다.
3. 오른쪽 데이터 노드를 사용하여 다음 루프 반복이나 프레임을 설정합니다. 오른쪽 데이터 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 다음 프레임 타이밍 설정 대화 상자를 엽니다. 또한 오른쪽 데이터 노드에 데이터를 연결하여 **다이나믹하게 다음 루프 반복 또는 프레임을 설정**할 수 있습니다.
4. 출력 노드는 입력 노드의 에러 입력에서 오는 에러 정보, 실행 중 구조에 의해 생성되는 에러 정보, Timed 루프 프레임 내에서 실행되는 서브다이아그램에서 오는 에러 정보를 반환합니다. 또한 출력 노드는 마지막 루프 반복이나 프레임의 타이밍 및 상태 정보를 반환합니다.



노트 기본적으로 Timed 루프의 노드는 사용 가능한 입력 및 출력 터미널을 모두 디스플레이하지는 않습니다. 노드의 크기를 조정하거나 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴를 사용하여 **숨겨진 노드 터미널을 디스플레이**할 수 있습니다.



Timed 구조 홈

Timed 구조의 절대 시간 및 상대 시간 이해하기

지정된 **타이밍 소스**는 Timed 구조 실행을 제어합니다. 그러나 모든 Timed 구조 설정 옵션이 타이밍 소스의 절대값을 참조하는 것은 아닙니다. 프레임을 가지는 Timed 루프와 프레임이 있는 Timed 시퀀스 구조에서 일부 설정 옵션은 이전 프레임 또는 현재 프레임의 타이밍과 관련이 있습니다. 프레임 실행의 시작이나 끝을 참조하는 옵션은 상대 시간을 사용하는 반면 타이밍 소스를 참조하는 옵션은 절대 시간을 참조합니다.

Timed 구조에서 오프셋은 절대 시간이 제어하며 예상되는 시작, 예상되는 종료, 실제 시작, 실제 종료와 같은 출력은 프레임과 반복 타이밍의 절대 시간 단위로 계산됩니다. 시작 시간, 데드라인, 각 프레임의 타임아웃은 상대 시간이 제어합니다. 시작 시간과 타임아웃은 이전 프레임 실행 시작에 상대적이며 데드라인은 현재 프레임 시작에 상대적입니다.

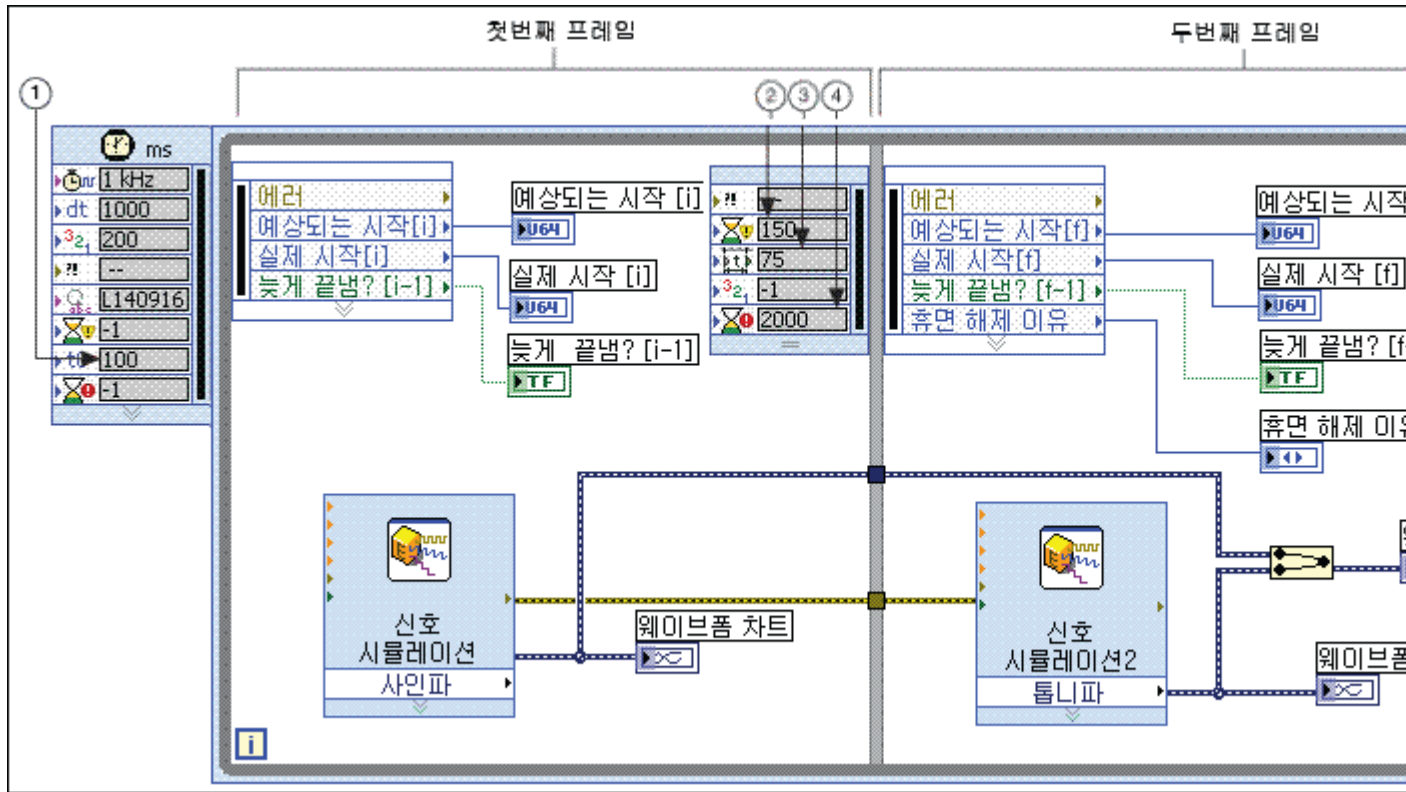
다음 블록다이아그램에서, 1 kHz 클럭이 제어하고 프레임이 있는 Timed 루프는 주기가 1000 ms이며 오프셋은 100 ms입니다. 오프셋은 절대 시간이 제어하므로, 루프는 타이밍

소스 클럭이 100 ms, 1100 ms, 2100 ms, 3100 ms 등에 도달할 때 반복되어서 루프 실행이 완료될 때까지 지속됩니다.



노트

늦어진 Timed 루프는 **Timed 루프의 모드**에 따라 절대 또는 상대 시간을 사용합니다.



1	오프셋	절대 시간—루프 타이밍 소스를 참조
2	데드라인	상대 시간—두번째 프레임의 시작을 참조
3	시작	상대 시간—첫번째 프레임의 시작을 참조
4	타임아웃	상대 시간—마지막 프레임의 시작을 참조

Timed 루프의 두번째 프레임은 시작 시간이 75 ms입니다. 시작 시간이 이전 프레임의 실행 시작에 상대적이기 때문에, 프레임 실행은 첫번째 프레임이 실행된 시간부터 75 ms가 지난 후에 시작됩니다. 타이밍 소스가 관리하는 절대 시간에 따르면 두번째 프레임의 시작 시간은 첫번째 반복에서 175 ms이며, 이는 첫번째 프레임이 타이밍 소스 클럭에서 100 ms에 시작된 후 두번째 프레임이 75 ms가 지나야 실행되기 때문입니다.

또한 프레임의 타임아웃 입력은 2000 ms입니다. 타임아웃은 이전 프레임이 실행을 시작한 후 또 다른 프레임의 시작을 기다리는 최대 시간을 설정합니다. 첫번째 프레임의 실행이 시작된 후 2000 ms 내에 두번째 프레임이 실행을 시작하지 않을 경우, LabVIEW는 시간에 관계 없이 남은 반복을 실행하고 왼쪽 데이터 노드의 휴면 해제 이유에 타임아웃을 반환합니다. 타임아웃인 2000 ms가 경과하기 전에 두번째 프레임이 실행을 시작하는 경우, 데드라인 150 ms는 프레임의 실행 소요 시간에 제한을 둡니다. 두번째 프레임이 실행을

시작한 순간부터 프레임이 늦게 끝냄 인디케이터에 참값을 출력하기 전까지 150 ms의 시간이 소요될 수 있습니다.

Timed 구조 흐름

Timed 루프 및 프레임이 있는 Timed 루프 설정하기 (Real-Time, Windows)

Timed 루프의 실행 방법을 설정하려면, Timed 루프 설정 또는 프레임이 있는 Timed 루프 설정 대화 상자를 사용합니다. 이 대화 상자를 디스플레이하려면, 입력 노드를 더블 클릭하거나 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택합니다. 이 대화 상자를 사용하여 **타이밍 소스** 및 **주기를 지정**하고 **고급 옵션을 설정**할 수 있습니다.

또한 데이터를 입력 노드에 연결하여 Timed 루프의 옵션을 설정할 수 있습니다. 기본 설정된 경우, Timed 루프의 입력 노드는 사용 가능한 입력 터미널을 모두 디스플레이하지는 않습니다. 입력 노드의 크기를 조정하거나 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴를 사용하여 **숨겨진 터미널을 디스플레이**할 수 있습니다.

Timed 루프 사용 요령

다음 리스트는 Timed 루프 사용에 대한 추가적인 정보를 제공합니다.

- While 루프와는 달리 Timed 루프의 조건 터미널은 연결할 필요가 없습니다.
- 데이터를 오른쪽 데이터 노드에 연결하여 **다이나믹하게 다음 루프 반복의 옵션을 설정**할 수 있습니다.
- Timed 루프에 **시프트 레지스터를 추가**하여 현재 반복에서 다음 반복으로 데이터를 전달할 수 있습니다.
- 예상과 다른 타이밍 동작을 피하려면, 시간 결정적인 우선순위의 실행이 설정된 VI와 Timed 루프를 동일한 블록 다이어그램에 추가하지 않습니다.

프레임이 있는 Timed 루프 사용 요령

다음 리스트는 프레임이 있는 Timed 루프 사용에 대한 추가적인 정보를 제공합니다.

- 데이터를 **프레임이 있는 Timed 루프 설정**의 입력 노드에 연결하거나, 입력 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 프레임이 있는 Timed 루프 설정 대화 상자를 디스플레이하고 옵션 값을 입력할 수 있습니다.
- 구조의 경계에서 마우스 오른쪽 버튼을 클릭한 후 프레임을 **추가**, **삭제**, **병합**할 수 있습니다.
- 각 프레임이 한 번씩만 실행되도록 하려는 경우, Timed 루프 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 Timed 시퀀스로 대체를 선택합니다.

Timed 루프를 사용하는 예제는 labview\examples\Structures\Timed Loop\Timed Loop.lvproj를 참조하십시오.

다음 프레임 설정하기

현재 프레임의 오른쪽 데이터 노드를 더블 클릭하거나 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 다음 프레임 타이밍 설정 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 다음 프레임에 대한 시작 시간, **우선 순위**, **데드라인**, **타임아웃**을 설정할 수 있습니다.

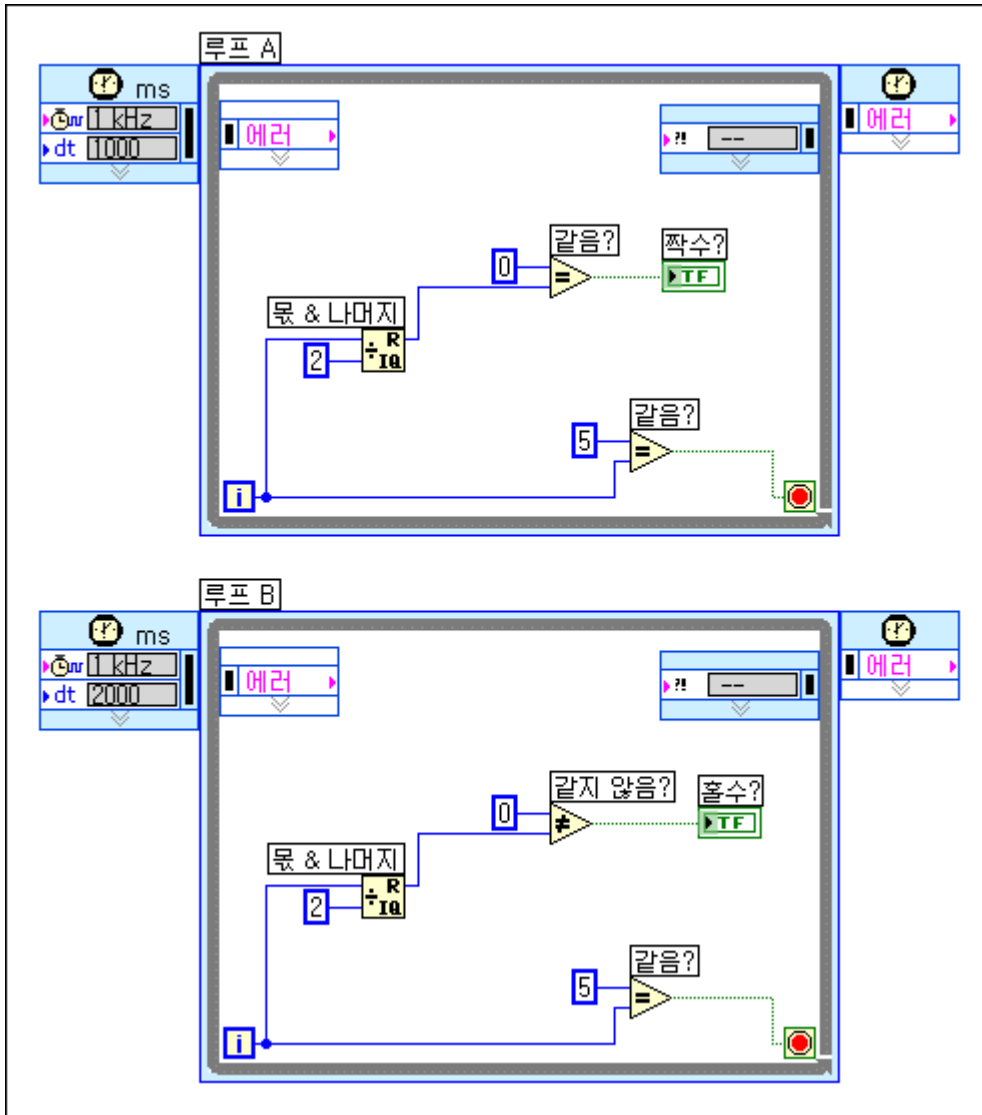
시작 시간은 다음 프레임이 실행을 시작하는 시간을 지정합니다. 이전 프레임의 시작에 상대적인 시작 시간 값을 프레임 타이밍 소스의 절대 단위로 지정해야 합니다. 시작 텍스트 박스에 값을 입력하여 시작 시간을 지정합니다.

또한 프레임의 오른쪽 데이터 노드의 입력 터미널을 사용하여 Timed 루프의 다음 반복에 대한 **옵션을 다이내믹하게 설정**하거나 **다음 프레임의 옵션을 다이내믹하게 설정**할 수 있습니다. 기본 설정된 경우, Timed 루프의 오른쪽 데이터 노드는 사용 가능한 입력 터미널을 모두 디스플레이하지는 않습니다. 오른쪽 데이터 노드의 크기를 조정하거나 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴를 사용하여 **숨겨진 터미널을 디스플레이**할 수 있습니다.

Timed 루프의 주기 설정하기

주기는 Timed 루프의 반복 사이의 시간의 길이로 구조 타이밍 소스의 절대 단위로 지정됩니다.

다음 블록 다이어그램의 Timed 루프는 기본 1 kHz 타이밍 소스를 사용합니다. 루프 A의 주기(dt)는 1,000 ms이고 루프 B의 주기는 2,000 ms입니다. 이는 루프 A는 1초마다 실행되고 루프 B는 2초마다 실행된다는 뜻입니다. 두 Timed 루프 모두 여섯 번 반복한 후 실행을 정지합니다. 루프 A는 6초 후 실행을 멈추고, 루프 B는 12초 후 실행을 멈춥니다.



Timed 루프의 서브다이어그램 또는 프레임을 가진 Timed 루프의 서브다이어그램은 각 반복에서 지정된 주기 전에 실행이 완료되어야 합니다. 서브다이어그램이 지정된 주기 전에 실행이 완료되지 않는 경우, Timed 루프는 이어지는 반복의 첫번째 프레임에서 출력 노드의 늦게 끝냄? [i-1] 출력 또는 왼쪽 데이터 노드의 늦게 끝냄? [i-1] 출력에서 참을 반환합니다.

Timed 루프 또는 프레임을 가진 Timed 루프의 서브다이어그램이 루프의 주기보다 먼저 실행이 완료되는 경우, Timed 루프는 휴면하게 되고 다른 코드나 구조가 블록다이어그램에서 실행되도록 허용합니다.



노트

Timed 루프의 주기를 0으로 설정하지 마십시오. 주기가 0인 경우 정의되지 않은 어플리케이션 동작을 초래할 수 있기 때문에 National Instruments는 주기가 0인 Timed 루프가 있는 어플리케이션은 지원하지 않습니다. Timed 루프의 우선 순위 실행 기능이 있으면서 최대한 빨리 실행되는 루프가 필요한 경우, 대신 Timed 시퀀스 구조 안에서 While 루프를 사용하십시오.

Timed 루프의 모드 설정하기

때때로 Timed 루프의 반복이 지정하는 시간보다 늦게 실행될 수도 있습니다. Timed 루프의 모드에 따라 루프가 늦은 실행을 처리하는 방법이 결정됩니다. Timed 루프 설정 또는 프레임이 있는 Timed 루프 설정 대화 상자의 늦은 반복에 대한 동작 섹션의 옵션을 사용하여 Timed 루프가 반복의 늦은 실행을 처리하는데 사용하는 모드를 지정합니다. 또한, 입력 노드의 모드 입력을 사용하여 Timed 루프의 모드를 지정할 수 있습니다.

다음 방법으로 Timed 루프의 늦은 실행을 처리할 수 있습니다.

- LabVIEW는 원래 수립된 스케줄대로 실행을 정렬할 수 있습니다.
- LabVIEW는 현재 시간에서 시작되는 새 스케줄을 정의할 수 있습니다.
- Timed 루프는 놓친 반복을 처리할 수 있습니다.
- Timed 루프는 놓친 반복을 뛰어넘을 수 있습니다.

예를 들어, 주기가 100 ms이고 오프셋이 30 ms인 Timed 루프를 설정하는 경우, 첫번째 루프 반복은 첫 타이밍 소스가 실행을 시작한지 30 ms 후에 실행되며 130 ms, 230 ms, 330 ms 등 100 ms의 배수로 실행되리라 예상합니다. 그러나 Timed 루프의 첫번째 실행은 240 ms가 지난 후 발생할 수 있습니다. 지정된 스케줄에서 다른 Timed 루프 또는 하드웨어 디바이스를 이미 실행하고 있을 수도 있으므로, 늦은 Timed 루프를 이미 실행 중인 글로벌 스케줄에 따라 정렬합니다. 이것은 Timed 루프가 최대한 빠르게 지정된 스케줄에 정렬할 수 있음을 의미합니다. 이 경우, 다음 Timed 루프 반복은 330 ms에서 실행되고 100 ms의 배수인 430 ms, 530 ms, 등에서 계속 실행됩니다. Timed 루프를 다른 Timed 루프나 다른 하드웨어 디바이스와 함께 정렬하는 것이 중요하지 않은 경우, Timed 루프는 즉시 실행되고 현재 시간을 실제 오프셋으로 사용할 수 있습니다. 이 경우, 다음 루프 반복은 240 ms, 340 ms, 440 ms와 같이 실행됩니다.

Timed 루프가 늦게 실행되는 경우, 다른 Timed 루프 또는 하드웨어 디바이스가 생성한 데이터를 잃어버릴 수 있습니다. 예를 들어, Timed 루프가 두 반복을 놓치고 현재 주기의 데이터 중 일부를 놓칠 경우, 버퍼는 놓친 반복의 데이터를 보유할 수 있습니다. Timed 루프가 지정된 스케줄에 따라 정렬하기 전에 놓친 데이터를 처리하도록 설정할 수도 있습니다. 그러나 놓친 반복을 처리하는 Timed 루프는 지터를 발생시킵니다. 지터는 루프 사이클 시간과 사용자가 지정한 시간 사이의 차이입니다. 놓친 데이터를 처리하지 않으려는 경우, Timed 루프는 루프 반복이 놓친 버퍼 안의 오래된 데이터를 무시하고, 다음 주기와 다음 반복에서 사용 가능한 데이터와 같은 최신 데이터만을 처리합니다.

설정 대화 상자를 사용하여 모드 설정하기

기본적으로, Timed 루프는 놓친 반복에서 생성된 모든 데이터를 버리고 원래 스케줄을 유지합니다. 이 모드 셋팅을 디스플레이하면, 입력 노드의 모드 입력 아이콘이 버림의 경우 D와 함께, 유지의 경우 M과 함께 나타납니다. Timed 루프 설정 또는 프레임이 있는 Timed 루프 설정 대화 상자의 놓친 주기 버림 확인란에서 확인 표시를 제거하여 놓치거나 늦은 루프 반복이 생성한 데이터를 처리합니다. 이 모드 셋팅을 디스플레이하면, 입력 노드의 모드 입력 아이콘은 프로세스를 나타내는 P와 함께 나타납니다. Timed 루프 설정 또는 프레임이 있는 Timed 루프 설정 대화 상자에 있는 원래 위상 유지 확인란에서 확인 표시를 제거하여 Timed 루프가 첫번째 반복에 기초한 새 일정에 맞춰 실행되도록 Timed 루프를 설정합니다.

입력 노드를 사용하여 모드 설정하기

입력 노드의 모드 입력을 사용하여 Timed 루프의 모드를 설정합니다. 모드 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **생성>>상수** 또는 **생성>>컨트롤**을 선택하여 열거형 상수나 컨트롤을 생성합니다. 이 상수나 컨트롤을 사용하여 모드를 선택할 수 있습니다.

입력 노드의 모드 입력을 사용하여 프로그램적으로 Timed 루프의 모드를 변경하거나 Timed 루프 설정 대화 상자의 현재 모드 설정을 유지합니다. 열거형 상수나 컨트롤에서 변경 없음을 선택하여 현재 모드 설정을 유지하거나, 옵션을 선택하여 Timed 루프의 늦거나 놓친 반복을 처리합니다.

Timed 루프에 대한 모드를 설정하는 예에 대해서는 labview\examples\Structures\Timed Loop에 있는 Timed Loop Mode VI를 참조하십시오.

Timed 구조 홈

Timed 시퀀스 구조 설정하기(Real-Time, Windows)

Timed 시퀀스 구조의 초기 실행 옵션을 설정하고 시퀀스의 첫번째 서브다이아그램 또는 프레임의 옵션을 설정하려면, Timed 시퀀스 설정 대화 상자를 사용합니다. 입력 노드를 더블 클릭하거나 입력 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 Timed 시퀀스 설정 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 **타이밍 소스를 지정**하고, **우선순위를 설정**하고, Timed 시퀀스 구조의 **고급 옵션을 설정**할 수 있습니다.

또한 값을 입력 노드의 입력 터미널에 연결하여 Timed 시퀀스 구조의 옵션을 설정할 수도 있습니다. 기본설정된 Timed 시퀀스 구조의 입력 노드는 사용 가능한 입력 터미널을 모두 디스플레이하지는 않습니다. 입력 노드의 크기를 조정하거나 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴를 사용하여 **숨겨진 터미널을 디스플레이**할 수 있습니다.

Timed 시퀀스 구조 사용 요령

다음 리스트는 Timed 시퀀스 구조 사용에 대한 추가적인 정보를 제공합니다.

- Timed 시퀀스 구조의 경계에서 마우스 오른쪽 버튼을 클릭한 후 프레임을 **추가**, **삭제**, **병합**할 수 있습니다. 플랫폼 시퀀스 구조와는 달리, Timed 시퀀스 구조의 경우 구조가 실행되기 전에 모든 프레임이 입력을 전부 가지고 있어야 합니다.
- Timed 시퀀스 구조의 마지막 프레임의 오른쪽 데이터 노드는 다른 프레임 또는 반복을 설정할 필요가 없으므로 설정 옵션을 포함하지 않습니다.
- Timed 시퀀스 구조는 시간 결정적인 우선순위로 실행되도록 설정되지 않은 모든 VI보다 먼저 블록다이아그램의 데이터 흐름 순서대로 실행됩니다. **Timed 시퀀스 구조의 우선순위를 변경**하면 블록다이아그램에 있는 Timed 구조 사이의 실행 순서를 변경할 수 있습니다.
- 예상과 다른 타이밍 동작을 피하려면, 시간 결정적인 우선순위의 실행이 설정된 VI와 Timed 시퀀스 구조를 동일한 블록다이아그램에 추가하지 않습니다.

- Timed 시퀀스 구조의 각 프레임을 한 번 이상 실행하려는 경우, Timed 시퀀스 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 Timed 루프로 대체를 선택합니다.

다음 프레임 설정하기

현재 프레임의 오른쪽 데이터 노드를 더블 클릭하거나 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 다음 프레임 타이밍 설정 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 다음 프레임의 시작 시간, **우선순위**, **데드라인**, **타임아웃**을 설정할 수 있습니다.

시작 시간은 다음 프레임이 실행을 시작하는 시간을 지정합니다. 현재 프레임의 시작에 상대적인 시작 시간 값을 프레임 타이밍 소스의 단위로 지정합니다. 다음 프레임 타이밍 설정 대화 상자의 시작 텍스트 박스에 값을 입력하여 시작 시간을 지정합니다. 다음 프레임이 지정된 시작 시간 전에 시작하지 않았을 경우 다음 프레임은 늦게 끝냄? 출력에 참을 반환합니다.

또한, 프레임의 오른쪽 데이터 노드의 입력 터미널을 사용하여 **Timed 루프의 다음 프레임 임 또는 다음 반복을 다이내믹하게 설정**할 수 있습니다. 기본설정된 Timed 시퀀스 구조 프레임의 오른쪽 데이터 노드는 사용 가능한 출력 터미널을 모두 디스플레이하지는 않습니다. 오른쪽 데이터 노드의 크기를 조정하거나 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴를 사용하여 **숨겨진 터미널을 디스플레이**할 수 있습니다.

Timed 구조 홈

Timed 구조의 셋팅 설정하기 (Real-Time, Windows)

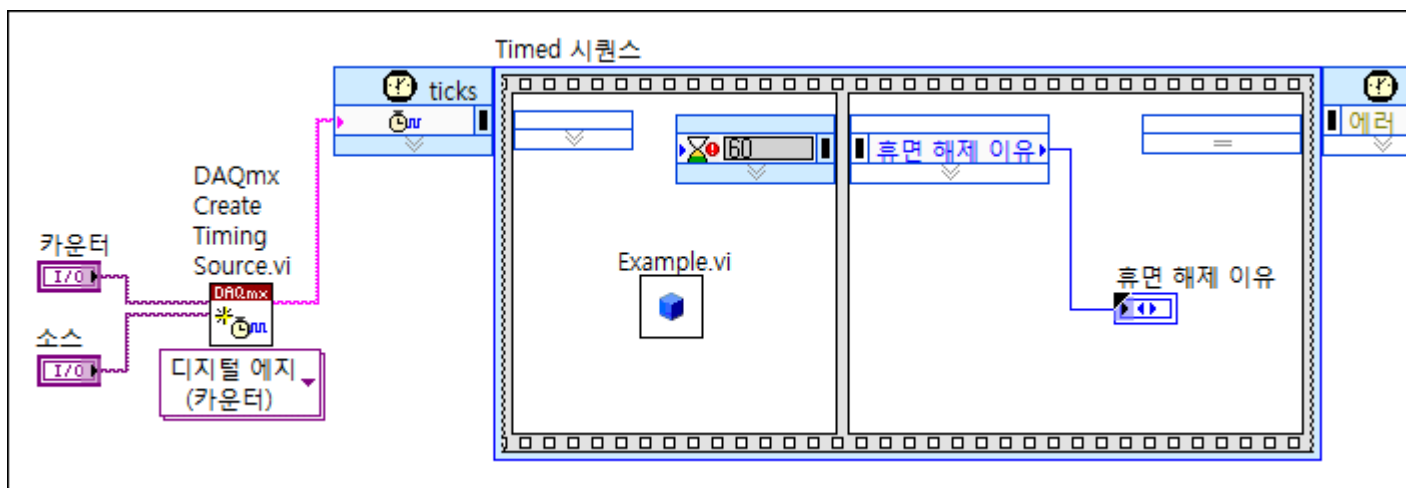
Timed 구조의 설정 대화 상자를 사용하여 Timed 구조의 데드라인, 타임아웃, 보조 타이밍 소스와 같은 고급 셋팅을 설정할 수 있습니다. 입력 노드를 더블 클릭하거나 입력 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 노드 설정을 선택하여 Timed 루프 설정, Timed 시퀀스 설정, 또는 프레임이 있는 Timed 루프 설정 대화 상자를 디스플레이합니다.

데드라인 설정하기

데드라인은 서브다이어그램 또는 프레임의 실행 시간으로 프레임의 시작에 상대적입니다. 데드라인을 사용하여 서브다이어그램 실행의 시간 제한 참조를 설정합니다. 프레임이 데드라인까지 완료되지 않을 경우, 다음 프레임의 왼쪽 데이터 노드는 늦게 끝냄? 출력에 참을 반환하고 계속 실행됩니다. 데드라인 값을 프레임 타이밍 소스와 같은 단위로 지정합니다.

다음 예제에서 Timed 시퀀스 구조의 첫번째 프레임은 데드라인 50으로 설정되었습니다. 데드라인은 서브다이어그램이 1 kHz 클럭의 50 Ticks, 또는 50 ms 이전에 완료되어야 함을 지정합니다. 그러나 프레임에 60 ms 기다림이 필요한 VI가 있으므로 프레임의 코드가 완료하려면 60 ms가 걸립니다. 프레임 실행이 지정된 데드라인을 충족시키지 못했기 때문에 두번째 프레임의 늦게 끝냄? 출력은 참을 반환합니다.

사용자는 디지털 에지 카운터나 태스크 소스로부터의 신호를 포함한 **여러 종류의 NI-DAQmx 타이밍 소스**를 이용하여 Timed 구조를 컨트롤하고 타임아웃을 지정합니다. 다음 예에서 Timed 시퀀스 구조의 두번째 프레임은 **[DAQmx 타이밍 소스 생성]** VI가 외부 디바이스로부터 펄스를 받을 때 실행됩니다. 외부 디바이스는 이상적인 조건 아래서는 50 ms 마다 펄스를 보내지만, 펄스 타이밍은 변할 수 있습니다. 두번째 프레임의 타임아웃 값 60 ms는 반복이 시작된 후 이 프레임이 시작할 때까지 60 ms 동안 대기할 수 있음을 나타냅니다. 두번째 프레임이 60 ms 전에 시작되지 않는 경우, 구조는 나머지 반복을 시간에 관계없이 계속 실행하며 두번째 프레임은 왼쪽 데이터 노드의 휴면 해제 이유 출력에 타임아웃을 반환합니다.



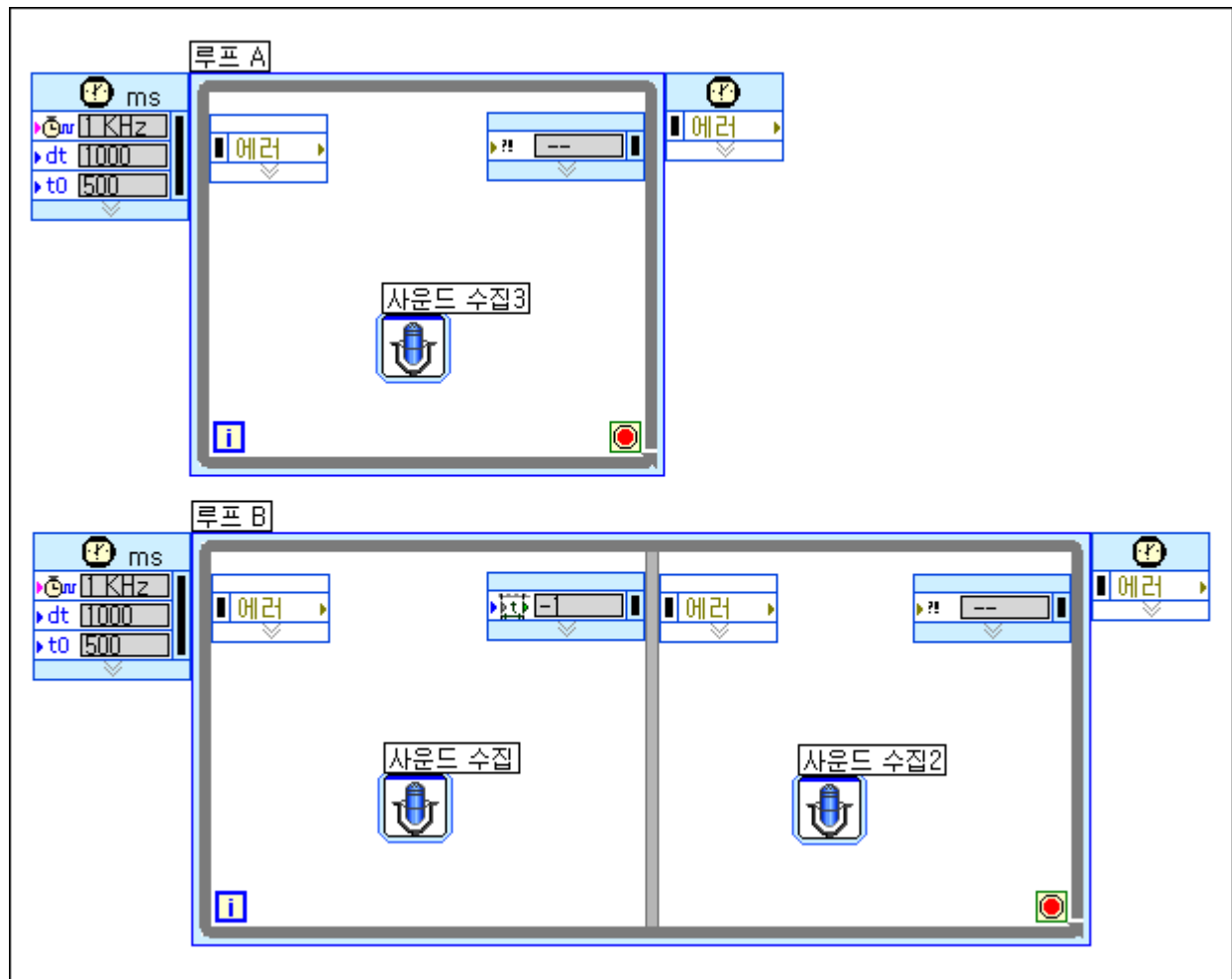
© National Instruments Corporation 391

Timed 구조 첫번째 프레임의 기본 타임아웃 값은 -1이며, 이는 서브다이어그램 또는 프레임의 시작을 무한히 기다림을 나타냅니다. 다른 프레임의 기본 타임아웃 값은 0이며, 이는 이전 프레임의 타임아웃 값이 변경되지 않음을 나타냅니다.

오프셋 설정하기

오프셋은 구조가 서브다이어그램 또는 프레임의 실행 시작을 기다리는 시간의 길이이며, Timed 구조의 시작에 상대적입니다. 오프셋은 구조 타이밍 소스의 절대 단위로 정의됩니다.

두 구조에 같은 타이밍 소스의 오프셋과 셋팅을 사용하여 서로 다른 Timed 구조의 위상을 정렬할 수 있습니다. 다음 블록다이어그램의 Timed 루프는 같은 1 kHz 타이밍 소스를 사용하고 500의 오프셋 값 t_0 을 가지므로 Timed 루프는 타이밍 소스가 반복의 실행을 시작하기 위해 트리거된 후 500 ms 동안 기다립니다.



노트

두 Timed 구조의 위상을 정렬해도 두 구조가 반드시 같은 시간에 시작하지는 않습니다. [Timed 구조 시작 동기화] VI를 사용하여 블록다이어그램에서 **Timed 구조의 시작을 동기화**합니다.

Timed 루프의 마지막 프레임에서 오른쪽 데이터 노드를 사용하여 **루프의 다음 반복에 대한 오프셋을 ديناميكية로 변경**할 수 있습니다. 그러나 다음 반복에 대한 오프셋을 ديناميكية로 변경할 때, 값을 오른쪽 데이터 노드의 모드 입력에 연결하여 반드시 모드를 지정해야 합니다.



노트 오른쪽 데이터 노드를 사용하여 오프셋을 변경한 경우 모드 입력에 변경 없음을 연결할 수 없습니다. 모드 값을 반드시 선택해야 합니다.

Timed 루프에 대한 오프셋을 설정하는 예에 대해서는 labview\examples\Structures\Timed Loop 디렉토리에 있는 Timed Loop Offset VI를 참조하십시오.

프로세서 할당하기

기본으로 LabVIEW는 자동으로 사용할 수 있는 프로세서를 할당하여 Timed 구조의 실행을 처리합니다. 그러나 보다 많은 실행 컨트롤을 위해 여러 프로세서에서 수동으로 Timed 구조를 로드할 수 있습니다.

Timed 루프 설정, 프레임이 있는 Timed 루프 설정, Timed 시퀀스 설정, 다음 프레임 타이밍 설정 또는 다음 반복 설정 대화 상자의 프로세서 지정 섹션의 모드 풀다운 메뉴에서 수동을 선택하면 Timed 구조를 처리하는 프로세서를 수동으로 설정할 수 있습니다. 또한 0에서 255 사이의 정수를 프로세서 필드에 입력해서 실행을 처리하는데 사용할 프로세서를 지정해야 하는데, 여기서 0은 시스템에서 처음으로 사용 가능한 프로세서를 나타냅니다. 지정된 범위 밖의 숫자를 입력하려고 하면 LabVIEW가 강제적으로 0부터 255 사이의 숫자로 만듭니다. 사용 가능한 프로세서의 개수를 넘는 숫자를 입력하는 경우, 런타임 에러가 발생하고 Timed 구조는 실행되지 않습니다.

모드 풀다운 메뉴에서 자동으로 선택하는 경우 LabVIEW가 자동으로 프로세서를 구조에 할당하고 프로세서 값은 자동으로 -2로 설정됩니다. 프레임 또는 이후 반복의 모드 풀다운 메뉴에서 변경 없음을 선택하는 경우, LabVIEW는 이전 프레임 또는 반복을 처리했던 프로세서와 같은 프로세서를 사용하고 프로세서 값은 자동으로 -1로 설정됩니다.

또한 값을 Timed 구조 또는 Timed 구조 프레임의 입력 노드의 프로세서 입력에 연결하여 프로세서 할당을 설정할 수 있습니다.

수동 프로세스 할당을 통한 Timed 구조의 성능 최적화에 대해 더 자세히 알려면 여러 CPU 시스템에서 RT 어플리케이션 최적화하기를 참조하십시오.

보조 타이밍 소스 설정하기

보조 타이밍 소스를 사용하여 Timed 구조의 서브다이아그램 또는 프레임을 구조 자체와 다른 클럭 소스로 설정할 수 있습니다. Timed 루프 설정, 프레임이 있는 Timed 루프 설정 또는 Timed 시퀀스 설정 대화 상자의 프레임 타이밍 소스 섹션을 사용하여 보조 타이밍 소스를 선택합니다.

Timed 구조는 다음 보조 타이밍 소스를 지원합니다:

- 시퀀스 또는 루프의 타이밍 소스를 사용—Timed 구조와 같은 타이밍 소스를 사용하여 프레임의 타이밍을 정합니다.

- 아래의 내장된 소스 사용—별도의 내장 타이밍 소스를 사용하여 프레임의 타이밍을 정합니다.
 - 1 kHz 클럭 <구조 시작 시 리셋>—1 kHz 클럭과 비슷한 타이밍 소스로 Timed 구조의 매 반복이 끝난 후 0으로 리셋합니다.
 - 1 MHz 클럭 <구조 시작 시 리셋>—1 MHz 클럭과 비슷한 타이밍 소스로 Timed 구조의 매 반복이 끝난 후 0으로 리셋합니다.

Timed 구조의 절대 시간 및 상대 시간

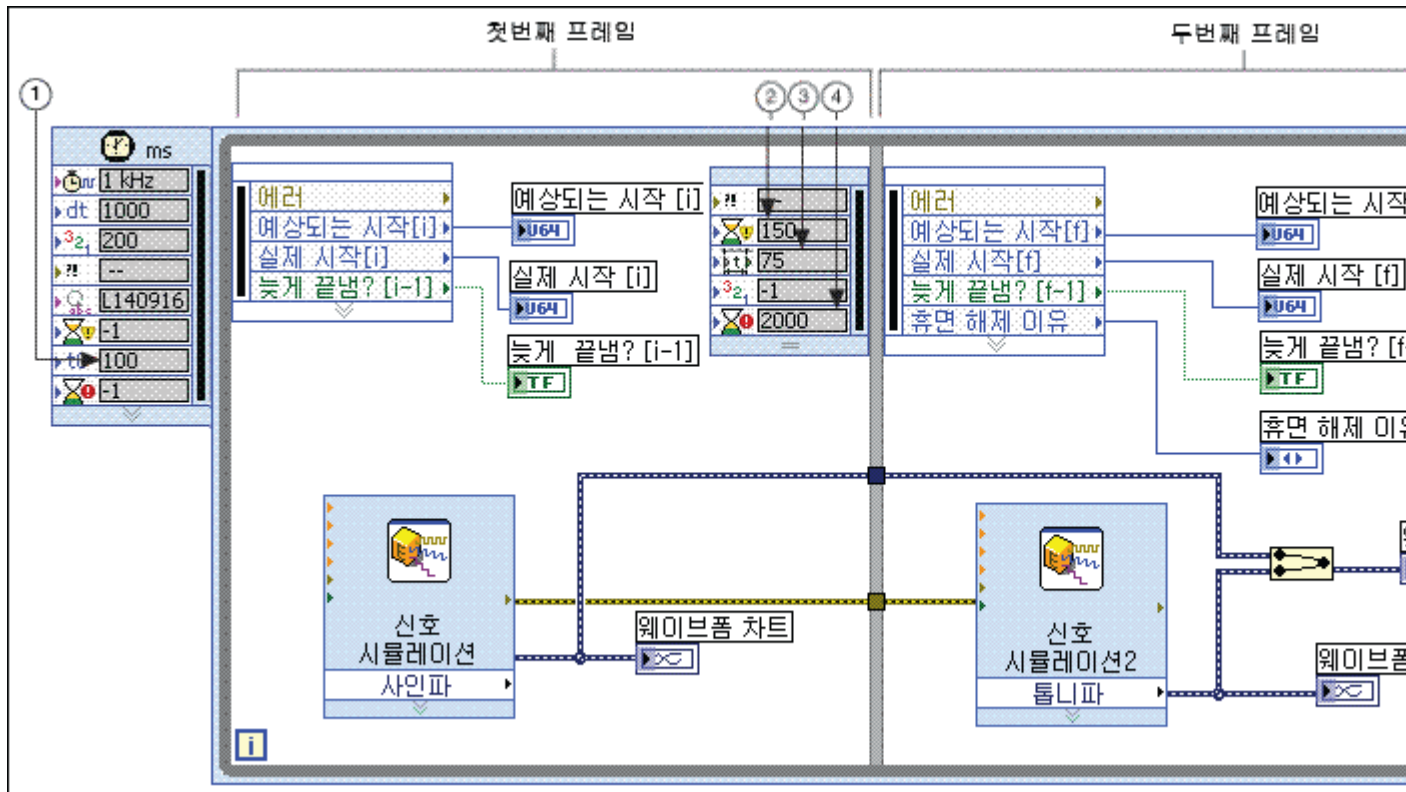
지정된 **타이밍 소스**는 Timed 구조 실행을 제어합니다. 그러나 모든 Timed 구조 설정 옵션이 타이밍 소스의 절대값을 참조하는 것은 아닙니다. 프레임을 가지는 Timed 루프와 프레임이 있는 Timed 시퀀스 구조에서 일부 설정 옵션은 이전 프레임 또는 현재 프레임의 타이밍과 관련이 있습니다. 프레임 실행의 시작이나 끝을 참조하는 옵션은 상대 시간을 사용하는 반면 타이밍 소스를 참조하는 옵션은 절대 시간을 참조합니다.

Timed 구조에서 오프셋은 절대 시간이 제어하며 예상되는 시작, 예상되는 종료, 실제 시작, 실제 종료와 같은 출력은 프레임과 반복 타이밍의 절대 시간 단위로 계산됩니다. 시작 시간, 데드라인, 각 프레임의 타임아웃은 상대 시간이 제어합니다. 시작 시간과 타임아웃은 이전 프레임 실행 시작에 상대적이며 데드라인은 현재 프레임 시작에 상대적입니다.

다음 블록 다이어그램에서, 1 kHz 클럭이 제어하고 프레임이 있는 Timed 루프는 주기가 1000 ms이며 오프셋은 100 ms입니다. 오프셋은 절대 시간이 제어하므로, 루프는 타이밍 소스 클럭이 100 ms, 1100 ms, 2100 ms, 3100 ms 등에 도달할 때 반복되어서 루프 실행이 완료될 때까지 지속됩니다.



노트: 늦어진 Timed 루프는 **Timed 루프의 모드**에 따라 절대 또는 상대 시간을 사용합니다.



1	오프셋	절대 시간—루프 타이밍 소스를 참조
2	데드라인	상대 시간—두번째 프레임의 시작을 참조
3	시작	상대 시간—첫번째 프레임의 시작을 참조
4	타임아웃	상대 시간—마지막 프레임의 시작을 참조

Timed 루프의 두번째 프레임은 시작 시간이 75 ms입니다. 시작 시간이 이전 프레임의 실행 시작에 상대적이기 때문에, 프레임 실행은 첫번째 프레임이 실행된 시간부터 75 ms가 지난 후에 시작됩니다. 타이밍 소스가 관리하는 절대 시간에 따르면 두번째 프레임의 시작 시간은 첫번째 반복에서 175 ms이며, 이는 첫번째 프레임이 타이밍 소스 클럭에서 100 ms에 시작된 후 두번째 프레임이 75 ms가 지나야 실행되기 때문입니다.

또한 프레임의 타임아웃 입력은 2000 ms입니다. 타임아웃은 이전 프레임이 실행을 시작한 후 또 다른 프레임의 시작을 기다리는 최대 시간을 설정합니다. 첫번째 프레임의 실행이 시작된 후 2000 ms 내에 두번째 프레임이 실행을 시작하지 않을 경우, LabVIEW는 시간에 관계 없이 남은 반복을 실행하고 왼쪽 데이터 노드의 휴면 해제 이유에 타임아웃을 반환합니다. 타임아웃인 2000 ms가 경과하기 전에 두번째 프레임이 실행을 시작하는 경우, 데드라인 150 ms는 프레임의 실행 소요 시간에 제한을 둡니다. 두번째 프레임이 실행을 시작한 순간부터 프레임이 늦게 끝냄 인디케이터에 참값을 출력하기 전까지 150 ms의 시간이 소요될 수 있습니다.

Timed 구조의 우선순위 설정하기 (Real-Time, Windows)

Timed 구조의 우선 순위는 블록 다이어그램의 Timed 구조들 중 어느 것이 먼저 실행될지 순서를 지정합니다. Timed 구조의 우선 순위 셋팅을 사용하여 같은 VI에서 서로를 실행시킬 수 있는 여러 작업을 가진 어플리케이션을 만듭니다. Timed 구조의 우선 순위가 높을수록 다른 Timed 구조보다 빨리 실행됩니다. 우선 순위 입력의 값은 1과 65,535 사이의 양의 정수여야 합니다.

블록 다이어그램의 각 Timed 구조는 단일 스레드를 포함하는 자체 실행 시스템을 생성하고, 그 시스템에서 실행되므로 병렬 작업이 일어날 수 없습니다. Timed 루프는 시간 결정적인 우선순위로 실행되도록 설정되지 않은 모든 VI보다 먼저 블록 다이어그램의 데이터 흐름 순서대로 실행됩니다.



노트

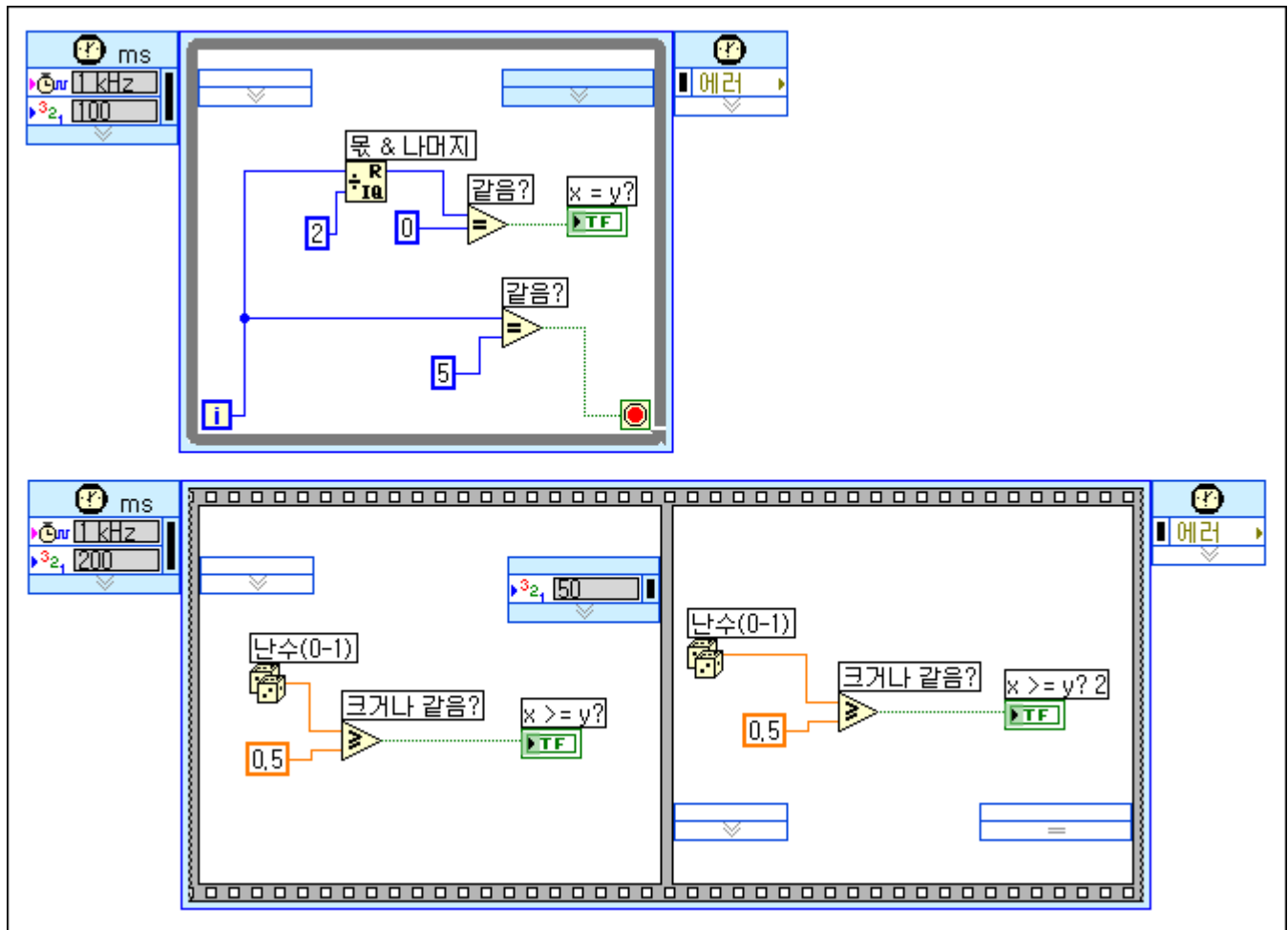
시간에 결정적인 우선순위로 실행되도록 설정된 VI와 Timed 루프를 같은 블록 다이어그램에 추가하면 예상치 않은 타이밍 동작이 발생할 수 있습니다.

프레임을 가지는 Timed 시퀀스 구조 또는 Timed 루프의 각 프레임에 우선순위 값을 설정할 수 있습니다. Timed 구조를 포함하는 VI를 실행할 때, LabVIEW는 블록 다이어그램에서 실행할 준비가 된 모든 프레임의 우선 순위를 확인하고 가장 높은 우선순위를 가진 프레임을 첫번째로 실행합니다.



노트(NI Linux Real-time) VI 안에서 32개 이상의 Timed 구조 우선 순위 레벨을 설정하지 않도록 합니다. 32개 이상의 Timed 구조 우선 순위 레벨을 설정하려고 시도하면 LabVIEW는 에러를 반환합니다.

다음 블록 다이어그램은 Timed 루프와 Timed 시퀀스 구조를 포함합니다. Timed 시퀀스 구조의 첫번째 프레임의 우선 순위 값(200)이 Timed 루프의 우선 순위 값(100)보다 높으므로 Timed 시퀀스 구조의 첫번째 프레임이 먼저 실행됩니다.



Timed 시퀀스 구조의 첫번째 프레임이 실행된 후, LabVIEW는 실행할 준비가 된 다른 구조 또는 프레임의 우선 순위를 비교합니다. Timed 루프의 우선 순위(100)는 Timed 시퀀스 구조의 두번째 프레임의 우선 순위(50)보다 높습니다. 이 예제에서, Timed 루프는 Timed 시퀀스 구조의 두번째 프레임이 실행되기 전에 완료됩니다.

입력 노드에서 마우스 오른쪽 버튼을 클릭하고 입력 노드 설정을 선택한 후 우선 순위 값을 변경하여 Timed 구조의 우선 순위를 설정할 수 있습니다. 또한 값을 마지막 프레임의 오른쪽 데이터 노드의 우선 순위 입력에 연결하여 Timed 루프의 **이어지는 반복의 우선 순위를 다이내믹하게 설정**할 수 있습니다. 값을 현재 프레임의 오른쪽 데이터 노드에 연결하여 Timed 구조의 **다음 프레임의 우선 순위를 다이내믹하게 설정**할 수 있습니다. 기본 설정된 프레임의 오른쪽 데이터 노드는 사용 가능한 출력 터미널을 모두 디스플레이하지는 않습니다. 오른쪽 데이터 노드의 크기를 조정하거나 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴를 사용하여 **숨겨진 터미널을 디스플레이**할 수 있습니다.



노트(NI Linux Real-Time) National Instruments는 Timed 구조 우선 순위 레벨을 대화식으로 변경하는 것을 권장합니다. Timed 구조 우선 순위 레벨을 다이내믹하게 변경하면 성능이 저하될 수 있습니다.

Timed 구조 홈

Timed 구조의 타이밍 소스 선택하기 (Real-Time, Windows)

타이밍 소스는 Timed 구조의 실행을 컨트롤합니다. 세가지 타입의 타이밍 소스 중에서 선택할 수 있습니다:

- **내부**—Timed 구조의 입력 노드에 대한 설정을 통해 사용자가 선택하는 내장된 타이밍 소스
- **소프트웨어 트리거**—[타이밍 소스 생성] VI의 소프트웨어 트리거 타이밍 소스 생성 인스턴스를 통해 사용자가 생성하는 타이밍 소스
- **외부**—[타이밍 소스 생성] VI와 [DAQmx - 데이터 수집] VI를 통해 사용자가 생성하는 타이밍 소스

내부 타이밍 소스

Timed 구조를 컨트롤을 위한 내부 타이밍 소스는 OS의 1 kHz 클럭과 지원하는 리얼타임(RT) 타겟의 1 MHz 클럭을 포함합니다. Timed 루프 설정, Timed 시퀀스 설정 또는 프레임이 있는 Timed 루프 설정 대화 상자의 루프 타이밍 소스 또는 시퀀스 타이밍 소스 섹션을 사용하여 내부 타이밍 소스를 선택합니다.

- 1 kHz 클럭—기본적으로, Timed 구조는 OS의 1 kHz 클럭을 타이밍 소스로 사용합니다. 1 kHz 클럭을 사용하여, 밀리초 분해능으로 Timed 구조의 스케줄을 설정할 수 있습니다. Timed 구조를 실행하는 모든 LabVIEW 플랫폼은 1 kHz 타이밍 소스를 지원합니다.
- 1 MHz 클럭—지원되는 타겟은 1 MHz 클럭 타이밍 소스를 사용하여 Timed 구조를 컨트롤할 수 있습니다. 1 MHz 클럭을 사용하여, 마이크로초 분해능으로 Timed 구조의 스케줄을 설정할 수 있습니다. 프로세서 또는 OS 제약 때문에 타겟이 마이크로초 분해능을 지원하지 않는 경우, 1 MHz 클럭을 사용할 수 없습니다.
- 1 kHz <절대 시간>—OS의 1 kHz 클럭을 선택합니다. 이 타이밍 소스를 선택하면 Timed 구조가 시작 및 종료 시간과 관련된 모든 입력과 출력 터미널에 타임스탬프를 사용합니다. 절대 타이밍 소스를 사용하여 구조를 실행할 특정 날짜와 시간을 지정할 수 있습니다.
- 1 MHz <절대 시간>—Intel Pentium III 또는 National NI PXI-817x 및 NI PXI-818x 디바이스 등의 최신 프로세서를 갖춘 RT 타겟에서 지원되는 1 MHz 클럭을 선택합니다. 이 타이밍 소스를 선택하면 Timed 구조가 시작 및 종료 시간과 관련된 모든 입력과 출력 터미널에 타임스탬프를 사용합니다. 절대 타이밍 소스를 사용하여 구조를 실행할 특정 날짜와 시간을 지정할 수 있습니다.
- 1 kHz 클럭 <구조 시작 시 리셋>—1 kHz 클럭과 비슷한 타이밍 소스로 Timed 구조가 시작할 때 0 Tick으로 리셋합니다.
- 1 MHz 클럭 <구조 시작 시 리셋>—1 MHz 클럭과 비슷한 타이밍 소스로 Timed 구조가 시작할 때 0 Tick으로 리셋합니다.
- 스캔 엔진에 동기화—Timed 구조를 **NI 스캔 엔진**에 동기화합니다. 이 타이밍 소스를 사용하여, 각각의 스캔을 마칠 때마다 Timed 구조가 실행됩니다. 각 반복 사이의 주기는 스캔 엔진 페이지에서 설정하는 스캔 주기(μ S) 셋팅에 대응합니다.

소프트웨어 트리거 타이밍 소스

소프트웨어 트리거 타이밍 소스를 생성하여 소프트웨어에 의해 정의된 이벤트를 기준으로 Timed 구조를 트리거할 수 있습니다. [타이밍 소스 생성] VI를 사용하여 소프트웨어 트리거 타이밍 소스를 생성합니다. [소프트웨어 트리거 타이밍 소스 발생] VI를 사용하여 소프트웨어 트리거 타이밍 소스에 의해 제어되는 Timed 루프를 프로그램적으로 트리거합니다.

소프트웨어 트리거 타이밍 소스를 사용하여 RT 호환 가능한 이벤트 핸들러로 이용하거나, 생산자-소비자 어플리케이션에서 새 데이터가 사용 가능할 때 이를 소비자의 Timed 루프에게 알려줄 수 있습니다. 또한 소프트웨어 트리거 타이밍 소스를 비연속적인 이벤트 시뮬레이션에서 사용할 수 있습니다. [소프트웨어 트리거 타이밍 소스 발생] VI의 Tick 개수 입력을 사용하여 비연속적인 각 이벤트 사이에서 경과하는 Tick의 개수를 지정합니다. 소프트웨어 트리거 타이밍 소스 발생 VI가 실행될 때, Timed 구조의 내부 Tick 카운터가 Tick 개수 만큼 진행됩니다. 내부 Tick 카운터가 하나 이상의 Timed 루프 주기를 뛰어 넘기면, Timed 루프는 해당 주기를 놓친 주기로 간주합니다. [소프트웨어 트리거 타이밍 소스 발생] VI를 한번 호출하여 여러 Timed 루프 반복을 트리거하려는 경우, Timed 루프 설정 대화 상자에서 놓친 주기 버림 확인란의 확인 표시를 제거하면 됩니다.

외부 타이밍 소스

NI-DAQmx 7.2 또는 이후 버전으로 Timed 구조를 컨트롤하기 위한 외부 타이밍 소스를 생성할 수 있습니다. [DAQmx 타이밍 소스 생성] VI를 사용하여 외부 타이밍 소스를 프로그램적으로 선택합니다. 또한, 주파수, 디지털 에지 카운터, 디지털 변화 감지기, 태스크 소스의 신호 등 몇몇 타입의 NI-DAQmx 타이밍 소스를 사용하여 Timed 구조를 컨트롤할 수 있습니다. DAQmx - Data Acquisition VI를 사용하여 다음 타입의 NI-DAQmx 타이밍 소스를 생성하여 Timed 구조를 컨트롤합니다.

- 주파수—일정한 주파수에서 Timed 구조를 실행시키는 타이밍 소스를 생성합니다.
- 디지털 에지 카운터—디지털 신호의 상승 또는 하강 에지에서 Timed 구조를 실행시키는 타이밍 소스를 생성합니다.
- 디지털 변화 감지기—디지털 라인 하나 이상의 상승 또는 하강 에지에서 Timed 구조를 실행시키는 타이밍 소스를 생성합니다.
- 태스크로부터의 신호—사용자가 지정하는 신호를 사용하여 언제 Timed 구조가 실행될지 결정하는 타이밍 소스를 생성합니다.

NI-DAQmx VI와 함수를 사용한 Timed 구조의 컨트롤에 대한 정보는 시작 >> 프로그램 >> National Instruments >> NI-DAQ >> NI-DAQmx 도움말을 선택하여 볼 수 있는 NI-DAQmx 도움말을 참조하십시오.

보조 타이밍 소스를 사용하여 Timed 구조의 프레임 실행을 컨트롤할 수 있습니다. 예를 들어, 1 kHz 클럭을 사용하여 Timed 루프의 반복을 컨트롤하고 1 MHz 클럭을 사용하여 매번 반복마다 프레임의 타이밍을 컨트롤할 수 있습니다.

Timed 구조 홈

다이나믹하게 Timed 구조의 입력 옵션 설정하기 (Real-Time, Windows)

프레임의 오른쪽 데이터 노드를 사용하여 Timed 구조의 다음 프레임의 입력 값을 설정하거나 Timed 루프의 다음 반복에 대한 입력 값을 설정합니다.

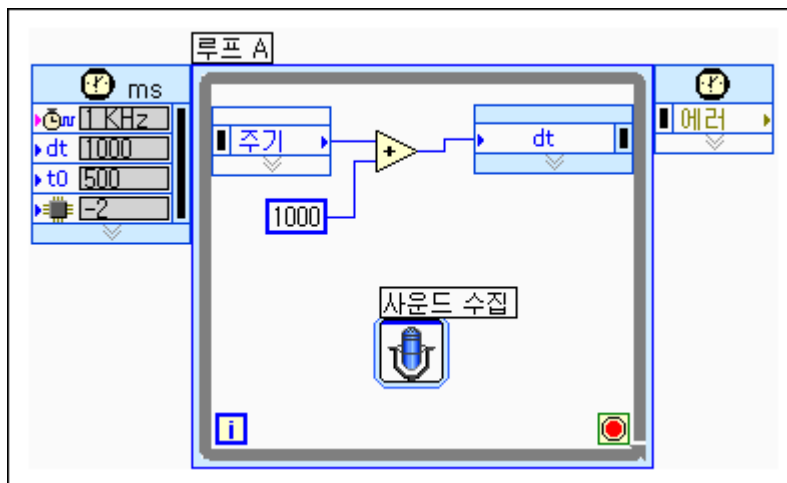


노트

(NI Linux Real-Time) National Instruments는 Timed 구조 우선 순위 레벨을 대화식으로 변경하는 것을 권장합니다. Timed 구조 우선 순위 레벨을 다이나믹하게 변경하면 성능이 저하될 수 있습니다.

Timed 루프의 다음 반복 설정하기

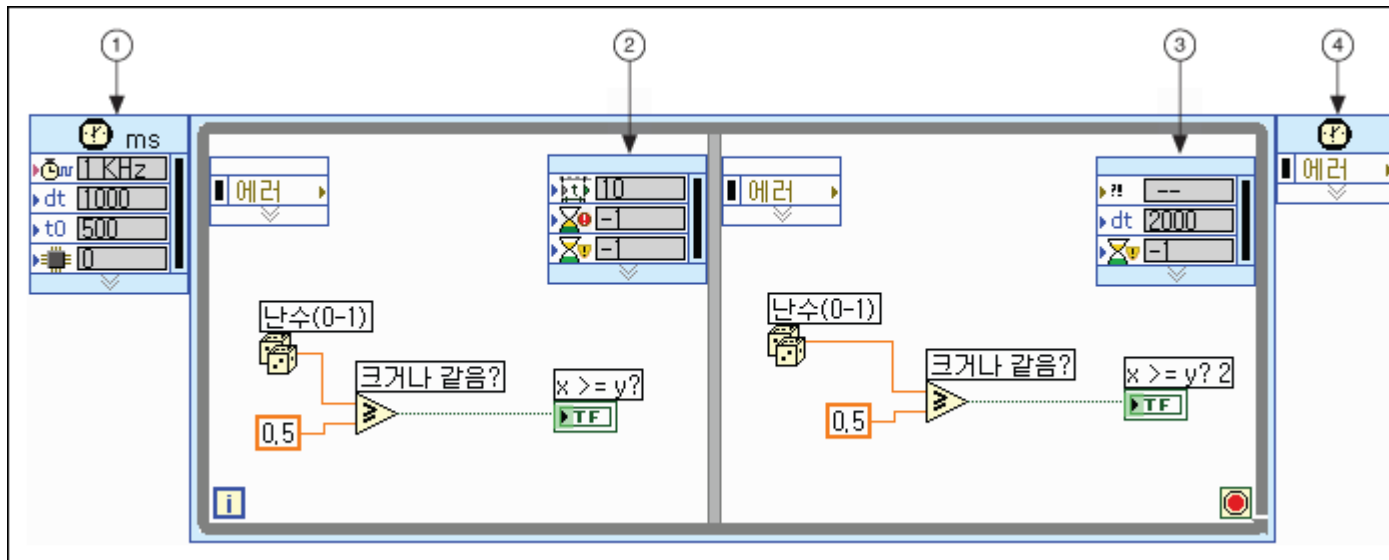
Timed 루프의 오른쪽 데이터 노드 또는 프레임이 있는 Timed 구조의 마지막 프레임의 오른쪽 데이터 노드를 사용하여 루프의 다음 반복에 대한 설정 옵션을 설정합니다. 다음 블록 다이어그램의 Timed 루프는 오른쪽 데이터 노드의 주기 입력을 사용하여 루프가 반복될 때마다 1 초(1,000 ms) 길게 실행되도록 설정합니다.



노트 값을 오른쪽 데이터 노드의 오프셋 입력에 연결하여 Timed 루프의 오프셋을 다이나믹하게 변경하는 경우, 오른쪽 데이터 노드의 모드 입력에도 반드시 모드를 지정해야 합니다.

Timed 구조의 다음 프레임 설정하기

또한 프레임의 오른쪽 데이터 노드를 사용하여 다이나믹하게 다음 프레임의 설정 옵션을 변경할 수 있습니다. 다음 블록 다이어그램의 Timed 루프에는 두 개의 프레임(A와 B)이 있습니다. 입력 노드 (1)은 Timed 루프의 초기 실행과 첫 번째 프레임의 옵션을 설정합니다. 첫 번째 프레임의 오른쪽 데이터 노드는 두 번째 프레임의 옵션을 설정합니다. 두 번째 프레임의 오른쪽 데이터 노드는 첫 번째 프레임과 다음 루프 반복의 옵션을 설정합니다.

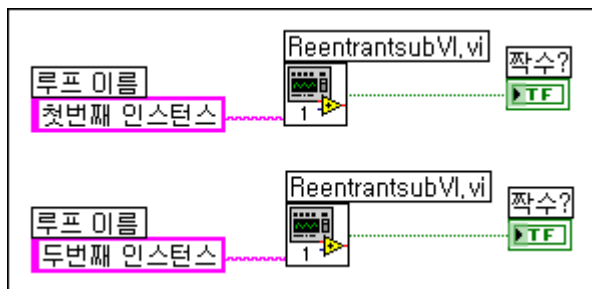


1	입력 노드
2	오른쪽 데이터 노드
3	오른쪽 데이터 노드
4	출력 노드

Timed 구조 이름 붙이기 (Real-Time, Windows)

기본적으로, LabVIEW는 자동적으로 사용자가 블록다이어그램에 놓은 각 Timed 구조를 이름으로 고유하게 식별하며, 이 이름은 Timed 루프 설정 또는 프레임이 있는 Timed 루프 설정 대화 상자의 구조 이름 텍스트 박스 또는 Timed 시퀀스 설정 대화 상자의 시퀀스 이름 텍스트 박스에 나타납니다. 이 텍스트 박스에 새 이름을 입력하여 Timed 구조의 이름을 변경합니다. Timed 구조의 고유한 이름을 Timed 구조와 VI 팔레트의 VI에 사용하여 프로그램적으로 구조를 정지하고 같은 시작 시간을 사용하는 Timed 구조 그룹을 동기화합니다.

재호출 VI가 Timed 루프를 포함하고 블록다이어그램에서 그 재호출 VI의 둘 이상의 인스턴스를 subVI로 사용할 때, 프로그램적으로 재호출 VI의 각 인스턴스의 Timed 구조 이름을 변경할 수 있습니다. 재호출 VI를 설정하려면 커넥터 팬의 입력 터미널을 블록다이어그램 Timed 루프 또는 Timed 시퀀스 구조의 이름 입력에 연결된 문자열 컨트롤에 연결하십시오. 재호출 VI의 둘 이상의 인스턴스가 subVI로 사용되는 블록다이어그램에서, 고유한 문자열 값을 재호출 subVI의 루프 이름 입력에 연결하여 재호출 subVI의 각 인스턴스 내의 각 Timed 구조를 고유하게 식별합니다. 예를 들어, 다음 블록다이어그램은 같은 재호출 VI의 두 인스턴스를 subVI로 포함합니다. SubVI의 인스턴스에 연결된 이름 문자열 상수는 블록다이어그램이 실행될 때마다 두 다른 이름을 재호출 VI의 블록다이어그램 상의 Timed 루프에 전달합니다.



Timed 구조 홈

Timed 구조의 시작 동기화하기 (Real-Time, Windows)

단일 컴퓨팅 디바이스에서 실행되거나 이더넷에 연결된 여러 분산 컴퓨팅 디바이스에서 실행되는 여러 Timed 구조의 시작을 동기화할 수 있습니다.

단일 계산 디바이스에서 실행되는 Timed 구조 동기화하기

I/O 변수를 사용하여 동기화하려는 Timed 구조 안의 I/O 데이터에 접근하려는 경우, Timed 구조를 NI 스캔 엔진에 동기화시킬 수 있습니다.

동기화하려는 Timed 구조 안의 I/O 작업을 수행하기 위해 I/O를 사용하는 것이 아닌 경우, [Timed 구조 시작 동기화] VI를 사용하여 단일 컴퓨팅 디바이스에서 실행되는 여러 Timed 구조의 시작을 동기화합니다. 예를 들어, 블록다이어그램에 두 개의 Timed 구조가 있는 경우 이 두 Timed 구조가 상대적으로 같은 스케줄에 따라 실행되도록 합니다. 첫 번째 Timed 구조를 먼저 실행하고 데이터를 생성하도록 설정할 수 있습니다. 그 후, 첫 번째 구조의 반복이 완료되면 이때 생성된 데이터를 두 번째 Timed 구조가 처리하도록 설정할 수 있습니다. Timed 구조의 시작을 동기화하여 두 구조가 실행에서 같은 시작 시간을 사용하도록 할 수 있습니다.

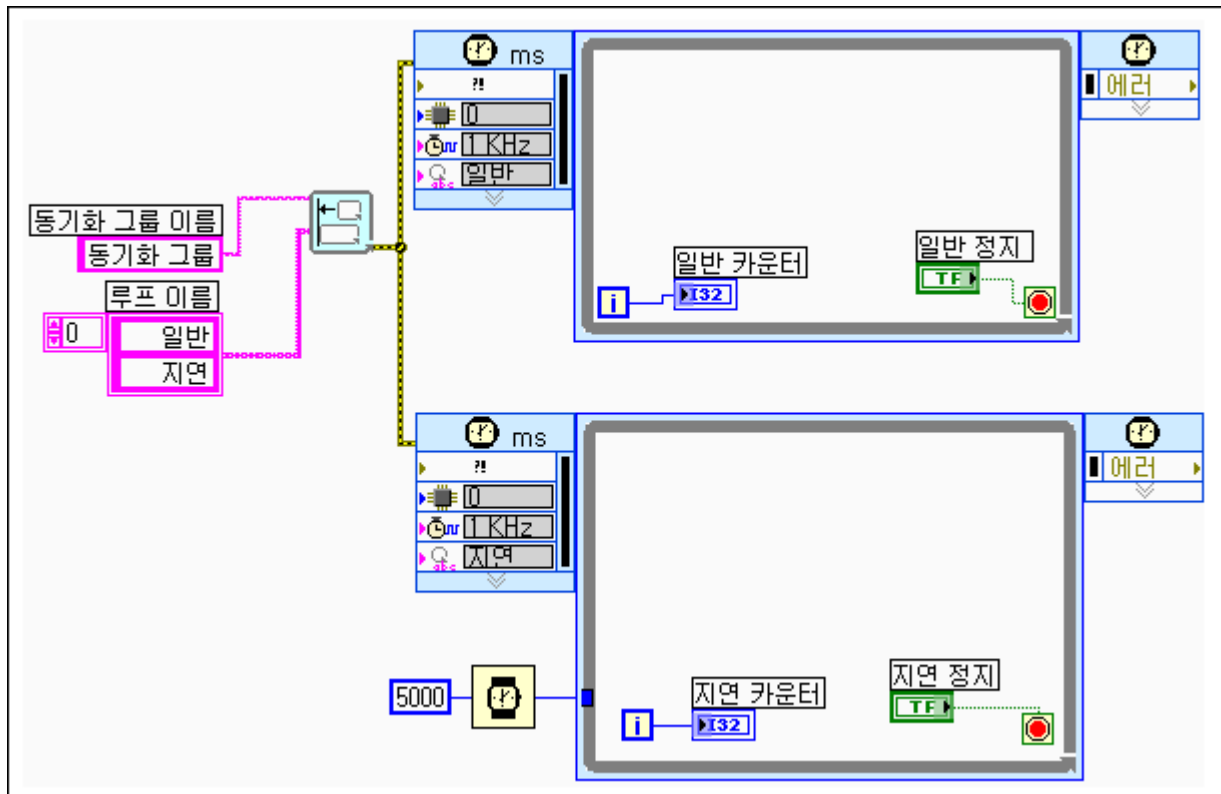


노트

National Instruments는 [Timed 구조 시작 동기화] VI를 오직 1 kHz 클럭 타이밍 소스 또는 1 MHz 클럭 타이밍 소스를 사용하도록 설정된 Timed 구조와 함께 사용하도록 권장합니다. National Instruments는 이 VI를 절대 시간 타이밍 소스와 함께 사용하거나, 구조 시작 타이밍 소스, 스캔 엔진에 동기화 타이밍 소스, 또는 기타 외부 타이밍 소스에서 리셋하는 것을 권장하지 않습니다.

동기화 그룹을 생성하여 동기화하려는 블록다이어그램의 구조를 지정할 수 있습니다. 그룹의 이름을 [Timed 구조 시작 동기화] VI의 동기화 그룹 이름 입력에 연결하고 Timed 구조 이름의 배열을 Timed 구조 이름 입력에 연결하여 동기화 그룹을 생성할 수 있습니다. 생성한 동기화 그룹은 VI가 실행을 완료할 때까지 활성화되어 있습니다.

다음 블록다이어그램의 [Timed 구조 시작 동기화] VI는 동기화 그룹을 생성하고, 2개의 Timed 루프인 정규와 지연을 동기화하여 동일한 시작 시간을 사용합니다.



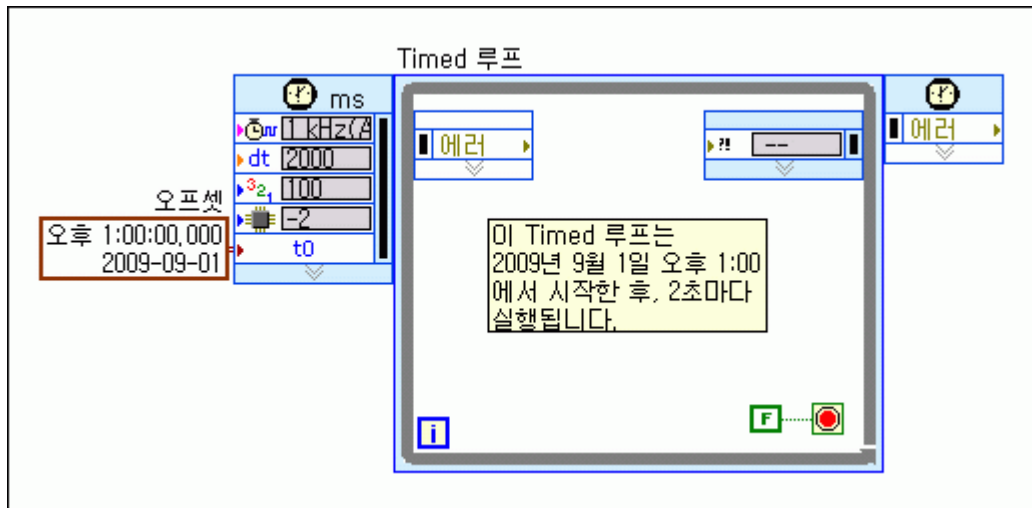
Timed 구조는 하나 이상의 동기화 그룹에 속할 수 없습니다. 이미 이전 그룹에 속해있는 Timed 구조를 새 동기화 그룹에 추가하려는 경우, LabVIEW는 이전 그룹에서 구조를 제거하고 새 그룹에 추가합니다. [Timed 구조 시작 동기화] VI의 대체 (참) 입력을 거짓으로 설정하여, Timed 구조가 이미 다른 동기화 그룹에 속한 경우에 그 구조가 동기화 그룹으로 이동하는 것을 막을 수 있습니다. 대체 (참) 입력을 거짓으로 설정한 경우에 이미 다른 동기화 그룹에 포함된 Timed 구조를 포함시키려고 시도하면, LabVIEW는 에러를 반환합니다.

여러 계산 디바이스에서 실행되는 Timed 구조 동기화하기

I/O 변수를 사용하는 경우, NI 스캔 엔진을 지원하는 각 디바이스 상에서 **NI 스캔 엔진에 Timed 구조를 동기화**할 수 있습니다. NI 스캔 엔진의 각 인스턴스는 사용자가 설정한 스캔 주기의 배수인 절대 시간에 맞춰서 실행을 시작합니다. 따라서, 각 타겟의 스캔 주기를 공통된 값 또는 공배수 값으로 설정함으로써, 여러 타겟에 있는 Timed 구조의 위상을 동기화할 수 있습니다.

I/O 변수를 사용하지 않는 경우, 절대 시간 타이밍 소스를 사용하여 이더넷에 연결된 1 개 이상의 분산 계산 디바이스에 있는 여러 Timed 구조의 시작을 동기화합니다. Timed 구조의 타이밍 소스를 1 kHz <절대 시간> 또는 1 MHz <절대 시간>으로 선택하는 경우, LabVIEW 타임스탬프 컨트롤이나 상수를 Timed 구조의 오프셋 입력에 와이어로 연결하여 구조를 시작할 정확한 날짜와 시간을 지정할 수 있습니다. 여러 Timed 구조에 똑같은 타임스탬프 값을 연결함으로써, 각 구조의 시작 시간을 동기화할 수 있습니다.

다음 블록 다이어그램의 Timed 루프는 1 kHz <절대 시간> 타이밍 소스를 사용하며 오프셋이 2009년9월1일 1:00 p.m.에 실행을 시작하도록 설정되었고, 주기는 매 2 초마다 실행하도록 설정되어 있습니다.

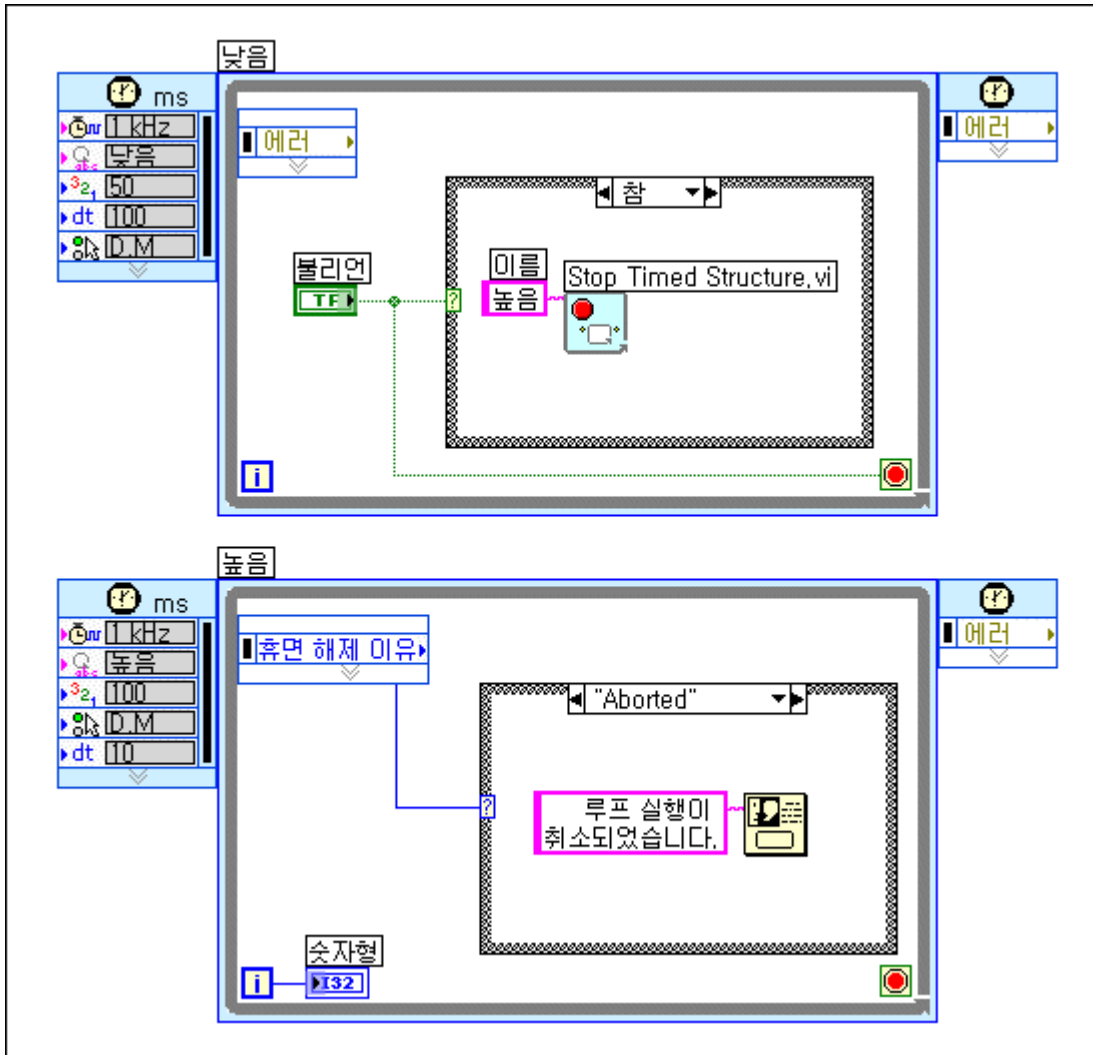


노트 NI 스캔 엔진과 절대 시간 타이밍 소수는 모두 Timed 구조가 실행되는 계산 디바이스의 시스템 클럭에 의존합니다. 따라서, Timed 구조의 동기화가 얼마나 정확하기 이루어졌는가는 시스템 클럭의 동기화 수준에 달려 있습니다. 각 디바이스의 시스템 클럭을 공통된 마스터 클럭에 맞춤으로써 클럭 동기화 수준을 향상시킬 수 있습니다. IEEE 1588 스펙에 기반한 NI 시간 동기화 소프트웨어에 대한 정보는 National Instruments 웹 사이트를 참조하십시오.

Timed 구조 홈

Timed 구조의 실행 강제 종료하기 (Real-Time, Windows)

[Timed 구조 정지] VI를 사용하여 프로그램적으로 Timed 구조의 실행을 강제 종료합니다. 문자열 상수나 컨트롤에 있는 구조 이름을 [Timed 구조 정지] VI의 이름 입력에 연결하여 강제 종료하려는 Timed 구조의 이름을 지정합니다. 예를 들어, 다음 블록 다이어그램에서 낮은 Timed 루프에는 [Timed 구조 정지] VI가 포함되어 있습니다. 높은 Timed 루프는 실행되며, 완료한 반복 수를 디스플레이합니다. 사용자가 프런트패널의 시간에 걸 정적인 루프 강제 종료 버튼을 클릭하는 경우, 왼쪽 데이터 노드의 휴면 해제 이유 출력은 강제 종료 값을 반환하고, 대화 상자가 나타나며, 사용자가 대화 상자에서 확인을 클릭하면 VI가 실행을 중지합니다.



Timed 구조 흐름

LabVIEW의 이벤트

이벤트 타입

LabVIEW에서 이벤트는 다음 소스로부터 생성됩니다:

- 사용자 인터페이스—사용자 이벤트에는 마우스 클릭, 키보드 입력 등이 있습니다.
- 외부 I/O—외부 I/O 이벤트에는 수집이 완료되거나 에러 조건이 발생했을 때 신호를 주는 하드웨어 타이머 또는 트리거가 있습니다.



노트

LabVIEW는 또한 프로그램식으로 생성할 수 있는 외부 I/O 이벤트인 ActiveX 및 .NET 이벤트를 지원합니다.

- 블록다이어그램—블록다이어그램 이벤트 또는 프로그램적으로 생성되는 이벤트에는 **사용자 이벤트**가 포함되며, 이는 사용자 정의 데이터로 이벤트를 생성하거나 블록다이어그램에서 발생한 다른 이벤트를 생성할 수 있게 합니다.

각 이벤트는 VI의 프런트패널에 있는 컨트롤, VI의 프런트패널 윈도우, VI의 구획 또는 LabVIEW 어플리케이션과 연계되어 있습니다.



노트 LabVIEW는 VI가 실행 중이거나 다른 실행 중인 VI가 해당 VI를 SubVI로 호출할 때에만 VI에 대한 이벤트를 생성합니다.

사용자 인터페이스 이벤트

사용자 인터페이스 이벤트를 사용하면, 사용자가 프런트패널에서 특정 작업을 수행할 때마다 LabVIEW에서 이벤트가 생성되는 어플리케이션을 설계할 수 있습니다. 프런트패널에 해당 이벤트가 발생할 때마다, 블록다이어그램은 해당 이벤트에 대해 작성된 코드에 따라 응답합니다. 루프에서 사용자 인터페이스를 폴링하여 상호작용하는 것과 달리, 사용자 인터페이스 이벤트는 프로그램의 CPU 요구사항을 줄이고, 블록다이어그램 코드를 단순화하며, 블록다이어그램이 작성된 모든 상호작용에 대해 응답할 수 있도록 합니다. LabVIEW는 사용자 인터페이스 이벤트를 두 개의 **이벤트 타입**으로 분류합니다.

프로그램적으로 생성된 이벤트

프로그램적으로 생성된 이벤트를 사용하면, 프런트패널에서의 사용자 작업이 없이도 블록다이어그램에서 이벤트를 생성할 수 있습니다. 프로그램적으로 생성된 이벤트는 데이터 흐름 의존성을 갖지 않는 프로그램의 다른 부분들과 통신합니다. 예를 들어 어플리케이션의 한 부분은 연속 루프를 실행하여 온도 센서를 모니터하고, 센서가 경고 레벨에 도달하면 사용자 이벤트를 생성합니다. 어플리케이션의 다른 부분에서는 이벤트 구조가 실행되어, 센서가 경고 레벨에 도달하여 생성된 사용자 이벤트와 사용자에게 경고 조건을 알리기 위해 이벤트 구조로 처리되어야 할 사용자 인터페이스 이벤트를 모두 처리합니다. 사용자 인터페이스 이벤트 및 프로그램적으로 생성된 이벤트를 하나의 이벤트 구조에서 처리하면, 이벤트를 사용한 큐된 상태 머신과 같은 고급 구조를 구현할 수 있습니다.



노트 일반적으로 LabVIEW는 사용자가 활성 프런트패널에서 직접 상호작용하는 것의 결과로만 사용자 인터페이스 이벤트를 생성합니다. 하지만 값(신호) 프로퍼티를 사용하여 프로그램적으로 값 변경 이벤트를 생성할 수 있습니다.

이벤트 드리븐 프로그래밍

LabVIEW에서는 데이터의 흐름이 블록다이어그램 상의 원소의 실행 순서를 결정합니다. 노드는 모든 필수 입력을 받은 후 실행되고, 이후 데이터를 생성하여 데이터 흐름 경로의 다음 노드로 전달합니다. 이벤트 구동 프로그래밍에서는 이벤트 또는 어떤 일의 발생에 대한 비동기적 알림을 제공하는 입력이 이러한 실행 방식에 영향을 끼칠 수 있습니다. 예를 들어 사용자가 프런트패널의 특정 키를 눌렀을 때 이벤트가 발생하기를 원하는 경우, 키 누름 이벤트가 발생할 때까지 대기하다가 해당 이벤트가 발생하면 처리한 후 다시 다음 이벤트를 기다리는 루프가 포함된 어플리케이션을 설계합니다. 루프가 해당 이벤트의 발생을 기다리는 동안에도 어플리케이션의 다른 부분은 계속해서 실행될 수 있습니다. 어플리케이션에서 각 이벤트가 처리되는 방법은 해당 이벤트에 대해 작성한 코

드에 따라 다릅니다. 프로그램의 일부 섹션은 처리하는 이벤트가 자주 발생하기 때문에 자주 실행되는 반면, 다른 섹션은 이벤트가 발생하지 않아서 전혀 실행되지 않을 수 있습니다.

자주 묻는 질문

LabVIEW에서 이벤트를 어떻게 사용할 수 있습니까?

어플리케이션이 이벤트에 응답하도록 어떻게 할 수 있습니까?

어플리케이션이 어떤 이벤트를 모니터하도록 어떻게 지정할 수 있습니까?

특정 이벤트만 어떻게 모니터 및 컨트롤할 수 있습니까?

사용자 정의된 데이터를 사용하는 이벤트를 어떻게 생성할 수 있습니까?

이벤트를 포함하도록 어플리케이션을 설계하는 가장 효과적인 방법은 무엇입니까?

어플리케이션이 실행 중일 때 이벤트 큐에 있는 이벤트를 어떻게 볼 수 있습니까?

어떤 타입의 사용자 인터페이스 이벤트를 사용할지 결정하기

LabVIEW는 **사용자 인터페이스 이벤트**를 알림과 필터라는 두 개의 다른 이벤트 타입으로 분류합니다. 다음 테이블은 이벤트의 타입과 각 타입을 사용해야 하는 경우를 설명합니다.

이벤트 타입	설명	예	용도
알림	알림 이벤트는 사용자 작업이 발생했다는 것을 알립니다.	사용자가 컨트롤의 값을 변경할 때 이벤트 구조가 실행되어 값 변경 이벤트를 처리하도록 설정할 수 있습니다.	오직 사용자가 특정 작업을 수행했는지만을 알려는 경우 알림 이벤트를 사용하십시오.

이벤트 타입	설명	예	용도
필터	필터 이벤트를 사용하면, LabVIEW가 해당 이벤트와 연계된 기본 동작을 수행하기 전에 사용자가 이벤트 데이터를 검증하거나 변경할 수 있습니다. 또한 변경 사항이 VI에 영향을 주는 것을 방지하려면 이벤트 전체를 무시할 수 있습니다.	사용자가 VI의 프론트패널을 닫는 것을 차단하기 위해, 이벤트 구조가 패널 닫기? 이벤트를 폐기하도록 설정할 수 있습니다.	사용자 동작 처리에 관여하려면 필터 이벤트를 사용하십시오.



노트
필터 이벤트는 알림 이벤트와 구별되도록 물음표로 끝나는 이름을 갖습니다. 이벤트 케이스가 해당 이벤트를 폐기하지 않는 경우, LabVIEW는 대부분의 필터 이벤트 이후 연계된 알림 이벤트를 생성합니다. 알림 이벤트는 필터 이벤트와 동일한 이름을 갖지만, 물음표 부호가 없습니다.

알림 이벤트 및 필터 이벤트용 데이터 흐름

알림 이벤트의 경우, LabVIEW는 해당 이벤트를 처리하도록 설정한 각 이벤트 구조에 해당 이벤트의 복사본을 병렬로 전송합니다.

필터 이벤트의 경우, LabVIEW는 해당 이벤트를 처리하도록 설정한 이벤트 구조에 이벤트를 순차적으로 보냅니다. LabVIEW가 각 이벤트 구조에 이벤트를 보내는 순서는 이벤트를 **등록**한 순서에 따릅니다. 각 이벤트 구조는 LabVIEW가 다음 이벤트 구조에 알리기 전에 반드시 필터 이벤트 케이스를 완료해야 합니다. 이벤트 구조 케이스가 이벤트 데이터를 변경하는 경우, LabVIEW는 변경된 데이터를 체인 내의 다음 이벤트 구조에 전달합니다. 체인 내의 이벤트 구조가 이벤트를 버릴 경우, LabVIEW는 체인 내의 남아있는 이벤트 구조에 이벤트를 전달하지 않습니다. LabVIEW는 설정된 모든 이벤트 구조에서 이벤트가 폐기되지 않고 처리된 후, 이벤트를 트리거한 사용자 동작의 처리를 완료합니다.

이벤트 구조에서 이벤트를 모니터하는 방법 선택하기

LabVIEW는 많은 다양한 이벤트를 생성할 수 있습니다. 이벤트 등록을 사용하여 이벤트 구조로 처리할 이벤트를 지정할 수 있습니다. 이벤트를 등록하면, LabVIEW는 특정 이벤트가 발생할 때 사용자에게 알립니다. LabVIEW에서는 정적으로 또는 다이내믹하게 이벤트를 등록할 수 있습니다. 정적 등록을 사용하면, LabVIEW는 이벤트가 발생하면 해당 이벤트를 항상 큐에 넣고 사용자에게 알립니다. 예를 들어 프론트패널 컨트롤에서 마우스 이동 이벤트를 정적으로 등록하면, LabVIEW는 사용자가 마우스를 해당 컨트롤 위로 움직일 때마다 이 이벤트를 큐에 넣습니다. 다이내믹 이벤트를 사용하면, 사용자가 지정한 특정 상황에서 해당 이벤트가 발생하는 경우에만 이벤트를 큐에 넣고 사용자에게 알리도록 어플리케이션을 설계할 수 있습니다. 예를 들어 사용자가 특정 키를 누를 때까지 마우스 이동 이벤트를 모니터하지 않을 경우가 있습니다. 또한 다이내믹 등록을 사용하여 어떤 VI나 컨트롤이 이벤트를 생성할지를 변경하고 LabVIEW가 이벤트를 생성하는 VI가 아닌 SubVI에서 이벤트를 처리하도록 할 수 있습니다.

정적 이벤트 등록

LabVIEW는 이벤트 구조가 포함된 VI를 실행하면 자동으로 정적 등록을 수행합니다. 이벤트 구조를 설정하여 각 이벤트 구조 케이스에서 처리하려는 VI의 프론트패널의 **사용자 인터페이스 이벤트**를 지정할 수 있습니다. 이벤트 편집 대화 상자를 사용하여 이벤트 구조가 **정적으로 등록된 이벤트를 처리**할 수 있도록 설정합니다.



노트

정적으로 이벤트 구조를 설정하여 다른 VI의 프론트패널에 대한 이벤트를 처리할 수 없습니다.

정적 이벤트 등록을 사용하는 예제는 examples\Structures\Event Structure\Event Structure.lvproj를 참조하십시오.

다이나믹 이벤트 등록

이벤트를 다이나믹하게 등록하면 LabVIEW가 언제 이벤트를 생성할지, LabVIEW가 어떤 이벤트를 생성할지 및 LabVIEW가 어디서 이러한 이벤트를 처리할지를 지정할 수 있습니다. 다이나믹 이벤트 등록을 사용하면 다음과 같은 혜택을 얻을 수 있습니다:

- 어플리케이션의 다른 부분이 실행되는 동안에도 이벤트를 등록하여 LabVIEW가 언제 이벤트를 생성할지 컨트롤할 수 있습니다. 예를 들어 데이터를 그래프에 플롯하는 버튼을 클릭한 후 그래프 위로 마우스를 움직이는 것과 같은 특정 상황에서서만 이벤트를 모니터하려는 경우가 있을 수 있습니다. 정적 등록을 사용하면, 마우스 이동 이벤트는 항상 발생합니다. 다이나믹 이벤트 등록을 사용하면, 사용자가 데이터를 그래프에 플롯하는 버튼을 클릭한 후에도 마우스 이동 이벤트를 등록할 수 있습니다.
- 실행 중에 어떤 VI 또는 컨트롤이 이벤트를 생성할지 수정할 수 있습니다. LabVIEW가 이벤트를 생성하는 객체를 변경하려면, 실행 중에 등록 정보를 **수정**하십시오.
- LabVIEW는 사용자가 프론트패널에서 직접 상호작용하지 않아도 이벤트를 생성할 수 있습니다. 다이나믹 이벤트 등록을 사용하면 블록 다이어그램 상에서 컨트롤 값을 업데이트하는 것과 같은 변경을 프로그램적으로 수행하는 어플리케이션을 생성할 수 있고, LabVIEW는 이러한 변경사항에 대한 이벤트를 생성합니다. 정적 이벤트 등록의 경우 LabVIEW가 이벤트를 생성하도록 하려면 사용자가 직접 상호작용을 해야 합니다.
- 이벤트가 발생한 VI에서만이 아닌 SubVI에서도 이벤트를 처리할 수 있으며, 이는 어플리케이션에 모듈화를 제공합니다. 예를 들어 DAQ 어플리케이션에서는 이벤트가 발생하는 VI에서 모든 이벤트를 처리하기보다 전체 어플리케이션의 모든 사용자 인터페이스 이벤트를 처리하는 SubVI, 모든 **사용자 이벤트**를 처리하는 SubVI 및 모든 DAQ 이벤트를 처리하는 SubVI를 생성할 수 있습니다.

LabVIEW의 데이터 그룹화

문자열, 클러스터, 배열, 집합을 이용한 데이터의 그룹화

LabVIEW에서는 다음 데이터 타입을 사용하여 데이터를 그룹화합니다.

- 문자열 — ASCII 문자의 시퀀스를 그룹화합니다.
- 클러스터 — 여러 타입의 데이터 원소를 함께 묶어 그룹화합니다.
- 배열 — 같은 타입의 데이터 원소를 정렬되지 않은 순서로 그룹화합니다. 2D 배열인 행렬 데이터 타입을 포함합니다. 이를 사용하면 보다 쉽게 수학 문제를 모델링할 수 있습니다.
 - 행렬 — 선형 대수 연산과 같은 일부 수학 연산에서 실수 또는 복소수 스칼라 데이터의 행이나 열을 그룹화합니다.
- 집합 — 다음 서브 타입을 포함합니다.
 - 세트 — 같은 타입의 데이터 원소를 정렬된 순서로 그룹화합니다.
 - 맵 — 키-값 쌍을 그룹화합니다. 이때 모든 키는 고유합니다. 키와 값은 각각 어떤 데이터 타입이든 될 수 있습니다.

관련 개념

[문자열을 이용한 데이터의 그룹화](#)

[배열과 클러스터를 이용한 데이터의 그룹화](#)

[집합을 이용한 데이터의 그룹화](#)

[행렬 사용하기](#)

문자열을 이용한 데이터의 그룹화

문자열은 디스플레이할 수 있거나 할 수 없는 ASCII 문자의 나열입니다. 문자열의 정보 및 데이터 포맷은 플랫폼에 의존적이지 않습니다. 일반적으로 문자열은 다음과 같은 작업에 사용됩니다.

- 단순 텍스트 메시지 생성하기.
- 인스트루먼트에 텍스트 명령을 보내서 인스트루먼트를 컨트롤하고, ASCII 또는 2진 문자열로 데이터 값을 반환한 후 숫자형 값으로 변환하기.
- 숫자형 데이터를 디스크에 저장하기. ASCII 파일에 숫자형 데이터를 저장하려면 디스크 파일에 데이터를 쓰기 전에 반드시 숫자형 데이터를 문자열로 변환해야 합니다.
- 대화 상자를 사용하여 사용자에게 지시 또는 입력요청.

프론트패널에서 문자열은 테이블, 문자 입력 박스 및 라벨로 나타납니다. LabVIEW에는 문자열 포맷, 분석, 기타 편집 등의 문자열 작업에 활용할 수 있는 VI와 함수들이 포함되어 있습니다.

ASCII 코드 및 변환 함수에 대한 추가적인 정보는 ASCII 코드 토픽을 참조하십시오.

프론트패널의 문자열

문자열 컨트롤과 인디케이터를 텍스트 입력 상자와 라벨로 사용합니다.

문자열 디스플레이 타입

프론트패널의 문자열 컨트롤과 인디케이터에서 마우스 오른쪽 버튼을 클릭하여 다음 테이블에서 보이는 디스플레이 타입을 선택할 수 있습니다. 또한 문자열 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 보이는 항목 » 디스플레이 스타일을 선택하여 디스플레이 타입을 나타내는 객체에서 문양을 디스플레이 합니다. 또한 테이블에는 각 디스플레이 타입에 대한 예제 메시지도 있습니다.

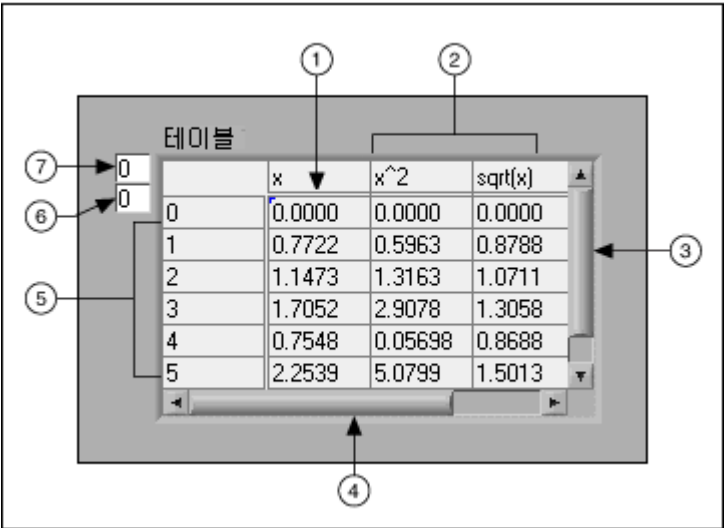
타입	디스플레이 타입	설명	메시지
n	일반 디스플레이	인쇄 가능한 문자를 컨트롤의 디스플레이합니다. 디스플레이할 수 없는 문자는 일반적으로 상자로 나타납니다.	There are four display types.\ is a backslash.
\	'\ '코드 디스플레이	디스플레이할 수 없는 문자를 모백래코드 포기합니다.	There\sare\sfour\sdisplay\stypes.\n\\\sis\sa\sbackslash.

타입	디스플레이	설명	메시지
P	암호 디스플레이	스페이스를 포함한 각 문자를 별표 *로 표기합니다.	*****
*	16진수 디스플레이	해당 문자열 값 대신에 각 문자의 ASCII 값을 16진수로 표기합니다.	<pre> 5468 6572 6520 6172 6520 666F 7572 2064 6973 706C 6179 2074 7970 6573 2E0A 5C20 6973 2061 2062 6163 6B73 6C61 7368 2E </pre>

테이블

테이블 컨트롤을 사용하여 프런트패널에 테이블을 생성합니다. 테이블의 각 셀이 문자열이고 각 셀은 열과 행에 상주합니다. 그러므로 테이블은 문자열의 2D 배열의 디스플레이입니다.

다음 설명은 테이블과 그 부분 모두를 보여줍니다.



1	인덱스 값으로 구분되는 셀
2	열 이름
3	수직 스크롤 막대
4	수평 스크롤 막대
5	행 이름
6	수평 인덱스
7	수직 인덱스

문자열 편집, 포맷, 분석하기

문자열 함수를 사용하여 다음과 같은 방법으로 문자열을 편집합니다:

- 문자열에서 문자나 문자열의 일부를 찾거나, 불러오고, 대체합니다.
- 문자열의 모든 문자를 대문자나 소문자로 바꿉니다.
- 문자열에서 일치하는 패턴을 찾고 가져옵니다.
- 문자열에서 한 개의 라인을 가져옵니다.
- 문자열에서 문자를 회전시키거나 뒤집습니다.
- 두 개 또는 그 이상의 문자열을 합칩니다.
- 문자열에서 문자를 삭제합니다.

문자열을 프로그램적으로 편집할 때 메모리 사용을 최소화하는 추가적인 팁은 [LabVIEW Style Checklist](#)를 참조하십시오. 문자열을 편집하기 위해 문자열 함수를 사용하는 것에 대한 예제는 labview\examples\Strings\Strings.lvproj를 참조하십시오.

문자열 포맷과 분석

데이터를 다른 VI, 함수, 또는 어플리케이션에서 사용하기 위해서 데이터를 문자열로 바꾸고 그 VI, 함수, 또는 어플리케이션에서 읽을 수 있는 포맷으로 바꾸어야 할 때가 있습니다. 예를 들어, Microsoft Excel은 문자열에 탭, 콤마, 또는 스페이스와 같은 구분 문자가 있어야 합니다. 엑셀은 이 경계 부호를 사용하여 숫자나 단어를 셀로 구분합니다.

예를 들어, [텍스트 파일에 쓰기] 함수를 사용하여 숫자형 값의 1D 배열을 스프레드시트에 쓰려면, 배열을 문자열 포맷으로 바꾸고 탭과 같은 구분 문자로 각 숫자를 분리해야 합니다. [구분된 스프레드시트 쓰기] VI를 사용하여 배열을 문자열로 포맷하고 구분 문자를 삽입할 수 있습니다.

문자열 함수를 이용하여 다음과 같은 작업을 수행합니다:

- 두 개 또는 그 이상의 문자열을 합칩니다.
- 문자열로부터 문자열의 부분을 빼냅니다.
- 데이터를 문자열로 바꿉니다.
- 워드 프로세싱 어플리케이션 또는 스프레드시트 어플리케이션에 사용할 수 있도록 문자열을 포맷합니다.

파일 I/O VI와 함수를 사용하여 문자열을 텍스트나 스프레드시트 파일로 저장합니다.

포맷 지정자

대부분의 경우, 문자열의 포맷을 위하여 [문자열] 함수의 포맷 문자열 파라미터에 하나 또는 그 이상의 포맷 지정자를 입력해야 합니다. 포맷 지정자는 어떻게 숫자 데이터에서 문자열로 혹은 그 반대로 바꿀 것인가를 지정하는 코드입니다. LabVIEW는 변환 코드를 이용하여 파라미터의 문자 포맷을 지정합니다. 예를 들어, 포맷 지정자 %x는 16진수 정수를 문자열로 혹은 그 반대로 바꿉니다.

[문자열로 포맷]과 [문자열로부터 스캔] 함수는 포맷 문자열 입력에서, 확장 가능한 함수의 각 입력 또는 출력에 하나씩, 여러 포맷 지정자를 사용할 수 있습니다.

[배열을 스프레드시트 문자열로]와 [스프레드시트 문자열을 배열로] 함수는 변환할 입력이 하나이므로, 포맷 문자열 입력에서 단 1 개의 포맷 지정자만 사용합니다. LabVIEW는 이 함수들에 추가로 더 입력하는 지정자를 특별한 의미가 없는 문자열로 간주합니다.

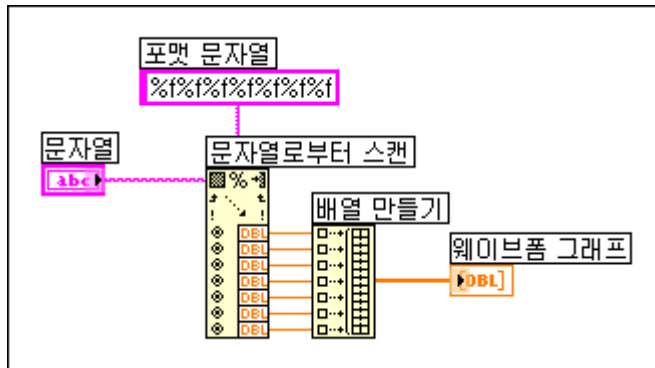
숫자와 문자열 데이터

문자열 데이터는 ASCII 문자이고 숫자 데이터는 그렇지 않으므로 숫자 데이터와 문자열 데이터는 다릅니다. 텍스트와 스프레드시트 파일은 문자열만 입력받습니다. 숫자 데이터를 텍스트나 스프레드시트 파일에 쓰기 위해서는 반드시 숫자 데이터를 문자열로 변환해야 합니다.

숫자값의 한 세트를 기존의 문자열에 추가하기 위해서, 숫자 데이터를 문자열로 변환하고 [문자열 연결]이나 다른 문자열 함수를 사용하여 새 문자열을 기존의 문자열에 추가합니다. 문자열/숫자 변환 함수를 사용하여 숫자값을 문자열로 변환합니다.

문자열은 그래프나 차트에 디스플레이하는 숫자값 세트도 가질 수 있습니다. 예를 들어, 차트에 플롯하려는 숫자값의 세트를 포함하고 있는 텍스트 파일을 읽을 수 있습니다. 그러나 이 숫자값들은 ASCII 텍스트이므로 차트에 숫자값을 플롯하기 전에 먼저 문자열로 읽은 다음 문자열을 숫자값 세트로 포맷을 바꾸어 주어야 합니다.

다음 블록 다이어그램은 숫자형 값의 세트를 가지고 있는 문자열에서 **문자열을 숫자값으로 변환**하고, 숫자형 값의 배열을 만든 후 그래프에 플롯한 것입니다.



문자열 검색하기

[문자열] 함수를 사용하여 정규식으로 문자, 단어, 구절, 또는 숫자에서 문자열을 검색합니다. 정규식을 사용하여 텍스트에서 리터럴 문자열을 검색할 수 있습니다. 예를 들어 "c"로 시작하고 "t"로 끝나는 단어 또는 한 문자로 시작하여 하나 또는 그 이상의 문자가 뒤따르는 부분문자열을 검색하여 "cat"라는 단어나 특정한 패턴과 일치하는 텍스트 내의 문자열을 검색할 수 있습니다.

[패턴 일치] 함수와 [패턴 검색과 대체] VI를 사용하여 정규식 입력에 리터럴 문자열이나 **특수 문자**를 입력하여 문자열을 검색합니다. [패턴 검색과 대체] 함수를 사용하여 문자열에서 **텍스트를 검색하고 대체**합니다.

표준 Perl Compatible Regular Expression (PCRE) 구문을 사용하는 더 복잡한 정규식 패턴 일치를 위해 [정규식 일치] 함수와 [문자열 검색과 대체] 함수를 사용합니다. [문자열 검색과 대체] 함수에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 정규식을 선택하여 고급 정규식 검색 및 대체 문자열에서 부분 일치 치환에 대한 함수를 설정합니다. [정규식 일치] 함수의 **크기를 조정**하여 문자열에서 발견된 부분 일치를 봅니다. [패턴 일치] 함수나 [정규식 일치] 함수를 사용하여 정규식을 위한 문자열을 검색할 수 있습니다. [정규식 일치] 함수는 문자열 일치에 더 많은 옵션을 제공하지만 [패턴 일치] 함수보다 느리게 수행됩니다.

정규식 지원은 PCRE 라이브러리 패키지에 의해 제공됩니다. PCRE 라이브러리 패키지의 재배포 라이선스에 대한 더 자세한 정보는 <National Instruments>_Legal Information 디렉토리를 참조하십시오.

Perl Compatible Regular Expressions (PCRE)에 대한 더 자세한 정보는 www.pcre.org/의 PCRE library package를 참조하십시오.

배열과 클러스터를 이용한 데이터의 그룹화

배열과 클러스터 컨트롤 그리고 함수를 이용하여 데이터를 그룹화합니다. 배열은 같은 타입의 데이터 원소를 그룹화합니다. 클러스터는 여러 타입의 데이터 원소를 하나의 그룹으로 묶습니다.

배열

배열은 원소와 차원으로 구성됩니다. 원소는 배열을 구성하는 데이터입니다. 차원은 배열의 길이, 높이, 또는 폭입니다. 배열은 하나 또는 그 이상의 차원을 가질 수 있으며, 메모리가 허용하는 경우 차원마다 최대 $(2^{31})-1$ 개의 원소를 가질 수 있습니다.

숫자형, 불리언, 경로, 문자열, 웨이브폼, 클러스터 데이터 타입의 배열을 만들 수 있습니다. 비슷한 데이터들로 작업을 하거나 반복된 연산을 수행하는 경우에 배열을 사용하면 유용할 수 있습니다. 배열은 웨이브폼에서 수집한 데이터나 루프에서 생성된 데이터를 저장하는데 이상적입니다. 이 경우, 루프는 매 반복마다 배열의 원소를 하나씩 생성하게 됩니다. 배열에서 Microsoft Excel과 같은 곳으로 데이터를 반출할 수 있습니다. 배열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반출을 선택하여 사용 가능한 반출 옵션을 봅니다.



노트

LabVIEW의 배열 인덱스는 0에서 시작합니다. 배열에 있는 첫번째 원소의 인덱스는 차원에 관계없이 0입니다.

제약

배열로 이루어진 배열은 만들 수 없습니다. 그렇지만, 다차원의 배열을 사용하거나 각각의 클러스터가 하나 또는 그 이상의 배열을 포함하는 클러스터의 배열을 생성할 수 있습니다. 또한 서브패널 컨트롤, 탭 컨트롤, .NET 컨트롤, ActiveX 컨트롤, 차트, 멀티플롯 XY 그래프 또는 여러 열 리스트박스의 배열도 생성할 수 있습니다.

인덱스

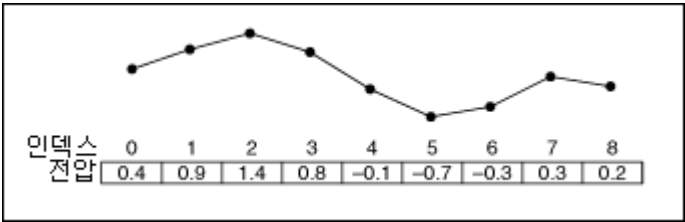
특정 원소를 배열에 위치시키기 위해서는 차원당 한 개의 인덱스가 필요합니다. LabVIEW에서 인덱스는 배열을 탐색하고 블록 다이어그램의 배열에서 원소나, 행, 열, 그리고 페이지를 불러오기 위해 사용됩니다.

배열의 예

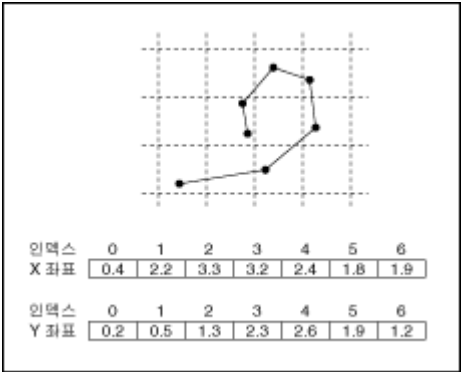
태양계의 8개 행성을 나열한 문자 배열이 간단한 배열의 예입니다. LabVIEW에서 이것은 8개의 원소가 있는 1D 문자열 배열로 나타납니다.

배열의 원소에는 순서가 있습니다. 배열에서는 인덱스를 사용하여 사용자가 특정 원소에 접근할 수 있도록 합니다. 인덱스는 0부터 시작합니다. 이는 n 개의 원소를 가진 배열에 대하여 0에서 $n-1$ 범위로 인덱스됨을 의미합니다. 예를 들어, $n=8$ 인 8개의 행성에 대하여 인덱스는 0에서 7이 됩니다. 지구는 세 번째 행성이므로 인덱스는 2입니다.

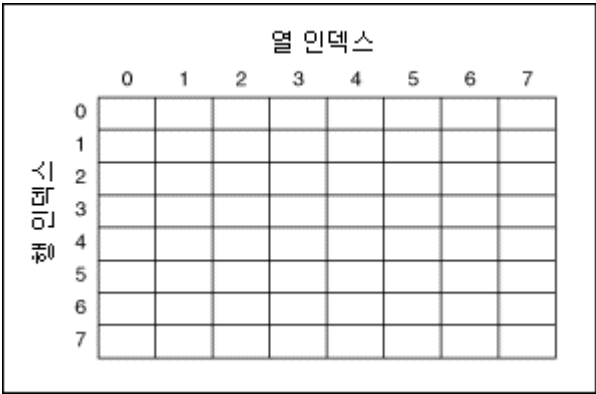
배열에 대한 또 다른 예는 다음 그림과 같이 연속적인 시간 간격에서 각 연속되는 원소가 연속되는 시간 간격에서의 볼트값인 숫자 배열로 표현되는 웨이브폼입니다.



좀더 복잡한 배열에 대한 예제는 다음 그림과 같이 포인트 배열로 나타내는 그래프입니다. 여기서 각 포인트는 X와 Y 좌표를 나타내는 숫자값의 쌍을 포함하는 클러스터입니다.

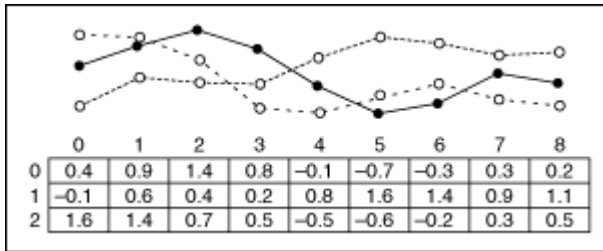


이전 예제는 1D 배열을 사용합니다. 2D 배열의 원소는 셀 안에 저장됩니다. 특정 원소를 찾으려면 행 인덱스와 열 인덱스가 필요하며, 이 인덱스는 0을 기준으로 시작됩니다. 다음 그림은 8 × 8 = 64 원소를 가진 2D 배열의 8행과 8열을 보여줍니다.



예를 들어, 체스판은 8행 8열이며 총 64개 자리가 있습니다. 각 자리는 비어있거나 한 개의 체스말이 위치할 수 있습니다. 체스판은 2D 문자열 배열로 나타낼 수 있습니다. 각 문자열은 체스판의 대응하는 위치에 있는 체스말의 이름이며, 빈 곳은 빈 문자열로 나타냅니다.

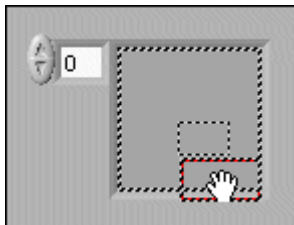
배열에 행을 추가하여 이전 1D 배열 예제를 2차원으로 일반화할 수 있습니다. 다음 그림은 2차원 숫자값 배열로 표현되는 웨이브폼들의 집합을 보여줍니다. 행 인덱스는 웨이브폼을 선택하고, 열 인덱스는 그 웨이브폼의 포인트를 선택합니다.



배열을 사용한 예제는 labview\examples\Arrays\Arrays.lvproj를 참조하십시오. 관련 예제 찾기

배열 컨트롤, 인디케이터, 상수 생성하기

아래의 프런트패널과 같이 프런트패널에 배열 셀을 추가한 후, 숫자형, 불리언, 문자열, 경로, 참조 번호 또는 클러스터 컨트롤이나 인디케이터와 같은 데이터 객체 또는 원소를 배열 셀 안으로 끌어와 배열 컨트롤이나 배열 인디케이터를 만들 수 있습니다.



배열 셀은 새 객체에 맞도록 자동으로 크기가 조절됩니다.

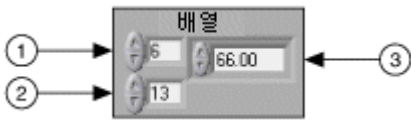
블록 다이어그램에서 배열 상수를 생성하려면, 함수 팔레트에서 배열 상수를 선택하고, 블록 다이어그램에 배열 셀을 놓은 후, 문자열 상수, 숫자형 상수, 불리언 상수 또는 클러스터 상수를 배열 셀 안에 놓습니다. 배열 상수는 다른 배열과 비교를 할 때의 기준으로 사용하거나, 상수 데이터를 저장하는 데 사용할 수 있습니다.

다차원 배열 생성하기

프런트패널에 다차원 배열을 추가하려면, 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차원 추가를 선택합니다. 또한 인덱스 디스플레이의 크기를 조정하여 원하는 수만큼의 차원을 만들 수 있습니다. 한 번에 한 개의 차원을 삭제하려면, 인덱스 디스플레이를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 차원 제거를 선택합니다. 또한, 인덱스 디스플레이를 크기 조정하여 차원을 제거할 수도 있습니다.

특정 원소를 프런트패널에 디스플레이하기 위하여, 인덱스 디스플레이에 인덱스 숫자를 입력하거나 또는 인덱스 디스플레이의 화살표를 이용하여 해당 숫자를 탐색합니다.

예를 들면 2D 배열은 행과 열을 가지고 있습니다. 다음 프런트패널에서 보이는 것처럼, 왼쪽에 있는 두 박스의 위쪽 디스플레이는 행 인덱스이고 아래쪽 디스플레이는 열 인덱스입니다. 행과 열의 표시를 합하여 지정된 위치의 값을 보여줍니다. 다음 프런트패널은 6 행, 13 열의 값이 66임을 보여줍니다.



1	행 인덱스
2	열 인덱스
3	행과 열 위치에서의 값

행과 열은 0을 기준으로 하며, 첫번째 행이 0번 행, 두번째 행이 1번 행이 됩니다. 다음 배열에서 인덱스 디스플레이를 1행, 2열로 변경하면 6의 값이 나타납니다.

0	1	2	3
4	5	6	7
8	9	10	11

배열 차원의 범위를 벗어난 영역의 행과 열을 보이려고 하면, 그곳에는 정의된 값이 없다는 것을 나타내기 위하여, 배열 컨트롤이 회색화되어 나타나며 LabVIEW는 해당하는 데이터 타입의 기본값을 보여줍니다. 데이터 타입의 기본값은 배열의 데이터 타입에 의존합니다.

위치 도구를 사용하여 한 번에 하나 이상의 행과 열이 보이도록 배열의 크기 조정을 합니다.

또한, 특정한 원소를 탐색하기 위해 배열의 스크롤 막대를 사용할 수 있습니다. 배열을 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 수직 스크롤 막대를 선택하거나 또는 보이는 아이템 >> 수평 스크롤 막대를 선택하여 배열에서 스크롤 막대를 디스플레이합니다.



노트 LabVIEW의 배열 함수는 행 우선 순서로 배열 데이터에 접근합니다. 2D 배열에서는 행이 우선하는 인덱스가 됩니다. 열은 보조 인덱스의 역할을 합니다. 더 큰 다차원 배열에서는 열은 마지막 인덱스의 역할을 하며, 그보다 우선하는 인덱스 여러 개가 그 앞에 추가됩니다. 그러므로 볼륨이 가장 우선순위가 높은 첫번째 인덱스가 되며, 그 다음에 페이지, 행, 열의 형태가 됩니다. 이러한 이름은 인덱스 식별자이며 다른 의미는 없습니다.

배열 함수

배열 함수를 사용하여 배열을 생성하고 조작합니다. 예를 들어, 다음 그림과 같은 태스크를 수행할 수 있습니다:

- 배열에서 개별 데이터 원소를 빼냅니다.
- 배열에서 데이터 원소를 삽입, 삭제, 또는 대체합니다.
- 배열을 분리합니다.

[배열 만들기] 함수를 사용하여 배열을 프로그램적으로 만듭니다. 또한, 배열을 만들기 위해 **루프를 사용**할 수 있습니다.

루프 안에서 배열 함수를 사용할 때 메모리 사용을 최소화하는 방법에 대한 추가적인 팁은 **LabVIEW Style Checklist**를 참조하십시오.

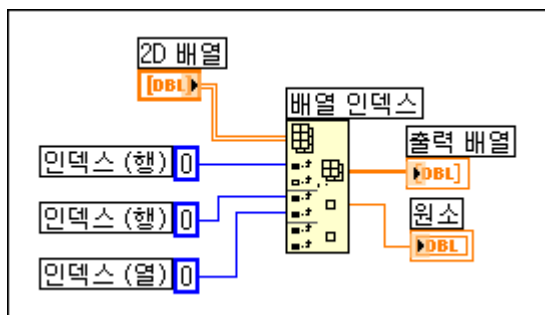
배열 함수의 자동 크기 조절

[배열 인덱스], [배열 부분 대체], [배열에 삽입], [배열로부터 삭제], [배열 잘라내기] 함수는 자동적으로 연결한 입력 배열의 차원에 맞도록 크기 조정합니다. 예를 들면, 1D 배열을 이들 함수에 연결하면 함수는 단일 인덱스 입력만 나타냅니다. 만약 2D 배열을 같은 함수에 연결하면 두 개의 인덱스 입력을 보여줍니다. 하나는 행, 다른 하나는 열에 대한 입력을 나타냅니다.

위치 도구를 사용하여 함수의 크기를 수동으로 조절하여 이들 함수에서 하나 이상의 원소나 부분배열(행, 열, 또는 페이지)에 접근할 수 있습니다. 이들 함수 중 하나를 확장할 때, 함수에 연결된 배열의 차원에 따라 늘어나는 폭이 결정됩니다. 1D 배열을 이들 함수에 연결한 경우, 함수는 한 개의 인덱스 입력씩 확장할 수 있습니다. 만약 2D 배열을 같은 함수에 연결한 경우, 함수는 행과 열에서 각각 하나씩인 두 개의 인덱스 입력을 확장할 수 있습니다.

연결한 인덱스 입력이 접근하거나 수정하려고 하는 부분배열의 형태를 결정합니다. 예를 들면, [배열 인덱스] 함수의 입력이 2D 배열이고 행 입력에만 연결했을 경우, 배열에서 1D 행을 추출합니다. 열 입력에만 연결했을 경우, 배열에서 1D 열을 통째로 추출합니다. 행 입력과 열 입력에 모두 연결했을 경우, 배열에서 한 개의 원소를 추출합니다. 각 입력 그룹은 독립적이며 배열의 모든 차원의 모든 부분에 접근할 수 있습니다.

다음 블록 다이어그램은 [배열 인덱스] 함수를 사용하여 2D 배열에서 행과 원소를 가져옵니다.



배열에서 여러 연속되는 값에 접근하기 위하여 [배열 인덱스] 함수를 확장합니다. 그러나 각 증분의 인덱스 입력에 값을 연결하지 않습니다. 예를 들어, 2D 배열에서 첫번째, 두번째, 그리고 세번째 행을 뽑아내려면, [배열 인덱스] 함수를 3 개의 인덱스로 확장한 다음 1D 배열 인디케이터를 각 부분배열 출력에 연결합니다.

배열의 기본 데이터

배열의 범위를 벗어나는 인덱스 선택은 배열 원소 파라미터에 **기본값**을 생성합니다. [배열 크기] 함수를 사용하여 배열의 크기를 결정할 수 있습니다.

While 루프를 사용하여 마지막 원소를 넘어 배열을 인덱스하거나, [배열 인덱스] 함수의 인덱스 입력에 너무 큰 값을 입력하거나, [배열 인덱스] 함수에 빈 배열을 공급하면, 의도치 않게 배열의 경계를 넘어 인덱스할 수 있습니다.

클러스터

클러스터는 여러 타입의 데이터 원소를 하나의 그룹으로 묶습니다. 클러스터의 예는 불리언 값, 숫자값, 문자열이 결합된 LabVIEW 에러 클러스터를 들 수 있습니다. 클러스터는 텍스트 기반 프로그래밍 언어의 레코드(record) 또는 구조(struct)와 유사합니다.

여러 데이터 원소를 클러스터로 묶으면, 블록 다이어그램에서 와이어의 복잡한 연결을 피할 수 있으며 SubVI에 필요한 커넥터 팬 터미널의 수를 감소시킬 수 있습니다. 커넥터 팬은 최대 28개의 터미널을 가질 수 있습니다. 프런트패널에 다른 VI로 전달하려는 컨트롤과 인디케이터가 28개 이상 있는 경우, 이중 일부를 하나의 클러스터로 그룹화하고, 이 클러스터를 커넥터 팬의 한 터미널에 할당합니다.

블록 다이어그램에서 대부분의 클러스터는 핑크색의 와이어 패턴과 데이터 타입 터미널을 가집니다. 에러 클러스터는 짙은 노란색의 와이어 패턴과 데이터 타입 터미널을 가집니다. 숫자값의 클러스터가 때때로 데이터 포인트를 나타내는 경우, 갈색의 와이어 패턴과 데이터 타입 터미널을 가집니다. 갈색의 숫자 클러스터를 숫자 함수, 예를 들어 [더하기]나 [제곱근]에 연결하여 클러스터 원소 모두에 같은 연산을 동시에 수행하게 할 수 있습니다.

클러스터 원소의 순서

클러스터와 배열 원소가 모두 순서를 갖고 있지만, 풀기 함수를 사용하여 한 번에 모든 클러스터 원소를 풀어야 합니다. 이름으로 풀기 함수를 사용하여 이름에 따라 클러스터 원소를 풀 수 있습니다. 이름으로 풀기 함수를 사용하는 경우, 각 클러스터 원소에 라벨이 있어야 합니다. 또한, 클러스터는 크기가 고정된다는 점에서 배열과 차이가 납니다. 배열과 마찬가지로 클러스터는 컨트롤이거나 인디케이터 둘 중 하나가 될 수 있습니다. 클러스터 안에는 컨트롤과 인디케이터가 함께 포함될 수 없습니다.

클러스터 원소는 셀에서의 그 위치에 관계 없이 논리적인 순서를 가지고 있습니다. 클러스터에 첫 번째로 놓인 객체는 원소 0이고, 두 번째 객체는 원소 1이 되는 방식으로 순서가 정해집니다. 원소를 삭제하면 순서가 자동으로 조절됩니다. 블록 다이어그램에 있는 [묶기] 및 [풀기] 함수에서 원소가 터미널로 표시되는 순서는 클러스터 순서에 의해 결정됩니다. 클러스터의 경계에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 클러스터 내의 컨트롤 순서 재설정을 선택하여 클러스터 순서를 보고 **클러스터의 순서를 수정**할 수 있습니다.

두 개의 클러스터를 연결하려면 반드시 양쪽이 같은 수의 원소를 가지고 있어야 합니다. 클러스터 순서로 결정되는 대응 원소는 반드시 호환되는 데이터 타입을 가져야 합니다. 예를 들어, 한 클러스터의 배정도 부동소수가 다른 클러스터의 문자열에 대응되면, 블록 다이어그램의 와이어는 깨지고 VI는 실행되지 않습니다. 숫자값이 다른 형을 가진 경우, LabVIEW는 같은 형으로 **강제 변환**합니다.

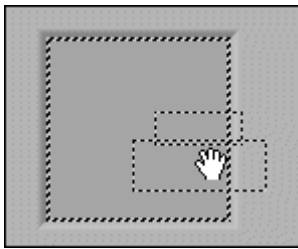
클러스터 함수

클러스터, 클래스, & 배리언트 VI와 함수를 사용하여 클러스터를 생성하고 조작합니다. 예를 들어, 다음 그림과 같은 태스크를 수행할 수 있습니다:

- 클러스터에서 개별 데이터 원소를 뺍니다.
- 클러스터에 개별 데이터 원소를 추가합니다.
- 클러스터를 개별 데이터 원소로 나눕니다.

클러스터 컨트롤, 인디케이터, 상수 생성하기

다음 프런트패널에서 보이는 것처럼 프런트패널에 클러스터 셸을 추가하여 숫자형, 불리언, 문자열, 경로, 참조 번호, 배열, 클러스터 컨트롤 또는 인디케이터 등의 데이터 객체 또는 원소를 클러스터 셸에 끌어다 놓아서 프런트패널에 클러스터 컨트롤 또는 인디케이터를 생성합니다.



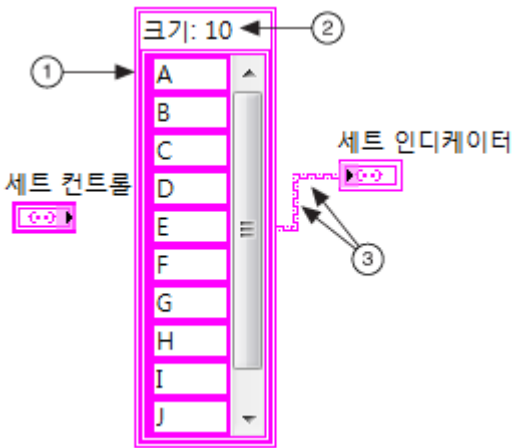
블록다이어그램에서 클러스터 상수를 만들려면 함수 팔레트에서 클러스터 상수를 선택하고, 블록다이어그램에 클러스터 셸을 위치시킨 후, 문자열 상수, 숫자형 상수, 불리언 상수, 또는 클러스터 상수를 클러스터 셸 안에 놓습니다. 클러스터 상수는 다른 클러스터와 비교를 하기 위한 기준으로 또는 상수 데이터를 저장하는데 사용할 수 있습니다.

집합을 이용한 데이터의 그룹화

집합은 같은 타입의 데이터를 모을 수 있는 컨테이너입니다. 집합의 각 객체는 집합의 원소 또는 엔트리입니다. 집합 데이터 타입을 사용하면 데이터 저장 및 접근에 필요한 고성능 데이터 구조를 생성할 수 있습니다. LabVIEW는 다음의 집합 데이터 타입을 포함합니다: 세트 및 맵.

집합 세트

세트는 같은 데이터 타입의 고유한 원소를 정렬된 순서로 그룹화합니다. 새 원소를 삽입하는 경우 해당 원소가 세트에 이미 존재하지 않는 경우에만 세트가 변경됩니다.

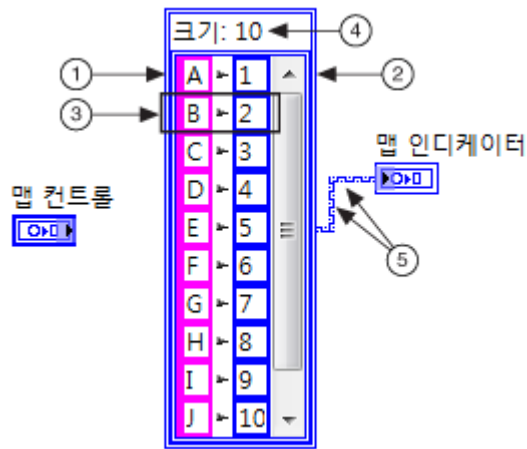


①	원소
②	크기—세트의 원소 개수
③	세트의 와이어—와이어 색이 원소의 데이터 타입 색과 일치합니다.

세트는 원소를 정렬된 순서로 관리하기 때문에, 세트를 사용하면 데이터 크기가 큰 경우에도, 배열과 같이 순서가 지정되지 않은 데이터 구조보다 더 신속하게 데이터에 대해 검색, 삽입, 수정, 제거 작업을 수행할 수 있습니다. 세트에는 메모리 오버헤드가 있어서 원소를 정렬된 순서로 관리합니다. 반면 배열에는 메모리 오버헤드가 없습니다.

집합 맵

맵은 키-값쌍을 정렬된 순서로 그룹화합니다. 이때 모든 키는 고유합니다. 키와 값은 각각 어떤 데이터 타입이든 될 수 있습니다. 값을 검색하는데 키를 사용하기 때문에 맵을 딕셔너리라고도 합니다. 키-값 쌍을 맵에 삽입하면 키가 맵에 이미 존재하는지 여부에 따라 해당 쌍이 맵에 추가되거나 맵에서 관련 키의 기존 값이 새 값으로 대체됩니다.



①	키
②	값

③	엔트리—관련된 키와 값의 쌍
④	크기—맵의 엔트리 개수
⑤	맵의 와이어—와이어 색이 값의 데이터 타입 색과 일치합니다.

배리언트 속성도 값을 검색하는 디렉터리로 사용될 수 있지만, 맵의 경우 키와 값이 각각 어느 LabVIEW 데이터 타입이든 될 수 있다는 점에서 배리언트 속성보다 유연성이 높습니다. 배리언트 속성의 경우, 키는 문자열이어야 하고 값은 배리언트여야 합니다. 또한 맵은 배리언트 속성보다 에러 발생 가능성이 낮습니다. 맵의 경우, 값이 특정 데이터 타입을 허용하기 때문에 LabVIEW는 편집 시에 와이어 연결 에러를 반환합니다. 배리언트 속성의 경우, 값이 모든 데이터를 허용하는 배리언트 타입이기 때문에 LabVIEW는 실행 시에 와이어 연결 에러를 반환합니다.

집합 vi와 함수, 집합 컨트롤과 인디케이터를 사용하여 집합을 생성하거나 조작합니다.



노트

집합의 임베디드 데이터는 대화식 또는 프로그램적으로 수정할 수 없습니다. 집합 데이터는 집합 프론트패널 터미널에 작성하거나 집합에서 마우스 오른쪽 버튼을 클릭한 후 데이터 작업>데이터 복사 또는 데이터 작업>데이터 붙여넣기를 선택하여 전체만 업데이트할 수 있습니다.

집합을 사용하여 데이터를 조작하는 예제는 다음 vi나 프로젝트를 참조하십시오.

- labview\examples\Collections\Set Collection - Word Counting.vi
- labview\examples\Collections\Map Collection - Comparing Test Results.vi
- labview\examples\Collections\Map Collection - Word Counting.vi
- labview\examples\Design Patterns\Registration Map\Registration Map Usage.lvproj

포맷 지정자 구문

문자열 포맷, 숫자를 문자열로 변환, 문자열에 디스플레이할 수 없는 문자를 삽입할 때 포맷 지정자를 사용합니다.

[문자열로 포맷]과 [배열을 스프레드시트 문자열로]와 같은 문자열을 출력으로 내보내는 함수에서, 포맷 지정자는 다음의 구문 원소를 사용합니다. 중괄호 []는 옵션 원소를 포함합니다.

%[\$][-][+][#][^][0][길이][.정밀도|_|_유효숫자][{ 단위 }][< 내장 정보 >] 변환 코드

길이는 반드시 제로보다 커야 하며, 정밀도와 유효숫자는 반드시 0과 같거나 커야 합니다.

[문자열로부터 스캔]과 [스프레드시트 문자열을 배열로]와 같이 문자열을 스캔하는 함수의 경우, 포맷 지정자는 다음과 같이 단순화된 구문 원소를 사용합니다.

%[폭] 변환 코드

[문자열로 포맷], [파일로 포맷], [문자열로부터 스캔], [파일로부터 스캔] 함수는, 확장 가능 함수의 각 입력 및 출력에 하나씩, 포맷 문자열 입력에서 여러 포맷 지정자를 사용할 수 있습니다.

포맷 지정자 구문 원소

다음 테이블은 포맷 지정자를 위한 구문 원소를 디스플레이합니다. 추가적인 정보는 [포맷 지정자의 예제](#)를 참조합니다.



노트

하나의 포맷 지정자에서 정밀도와 유효 숫자를 함께 사용할 수 없습니다.

포맷 지정자 예제

다음 테이블은 포맷 지정자에 대해 [구문 원소](#)를 사용하는 예를 디스플레이합니다.

타입	인수	포맷 문자열	결과 문자열	주석
자동 포매팅 %g	12.00	%#g	12	#으로 지정했다면, LabVIEW는 뒤따르는 제로를 제거합니다. g로 지정했다면, LabVIEW는 포맷할 숫자를 기준으로 과학적 표기법 또는 부동소수 표기법을 선택합니다.
12000000	%#g	1.2E+7		

타입	인수	포맷 문자열	결과 문자열	주석
소수점 %d	12.67	score= %d%%	score= 13%	%d를 지정하면, LabVIEW는 정수를 10진법으로 디스플레이합니다. - 비트 크기에 제한 없이 부호 있는 정수 또는 부호 없는 정수이면, LabVIEW는 정수를 수정하지 않고 그대로 디스플레이합니다. - LabVIEW는 부동소수점을 부호 있는 64비트 정수로 반올림합니다. NaN과 부호 있는 64비트 정수의 최대값보다 큰 값은 부호 있는 64비트 정수의 최대값인 9223372036854775807로 반올림됩니다. 부호 있는 64비트

타입	인수	포맷 문자열	결과 문자열	주석
부동소수점 %f	12.67	온도: %5.1f	온도: 12.7	포맷 문자열 섹션에서 5는 길이 5를 지정하며, 1은 소수점 또는 정밀도의 오른쪽에 있는 자릿수를 지정합니다.
12.67 N	%5.3f	12.670 N	단위는 [문자열로 포맷] 함수를 사용해야만 유효합니다. 이들은 물리량 입력의 예입니다. 두 번째 예제는 하나의 단위를 다른 단위로 변환하는 방법을 보여줍니다. 물음표 표시는 포맷 지정자의 단위가 입력 단위와 충돌이 일어났음을 알려줍니다.	
12.67 N	%5.3{mN}f	12670.000 mN		
12.67 N	%5.3{kg}f	12.670 ?kg		
과학적 표기법 %e	12.67	%.3e	1.267E+1	지수가 언제나 3의 배수인 경우 ^를 추가하여 공학적 표기법으로 변경합니다.
12.67	%^.3e	12.670E+0		

타입	인수	포맷 문자열	결과 문자열	주석
SI 표기법 %p	120000000	%.2p	12.00M	.2의 값은 결과 문자열에서 2 자릿수의 정밀도로 나타나도록 지정합니다. _2의 값은 결과 문자열에서 2개의 유효숫자로 나타나도록 지정합니다. %p는 SI 접두어를 추가합니다.
120000000	%.2p	12M		
16진 %x	12	%02x	0C	A — 결과 문자열의 왼쪽으로 자리맞춤되도록 지정합니다. A0은 결과 문자열에 제로로 채우도록 지정합니다. 숫자는 결과 문자열의 길이를 지정합니다. 즉, -n의 값은 결과 문자열을 n 폭만큼 공백으로 채웁니다. b는 밀이 2이고, o는 밀이 8이며, x는 밀이 16입니다.
8진 %o	12	%06o	000014	
2진 %b	12	%b	1100	

타입	인수	포맷 문자열	결과 문자열	주석
상대 시간 %t	91.80	%.2t	01:31.80	LabVIEW는 경과 시간을 완전한 주 %W, 날 %D, 시간 %H, 분 %M, 초 %S, 초 이하 단위 %u로 포맷합니다. 시간 포맷 문자열에 대한 추가적인 정보는 시간 포맷 문자열을 위한 포맷 코드를 참조하십시오.
91.8	%<시간:%H 분:%M 초:%S>t	시간:00 분:01 초:31		
절대 시간 %T	2001년 1월 1일 오전 00:00:00.000 (세계시)	%<%.3X %x>T	2001년 01월 01일 오전 12:00:00.000	%T는 절대 시간을 지정합니다. < and > 괄호 인디케이터에 포함되어 있는 정보는 절대 시간을 포맷하려는 방법을 나타냅니다. 구두점을 포함한 이 포맷은 컴퓨터의 지역 설정에 따라 변경됩니다. 시간은 컴퓨터에 설정된 표준 시간대에 따라 변경됩니다. 시간 포맷 문자열에 대한 추가적인 정보는 시간 포맷 문자열을 위한 포맷 코드를 참조하십시오.

타입	인수	포맷 문자열	결과 문자열	주석
2001년 1월 1일 오전 00:00:00.000 (세계시)	%<%Y.%m.%d>T	2001년 1월 1일		
2001년 1월 1일 오전 00:00:00.000 (세계시)	%^<%.3X %x>T	2000년 12월 31일 오전 06:00:00.000	^을 추가하여 시간을 세계시로 포맷합니다.	
문자열 %s	Smith John	이름: %s, %s.	이름: Smith, John.	문자열은 [문자열로 포맷] 함수를 사용해야만 유효합니다. 두 번째 예제에서, 포맷 문자열은 Hello, World 문자열에서 적어도 6개의 문자 사용을 지정하고 총 문자열의 길이가 10과 같도록 공백으로 채웁니다.
Hello, World	문자열: %10.6s	문자열: Hello,		

타입	인수	포맷 문자열	결과 문자열	주석
변수 순서 %\$	Smith John	%2\$s %1\$s	John Smith	이 예제에서, 스미스는 첫번째 변수이고 존은 두번째 변수입니다. 특정 변수의 포맷 문자열에 \$ 구문 원소를 사용하는 경우, \$ 구문 원소가 앞에 오지 않는 이후의 구문 원소는 다음 변수에 적용됩니다. 변수에 \$ 구문 원소를 사용하는 경우, 문자열의 다른 변수에 \$ 구문 원소를 사용하여 더 쉽게 읽을 수 있도록 합니다.
퍼센트 문자 %%	없음	%%	%	%%는 단일 %를 포맷합니다. 이 포맷을 사용하면 문자열에 실제 퍼센트 문자를 포함시킬 수 있습니다. 이는 [문자열로 포맷] 또는 [문자열로부터 스캔] 함수에 터미널을 추가하지 않습니다.

LabVIEW의 그래프와 차트

그래프와 차트

데이터를 수집 또는 생성한 후에 그래픽 방식으로 데이터를 디스플레이하기 위해 그래프 또는 차트를 사용합니다.

그래프와 차트는 데이터를 디스플레이하고 업데이트하는 방법에서 차이가 납니다. 그래프가 포함된 vi는 일반적으로 배열로 데이터를 수집한 후, 이 데이터를 그래프로 플롯합니다. 이 과정은 데이터를 먼저 저장한 후 데이터의 플롯을 생성하는 스프레드시트와 비슷합니다. 데이터가 플롯되면, 그래프는 이전에 플롯된 데이터를 버리고 새로운 데이터만 디스플레이합니다. 일반적으로 그래프는 연속적으로 데이터를 수집하는 속도가 빠른 프로세스에 사용됩니다.

그와 대조적으로, 차트는 새 데이터 포인트를 이미 그려진 데이터 뒤에 덧붙여서 디스플레이하고 히스토리를 생성합니다. 차트에서는 현재 읽은 값 또는 측정된 값을 이전에 수집된 데이터와 함께 볼 수 있습니다. 차트에서 디스플레이할 수 있는 것보다 더 많은 데이터 포인트가 추가되면, 차트가 스크롤되며 오래된 데이터 포인트는 왼쪽으로 사라지고 새로운 데이터 포인트가 오른쪽에 추가됩니다. 일반적으로 차트는 초당 아주 적은 수의 데이터 포인트만이 플롯에 추가되는 느린 프로세스에 사용됩니다.



관련 정보

[그래프와 차트의 종류](#) (433 페이지 참조)

[그래프와 차트의 종류](#) (433 페이지 참조)

[그래프와 차트 사용자 정의](#) (449 페이지 참조)

[그래프 또는 차트 설정하기](#) (1146 페이지 참조)

[그래프 또는 차트에서 X축과 Y축 설정하기](#) (1146 페이지 참조)

[데이터와 이미지 반출하기](#) (1132 페이지 참조)

그래프와 차트의 종류

LabVIEW는 다음 타입의 그래프와 차트를 가집니다:

- 웨이브폼 차트와 그래프—일정한 속도로 수집된 데이터를 디스플레이합니다.
- XY 그래프—일정하지 않은 속도로 얻은 데이터와 여러 값을 가진 함수를 디스플레이합니다.
- 강도 차트와 그래프—3차원 값을 디스플레이하기 위해 색을 이용하여 2D 플롯에서 3D 데이터를 디스플레이합니다.
- 디지털 웨이브폼 그래프—펄스나 디지털 라인의 그룹으로 데이터를 디스플레이합니다.
- 혼합된 신호 그래프—웨이브폼 그래프, XY 그래프, 디지털 웨이브폼 그래프에 의해 받는 데이터 타입을 디스플레이합니다. 또한 이 데이터 타입의 모든 조합이 있는 모든 클러스터도 받습니다.
- 2D 그래프—2D 프런트패널 플롯에 2D 데이터를 디스플레이합니다.
- 3D 그래프—3D 프런트패널 플롯에 3D 데이터를 디스플레이합니다.



노트

3D 그래프 컨트롤은 LabVIEW Full과 Professional Development Systems에서만 제공됩니다.

- ActiveX 3D 그래프—프론트패널 **ActiveX 객체**의 3D 플롯에 3D 데이터를 디스플레이합니다.



노트 ActiveX 3D 그래프 컨트롤은 Windows의 LabVIEW Full과 Professional Development Systems에서만 지원됩니다.

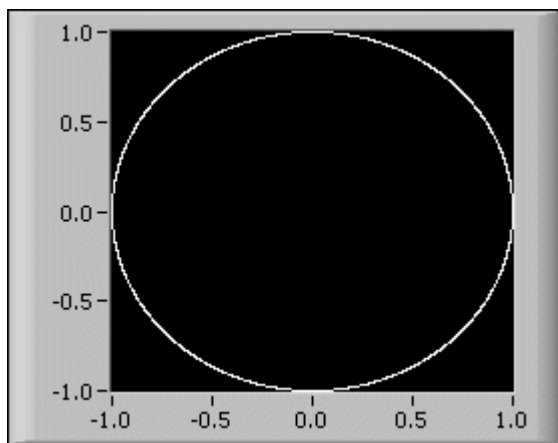
그래프와 차트에 대한 예제는 labview\examples\Controls and Indicators\Graphs and Charts 디렉토리를 참조하십시오.

XY 그래프

XY 그래프는 구형이나 가변 타임 베이스를 갖는 웨이브폼과 같은, 여러값의 함수를 플롯하는 일반적인 목적을 가진 직각좌표 그래픽 객체입니다. XY 그래프는 샘플링 속도의 균일도에 상관없이 모든 포인트 세트를 표시합니다.

또한, XY 그래프에서 나이퀴스트 평면, 니콜스 평면, S 평면, Z 평면을 **디스플레이**할 수 있습니다. 이들 평면의 라인과 라벨은 직각좌표 라인과 같은 색이며 선도 라벨 폰트를 수정할 수 없습니다.

다음 프론트패널은 XY 그래프의 예제를 보여줍니다.



XY 그래프는 임의 개수의 포인트를 가진 플롯을 디스플레이할 수 있습니다. XY 그래프는 데이터 호환성의 불편을 최소화하도록 여러 데이터 타입을 수용합니다.

XY 그래프에서 단일 플롯 디스플레이하기

XY 그래프는 단일 플롯 XY 그래프에서 세가지 데이터 타입을 수용합니다. XY 그래프는 x 배열과 y 배열을 포함하는 클러스터를 수용합니다.

또한, XY 그래프는 x 값과 y 값을 가지는 클러스터인 포인트의 배열을 수용합니다. 또한, XY 그래프는 실수 부분을 x축에 플롯하고 허수부분을 y축에 플롯하여 복소수 데이터의 배열을 수용합니다.

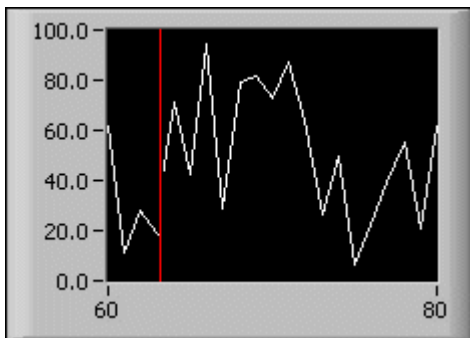
XY 그래프에서 여러 개의 플롯 디스플레이하기

XY 그래프는 여러 개의 플롯을 디스플레이하기 위해 세가지 데이터 타입을 수용합니다.. XY 그래프는 x 배열과 y 배열을 가진 클러스터인 플롯의 배열을 입력 받을 수 있습니다.

또한, XY 그래프는 포인트의 배열인 플롯의 클러스터 배열을 수용합니다. 포인트는 x 값과 y 값을 가지는 클러스터입니다. 뿐만 아니라, 멀티 플롯 XY 그래프는 플롯으로 이루어진 클러스터의 배열을 받을 수 있습니다. 이때 한 플롯은 복소수 데이터의 배열로 나타나며, 실수 부분은 x축에 플롯되고 허수 부분은 y축에 플롯됩니다.

웨이브폼 차트

웨이브폼 차트는 일정한 속도로 수집되는 데이터를 하나 또는 그 이상의 플롯으로 나타내는 특수한 타입의 숫자형 인디케이터입니다. 다음 프런트패널은 웨이브폼 차트의 예를 보여줍니다.



웨이브폼 차트는 이전 업데이트에서 데이터 히스토리, 또는 버퍼를 유지합니다. 웨이브폼 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차트 히스토리 길이를 선택하여 버퍼를 설정합니다. 웨이브폼 차트의 기본 **차트 히스토리 길이**는 1,024 데이터 포인트입니다. 차트에 데이터를 보내는 주기가 차트를 얼마나 자주 업데이트하는지를 결정합니다.

웨이브폼 그래프에서 단일 플롯 디스플레이하기

한 번에 하나 또는 여러 개의 값을 차트로 전달하는 경우, LabVIEW는 차트에서 데이터를 포인트로 해석하고 x=0에서 시작하여 x 인덱스를 하나씩 증가시킵니다. 차트는 이러한 입력값을 단일 플롯의 새 데이터로 간주합니다.

웨이브폼 차트는 웨이브폼의 데이터, 시작 시간, 델타 t를 전달하는 **웨이브폼 데이터 타입**을 수용합니다. [웨이브폼 만들기 (아날로그 웨이브폼)] 함수를 사용하여 차트의 x축의 시간을 플롯하고 자동적으로 차트의 x스케일에서 마커 사이의 정확한 간격을 사용합니다. t0과 단일 원소 y 배열을 지정하는 웨이브폼은 각 데이터 포인트가 고유의 타임스탬프를 갖기 때문에 균일하게 샘플되지 않은 데이터를 플롯하는데 유용합니다.

웨이브폼 차트의 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts\Waveform Graphs and Charts.lvproj를 참조하십시오.

웨이브폼 차트에서 여러 개의 플롯 디스플레이하기

웨이브폼 차트에 여러 개의 플롯 데이터를 전달하기 위해서, 데이터를 스칼라 숫자값의 클러스터로 묶을 수 있습니다. 여기서 각 숫자값은 플롯의 단일 포인트를 나타냅니다.

단일 업데이트에서 플롯당 여러 포인트를 통과시키려면, 차트에 숫자값의 클러스터 배열을 연결합니다. 각 숫자는 각 플롯의 단일 y값 포인트를 나타냅니다.

웨이브폼 데이터 타입을 사용하여 웨이브폼 차트에 여러 개의 플롯을 생성할 수 있습니다. [웨이브폼 만들기] 함수를 사용하여 차트의 x축의 시간을 플롯하고 자동적으로 차트의 x스케일에서 마커 사이에 정확한 간격을 사용합니다. 각각 t_0 와 단일 원소 y배열을 지정하는 웨이브폼의 1D 배열은 균일하게 샘플되지 않은 데이터의 플롯에 유용합니다. 각 데이터 포인트가 고유의 타임스탬프를 가지기 때문입니다.

실행 시점까지 몇 개의 플롯을 디스플레이할지 알 수 없는 경우, 또는 여러 개의 플롯에 다수 포인트를 단일 업데이트로 전달하고 싶을 때, 차트에 숫자값의 2D 배열 또는 웨이브폼을 연결합니다. 기본적으로 웨이브폼 차트는 배열의 각 열을 단일 플롯으로 다룹니다. 차트에 2차원 배열 데이터 타입을 연결한 다음, 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 배열 전치를 선택하여 배열의 각 행을 단일 플롯으로 다룹니다.

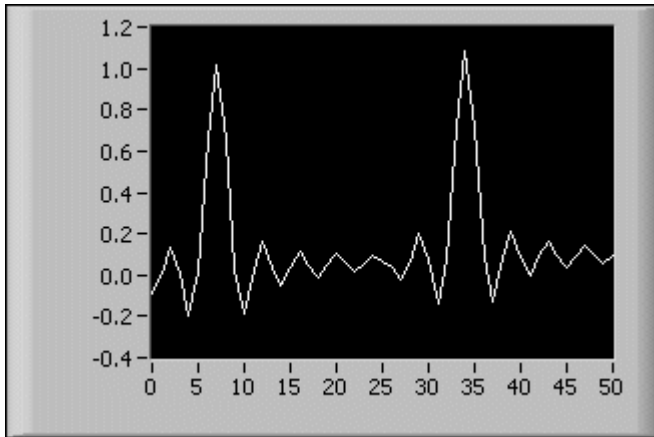
웨이브폼 차트의 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts\Waveform Graphs and Charts.lvproj를 참조하십시오.

웨이브폼 데이터 타입

웨이브폼 데이터 타입은 웨이브폼의 데이터, 시작 시간, 델타 t를 전달합니다. 웨이브폼은 [웨이브폼 만들기] 함수로 생성할 수 있습니다. 웨이브폼을 수집 또는 분석하는 대부분의 VI와 함수는 기본적으로 웨이브폼 데이터를 받고 반환합니다. 웨이브폼 데이터를 **웨이브폼 그래프** 또는 **웨이브폼 차트**에 연결하면, 그래프나 차트는 자동적으로 웨이브폼의 데이터, 시작 시간, 델타 x를 기반으로 웨이브폼을 플롯합니다. 웨이브폼 데이터의 배열을 웨이브폼 그래프나 차트에 연결하면, 그래프와 차트는 자동으로 모든 웨이브폼을 플롯합니다.

웨이브폼 그래프

웨이브폼 그래프는 균일하게 샘플된 **하나** 또는 **그 이상**의 측정치에 대한 플롯을 디스플레이합니다. 웨이브폼 그래프는 $y=f(x)$ 에서 보는 대로 시간에 따라 변하는 웨이브폼과 같이 x축을 따라 포인트가 균일하게 분포하는 단일 값 함수만을 플롯합니다. 다음 프런트 패널은 웨이브폼 그래프의 예제를 보여줍니다.



웨이브폼 그래프는 임의 개수의 포인트를 가진 플롯을 디스플레이 할 수 있습니다. 그래프는 데이터 호환성의 불편을 최소화하도록 여러 데이터 타입을 수용합니다.



노트 **디지털 웨이브폼 그래프**를 사용하여 디지털 데이터를 디스플레이합니다.

웨이브폼 그래프에서 단일 플롯 디스플레이하기

웨이브폼 그래프는 단일 플롯 웨이브폼 그래프에서 여러 데이터 타입을 받습니다. 그래프는 단일 배열 값을 수용하고 이 배열을 그래프의 포인트로 해석합니다. 그리고 $x=0$ 에서 시작하여 하나씩 x 인덱스가 증가합니다. 그래프는 초기 x 값, 델타 x 값, y 데이터의 배열의 클러스터를 받습니다. 또한, 그래프는 웨이브폼의 데이터, 시작 시간, 델타 t 를 전달하는 **웨이브폼 데이터 타입**을 수용합니다.

또한 웨이브폼 그래프는 익스프레스 VI와 함께 사용하기 위해 **다이나믹 데이터 타입**을 받을 수 있습니다. 다이나믹 데이터 타입은 신호와 연관된 데이터 및 신호의 이름 또는 데이터가 수집된 날짜와 시간 등의 신호와 연관된 정보를 제공하는 속성을 포함합니다. 속성은 신호가 웨이브폼 그래프에 어떻게 나타날지를 지정합니다. 다이나믹 데이터 타입이 단일 숫자값을 가질 때, 그래프는 단일 값을 플롯하고 자동적으로 플롯 범례와 x 스케일 타임스탬프를 포맷합니다. 다이나믹 데이터 타입이 단일 채널을 가질 때, 그래프는 모든 웨이브폼을 플롯하고 자동적으로 플롯 범례와 x 스케일 타임스탬프를 포맷합니다.

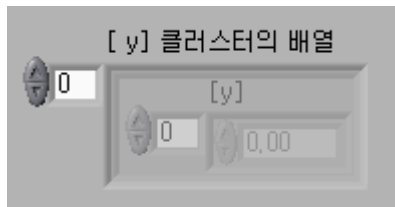
웨이브폼 그래프가 받는 데이터 타입의 예는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts 디렉토리의 Waveform Graph Data Type VI를 참조하십시오.

웨이브폼 그래프에서 여러 개의 플롯 디스플레이하기

웨이브폼 그래프는 여러 개의 플롯을 디스플레이하기 위해 여러 데이터 타입을 수용합니다. 웨이브폼 그래프는 배열의 각 행이 하나의 단일 플롯 값을 나타내는 2D 배열을 받을 수 있습니다. 그래프는 해당 데이터를 그래프의 포인트로 해석합니다. 그리고 $x=0$ 에서 시작하여 하나씩 x 인덱스가 증가합니다. 그래프에 2차원 배열 데이터 타입을 연결한 다음, 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 배열 전치를 선택하여 배열의 각 열을 플롯으로 처리합니다. 이는 특히 DAQ 디바이스에서 여러 채널을 샘플링하는 경우에 유용합니다. DAQ 디바이스는 여러 채널이 각각 별도의 열로 저장된 2D 배열로 데이터를 반환할 수 있기 때문입니다.

웨이브폼 그래프는 초기 x 값, 델타 x 값, y 데이터의 2D 배열로 이루어진 클러스터도 받을 수 있습니다. 그래프는 y 를 그래프의 포인트로 해석하고 초기 x 값에서 시작하여 델타 x 만큼씩 x 인덱스를 증가시킵니다. 이 데이터 타입은 동일한 속도로 샘플링되는 여러 개의 신호를 나타낼 때 유용합니다.

웨이브폼 그래프는 클러스터를 포함하는 배열이 있는 플롯 배열을 받을 수 있습니다. 각 클러스터는 y 데이터를 포함하는 1D 배열을 가집니다. 내부 배열은 플롯 안의 포인트를 나타내며, 외부 배열은 각 플롯마다 하나의 클러스터를 갖습니다. 다음 프런트패널은 y 클러스터의 이와 같은 배열을 보여줍니다.



원소의 개수가 각 플롯마다 다르면 2D 배열 대신에 플롯 배열을 사용합니다. 예를 들어, 각 채널에서 서로 다른 시간 동안 여러 채널에서 데이터를 샘플링한 경우, 행의 원소 개수가 일정해야 하는 2D 배열 대신 이 데이터 구조를 사용합니다. 클러스터 배열에 있는 내부 배열의 원소 개수는 서로 동일하지 않아도 됩니다.

웨이브폼 그래프는 초기 x 값, 델타 x 값, 클러스터를 포함하는 배열로 이루어진 클러스터를 받을 수 있습니다. 각 클러스터는 y 데이터를 포함하는 1D 배열을 가집니다. [묶기] 함수를 사용하여 배열을 클러스터로 묶고, [배열 만들기] 함수를 사용하여 결과 클러스터를 하나의 배열로 묶습니다. 또한 [클러스터 배열 만들기] 함수를 사용하여 사용자가 지정한 입력을 가지는 클러스터의 배열을 생성할 수 있습니다.

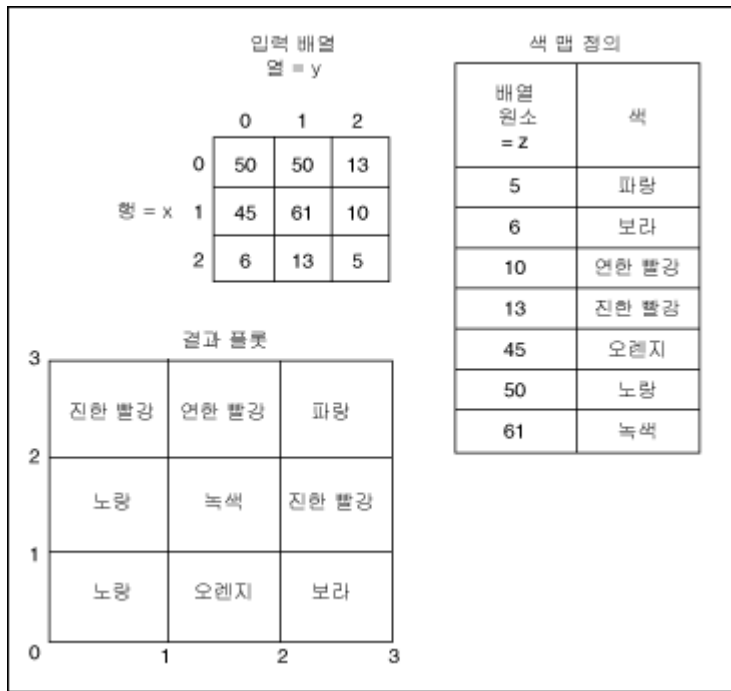
웨이브폼 그래프는 x 값, 델타 x 값, y 데이터 배열로 이루어진 클러스터의 배열을 받을 수 있습니다. 이는 멀티 플롯 웨이브폼 그래프의 가장 일반적인 데이터 타입입니다. 이러한 웨이브폼 데이터 타입을 사용하면 여러 플롯들의 x 스케일에 대해 각각 고유의 시작점 및 증가분을 지정할 수 있기 때문입니다.

또한 웨이브폼 그래프는 익스프레스 VI와 함께 사용하기 위해 **다이내믹 데이터 타입**을 받을 수 있습니다. 다이내믹 데이터 타입은 신호와 연관된 데이터 및 신호의 이름 또는 데이터가 수집된 날짜와 시간 등의 신호와 연관된 정보를 제공하는 속성을 포함합니다. 속성은 신호가 웨이브폼 그래프에 어떻게 나타날지를 지정합니다. 다이내믹 데이터 타입이 여러 채널을 가질 때, 그래프는 각 채널에 대한 플롯을 디스플레이하고 자동적으로 플롯 범례와 x 스케일 타임스탬프를 포맷합니다.

강도 차트

강도 그래프와 차트

강도 그래프와 차트를 사용하여 직각 평면에 색 블록을 놓아서 3D 데이터를 2D 플롯에서 디스플레이합니다. 예를 들어, 강도 그래프 또는 차트를 사용하여, 온도 패턴이나 고도를 등급으로 나타내는 지형과 같은 패턴화된 데이터를 디스플레이할 수 있습니다. 강도 그래프와 차트는 숫자의 3D 배열을 받습니다. 배열에서 각 숫자는 특정 색을 나타냅니다. 2D 배열 원소의 인덱스는 해당 색에 대한 플롯 위치를 지정합니다. 다음 그림은 강도 차트 작업에 대한 개념을 설명해 줍니다.



데이터의 행은 그래프 또는 차트의 새로운 열로 디스플레이에 전달됩니다. 행을 디스플레이 상에서 행으로 나타내려면 그래프 또는 차트에 2D 배열 데이터 타입을 연결하고, 그래프 또는 차트에서 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 배열 전치를 선택합니다.

배열의 인덱스는 색 블록의 왼쪽 밑 정점에 대응합니다. 색 블록은 배열 인덱스로 정의되는 두 포인트 사이의 영역인 단위 영역을 가지고 있습니다. 강도 그래프 또는 차트는 최대 256개의 비연속적인 색을 디스플레이할 수 있습니다.

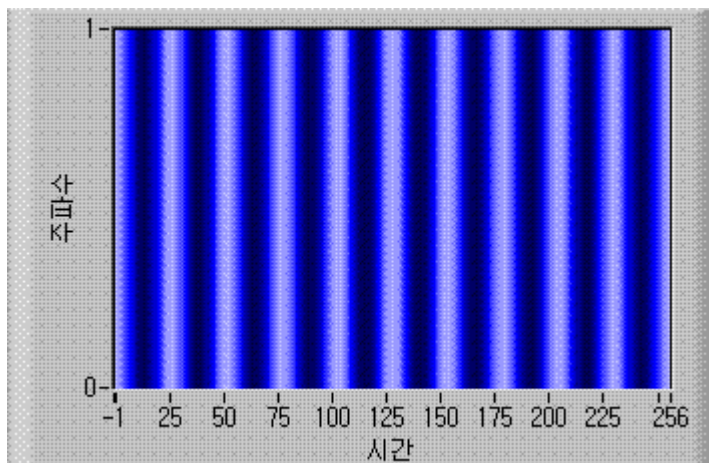
강도 그래프와 차트에 대한 예제는 `labview\examples\Controls and Indicators\Graphs and Charts\Intensity Graphs and Charts\Intensity Graphs and Charts.lvproj`를 참조하십시오.

관련 예제 찾기

강도 차트

강도 차트에 데이터 블록을 플롯한 후에, 직각 평면의 원점은 마지막 데이터 블록의 오른쪽으로 이동합니다. 차트가 새 데이터를 처리할 때, 새 데이터 값은 이전 데이터 값의 오른쪽에 나타납니다. 차트 디스플레이가 꽉 차면, 가장 오래된 데이터 값은 차트의 왼쪽으로 스크롤되어 사라집니다. 이런 동작 특성은 [스트립 차트](#)와 유사합니다.

다음 프런트패널은 강도 차트의 예제를 보여줍니다.



강도 차트는 **스케일 범례**와 **그래프 팔레트** 등과 같은 **웨이브폼 차트**의 옵션 부분을 상당히 많이 공유하며, 이러한 부분은 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목을 선택하여 보이거나 감출 수 있습니다. 추가적으로, 강도 차트는 3차원의 색을 가지고 있기 때문에, 색 램프 컨트롤과 유사한 스케일이 **색 값의 맵핑**과 범위를 지정합니다.

웨이브폼 차트와 마찬가지로, 강도 차트는 이전 업데이트에서 데이터 히스토리, 혹은 버퍼를 보존합니다. 웨이브폼 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차트 히스토리 길이를 선택하여 버퍼를 설정합니다. 강도 차트에서 히스토리 버퍼 크기의 기본 설정은 128 데이터 포인트입니다. 강도 차트를 디스플레이하기 위해서 많은 메모리가 필요할 수 있습니다.



팁 그래프와는 다르게, 차트는 이전에 쓴 데이터의 히스토리를 갖습니다. 차트가 연속적으로 실행될 때, 차트 히스토리는 커지고 추가적인 메모리 공간을 필요로 하게 됩니다. 차트 히스토리가 가득 찰 때까지 지속되는데, 이후 LabVIEW는 더이상의 메모리를 차지하지 않게 됩니다. LabVIEW는 VI가 재시작할 때 자동으로 차트 히스토리를 지우지 않습니다. 프로그램이 실행되는 동안, 차트 히스토리를 지울 수 있습니다. 이를 위해, 차트에 대한 히스토리 데이터 속성 노드에 빈 배열을 작성하십시오.

강도 그래프

강도 그래프는 이전 데이터 값을 유지하지 않고 **업데이트 모드**를 포함하지 않는 부분을 제외하면, **강도 차트**와 동일하게 작동합니다. 강도 그래프로 새 데이터 값이 전달될 때마다 새 데이터는 이전 데이터 값을 대체합니다. 다른 그래프와 같이 강도 그래프에는 **커서**가 있습니다. 각 커서는 그래프에서 특정 포인트의 x, y, z 값을 디스플레이합니다.

강도 그래프와 차트에서 색 맵핑 사용하기

강도 그래프 또는 **강도 차트**는 색을 사용하여 2D 플롯에서 3D 데이터를 디스플레이합니다. 강도 그래프와 차트에서 색 맵핑을 설정하려면, 그래프 또는 차트의 색 스케일을 설정해야 합니다. 색 스케일은 각각의 숫자값과 대응하는 디스플레이 색을 가지는 최소 두 개의 마커로 구성되어 있습니다. 강도 그래프와 차트에서 디스플레이되는 색은 지정된 색과 연관있는 숫자값으로 대응됩니다. 색 맵핑은 임계점 값을 넘는 플롯 데이터 등에서 데이터 범위를 시각적으로 지정하는데 유용합니다.

색 램프 숫자 컨트롤에서 색을 정의하는 것과 같은 방법으로 강도 그래프와 차트에서 **색 매핑을 대화식으로 설정**할 수 있습니다.

두 가지 방법으로 프로퍼티 노드를 사용하여 프로그램적으로 강도 그래프와 차트의 색 매핑을 설정할 수 있습니다. 일반적으로, 프로퍼티 노드에서는 데이터 값을 색으로 지정합니다. 이와 같은 방법으로 z 스케일의 Z 스케일: 마커 값 프로퍼티를 지정합니다. 이 프로퍼티는 클러스터의 배열로 구성되며, 각 클러스터는 이 값을 디스플레이하기 위한 숫자 리미트 값과 대응되는 색을 가지고 있습니다. 이런 방법으로 색 매핑을 지정하면, 높은 쪽으로 범위를 벗어났을 때에는 색을 지정하기 위해 z 스케일의 Z 스케일: 높은 색 프로퍼티를 사용하고 낮은 쪽으로 범위를 벗어났을 때에는 z 스케일의 Z 스케일: 낮은 색 프로퍼티를 사용합니다. 강도 그래프와 차트는 합계 254색으로 제한되며, 위쪽과 아래쪽의 범위 바깥 색을 포함하면 총 256색입니다. 254색보다 많게 지정하면, 강도 그래프와 차트는 지정한 색상을 조절하여 254색 테이블을 만듭니다.

강도 그래프에 비트맵을 디스플레이하려면, 색 테이블 프로퍼티를 사용하여 색 테이블을 지정합니다. 이와 같은 방법으로, 배열에 256색까지 지정합니다. 차트에 전달되는 데이터는 강도 차트의 색 스케일에 기반하여 색 테이블의 인덱스에 매핑됩니다. 만일 색 스케일의 범위가 0과 100이라면, 0은 인덱스 1로 매핑되고, 100은 인덱스 254로 매핑되며, 중간 값은 1과 254사이로 지정됩니다. 0보다 작은 값은 인덱스 0, 그리고 100보다 큰 값은 인덱스 255로 매핑됩니다.



노트 강도 그래프와 차트에서 디스플레이할 수 있는 색은 비디오 카드에서 디스플레이할 수 있는 정확한 색과 색의 수로 제한됩니다. 또한 모니터가 제공하는 색상의 수로 제한됩니다.

색 매핑에 대한 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Intensity Graphs and Charts에서 Modifying the Color Table VI를 참조하십시오.

디지털 웨이브폼 그래프

디지털 웨이브폼 그래프를 사용하여 디지털 데이터를 디스플레이합니다. 특히 타이밍 다이어그램이나 논리 분석기와 함께 사용할 수 있습니다.

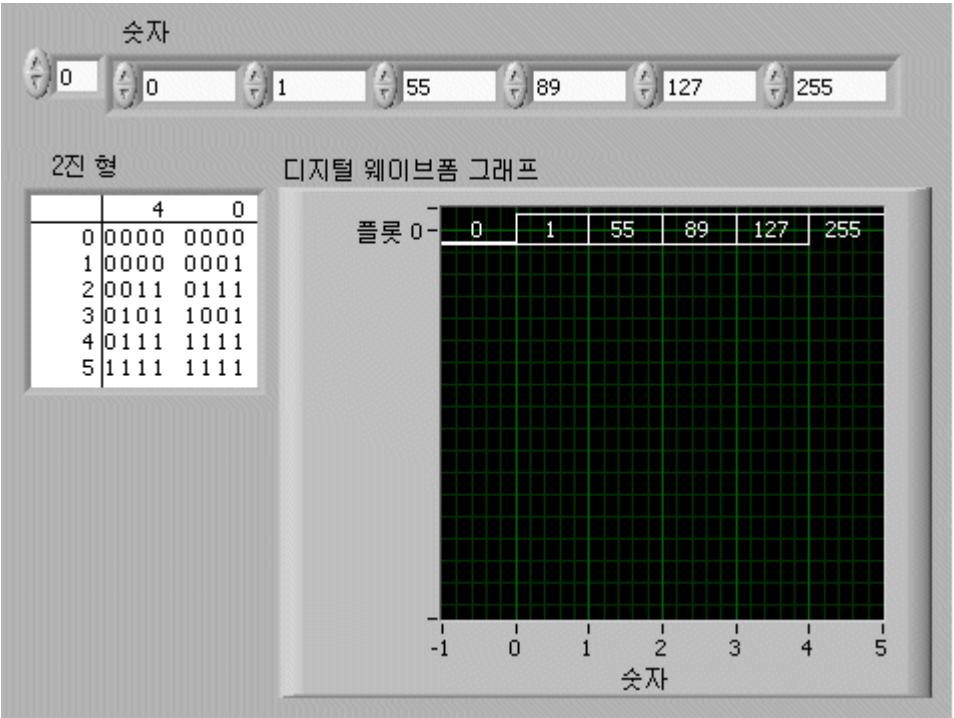
디지털 웨이브폼 그래프는 **디지털 웨이브폼 데이터 타입**, **디지털 데이터 타입**, 이 데이터 타입들의 배열을 입력으로 받을 수 있습니다. 기본으로, 디지털 웨이브폼 그래프는 데이터를 플롯 영역에 디지털 라인과 버스로 디스플레이합니다. **디지털 웨이브폼 그래프를 사용자 정의**하여 디지털 버스, 디지털 라인, 또는 디지털 버스와 디지털 라인의 조합을 디스플레이합니다. 배열의 각 원소가 버스를 나타내는 디지털 데이터 배열을 연결하면, 디지털 웨이브폼 그래프는 배열의 각 원소를 다른 라인으로 플롯합니다. 이 때 순서는 배열 원소를 그래프에 그리는 순서입니다.

플롯 범례의 트리 보기에서 디지털 버스를 확장하거나 축소하려면 디지털 버스 왼쪽의 확장/축소 기호를 클릭하십시오. 플롯 범례의 트리 보기에서 디지털 버스를 확장하거나 축소하면 그래프의 플롯 영역에 있는 버스도 확장되거나 축소됩니다. 플롯 범례가 표준 보기일 때 디지털 버스를 확장하거나 축소하려면 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 Y 스케일 >> 디지털 버스 확장을 선택합니다.



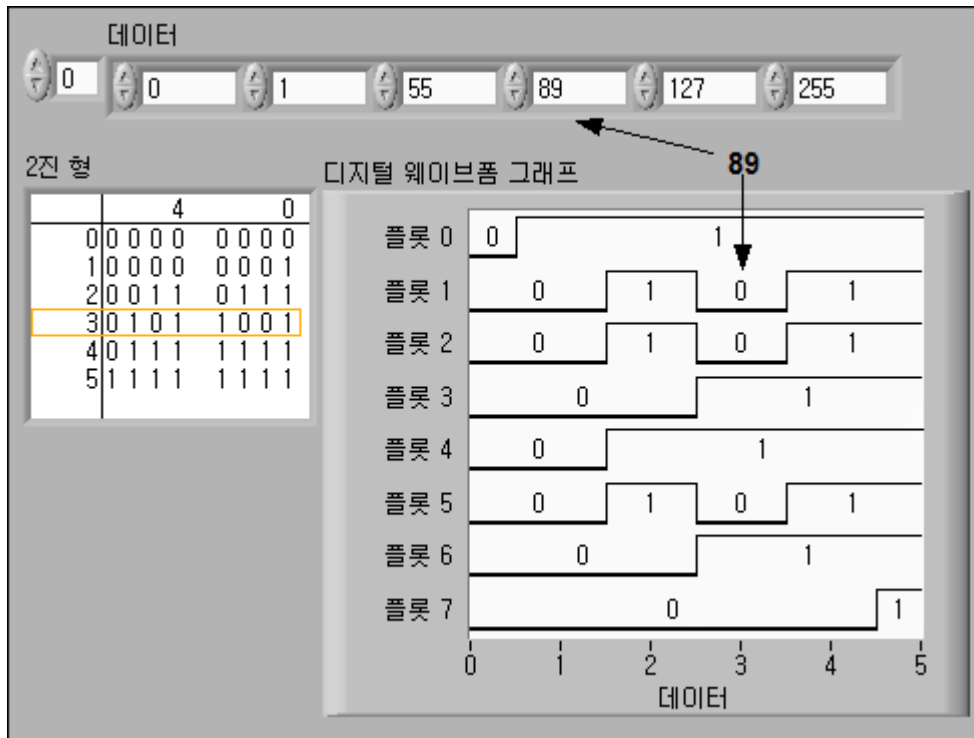
노트 Y 스케일>>디지털 버스 확장은 라인 있는 버스 보이기를 비활성화하고 플롯 범례가 표준 보기인 경우에만 사용 가능합니다. 라인 있는 버스 보이기를 비활성화하려면, 플롯 범례를 표준 보기로 변경하고, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 라인 있는 버스 보이기를 선택하여 메뉴 아이템 옆의 확인 표시를 제거합니다.

다음 프런트패널의 디지털 웨이브폼 그래프는 디지털 데이터를 버스로 플롯합니다. VI는 숫자 배열의 숫자를 디지털 데이터로 변환하고 숫자의 2진 표현을 디지털 데이터인 디케이터의 2진 형에 디스플레이합니다. 디지털 그래프에서 숫자 0은 모든 비트 값이 0이므로 상위 라인 없이 나타납니다. 숫자 255는 모든 비트 값이 1이므로 아래쪽 선 없이 나타납니다.



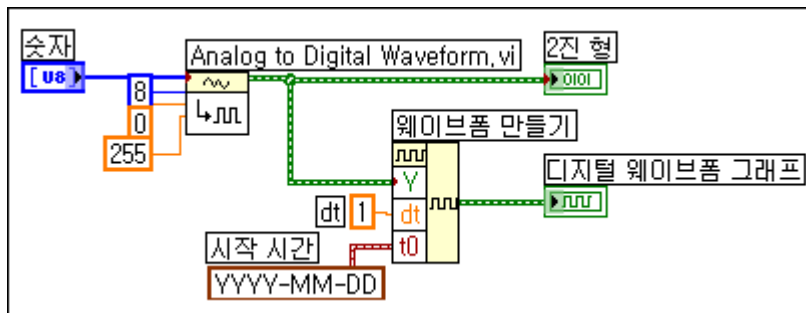
디지털 데이터의 각 샘플을 플롯하려면 y스케일을 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 디지털 버스 확장을 선택합니다. 각 플롯은 디지털 패턴에서 다른 비트를 나타냅니다. 디지털 웨이브폼 그래프에 플롯된 데이터의 모양을 사용자가 정의할 수 있습니다.

다음 프런트패널에서 디지털 웨이브폼 그래프는 숫자 배열에 있는 여섯 개의 숫자를 디스플레이합니다.



2진 형 디지털 데이터 인디케이터는 숫자를 2진 형으로 디스플레이합니다. 테이블의 각 열은 하나의 비트를 나타냅니다. 예를 들어, 숫자 89는 7비트 메모리(7열의 0은 사용되지 않은 비트를 나타냅니다)가 요구됩니다. 디지털 웨이브폼 그래프의 포인트 3은 7 비트를 플롯하여 번호 89를 나타내고, 0의 값을 플롯하여 플롯 7에서 사용되지 않는 8번째 비트를 나타냅니다. 데이터는 오른쪽에서 왼쪽으로 읽습니다.

다음 vi는 숫자 배열을 디지털 데이터로 변환하고 웨이브폼 만들기 함수를 이용하여 시작 시간, 델타 t , 그리고 **디지털 데이터 컨트롤**에 입력한 숫자를 모으고 디지털 데이터를 디스플레이합니다.



디지털 웨이브폼 그래프의 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Digital Waveform Graph\Digital Waveform Graph.lvproj를 참조하십시오.

디지털 웨이브폼 데이터 타입

디지털 웨이브폼 데이터 타입은 시작 시간, 델타 x , 데이터, 그리고 디지털 웨이브폼의 속성으로 구성됩니다. [디지털 웨이브폼 만들기] 함수를 사용하여 디지털 웨이브폼을 생성할 수 있습니다. 디지털 웨이브폼 데이터를 **디지털 웨이브폼 그래프**에 연결하면, 그래프

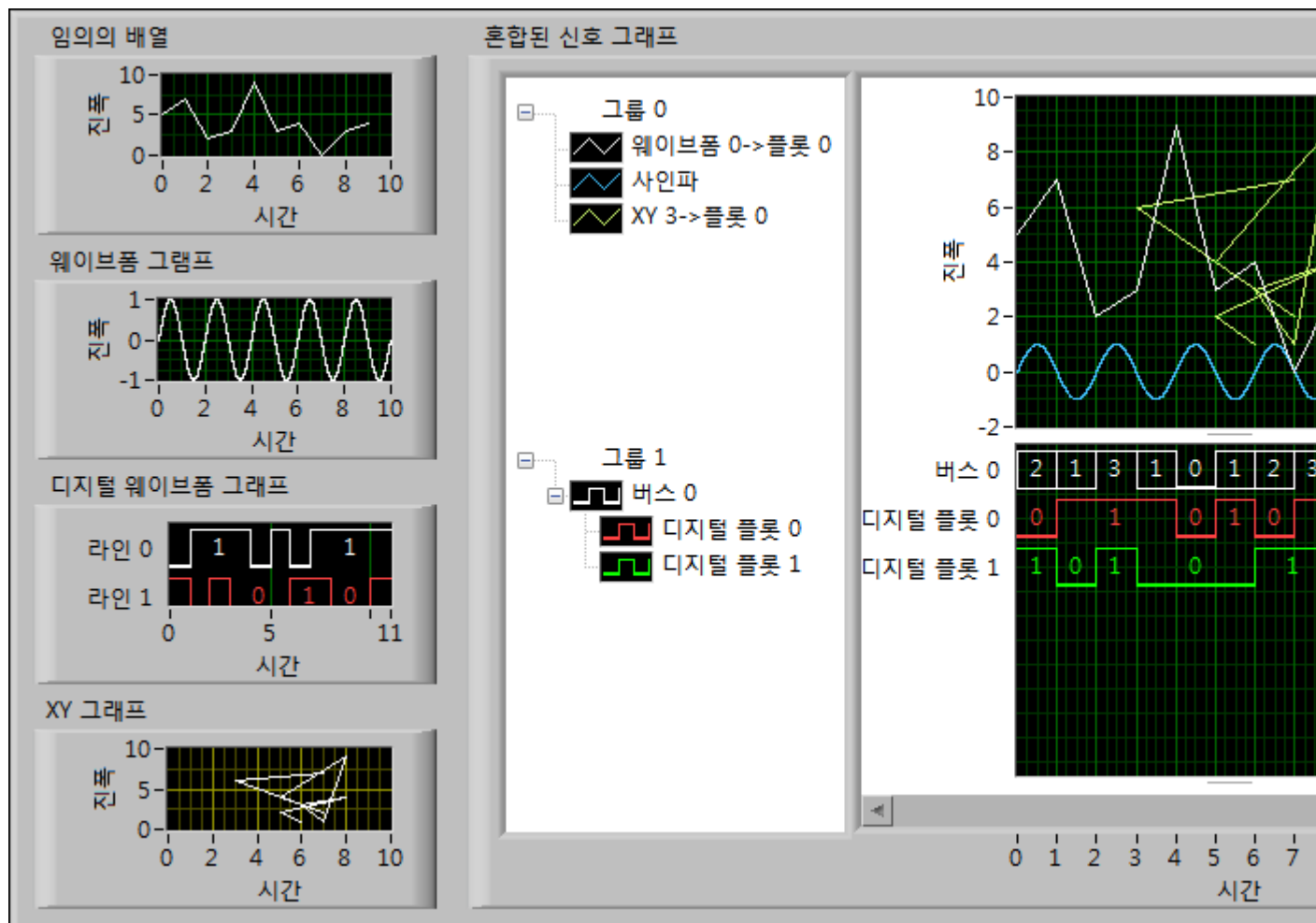
는 디지털 웨이브폼의 시간 정보와 데이터에 기반한 웨이브폼을 자동적으로 플롯합니다. 디지털 웨이브폼 데이터를 디지털 데이터 인디케이터에 연결하여 디지털 웨이브폼의 샘플과 신호를 봅니다.

혼합된 신호 그래프

혼합된 신호 그래프는 아날로그와 디지털 데이터 모두를 디스플레이할 수 있으며, **웨이브폼 그래프**, **XY 그래프** 및 **디지털 웨이브폼 그래프**에 의해 받는 모든 데이터 타입을 받습니다.

혼합된 신호 그래프는 여러 개의 플롯 영역을 가질 수 있습니다. 주어진 플롯 영역은 디지털이나 아날로그 플롯 중 하나만을 디스플레이할 수 있습니다. 이 플롯 영역은 LabVIEW가 그래프에서 데이터를 그리는 부분입니다. 혼합된 신호 그래프는 아날로그와 디지털 데이터에 맞춰서 필요하면 자동적으로 플롯 영역을 생성합니다. 멀티플롯 영역을 혼합된 신호 그래프에 추가할 때, 각 플롯 영역은 그 자체가 y 스케일을 가집니다. 모든 플롯 영역은 공통적인 x 스케일을 공유하므로 디지털과 아날로그 데이터의 여러 신호를 비교할 수 있습니다.

기본으로, 혼합된 신호 그래프는 디지털 데이터를 플롯 영역에 디지털 라인과 버스로 디스플레이합니다. 각 배열 원소가 버스를 나타내는 디지털 데이터의 배열을 연결하면, 혼합된 신호 그래프는 디지털 데이터의 열을 각각 다른 라인으로 플롯합니다. 다음 프런트 패널은 혼합된 신호 그래프의 예제를 보여줍니다.



플롯을 설정하여 혼합된 신호 그래프가 숫자 및 플롯 라인을 디스플레이하는 방식을 변경할 수 있습니다. 또한, 플롯 프로퍼티를 사용하여 디지털 플롯의 숫자의 포맷을 프로그램적으로 설정할 수 있습니다.

혼합된 신호 그래프에서 단일 플롯 디스플레이하기

혼합된 신호 그래프는 단일 플롯 혼합된 그래프에서 **웨이브폼 그래프**, **XY 그래프** 및 **디지털 웨이브폼 그래프**와 같은 데이터 타입을 받습니다.

혼합된 신호 그래프가 받는 데이터 타입의 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Mixed Signal Graph의 Mixed Signal Graph VI를 참조하십시오.

혼합된 신호 그래프에서 여러 개의 플롯 디스플레이하기

혼합된 신호 그래프는 여러 개의 플롯 디스플레이에서 **웨이브폼 그래프**, **XY 그래프** 및 **디지털 웨이브폼 그래프**와 같은 데이터 타입을 받습니다.

플롯 영역은 아날로그나 디지털 데이터 중 한 가지만을 받을 수 있습니다. 데이터를 혼합된 신호 그래프에 연결하면 LabVIEW는 아날로그와 디지털 데이터의 조합에 맞춰서 자동으로 플롯 영역을 생성합니다. 혼합된 신호 그래프에 여러 개의 플롯 영역이 있을 경우, 플롯 영역에서 분리자 막대를 사용하여 각 플롯 영역의 크기를 재조정할 수 있습니다.

혼합된 신호 그래프의 플롯 범례는 세가지 컨트롤로 구성되어 있고 그래프 플롯 영역의 왼쪽에 디스플레이됩니다. 각각의 트리 컨트롤은 하나의 플롯 영역을 나타냅니다. 플롯 영역에는 그룹 X라는 라벨이 붙습니다. 여기서 X는 LabVIEW나 사용자가 그래프에서 플롯 영역을 놓는 순서에 대응하는 숫자입니다. 플롯 범례를 사용하여 **한 곳의 플롯 영역에서 다른 플롯 영역으로 플롯을 이동할 수 있습니다**. 플롯 영역과 플롯 범례 사이에 있는 분리자 막대를 움직여 플롯 범례의 크기를 재조정하거나 숨길 수 있습니다.

혼합된 신호 그래프의 여러 개의 플롯을 디스플레이하는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Mixed Signal Graph의 Mixed Signal Graph VI를 참조하십시오.

2D 그래프

2D 그래프는 x와 y 데이터를 사용하여, **그래프에 포인트를 그리고** 각 포인트를 연결하면서 데이터를 2차원 표면에서 보여줍니다. 2D 그래프를 통해 2 차원 데이터를 XY 그래프로 시각화할 수 있는데, 이는 2D 그래프가 모두 XY 그래프이기 때문입니다. 2D 그래프에 데이터가 표시되는 방식을 수정하려면, 2D 그래프 프로퍼티를 사용하십시오.

프론트패널에 2D 그래프를 추가할 때, LabVIEW는 사용자가 선택하는 2D 그래프에 따라 블록 다이어그램에서 그래프를 보조 VI에 연결합니다. 보조 VI는 입력 데이터 타입을 2D 그래프가 받을 수 있는 일반적인 데이터 타입으로 변환시킵니다. LabVIEW는 다음과 같은 타입의 2D 그래프를 가집니다:

- 콤팩스 플롯—콤팩스 그래프의 중심에서 발산하는 벡터를 그래프에 그립니다.
- 에러 막대 플롯—라인 그래프 위와 아래의 각 포인트에서 에러 막대를 그래프에 그립니다.

- 화살표 플롯—수평 축을 따라 균일한 간격의 포인트에서 발산하는 벡터를 그래프에 그립니다.
- XY 플롯 행렬—산포 그래프의 행과 열을 그래프에 그립니다.

2D 그래프에 데이터를 플롯하는 예제는 `labview\examples\Controls and Indicators\Graphs and Charts\Math Plots - 2D\Math Plots - 2D.lvproj`를 참조하십시오.

관련 예제 찾기

3D 그래프

표면의 온도 분포, 조인트 시간-주파수 분석, 비행기의 움직임과 같은 대부분의 실세계 데이터 세트는 3차원으로 데이터를 시각화할 필요가 있습니다. 3D 그래프는 3차원 데이터를 시각화할 수 있으며, 3D 그래프 프로퍼티를 수정하여 데이터를 나타내는 방법을 바꿀 수 있습니다.

LabVIEW는 다음과 같은 타입의 3D 그래프를 가집니다:

- 산포—두 데이터 세트가 갖는 성향에 대한 통계 및 두 데이터 세트 사이의 관계를 보여줍니다.
- 줄기형—임펄스 응답을 디스플레이하고 분포도에 따라 데이터를 정리합니다.
- 헤성형—데이터 포인트를 따라서 원을 포함하는 **애니메이션 그래프**를 생성합니다.
- 표면—연결하는 표면과 함께 데이터를 그래프로 나타냅니다.
- 컨투어—컨투어 라인이 있는 그래프를 그립니다.
- 메시—열린 공간의 메시 표면 그래프를 그립니다.
- 워터폴—데이터 포인트 아래에 데이터의 표면과 Y 축 영역으로 그래프를 그립니다.
- 화살형—법선 벡터의 플롯을 생성합니다.
- 리본—평행선의 플롯을 생성합니다.
- 막대—수직형 막대의 플롯을 생성합니다.
- 파이—파이 차트를 생성합니다.
- 3D 표면 그래프—3차원 공간 위에 표면을 그립니다.
- 3D 파라미터형 그래프—3차원 공간 위에 파라미터형 표면을 그립니다.
- 3D 라인 그래프—3차원 공간 위에 라인을 그립니다.



노트 3D 그래프 컨트롤은 LabVIEW Full과 Professional Development Systems에서만 제공됩니다.

- ActiveX 3D 표면 그래프—ActiveX 기술을 사용하여 3차원 공간 위에 표면을 그립니다.

- ActiveX 3D 파라미터형 그래프—ActiveX 기술을 사용하여 3차원 공간 위에 파라미터형 표면을 그립니다.
- ActiveX 3D 커브 그래프—ActiveX 기술을 사용하여 3차원 공간 위에 선을 그립니다.

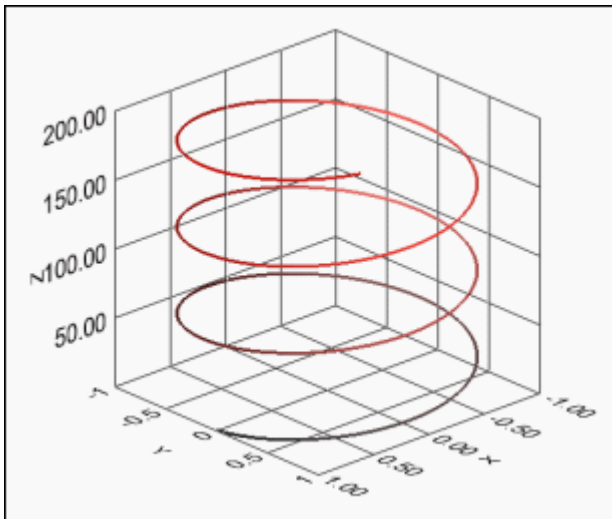


노트 ActiveX 3D 그래프 컨트롤은 Windows의 LabVIEW Full과 Professional Development Systems에서만 지원됩니다.

3D 표면, 3D 파라미터형, 3D 커브 그래프를 제외한 나머지 3D 그래프를 3D 플롯 프로퍼티 대화 상자과 함께 사용하여 3차원적으로 **그래프를 그립니다**. 3D 그래프에 데이터를 플롯하는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Math Plots - 3D 디렉토리를 참조하십시오.

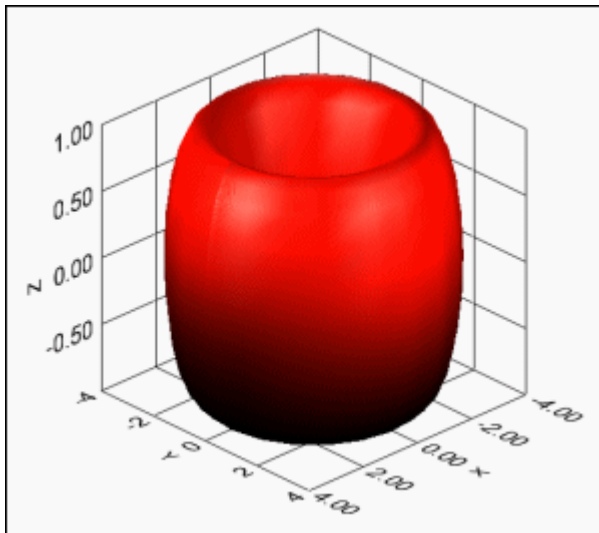
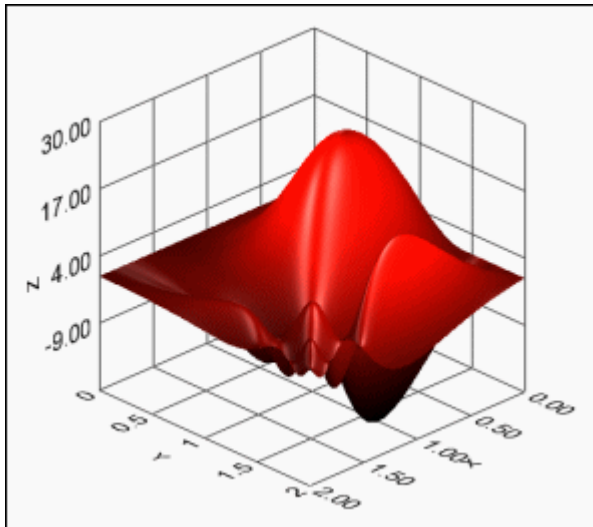
관련 예제 찾기

3D 표면, 3D 파라미터형, 3D 커브 그래프를 3D 그래프 프로퍼티 대화 상자과 함께 사용하여 커브와 표면을 그립니다. 커브는 그래프의 각 포인트들을 가집니다. 각 포인트는 x, y, z 좌표를 가지고 있습니다. VI는 이 포인트들을 하나의 선으로 연결합니다. 커브는 비행기의 비행 경로와 같이 움직이는 객체의 경로를 시각화하는데 이상적입니다. 다음 그림은 3D 라인 그래프의 예를 보여주는데, ActiveX 3D 커브 그래프와 유사합니다.



노트 3D 그래프 프로퍼티 VI를 사용하여 ActiveX 3D 그래프에 커브와 표면을 플롯합니다.

표면 플롯은 x, y, z 데이터를 사용하여 그래프에서 포인트를 플롯합니다. 표면 플롯은 이러한 포인트들을 연결하여, 데이터의 3차원 표면 보기를 형성합니다. 예를 들어, 지형 맵핑을 위해 표면 플롯을 사용할 수 있습니다. 파라미터형 플롯은 파라미터 함수의 파라미터를 사용하여 플롯의 커브를 결정하는 표면 플롯입니다. 기하 입체 객체를 그래프로 나타내는데 파라미터형 플롯을 사용할 수 있습니다. 다음 그림은 3D 표면 그래프와 3D 파라미터형 그래프의 예를 보여줍니다.



프런트패널에 3D 그래프를 추가할 때, LabVIEW는 사용자가 선택하는 3D 그래프에 따라 블록다이어그램에서 그래프를 보조 VI에 연결합니다. 보조 VI는 입력 데이터 타입을 3D 그래프가 받을 수 있는 일반적인 데이터 타입으로 변환시킵니다.

3D 그래프는 렌더링 윈도우에서 그래픽 하드웨어 가속도를 사용하여, 성능상의 이점을 제공합니다. 3D 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 렌더링 윈도우를 선택하여, 렌더링 윈도우에서 3D 그래프를 봅니다.

ActiveX 3D 그래프는 ActiveX 기술과 3D 형을 다루는 VI를 사용합니다. ActiveX 3D 그래프를 선택할 때, LabVIEW는 ActiveX 컨테이너를 3D 그래프 컨트롤을 포함하고 있는 프런트패널에 추가합니다. 또한 LabVIEW는 블록다이어그램에 ActiveX 3D 그래프 컨트롤에 참조를 놓습니다. LabVIEW는 이 참조를 3 개의 3D 그래프 VI중 하나에 연결합니다.

(Windows) ActiveX 3D 그래프는 프런트패널 윈도우에서 그래픽 하드웨어 가속도를 사용합니다.

3D 그래프에 데이터를 플롯하는 예제는 `labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts` 디렉토리를 참조하십시오.

그래프와 차트 사용자 정의

각 그래프와 차트는 모양의 사용자 정의, 부가 정보 전달, 데이터 하이라이트를 사용할 수 있도록 많은 옵션을 가집니다. 그래프와 차트는 다른 방법으로 데이터를 그리지만, 바로 가기 메뉴에서 접근할 수 있도록 여러가지 공통된 옵션이 제공됩니다. 그러나, 어떤 옵션은 **그래프** 또는 **차트**의 특정한 타입에서만 사용 가능합니다.

여러 X, Y 스케일 사용

그래프나 차트의 여러 스케일을 사용하여 공통적인 x 또는 y 스케일을 공유하지 않는 여러 개의 플롯을 디스플레이합니다. 그래프나 차트에 여러 스케일을 추가하기 위해서는 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 스케일 복제를 선택합니다. 플롯 범례의 플롯에서 마우스 오른쪽 버튼을 클릭한 후 X 스케일 또는 Y 스케일 바로 가기 메뉴에서 플롯에 연계된 스케일을 선택합니다.

XY, 강도, 디비털 웨이브폼, **웨이브폼 그래프에 여러 x와 y 스케일 추가**가 가능합니다. 혼합된 신호 그래프와 모든 차트는 여러 y 스케일만을 지원합니다.

오토스케일링

모든 그래프와 차트(3D 그래프 제외)는 자동적으로 연결한 데이터에 맞도록 수평과 수직 스케일을 조정(즉, 오토스케일링)할 수 있습니다. 기본적으로, 그래프와 차트의 오토스케일링은 활성화되어 있습니다. 하지만, 오토스케일링은 실행을 느리게 할 수 있습니다. 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 X 스케일 >> X 오토스케일 또는 Y 스케일 >> Y 오토스케일을 선택하여 오토스케일링을 켜거나 끌 수 있습니다.



노트

컴파스 플롯, 에러 막대 플롯, 화살표 플롯, XY 플롯 행렬의 경우, 바로 가기 메뉴에서 X 오토스케일 또는 Y 오토스케일을 선택합니다.

수행 도구나 라벨링 도구를 사용하여 직접 수직 또는 수평 스케일을 바꿉니다.



노트 LabVIEW는 사용자가 그래프나 차트의 축을 오토스케일할 때 숨겨진 플롯을 포함하지 않습니다. 오토스케일할 때 숨겨진 플롯을 포함시키려는 경우, 플롯을 숨기는 대신 투명하게 하십시오. 플롯 범례의 플롯 이미지에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 색을 선택하여 플롯의 색을 변경합니다.

그래프 또는 차트 스케일의 크기를 조정할 때, 그래프 또는 차트의 다른 원소는 이동하고 크기가 조정됩니다. 이 동작을 비활성화하고 플롯 영역 크기를 고정시키려면, 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급 >> 오토스케일 조정을 선택합니다. 이 동작을 비활성화하면, 스케일이 잘리거나 서로 겹칠 수도 있습니다.



노트 오토스케일 조정 옵션은 컴파스 플롯, 에러 막대 플롯, 화살표 플롯, XY 플롯 행렬 또는 3D 그래프에 적용되지 않습니다.

여러 X, Y 스케일 포맷하기

그래프 또는 차트의 스케일을 설정하려면, 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **프로퍼티**를 선택하여 그래프 프로퍼티 대화 상자 또는 차트 프로퍼티 대화 상자를 디스플레이합니다. 그래프와 차트(3D 그래프 제외)에서 x축과 y축의 스케일이 어떻게 나타나는지 지정하려면 **프로퍼티** 대화 상자의 디스플레이 포맷 페이지를 사용합니다. 기본적으로, x 스케일은 시간의 라벨을 가지며, y 스케일은 진폭의 라벨을 가집니다. 3D 그래프에서 x축과 y축의 스케일이 어떻게 나타나는지 지정하려면, 3D 플롯 프로퍼티 또는 3D 그래프 프로퍼티 대화 상자의 포맷 페이지를 사용합니다.

또한 디스플레이 포맷 또는 포맷 페이지를 사용하여 그래프 또는 차트의 스케일을 위해 숫자형 포맷을 지정합니다. 직접 포맷 문자열을 입력할 수 있는 텍스트 옵션을 디스플레이하려면 고급 편집 모드를 선택합니다. **포맷 문자열**을 입력하여 스케일의 모양과 숫자 정밀도를 사용자 정의합니다.

스케일의 이름을 변경하고 x, y축 스케일의 모양을 포맷하려면, 프로퍼티 대화 상자의 스케일 탭, 3D 그래프 프로퍼티 대화 상자의 축 탭, 또는 3D 플롯 프로퍼티 대화 상자의 축 페이지를 클릭합니다.

그래프와 차트의 모양 사용자 정의하기

바로 가기 메뉴 옵션을 사용하여 그래프 또는 차트의 모양을 사용자 정의합니다. 그래프 또는 차트를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 보이는 항목을 선택하여 다음 옵션들을 숨기거나 보입니다:



노트 3D 그래프에서는 이 옵션을 사용할 수 없습니다.

- 플롯 범례—플롯의 색과 스타일을 정의합니다. 여러 개의 플롯을 표시하려면 플롯 범례의 크기를 조정합니다. 이 옵션은 강도 그래프나 차트에는 적용되지 않습니다.
- 스케일 범례—스케일의 라벨을 정의하고 스케일 프로퍼티를 설정합니다.
- 그래프 팔레트—VI가 실행되는 동안 그래프 또는 차트에서 커서를 움직이고 줌하고 팬할 수 있습니다.
- 커서 범례—정의된 포인트 좌표에 마커를 디스플레이합니다. 그래프에 여러 커서를 디스플레이할 수 있습니다. 이 옵션은 그래프에만 적용됩니다.
- X 스크롤 막대—그래프 또는 차트의 데이터를 스크롤합니다. 스크롤 막대를 사용하여 현재 보이지 않는 그래프나 차트의 데이터를 관찰합니다.
- X 스케일과 Y 스케일—x와 y스케일을 포맷합니다.
- 디지털 디스플레이—차트의 숫자값을 디스플레이합니다 이 옵션은 웨이브폼 차트에만 적용됩니다.

플롯 범례 사용하기

그래프와 차트의 플롯 범례를 사용하여 LabVIEW가 플롯 영역에 그리는 플롯을 보거나 플롯이 그래프나 차트에 나타나는 방식을 사용자 정의할 수 있습니다. 프로그램적으로 범례 프로퍼티를 사용하여 플롯 범례를 사용자 정의합니다.



노트 콤파스 플롯, 에러 막대 플롯, 화살표 플롯의 **플롯 범례**는 다른 그래프의 플롯 범례와 다릅니다.

플롯 범례에 플롯 추가하기 및 디스플레이하기

플롯 범례를 추가하려면, **위치 도구를 사용**하십시오. 그래프 또는 차트 **프로퍼티** 대화 상자의 모양 페이지를 사용하여 그래프 또는 차트의 플롯 범례에서 **플롯의 수를 지정**합니다. 또한 범례:행의 개수 프로퍼티를 사용하여 플롯 범례에 나타내는 플롯의 개수를 프로그램적으로 설정할 수 있습니다.

그래프나 차트에 디스플레이할 플롯 선택하기

그래프나 차트에 플롯을 디스플레이할지 선택하려면, 플롯 범례의 문양에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 플롯 보이기를 선택합니다. 여러 개의 플롯이 있는 경우, 플롯 범례에서 마우스 오른쪽 버튼을 클릭한 후 보이는 아이템 >> 플롯 보이기 확인란을 선택하여 각 플롯 옆에 확인란을 디스플레이할 수도 있습니다. 확인란을 사용하여 플롯 영역에 디스플레이하려는 플롯을 선택합니다.

플롯 범례 사용하여 플롯 모양 사용자 정의하기

그래프 또는 차트의 플롯 영역에 나타나는 플롯을 사용자 정의하면 각 플롯들을 구분하기 편해집니다. 플롯 범례를 사용하여 각 플롯이 그래프 또는 차트의 플롯 영역에서 어떻게 나타나는지 사용자 정의합니다. 플롯 범례 안에 있는 문양을 클릭한 후 바로 가기 메뉴에서 다음 옵션 중 하나를 선택하여, 그래프 또는 차트의 플롯 영역에 어떻게 플롯이 나타나는지 사용자 정의합니다.

- 플롯 보이기—그래프나 차트에 플롯을 디스플레이할지 설정합니다.
- 일반 플롯—플롯 타입에 대한 옵션을 제공합니다. 라인, 산포, 라인과 기호, 채우기, 빔, 막대 플롯 타입을 선택할 수 있습니다.
- 색—**색 선택기**를 디스플레이하여 플롯의 색을 선택할 수 있습니다.
- 라인 스타일—기본 라인과 점선 라인 스타일을 제공합니다.
- 라인 폭—라인 폭을 제공합니다.
- 엘리어스 제거—라인 플롯을 자연스럽게 나타내기 위해서 이 아이템의 옆에 확인 표시를 합니다. 그러나, 엘리어스 제거 라인은 복잡한 계산이 필요하므로 성능을 저하시킬 수 있습니다.
- 막대 플롯—막대 플롯 옵션을 제공합니다.
- 베이스 라인 채우기—플롯과 같은 색으로 플롯 아래 공간을 채우는 옵션을 제공합니다. 그래프 또는 차트에 여러 개의 플롯이 있으면, 바로 가기 메뉴의 아래에서 플롯을 선택하여 두 플롯간 공간을 채울 수 있습니다.
- 보간—데이터 포인트만을 플롯하는 것을 포함하여 보간 옵션을 제공합니다.
- 포인트 스타일—데이터 포인트의 포인트 스타일을 제공합니다.
- X 스케일—X 축과 연계할 변수를 설정합니다.
- Y 스케일—Y 축과 연계할 변수를 설정합니다.

또한 플롯 프로퍼티를 사용하여 프로그램적으로 플롯이 플롯 영역에 어떻게 나타날지를 사용자 정의할 수 있습니다.



노트 일반 플롯, 라인 폭, 엘리먼트 제거, 막대 플롯, 베이스 라인 채우기, 보간, 포인트 스타일 옵션은 디지털 웨이브폼 그래프에서는 사용할 수 없습니다.

플롯 범례에 스크롤 막대 추가하기

그래프나 차트의 플롯 범례에 **수평 또는 수직 스크롤 막대**를 추가할 수 있습니다. 스크롤 막대를 사용하여 한 번에 모든 플롯을 표시하지 않고도 플롯 범례의 플롯을 보십시오.

그래프 팔레트 사용하기

그래프 팔레트를 사용하여 VI를 실행하는 동안 그래프나 차트와 상호 작용합니다. 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목>> 그래프 팔레트를 선택하여 다음과 같이 그래프 팔레트를 디스플레이합니다.



그래프 팔레트에서 버튼을 클릭하여 **커서**를 움직이거나, **디스플레이를 줌**하거나, 디스플레이를 팬합니다. 각 버튼은 활성화시 녹색 LED로 디스플레이됩니다. 그래프 팔레트에는 다음 버튼이 왼쪽에서 오른쪽의 순서로 나타납니다:

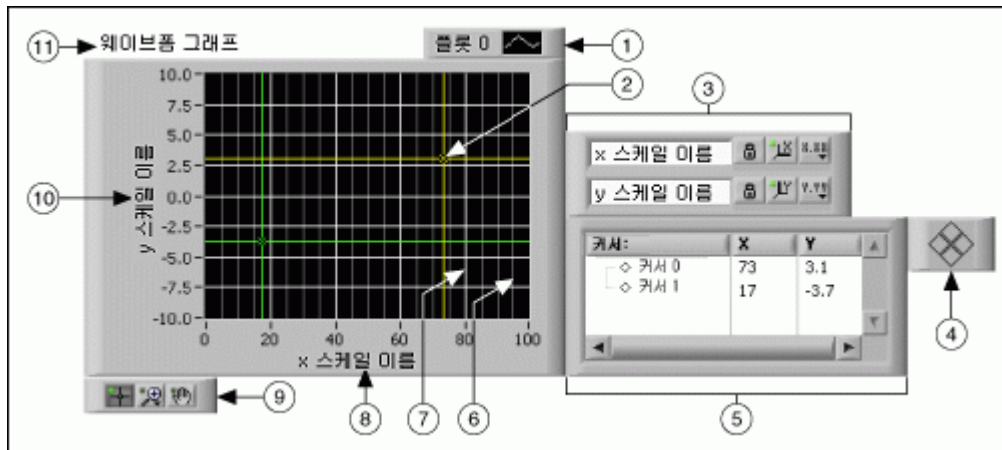
- 커서 이동 도구—디스플레이에서 **커서**를 움직입니다. 이 버튼은 그래프에만 적용됩니다.
- 줌—디스플레이를 줌 인, 줌 아웃합니다.
- 팬 도구—플롯을 선택하여 플롯을 디스플레이에서 움직입니다.



노트 그래프 팔레트는 3D 그래프에서는 사용할 수 없습니다.

그래프 사용자 정의하기

각 그래프는 사용자의 디스플레이 요구사항에 맞게 그래프를 사용자 정의할 수 있는 옵션을 포함합니다. 예를 들어, **그래프 커서**의 동작과 모양을 수정하거나 **그래프 스케일**을 설정할 수 있습니다. 다음 그림은 그래프의 원소를 보여줍니다.



1	플롯 범례
2	커서
3	스케일 범례
4	커서 이동자
5	커서 범례
6	보조 눈금 마크
7	눈금 마크
8	X 스케일
9	그래프 팔레트
10	Y 스케일
11	라벨



노트 혼합된 신호 그래프와 디지털 웨이브폼 그래프의 플롯 범례는 그래프의 왼쪽에 있습니다.

그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이টে를 선택하고 적절한 항목을 선택하여, 플롯 범례, 스케일 범례, 커서 범례, 그래프 팔레트, 라벨을 추가합니다.



노트 플롯 범례, 커서, 스케일 범례, 커서 이동자, 커서 범례, 그래프 팔레트는 3D 그래프에는 적용되지 않습니다.

커서 범례 사용하기

그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이টে를 선택합니다. **커서를 그래프에 추가**하려면, 커서 범례에서 어디든 마우스를 놓고

마우스 오른쪽 버튼을 클릭하고, 커서 생성을 선택하고, 바로 가기 메뉴에서 커서 모드를 선택합니다.

커서 위치는 커서 모드에서 정의합니다. 커서에는 다음과 같은 모드가 있습니다.

- 독립 — 플롯 위치에 구애받지 않고 플롯 영역에서 자유롭게 커서를 움직입니다.
- 단일 플롯 — 커서와 연관되어 있는 플롯에만 커서를 위치시킵니다. 커서를 대응되는 플롯에 따라 움직일 수 있습니다. 커서 범례에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고정을 선택하여 하나 또는 모든 플롯을 커서와 연관시킵니다.
- 멀티플롯 — 플롯 영역의 특정 데이터 포인트에서만 커서가 위치합니다. 멀티플롯 커서는 커서가 연관된 모든 플롯의 지정된 x값에서 값을 리포트합니다. 플롯 영역의 어느 플롯에나 커서를 놓을 수 있습니다. 커서 범례에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고정을 선택하여 하나 또는 모든 플롯을 커서와 연관시킵니다. 이 모드는 복합 신호 그래프에서만 사용 가능합니다.



노트 커서를 생성한 후에는 커서의 모드를 변경할 수 없습니다. 반드시 **커서를 삭제**하고 다른 커서를 생성해야 합니다.

그래프에서 커서를 사용하여 플롯에서의 정확한 포인트 값을 읽습니다. 커서 값은 커서 범례에 디스플레이됩니다.

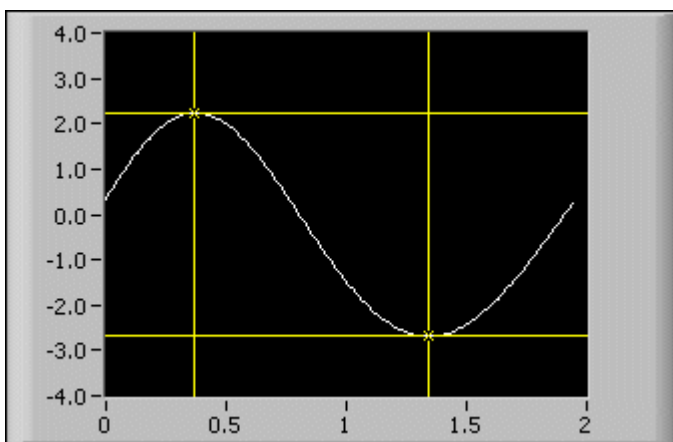


노트 멀티플롯 그래프 또는 차트에서 때때로 커서 범례의 y 값이 괄호로 표시될 수 있습니다. 이것은 커서의 현재 x 위치에 해당하는 플롯에 y 값이 없으므로 y 값을 보간해야 하는 경우에 발생합니다.



팁 커서 대신 **주석**을 사용하여 플롯의 데이터 포인트를 하이라이트합니다.

다음 프런트패널은 여러 커서를 사용한 그래프의 예제를 보여줍니다.



그래프 커서를 끌어서 그래프 경계를 지나도록 만들면, 그래프는 커서가 이동한 방향을 스크롤됩니다. 이 동작을 비활성화하려면, 그래프에서 마우스 오른쪽 버튼을 클릭하고

바로 가기 메뉴에서 고급 >> 커서 스크롤 그래프를 선택합니다. 이 동작을 비활성화하면, 커서를 끌어서 그래프 경계를 지나더라도 스케일은 업데이트되지 않습니다.



노트 커서 스크롤 그래프 옵션은 콤파스 플롯, 에러 막대 플롯, 화살표 플롯, XY 플롯 행렬에 적용되지 않습니다.

여러가지 방법으로 커서의 모양을 사용자가 정의할 수 있습니다. 플롯에서 커서에 라벨링하고, 커서의 색을 지정하고, 라인, 포인트, 커서 스타일을 지정할 수 있습니다. 커서 범례 열에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이টে를 선택하여 **커서를 사용자 정의**합니다.

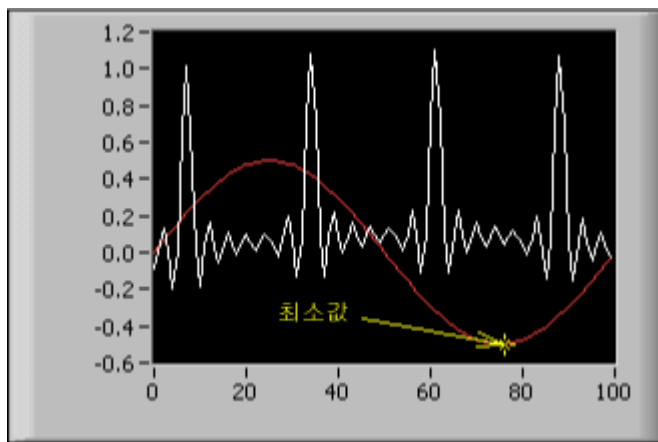
그래프 주석 사용하기

그래프에 주석을 사용하여 플롯 영역의 데이터 포인트를 하이라이트합니다. 주석은 주석과 데이터 포인트를 식별하는 라벨과 화살표를 포함합니다. 그래프는 여러 개의 주석을 가질 수 있습니다.



노트 3D 그래프에서는 주석을 사용할 수 없습니다.

다음 프런트패널은 주석을 사용한 그래프의 예제를 보여줍니다.



그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리 >> 주석 생성을 선택하여 주석 생성 대화 상자를 디스플레이합니다. 주석 생성 대화 상자를 사용하여 주석 이름을 지정하고 어떻게 주석이 플롯 영역의 플롯에 고정되는지 지정합니다.

주석 생성 대화 상자에서 고정 스타일 풀다운 메뉴를 사용하여 어떻게 주석이 플롯 영역의 플롯에 고정되는지 지정합니다. 고정 스타일 구성 요소에는 다음 옵션이 있습니다:

- 독립—주석을 플롯 영역의 어디로든 이동시킬 수 있습니다. LabVIEW는 주석을 플롯 영역의 어느 플롯에도 고정시키지 않습니다.
- 모든 플롯에 고정—플롯 영역에서 주석을 가장 가까운 데이터 포인트를 가진 플롯으로 이동할 수 있도록 합니다.
- 한 플롯에 고정—지정된 플롯을 따라서만 주석을 이동할 수 있습니다.

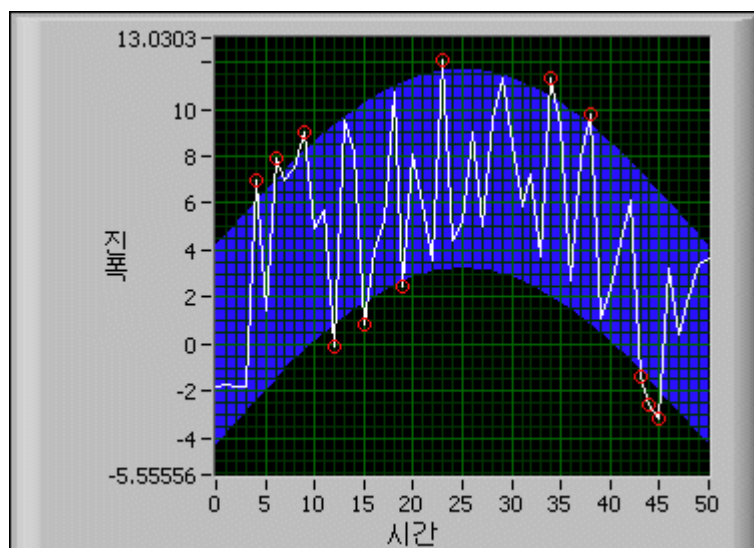
여러가지 방법으로 주석의 모양을 사용자가 정의할 수 있습니다. 플롯 영역에 주석 이름이나 화살표를 숨기거나 보여주고, 주석의 색을 지정하고, 라인, 포인트, 주석스타일을 지정할 수 있습니다. 주석에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 옵션을 선택하여 **주석을 사용자 정의**합니다.

주석을 삭제하려면, 주석에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 주석 삭제를 선택합니다. 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리>>모든 주석 삭제를 선택하여 플롯 영역의 모든 주석을 삭제합니다.

그래프 플롯 영역에 그리기

플롯 영역에 배경, 전경, 혹은 중간 이미지 그리기를 하여 강도, 혼합된 신호, 웨이브폼 그래프의 그래프 플롯 영역을 사용자 정의합니다. 플롯 이미지 그리기 캔버스는 그래프 플롯 영역의 가장 왼쪽 위 가장자리에 항상 원점 (0,0)이 위치하는 좌표 시스템을 갖습니다. 강도 그래프, 혼합된 신호 그래프, 웨이브폼 그래프 클래스의 플롯 이미지 프로퍼티를 사용하여 그래프 플롯 영역의 배경에 사용자 눈금, 중간에 데이터 envelope를 생성하거나 전경에 데이터 포인트를 설명하는 모양을 사용할 수 있습니다.

다음 프런트패널은 데이터 포인트를 설명하는 원을 만들고 데이터에 설정된 허용오차를 지정하는 envelope를 생성하는 예를 보여줍니다.



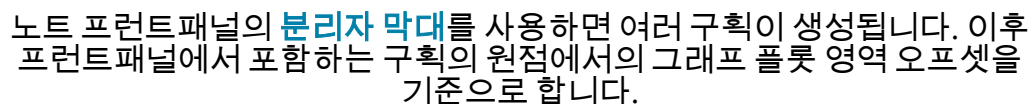
배경 이미지는 플롯 데이터와 그리드 라인 뒤에 위치합니다. 플롯 이미지:뒤 프로퍼티를 사용하여 혼합된 신호와 웨이브폼 그래프에 대한 배경 이미지를 설정하십시오.

전경 이미지는 플롯 데이터와 그리드 라인 앞에 위치합니다. 플롯 이미지:앞 프로퍼티를 사용하여 강도, 혼합된 신호, 웨이브폼 그래프에 대한 배경 이미지를 설정하십시오.

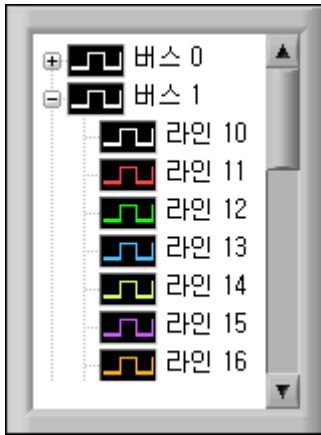
중간 이미지는 그리드 라인과 플롯 데이터 사이에 위치합니다. 플롯 이미지:중간 프로퍼티를 사용하여 혼합된 신호와 웨이브폼 그래프에 대한 배경 이미지를 설정하십시오.



노트 강도 그래프의 전경에서만 이미지를 그릴 수 있습니다.



© National Instruments Corporation 457



플롯 범례의 트리 보기에서 버스를 확장 및 축소하려면, 버스 왼쪽의 확장/축소 기호를 클릭하십시오. 버스나 라인 이름이 플롯 범례의 크기를 초과하고 라인, 버스 또는 라인과 버스의 양이 플롯 범례의 크기를 초과하면 스크롤 막대가 자동으로 나타납니다.

디지털 웨이브폼 그래프에 나타나는 라인의 굵기를 **사용자 정의**하고, 변이 타입과 위치를 설정하고, 그래프 라벨을 포맷하고, 디지털 라인과 버스의 이름을 지정할 수 있습니다. [디지털 신호 그룹화] 익스프레스 VI를 사용하여 디지털 웨이브폼 그래프에서 보고 싶은 순서대로 디지털 웨이브폼을 그룹화할 수 있습니다.

다음 플롯 범례는 플롯 범례의 표준 보기로, 이 보기를 사용하여 디지털 라인이나 버스를 볼 수 있습니다.



플롯 범례를 트리 보기에서 표준 보기로 변경하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 플롯 범례를 표준 보기로 변경을 선택합니다. 플롯 범례의 트리 보기로 다시 되돌아가려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 플롯 범례를 트리 보기로 변경을 선택합니다.



노트 플롯 범례의 표준 보기에서 트리 보기로 변경하면 LabVIEW는 **사용자 라인 셋팅**을 포함하여 모든 **할당한 라인**을 제거합니다. 또한, 트리 보기가 활성화된 경우, 라인을 플롯 범례에 할당할 수 없습니다.

선택한 보기에 상관없이 플롯 범례를 비활성화하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급 >> 플롯 범례 비활성화를 선택하여 메뉴 아이템 옆에 확인표시를 합니다. 또는 범례:비활성화 프로퍼티를 사용하여 프로그램적으로 플롯 범례를 비활성화할 수 있습니다.

플롯 범례의 크기를 변경하려면, 위치 도구를 플롯 범례 경계로 이동합니다. 크기 조정 핸들이 범례의 모서리에 나타나면, 핸들을 위, 아래, 왼쪽 또는 오른쪽으로 끌고 범례가 원하는 크기가 될 때 마우스 버튼을 놓습니다.

플롯 영역에서 데이터 사용자 정의하기

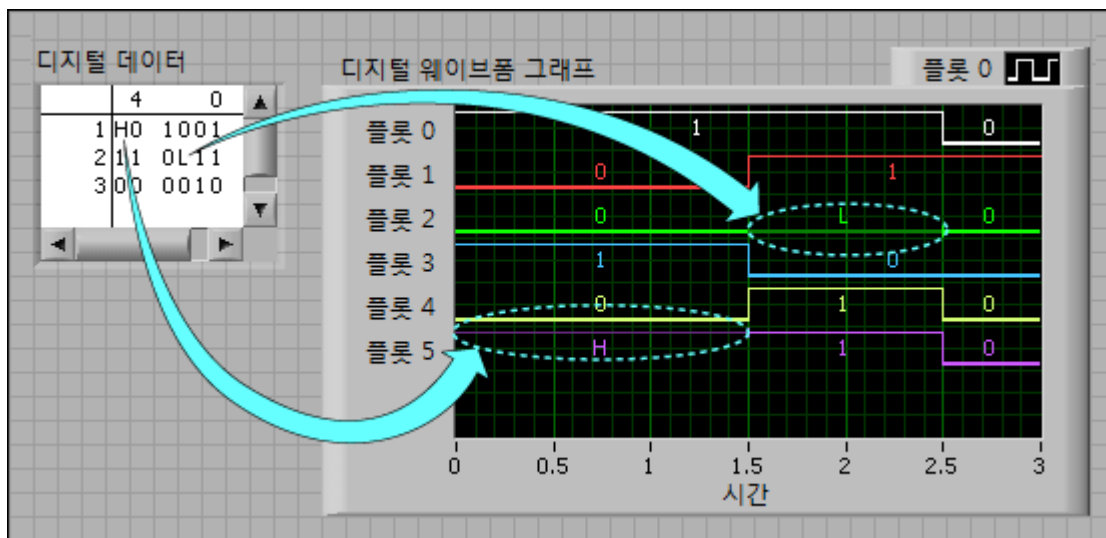
기본으로, 디지털 웨이브폼 그래프는 플롯 영역에 디지털 라인과 버스 모두를 디스플레이합니다. 디지털 웨이브폼 그래프의 **플롯 영역을 사용자 정의**하여 보려는 데이터만을 디스플레이합니다.

디지털 웨이브폼 그래프에 드라이브 및 비교 로직 상태 모두의 디지털 데이터가 포함되어 있는 경우, 기본으로 비교 데이터는 플롯에서 드라이브 데이터보다 어둡게 나타납니다. 비교 데이터가 어둡게 나타나지 않게 하려면, 플롯에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 비교 데이터 어둡게를 선택하고 확인 표시를 제거합니다. 또한 비교 데이터 어둡게 프로퍼티를 사용하여 비교 데이터를 프로그램적으로 어둡게 할 수도 있습니다.



노트 이 기능은 주로 디지털 I/O 신호를 생성하는 사용자에게 적용됩니다. 비교 또는 예상되는 응답 데이터는 2진 데이터 스트림에서 하이 또는 로우로 표시된 데이터를 참조합니다. LabVIEW는 이 값을 어둡게하여 높고 낮은 데이터를 다른 드라이브나 자극 데이터와 구별합니다.

다음 프런트패널은 어둡게 표시된 비교 데이터를 보여줍니다.



2D 그래프 사용자 정의하기

2D 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 프로퍼티 대화 상자를 디스플레이하면 콤팩스 플롯, 에러 막대 플롯, 화살표 플롯, XY 플롯 행렬의 모양, 포맷, 플롯 스타일, 스케일, 커서를 사용자 정의할 수 있습니다.

그래프 오른쪽에 있는 플롯 범례를 활용하여 콤팩스 플롯, 에러 막대 플롯, 화살표 플롯의 플롯을 사용자 정의합니다. 플롯 이름을 클릭하여 이름을 편집할 수 있습니다. 플롯의 모양을 바꾸려면, 플롯 이름 오른쪽에 있는 이미지를 클릭하여 다음 옵션을 디스플레이합니다:

- 라인 스타일—플롯의 라인 스타일을 설정합니다.
- 라인 폭—플롯의 폭을 설정합니다.

- 크기—콤파스 및 화살표 그래프의 화살표 크기를 설정하고 에러 막대 그래프의 에러 막대 크기를 설정합니다.
- 색—플롯 라인의 색을 설정합니다.



노트 이 대화 상자를 닫으려면 확인 표시를 클릭하여 변경 사항을 적용하거나 빨간색 x 표시를 클릭하여 변경 사항을 취소합니다.

3D 그래프에 데이터를 플롯하는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Math Plots - 2D 디렉토리를 참조하십시오.

관련 예제 찾기

3D 그래프 사용자 정의하기

3D 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 프로퍼티 대화 상자를 디스플레이하면, 모든 **3D 그래프**의 모양을 사용자 정의할 수 있습니다.

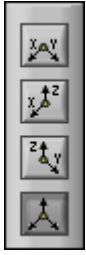
3D 라인, 3D 파라미터형, 3D 표면 그래프를 제외한 나머지 3D 그래프의 경우, 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 3D 플롯 프로퍼티를 선택하여 3D 플롯 프로퍼티 대화 상자를 디스플레이합니다. 이 대화 상자를 통해 그래프, 빛, 축, 값 인덱스, 포맷, 커서를 사용자 정의할 수 있습니다.

3D 라인, 3D 파라미터형, 3D 표면 그래프의 경우, 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 3D 그래프 프로퍼티를 선택하여 3D 그래프 프로퍼티 대화 상자를 디스플레이합니다. 3D 그래프 프로퍼티 대화 상자를 사용하여 3D 그래프 프로퍼티를 포맷합니다. 대화 상자의 각 페이지에 있는 컨트롤을 변경하여 3D 그래프의 모양을 바꿉니다.

(Windows) ActiveX 3D 그래프는 **ActiveX 기술**과 3D 형태를 다루는 VI를 사용하기 때문에 다른 그래프의 옵션을 설정과는 다르게 ActiveX 3D에 대한 옵션을 설정합니다. 어플리케이션을 생성하는 동안 **ActiveX 컨트롤 프로퍼티 브라우저**를 사용하여 ActiveX 3D 그래프의 프로퍼티를 설정합니다. ActiveX 3D 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티 브라우저를 선택하여 ActiveX 컨트롤 프로퍼티 브라우저를 디스플레이합니다. 사용자가 런타임에서 일반 프로퍼티를 변경할 수 있도록 허용하거나 프로그램적으로 프로퍼티를 설정할 필요가 있다면 3D 그래프 프로퍼티 VI를 사용합니다.

막대 그래프와 파이 그래프의 경우, 그래프 오른쪽에 있는 색 팔레트를 사용하여 각 막대와 파이 조각을 사용자 정의합니다. 막대 또는 파이 조각의 이름을 변경하려면, 이름을 하이라이트하고 새 텍스트를 입력합니다. 막대 또는 파이 조각의 색을 변경하려면, 이름 오른쪽에 있는 색을 클릭하고 새 색을 선택합니다.

3D 그래프에서 평면 투영 보기를 사용자 정의하려면, 그래프 오른쪽 하단 모서리 근처에 있는 다음의 투영 팔레트를 사용하십시오:



투영 팔레트에는 다음의 버튼이 포함되어 있습니다:

- X-Y 평면 투영—플롯 투영을 X-Y 평면에 디스플레이합니다.
- X-Z 평면 투영—플롯 투영을 X-Z 평면에 디스플레이합니다.
- Y-Z 평면 투영—플롯 투영을 Y-Z 평면에 디스플레이합니다.
- 기본 투영—3D 평면에 플롯을 디스플레이합니다.



노트 투영 팔레트는 막대, 3D 라인, 3D 파라미터형, 3D 표면 그래프에 나타나지 않습니다.

3D 그래프에 데이터를 플롯하는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Math Plots - 3D 디렉토리를 참조하십시오.

관련 예제 찾기

3D 그래프에 데이터를 플롯하는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Math Plots - 3D 디렉토리를 참조하십시오.

관련 예제 찾기

차트 사용자 정의하기

저장된 데이터를 새 데이터로 덮어쓰고 디스플레이하는 그래프와 달리, 차트는 이전에 저장된 데이터의 히스토리를 보존하고 주기적으로 업데이트합니다.

사용자의 데이터 디스플레이 요구에 맞도록 차트를 사용자 정의할 수 있습니다. 모든 차트에서 스크롤 막대, [스케일 범례](#), [그래프 팔레트](#), 디지털 디스플레이, 시간에 대한 스케일 형의 옵션이 사용 가능합니다. 또한 [차트 히스토리 길이](#), [업데이트 모드](#), 그리고 플롯 디스플레이의 차트 동작을 수정할 수 있습니다.

받을 수 있는 다양한 차트의 종류 및 데이터 타입에 대한 예는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts 디렉토리의 Chart Data Types and Update Modes VI를 참조하십시오.

차트 히스토리 길이 설정하기

LabVIEW는 버퍼, 또는 차트 히스토리에 이미 추가된 데이터 포인트를 저장합니다. 웨이브폼 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차트 히스토리 길이를 선택하여 히스토리 버퍼를 설정합니다.



팁 커다란 차트 히스토리 값은 많은 메모리가 필요할 수 있습니다. 그래프와는 다르게, 차트는 이전에 쓴 데이터의 히스토리를 갖습니다. 차트가 연속적으로 실행될 때, 차트 히스토리는 커지고 추가적인 메모리 공간을 필요로 하게 됩니다. 차트 히스토리가 가득 찰 때까지 지속되는데, 이후 LabVIEW는 더이상의 메모리를 차지하지 않게 됩니다. LabVIEW는 VI가 재시작할 때 자동으로 차트 히스토리를 지우지 않습니다. 프로그램이 실행되는 동안, 차트 히스토리를 지울 수 있습니다. 이를 위해, 차트에 대한 히스토리 데이터 속성 노드에 빈 배열을 작성하십시오.

차트의 스크롤 막대를 이용해 이전에 수집된 데이터를 살펴봅니다. 스크롤 막대를 나타내려면 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목>> X 스크롤 막대를 선택합니다.

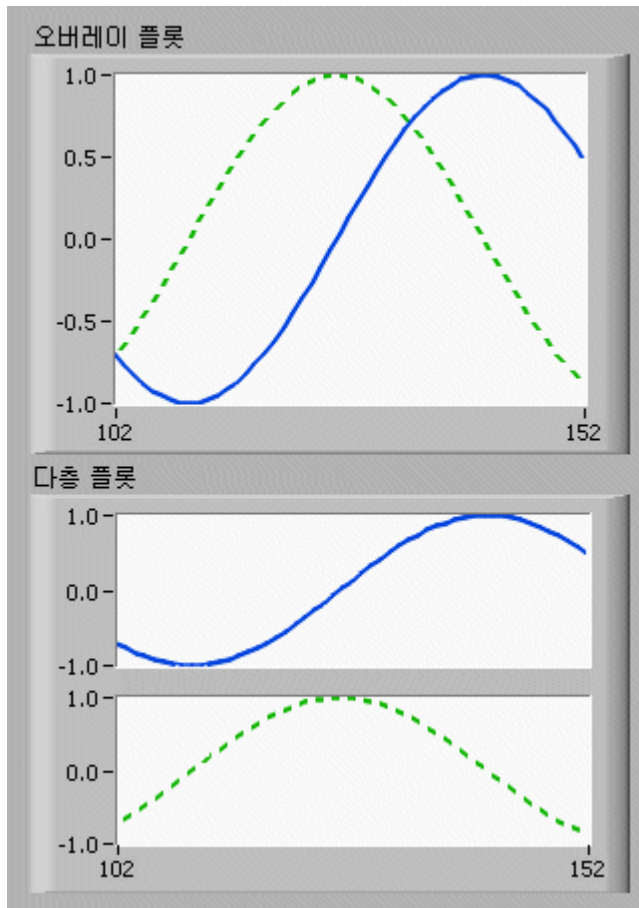
차트 업데이트 모드 설정하기

새로운 데이터를 디스플레이하기 위해 차트를 업데이트 하는 방법을 설정합니다. 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>업데이트 모드를 선택하여 차트의 업데이트 모드를 설정합니다. 차트에 데이터를 디스플레이할 때 사용할 수 있는 모드는 다음과 같습니다:

- 스트립 차트—차트의 이전 데이터는 왼쪽 그리고 새 데이터는 오른쪽에 연속적으로 스크롤되면서 데이터를 보여줍니다. 스트립 차트는 종이 테이프를 사용하는 스트립 차트 기록기와 유사합니다. 스트립 차트는 기본 업데이트 모드입니다.
- 스코프 차트—펄스 또는 웨이브와 같은 데이터의 한 항목을 왼쪽에서 오른쪽으로 스크롤하며 보여줍니다. 각 새로운 값에 대해, 차트는 최근값의 오른쪽에 값을 플롯합니다. 플롯 영역의 오른쪽 경계 끝까지 데이터를 표시하고 나면, LabVIEW는 그 플롯을 지우고 왼쪽 경계에서부터 다시 새 데이터를 나타내기 시작합니다. 스코프 차트에서 이렇게 플롯 영역을 되돌린 후에 다시 데이터를 표시하는 회귀 디스플레이 방식은 오실로스코프와 유사합니다.
- 스위치 차트 — 수직선을 기준으로 오른쪽에는 이전 데이터가 유지되고 왼쪽에는 새 데이터가 그려진다는 점을 제외하면 스코프 차트와 유사합니다. 플롯이 영역의 오른쪽 경계 끝까지 데이터가 표시될 때, LabVIEW 스위치 차트의 플롯은 지워지지 않습니다. 스위치 차트는 EKG 디스플레이와 비슷합니다.

오버레이와 다층 플롯 사용하기

하나의 수직 스케일인 오버레이 플롯이나 여러 수직 스케일인 다층 플롯을 사용하여 웨이브폼 차트에 여러 개의 플롯을 디스플레이할 수 있습니다. 다음 프런트패널은 오버레이 플롯과 다층 플롯의 예입니다.



차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다층 플롯을 선택하여 여러 개의 수직 스케일로 플롯하는 차트를 디스플레이할 수 있습니다. 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 오버레이 플롯을 선택하면 하나의 수직 스케일로 플롯하는 차트를 디스플레이할 수 있습니다.

그래프, 차트, 테이블, 배열에서 데이터 반출하기

그래프, 차트, 테이블과 배열에서 Microsoft Excel이나 클립보드 또는 DIAdem으로 **데이터를 반출**할 수 있습니다.



노트 Excel로 데이터를 반출하려면 Microsoft Excel이 설치되어 있어야 합니다. DIAdem로 데이터를 반출하려면 DIAdem이 설치되어 있어야 합니다. DIAdem에 관한 정보와 최신 버전을 다운로드하려면 National Instruments 웹 사이트 ni.com/diadem을 참조하십시오.

그래프와 차트에서는 프론트패널 윈도우에서 보이는 데이터만을 반출할 수 있습니다. 배열과 테이블에서는 셀과 열, 행 중 원하는 부분을 선택하여 반출합니다. 데이터를 반출하기 전에 여러 기법을 사용하여 **데이터가 디스플레이되는 방식을 사용자 정의**할 수 있습니다.

다음 정보는 데이터를 반출할 때의 유의점과 권장 사항에 대한 설명입니다:

- 여러 개의 플롯을 가진 그래프 전체의 반출이 가능합니다. 그래프에서 단일 플롯만 반출할 수도 있습니다.
- 그래프나 차트에서 반출하는 데이터는 각각의 플롯을 두 개의 열로 나타냅니다. 두 개의 열 중 첫번째 열은 플롯의 x 값을 나타내며, 두번째 열은 플롯의 y 값을 나타냅니다.
- **강도 그래프**에서 반출하는 데이터는 각 플롯을 한 개의 열로 나타냅니다. 한 개의 열로 한 개의 플롯을 나타내며, 각각의 셀은 플롯의 x/y 위치에서의 z 값을 나타냅니다.
- 1D 또는 2D **문자열 배열과 숫자형 배열**을 반출할 수 있습니다.
- 또한 2차원을 초과하는 배열을 반출할 수도 있습니다. 그러나 2차원을 초과하는 배열을 반출할 때는 프런트패널 윈도우에서 보이는 차원만 반출할 수 있습니다.

그래프, 차트, 테이블의 이미지 반출하기

그래프, 차트, 테이블, 그림 컨트롤, 디지털 데이터, 디지털 웨이브폼 컨트롤과 인디케이터의 이미지를 프리젠테이션, E-메일, 텍스트 문서 등에 **반출**시킬 수 있습니다. 단순화된 이미지 반출시, LabVIEW는 컨트롤 또는 인디케이터, 디지털 디스플레이, 플롯 범례, 인덱스 디스플레이만을 반출하며 스크롤 막대, 스케일 범례, 그래프 팔레트, 또는 커서 범례는 반출하지 않습니다. 이미지를 반출하려면, 그래프나 차트에서 마우스 오른쪽 버튼을 클릭한 후, 반출» 단순화된 이미지 반출을 선택합니다.



노트 단순화된 이미지 반출 메뉴 아이템은 강도 그래프, 강도 차트 또는 3D 그래프에서 사용할 수 없습니다. 3D 그래프(3D 표면 그래프, 3D 파라미터형 그래프, 3D 라인 그래프 제외)에서 이미지를 반출하려면, 3D 그래프에서 마우스 오른쪽 버튼을 클릭한 후, 이미지 반출» 파일로 반출을 선택합니다.

다음 포맷으로 이미지를 반출할 수 있습니다:

(Windows) .emf, .bmp, .eps 파일

(macOS) .pict, .bmp, .eps 파일

(Linux) .bmp, .eps 파일

반출된 이미지를 클립보드 또는 디스크에 저장합니다.

앨리어스 제거 라인 플롯 사용하기

그래프 또는 차트의 라인 플롯의 모양은 앨리어스 제거 라인을 사용하여 개선할 수 있습니다. 앨리어스 제거 라인 그리기를 활성화시키면, 라인 플롯이 더 부드러워집니다. 앨리어스 제거 라인 그리기는 라인 두께, 라인 스타일, 포인트 스타일 등을 변경하지 않습니다.



노트

앨리어스 제거 라인은 디지털 웨이브폼 그래프에는 제공되지 않습니다.

앨리어스 제거 라인 플롯을 활성화하기 위해서, 플롯 범례에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 앨리어스 제거를 선택합니다. 플롯 범례가 보이지 않는 경우, 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 플롯 범례를 선택합니다.



노트 앨리어스 제거 라인은 내부적으로 복잡한 계산이 필요하므로 성능 저하를 유발할 수 있습니다.

스케일 범례 사용하기

그래프 또는 차트에서 스케일에 라벨링하고 스케일 프로퍼티를 설정하기 위해 스케일 범례를 사용합니다. 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 스케일 범례를 보기 위해 보이는 아이템 >> 스케일 범례를 선택합니다.

다음에 보이는 것처럼 수행 도구를 통해 스케일 잠금 버튼을 클릭하여, 각 스케일에 대한 오토스케일링을 토글합니다.



다음에서 보이는 것처럼 오토스케일 버튼을 사용하여, 그래프 또는 차트에 연결하는 데이터 반영하도록 스케일을 조정합니다.



다음에 보이는 것처럼 스케일 포맷 버튼을 사용하여, 포맷, 정밀도, 맵핑 모드, 스케일의 가시화 여부와 스케일 라벨을 설정하고 눈금 색을 지정합니다.



그래프 또는 차트에 **여러 y스케일을 추가**하기 위해 스케일 범례를 사용할 수 있습니다. 위치 도구를 이용하여 스케일 범례를 확장하여 그래프 또는 차트에 y스케일을 추가합니다. 스케일 범례 행에서 아무 곳이나 클릭하여 스케일을 활성화합니다.

다이나믹하게 그래프 포맷하기

다이나믹 데이터 또는 웨이브폼 데이터를 그래프에 와이어로 연결하면 자동으로 그래프의 플롯 범례와 x 스케일 타임스탬프가 포맷됩니다. 예를 들어, [신호 시뮬레이션] 익스프레스 VI로 사인파를 생성하고 절대 시간을 사용하게 하여 웨이브폼 그래프에 [신호 시뮬레이션] 익스프레스 VI 출력을 연결할 경우, 그래프의 플롯 범례는 자동으로 플롯 라벨을 사인으로 업데이트하며, x스케일은 VI가 실행되는 날짜와 시간을 디스플레이합니다.

다이나믹 또는 웨이브폼 데이터가 포함하는 플롯 범례를 무시하려면, 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 속성 무시를 선택합니다. 속성 무시를 선택하면, 플롯 범례 라벨을 프로그램적으로 변경할 수 있습니다.

다이나믹 또는 웨이브폼 데이터가 포함하는 타임스탬프 설정을 무시하려면, 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타임스탬프 무시를 선택합니다.



노트

그래프에 디스플레이한 데이터가 절대 시간에 대해 포맷한 타임스탬프를 포함하는 경우, 타임스탬프 무시를 선택하여 타임스탬프를 무시합니다.

자연스러운 업데이트 사용하기

LabVIEW가 자연스러운 업데이트를 끄고 객체를 업데이트하면, 객체의 내용을 지우고 새 값을 그리면서 눈에 뵈 정도로 깜빡거립니다. 자연스러운 업데이트를 사용하면, 지우기와 그리기로 인해 발생하는 깜빡거림을 방지할 수 있습니다. 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 자연스러운 업데이트를 선택하여 깜빡임을 최소화하도록 오프스크린 버퍼를 사용합니다. 자연스러운 업데이트 기능을 사용할 경우 컴퓨터와 비디오 시스템에 따라 성능 저하를 유발할 수 있습니다.

2D 그림 컨트롤 사용하기

2D 그림 컨트롤은 라인, 원, 텍스트, 다른 타입의 그래픽 형태 등을 가진 그림을 디스플레이하기 위한 그리기 명령 세트를 가집니다. 2D 그림 컨트롤에서 픽셀 단위로 컨트롤이 가능하므로, 거의 어떤 형태의 그래픽 객체라도 제작이 가능합니다. 그래픽 어플리케이션 대신, LabVIEW의 2D 그림 컨트롤과 그래픽 VI를 이용하여 LabVIEW에서 그래픽을 생성, 수정하고 볼 수 있습니다.



노트

또한 3D 그림 컨트롤을 사용하여 **3D 장면** 객체의 그래픽 형태를 디스플레이할 수 있습니다. 또한 빛 소스의 스타일과 위치 및 3D 장면과 사용자 컨트롤 카메라와의 상호작용 등 3D 장면의 특성을 지정할 수 있습니다.

2D 그림 컨트롤은 컨트롤의 왼쪽 위 코너의 픽셀을 원점 (0, 0)으로 하는 픽셀 기반의 좌표 시스템을 가지고 있습니다. 좌표의 수평 x 성분은 오른쪽으로 증가하고, 수직 y 성분은 아래쪽으로 증가합니다.

2D 그림 컨트롤로 나타내기에 그림이 너무 큰 경우, 그림의 일부가 잘리게 되어 사용자는 컨트롤의 디스플레이 영역에 포함된 픽셀만 볼 수 있습니다. 위치 도구를 사용하여 컨트롤의 크기를 조절하고 전체 그림을 볼 수 있도록 VI를 다시 실행합니다. 또한 수직 스크롤 막대나 수평 스크롤 막대를 사용하여 2D 그림 컨트롤의 디스플레이 영역에 나타나지 않는 부분을 볼 수 있습니다. 컨트롤에서 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 스크롤 막대를 선택하여 컨트롤 안에 스크롤 막대를 나타냅니다.

그림 클래스의 **VI 서버** 프로퍼티를 사용하여 2D 그림 컨트롤 또는 컨트롤의 그림 영역의 크기를 변경하는 등 프로그램적으로 2D 그림 컨트롤을 수정할 수 있습니다.

2D 그림 컨트롤을 프런트패널에 놓으면 빈 사각형 영역이 나타나고, 다음과 같이 이에 상응하는 터미널이 블록 다이어그램에 나타납니다.



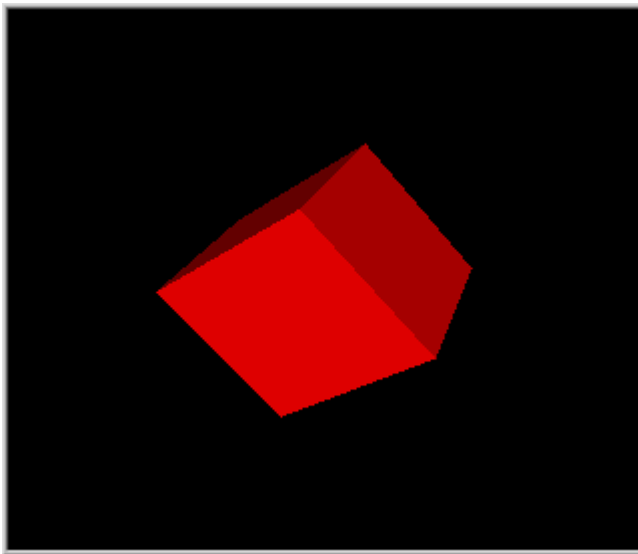
이미지를 2D 그림 컨트롤에 디스플레이하려면, 프로그램적으로 이미지를 컨트롤에 그려야 합니다. 클립보드에 이미지를 복사한 후 2D 그림 컨트롤에 붙여 넣을 수 없습니다. **그림 함수 VI를 사용**하여 그리기 명령 세트를 지정할 수 있습니다. 각 VI는 그리기 명령을 설명하는 입력 세트를 가집니다. 이러한 입력을 기반으로, VI는 디스플레이할 2D 그림 컨트롤에 전달되는 스펙의 간단한 설명을 생성합니다.

2D 그림 컨트롤은 기본적으로 이미지를 그리기 전에 기존 디스플레이를 삭제합니다. 2D 그림 컨트롤이 이전 이미지 위에 새 이미지를 그리도록 하려면, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 먼저 삭제 바로 가기 메뉴 아이템 옆의 확인 표시를 제거합니다.

3D 그림 컨트롤 사용하기

3D 그림 컨트롤은 **3D 객체**의 그래픽 형을 디스플레이합니다. 3D 장면은 3D 그림 컨트롤이나 별도의 장면 윈도우에서 볼 수 있는 3D 객체 또는 3D 객체의 모음입니다. 3D 장면을 디자인하면서 여러 3D 객체를 생성하고 **방향**, **모양**, 장면에서 **다른 객체와의 관계**를 지정할 수 있습니다. 또한 **빛 소스**의 스타일과 위치 및 3D 장면과 **사용자 컨트롤 카메라와의 상호작용** 등 3D 장면의 특성을 설정할 수 있습니다.

다음 이미지는 단순한 3D 장면이 포함된 3D 그림 컨트롤을 보여줍니다.



3D 객체 생성하기

3D 장면에 나타나는 객체는 물리적인 객체의 그래픽 표현입니다. 3D 객체는 3D 공간에 존재하는 정점의 세트와 그 정점 사이의 연결입니다. **3D 장면을 생성**할 때 하나 또는 그 이상의 3D 객체를 생성하여 3D 장면을 구성합니다. [객체 생성] VI를 사용하여 3D 객체를 생성합니다.

3D 객체를 생성한 후, 객체가 3D 장면에 어떻게 나타나는지 정의해야 합니다. 다음의 그리기 가능 속성을 3D 객체에 설정할 수 있습니다:

- 기하—기하는 3D 객체의 모양을 지정합니다. [상자형 생성] 및 [구형 생성]과 같은 기하 VI는 간단한 기하 객체의 형을 정의합니다. 또한 **메시**를 사용하여 더욱 기하학적으로 복잡한 객체도 생성할 수 있습니다.

- 텍스트—객체에 정의된 기하가 없는 경우, 객체에 **텍스트 적용**을 하여 텍스트 객체를 생성할 수 있습니다. 기하가 정의된 객체와 같이, 텍스트는 3D 장면을 제공할 때 나타납니다. [텍스트 생성] VI를 사용하여 새 텍스트를 생성하고 옵션으로 색이나 폰트 등과 같은 텍스트 속성을 설정합니다. 또한 장면 텍스트 프로퍼티와 장면 텍스트 메소드를 사용해서 텍스트의 속성을 설정할 수 있습니다.

LabVIEW는 기본 크기와 색으로 새 객체를 그린 후 3D 장면의 중앙에 객체를 디스플레이합니다. 그러나 **변환** 및 **텍스처**를 사용하여 3D 장면에 있는 3D 객체의 크기, 방향, 위치, 또는 표면 모양을 변경할 수 있습니다. 또한 장면 객체 프로퍼티와 장면 객체 메소드를 사용하여 3D 객체를 설정할 수 있습니다.



노트

텍스트 객체에는 **텍스처**를 적용할 수 없습니다.

변환 적용하기

변환은 수학적 규칙에 따라 3D 객체의 방향이나 모양을 바꿉니다. 변환 VI를 사용하여 3D 장면에서 객체의 크기, 방향, 위치를 변경합니다. 다음의 세 가지 일반적인 변환을 3D 객체에 적용할 수 있습니다:

- 변환—3D 객체를 이동시키면 3D 장면에서 객체가 새 위치로 움직입니다. 3D 객체의 새 위치를 지정하려면 객체를 이동시킬 벡터를 정의해야 합니다.
- 회전—3D 객체를 회전하면 3D 객체의 방향이 변경됩니다. 3D 객체의 새 방향을 지정하려면 객체 회전의 기준이 되는 축과 각도를 지정해야 합니다.
- 스케일링—3D 객체의 스케일링은 3D 객체의 크기를 변경합니다. 3D 객체의 새 스케일을 지정하려면, 3D 장면의 x, y, z 평면에서 객체의 크기를 변경하려는 만큼 인자를 설정합니다.



노트 3D 객체의 모양이 왜곡되지 않도록 하려면 객체를 절단 평면 이상으로 스케일하지 마십시오. 뷰어에서 가장 가까운 평면을 정의하는 근거리 절단 평면의 기본값은 0 단위입니다. 뷰어에서 가장 먼 평면을 정의하는 원거리 절단 평면의 기본값은 10000 단위입니다. 투영:투시법 메소드를 사용하여 절단 평면의 값을 프로그램적으로 설정할 수 있습니다.

각 변환에서, 객체의 초기 위치에서 3D 객체를 회전하는 절대적인 회전이나 객체의 현재 위치에서 3D 객체를 회전하는 상대적인 회전을 수행할 수 있습니다. 예를 들어, [이동 설정] VI를 사용하여 3D 객체를 절대적으로 이동합니다. [객체 이동] VI를 사용하여 3D 객체를 상대적으로 이동합니다.

3D 객체에 변환을 적용하는 방법에 대한 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 Order of Transformations VI를 참조하십시오.

텍스처 적용하기

3D 객체 주변에 **2D 이미지를 둘러싸서** 3D 객체의 표면 모양을 변경할 수 있습니다. 이 과정을 텍스처 적용이라고 부릅니다. 예를 들면, 구형에 특정 행성 표면의 2D 이미지를 적

용하여 구형을 행성으로 만들 수 있습니다. 행성 표면의 이미지가 텍스처입니다. 특수 기능: 텍스처링 프로퍼티를 사용하여 텍스처를 적용하려는 각 3D 객체에 텍스처링을 활성화합니다. 그 후 [텍스처] VI를 사용하여 새 텍스처를 생성합니다. 장면 텍스처 프로퍼티와 장면 텍스처 메소드를 사용하여 텍스처를 설정합니다.

3D 객체에 텍스처를 적용하는 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

3D 객체간 관계 생성하기

3D 장면에서 **객체 사이의 관계를 생성**할 수 있습니다. 객체는 상위 또는 하위로 동작합니다. 상위 객체에 변경 사항을 적용하면 하위 객체에도 변경 사항이 반영됩니다. 큰 3D 장면과 객체의 움직임이 서로 관련된 3D 장면에서 객체 관계 기능을 사용하여 보다 효율적으로 프로그램할 수 있습니다. 예를 들어 태양계 시스템 모형에서 지구의 공전 속도 변화는 달의 공전 속도에 영향을 미칩니다. 달을 지구의 직하위로 생성하여 두 객체 모두에 대한 변환 변경 설정을 피합니다.

3D 객체 간에 관계를 생성하는 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

빛 소스 설정하기

LabVIEW는 빛 소스 설정 유무에 관계없이 보이는 3D 장면을 제공할 수 있습니다. **빛 소스를 설정**하여 3D 장면에서 햇빛이나 스포트라이트와 같은 시각 효과를 얻을 수 있습니다. 3D 장면에서 세 가지 타입의 빛을 생성할 수 있습니다: 빛 **방향**, 빛 **위치**, **스�포트라이트**. [새 빛] VI를 사용하여 새 빛 소스를 생성합니다. 또한 장면 빛 프로퍼티를 사용하여 색, 감쇠, 강도와 같은 빛 소스의 추가적인 속성을 설정할 수 있습니다.

빛 방향

빛 방향을 사용하여 3D 장면에서 햇빛 효과처럼 객체에서 무한히 멀리 있는 빛 소스를 생성할 수 있습니다. 빛 방향의 위치가 무한대이므로, 빛의 방향만 지정하면 됩니다. 방향 프로퍼티를 사용하여 빛 방향을 설정합니다.

빛 위치

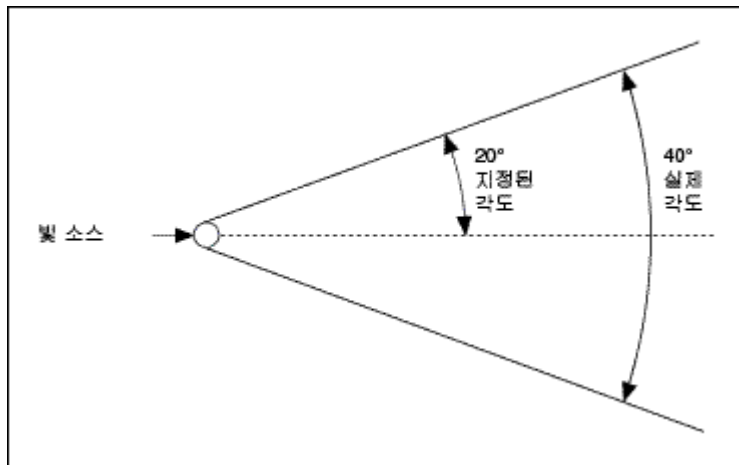
빛의 특정한 위치 또는 근원을 설정하려는 경우 빛 위치를 사용합니다. 빛 위치의 경우, 빛 소스의 위치와 빛의 방향에 대한 x, y, z 좌표를 지정하십시오. 빛의 위치를 지정하는 x, y, z 좌표는 해당 빛이 속해 있는 장면의 객체에 상대적입니다. 이때 소유 객체의 위치는 좌표 0, 0, 0을 나타냅니다. 예를 들어 빛 소스를 3D 객체 바로 아래에 놓으려는 경우, 좌표 0, -1, 0을 사용하여 빛 소스의 위치를 3D 객체의 아래의 1 객체 좌표로 설정할 수 있습니다. 다른 3D 객체가 위치 빛의 아래에 나타나는 첫번째 객체에 상대적으로 0, -3, 0의 위치에 있으며 빛의 방향이 첫번째 객체를 향하고 있다고 가정하면 빛 소스는 두번째 객체에 영향을 주지 않습니다.

3D 장면에 위치 빛을 추가하려면, 객체 참조 번호와 빛 참조 번호를 빛:빛 추가 메소드에 연결하여 빛을 3D 장면의 객체에 연결합니다. 그러면 객체가 빛을 소유하게 됩니다. 방

향 프로퍼티를 사용하여 빛 방향을 설정하고, 위치 프로퍼티를 사용하여 빛 위치를 설정합니다.

스포트라이트

빛의 위치 및 방향을 모드 지정하려는 경우 스포트라이트를 사용합니다. 스포트라이트는 방향과 각도가 정의된 빛 위치입니다. 빛을 위치 빛으로 설정하되, 컷오프 프로퍼티를 사용하여 스포트라이트의 각도를 정의합니다. 이 프로퍼티에 연결하는 값은 실제 스포트라이트 각도의 절반 값입니다. 예를 들어, 스포트라이트가 빛의 40도 각도에서 방출되게 하려면, 이 프로퍼티를 20으로 설정하십시오. 이 프로퍼티는 0 ~ 90 및 180의 값을 받습니다. 이때 180은 컷오프가 없는 상태를 나타냅니다.



빛 소스 설정에 대한 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 Using Meshes VI를 참조하십시오.

3D 장면 보기

3D 장면과 사용자 컨트롤 카메라가 상호작용하는 방식을 설정하고, 3D 장면을 디스플레이하는 다른 장면 윈도우를 설정하고, 3D 장면이 실시간 데이터를 디스플레이하도록 연속적으로 업데이트하도록 설정할 수 있습니다.

장면 카메라 설정하기

3D 그림 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 카메라 컨트롤을 선택하여 카메라가 어떻게 3D 장면과 상호작용하는지 지정합니다. 수행 도구는 3D 장면에서 이동시킬 때 카메라와 같이 동작합니다. 장면 그래프 디스플레이 프로퍼티나 장면 윈도우 프로퍼티를 사용하여 프로그램적으로 장면 카메라를 설정할 수 있습니다.

장면 윈도우 설정하기

VI를 실행할 때 3D 장면을 디스플레이하는 다른 장면 윈도우를 설정할 수 있습니다. [윈도우 설정] VI를 사용하여 새 장면 윈도우를 설정합니다. 또한 장면 윈도우 프로퍼티와 장면 윈도우 메소드를 사용하여, 3D 장면의 투영, 장면 윈도우의 크기 및 활성 상태와 같은 속성을 설정할 수 있습니다.

라이브 데이터

실시간(또는 연속적으로 변화하는) 3D 데이터를 계속 업데이트하도록 3D 장면을 설정할 수 있습니다. 루프 안에 3D 장면 포함하고 다음 방법 중 하나 사용하기:

- (권장) 루프 안의 3D 그림 또는 다시 그리려는 장면에 다시 그리기 메소드를 사용합니다.
- 새로 고치려는 장면에 3D 그림 컨트롤 터미널 또는 로컬 변수를 놓습니다.

장면 카메라와 장면 윈도우를 설정하는 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

3D 모델에 데이터 디스플레이하기

독립 센서 및 NI-DAQmx 채널을 사용하여 현실 세계의 데이터를 3D 모델에 맵핑할 수 있습니다. 센서 맵핑 익스프레스 VI를 사용하여 센서 설정 대화 상자를 통해 설정하는 센서를 나타내는 데이터 값 배열을 와이어로 연결합니다. 예를 들어, 2 개의 채널을 포함하는 NI-DAQmx 태스크를 갖고 있고 2 개의 독립 센서를 3D 모델에 놓은 경우, LabVIEW가 기대하는 데이터 값 입력은 daqsensor1, daqsensor2, freesensor1, freesensor2 입니다. 최소한 1 개의 NI-DAQmx 채널을 3D 모델에 놓는 경우, LabVIEW는 NI-DAQmx 태스크에 있는 모든 채널에 대해 각각 공간을 할당합니다. NI-DAQmx 채널의 사용에 대한 추가적인 정보는 NI-DAQmx 도움말을 참조하십시오.



노트 DAQ 어시스턴트 익스프레스 VI를 사용하여 NI-DAQmx 태스크를 생성할 수 있습니다. 그러나, 센서 맵핑 익스프레스 VI는 DAQ 어시스턴트 익스프레스 VI로부터 채널을 가져올 수 없습니다. 데이터를 센서 맵핑 익스프레스 VI의 센서 값 입력에 와이어로 연결할 수 있습니다. 센서를 놓으려면 3D 모델에서 마우스 오른쪽 버튼을 클릭하여 채널을 3D 모델에 맵핑합니다.

3D 그림 컨트롤을 익스프레스 VI의 장면 참조 출력에 와이어로 연결합니다. 익스프레스 VI에 연결하는 데이터는 3D 모델에 나타납니다. 3D 모델에 놓는 센서는 센서가 나타내는 데이터 값에 따라 센서 포인트에 있는 3D 모델의 색상에 영향을 미칩니다.

메시를 사용하여 3D 객체 그리기

3D 장면에 나타나는 객체는 물리적인 객체의 그래픽 표현입니다. 3D 객체는 3D 공간에 존재하는 정점의 세트와 그 정점 사이의 연결입니다. [상자형 생성] 및 [구형 생성]과 같은 기하 VI는 간단한 기하 객체의 형을 정의합니다. 기하학적으로 더욱 복잡한 객체를 생성하려는 경우, 또는 3D 객체의 모양을 정의하는 **정점, 법선, 인덱스, 색, 텍스처 맵핑**에 특정한 값을 정의하려는 경우, 메시를 설정할 수 있습니다.



노트

고급 3D 그래픽 디자인 기술에 메시를 사용합니다. 3D 그래프를 디자인한 경험이 있다면 이 기능을 사용하는데 도움이 될 수도 있습니다.

메시는 속성의 모음으로, 이러한 속성을 특정한 방법으로 조합하면 3D 표면을 생성할 수 있습니다. 메시의 속성에는 정점, 법선, 색, 텍스처 좌표 및 윤곽을 그리거나 정점을 연결하는 포인트, 라인, 삼각형, 다각형 등이 포함됩니다. 메시를 정의할 때, 메시의 정점, 법선, 색, 텍스처 좌표를 나타내는 값의 배열을 정의하게 됩니다. 또한 메시의 어디에 각 배열의 값을 적용할 것인지 결정하는 **인덱스** 배열도 정의할 수 있습니다. 마지막으로 각 배열의 값을 연결하는 모드를 정의합니다. 메시를 이해하려면 다음과 같은 용어를 잘 숙지해야 합니다:

- 정점—메시의 코너를 생성하는 3D 공간의 포인트. x, y, z 좌표는 각 정점을 정의합니다.
- 원형—개별적이고 보이는 메시의 표면. 예를 들어 메시가 세 개의 삼각형으로 구성되어 있다면 각 삼각형은 메시의 원형입니다. 포인트, 라인, 삼각형, 사각형, 기타 다각형이 모두 원형이 될 수 있습니다.
- 법선—메시의 표면에서 수직인 벡터. 메시의 각 정점 또는 각 다각형에 법선을 정의할 수 있습니다. 메시의 법선은 빛 소스에 상대적인 메시의 표면 방향을 정의합니다.

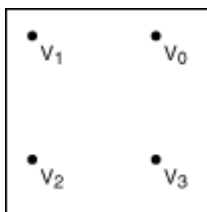
LabVIEW에서 [메시 생성] VI를 사용하여 새로운 메시의 참조를 생성할 수 있습니다. 그 후 장면 메시 프로퍼티와 메소드를 사용하여 프로그램적으로 메시지를 설정할 수 있습니다. 다음 섹션은 메시의 속성에 대한 더욱 자세한 정보를 제공합니다.

그리기 모드

메시에서 사용자가 선택하는 그리기 모드는 메시지를 그리는 방식 또는 메시의 정점을 연결하는 방식을 지정합니다. 다음 그리기 모드 중에서 하나를 선택할 수 있습니다.

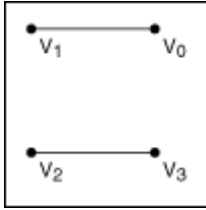
포인트

각 정점 위에 포인트를 그립니다. 이 모드는 서로 떨어져있는 정점을 연결하지는 않으나, 포인트는 3D 공간에 나타납니다.



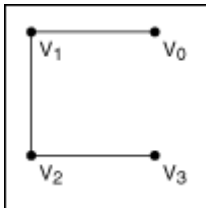
라인

두 정점 사이에 비연속적인 라인 선분을 그립니다. 예를 들어 LabVIEW는 다음과 같은 네 개의 정점과 두 개의 라인으로 메시지를 그립니다. V_0V_1 및 V_2V_3 . 정점의 총수가 홀수인 경우, LabVIEW는 마지막 정점을 무시합니다.



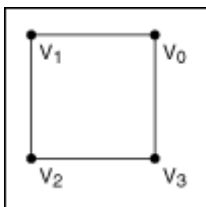
라인 스트립

정점 사이에 V_0 에서 시작해서 V_{n-1} 에서 끝나는 연속적인 라인 선분을 그립니다. 이 때 n 은 정점의 총 개수입니다. 이 모드는 모든 정점을 연결하지만 닫힌 다각형을 생성하지는 않습니다.



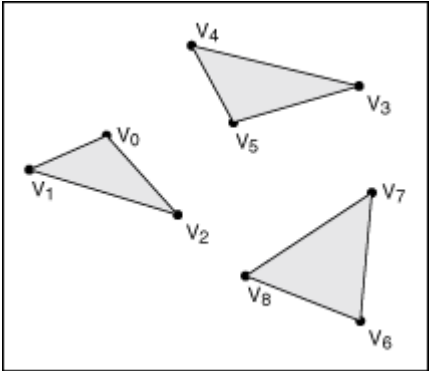
라인 루프

정점 사이에 V_0 에서 시작하고 끝나는 연속적인 라인 선분을 그립니다. 이 모드는 닫힌 다각형을 생성하지만, 라인 선분으로 둘러싸인 공간을 채우지는 않습니다.



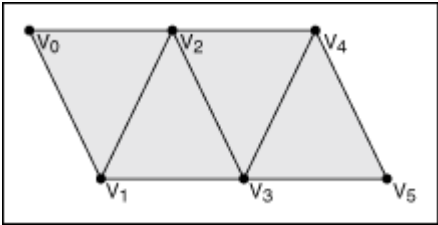
삼각형

세 개의 정점을 연결하는 비연속적인 삼각형 여러 개를 그리고 각 삼각형 내부의 공간을 채웁니다. 예를 들어 LabVIEW는 다음과 같은 여섯 개의 정점과 두 개의 삼각형으로 메시지를 그립니다. V_0, V_1, V_2 및 V_3, V_4, V_5 . 이 모드는 삼각형을 서로 연결하지 않습니다. 정점 총 개수가 3의 배수가 아닌 경우, LabVIEW는 마지막 한두 개의 정점을 무시합니다.



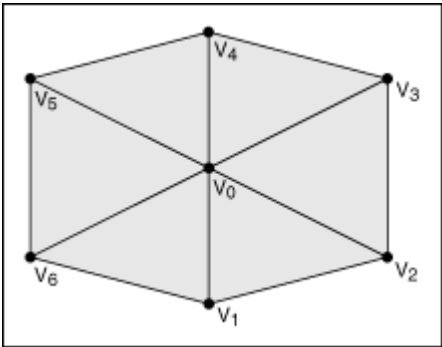
삼각 스트립

여러 개의 연속적인 속이 채워진 삼각형을 그립니다. LabVIEW는 삼각형의 방향이 실선 스트립을 형성하도록 점점을 연결하여 객체 표면의 일부분을 생성합니다. 예를 들어 첫 번째 삼각형은 정점 $V_0 V_1 V_2$ 을 연결하고, 두 번째 삼각형은 정점 $V_2 V_1 V_3$ 을 연결하고, 세 번째 삼각형은 정점 $V_2 V_3 V_4$ 을 연결하는 식입니다.



삼각 팬

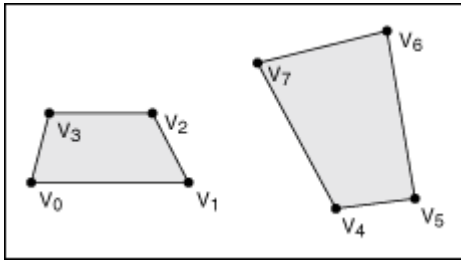
여러 개의 연속적인 속이 채워진 삼각형을 그립니다. 각 삼각형의 첫 번째 정점은 같으며 LabVIEW는 각 다음 삼각형에 두 번째와 세 번째 정점을 하나씩 증가시킵니다. 이렇게 되면 삼각형이 중심 포인트 V_0 주변에 팬을 생성합니다. 예를 들어 첫 번째 삼각형은 $V_0 V_1 V_2$ 를 연결하고, 두 번째 삼각형은 $V_0 V_2 V_3$ 을 연결하는 식입니다.



쿼드

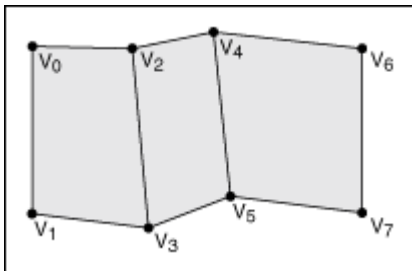
네 개의 정점을 연결하는 비연속적인 사각형을 여러 개 그리고 각 사각형 내부의 공간을 채웁니다. 예를 들어 LabVIEW는 다음과 같은 여덟 개의 정점과 두 개의 사각형으로 메시

를 그립니다. $V_0 V_1 V_2 V_3$ 및 $V_4 V_5 V_6 V_7$. 정점 총 개수가 4의 배수가 아닌 경우, LabVIEW는 마지막 1~3 개의 정점을 무시합니다.



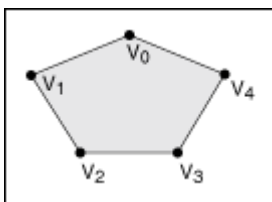
쿼드 스트립

연속적인 사각형을 여러 개 그리고 각 사각형으로 둘러싸인 공간을 채웁니다. 예를 들어 LabVIEW는 다음과 같은 여덟 개의 정점과 세 개의 사각형으로 메시를 그립니다. $V_0 V_1 V_3 V_2$, $V_2 V_3 V_5 V_4$ 및 $V_4 V_5 V_7 V_6$. 정점의 총수가 홀수인 경우, LabVIEW는 마지막 정점을 무시합니다.



다각형

V_0 에서 시작해서 V_{n-1} 에서 끝나도록 모든 정점을 연결하는 속이 채워진 다각형을 하나 그립니다. 이때 n 은 정점의 총 개수입니다. 정점의 총 개수는 3개 이상이 되어야 합니다. 그렇지 않은 경우 LabVIEW는 아무것도 그리지 않습니다. 또한 다각형은 교차할 수 없습니다.



정점 배열

메시의 정점 배열은 메시에서 각 정점의 위치를 정의합니다. 배열의 각 원소는 메시에서 하나의 정점을 나타냅니다. LabVIEW는 [인덱스](#) 배열에서 지정한 값에 따라 배열을 인덱스합니다. 정점 배열은 클러스터의 배열입니다. 이때 각 클러스터에는 3D 장면에서 한 정점의 위치를 정의하는 x, y, z 값이 포함되어 있습니다.

인덱스

인덱스는 LabVIEW가 메시지를 생성하는 여러 배열을 어떻게 인덱스하는지 정의하는 정수의 배열입니다. 예를 들어 인덱스 배열의 위치 3에 있는 원소가 0인 경우, 색 배열, **정점 배열**, **법선 배열**, **텍스처 좌표 배열**의 위치 3에 있는 원소는 모두 0으로 인덱스됩니다. 인덱스 배열을 사용하여 배열 원소 네 개의 인덱스를 변경함으로써 개별적으로 배열 내 원소의 순서를 변경하지 않고도 메시지의 모양을 변경합니다.

색 연결 모드

메시에서 선택하는 색 연결 모드는 LabVIEW가 **색 배열**의 원소를 메시에 연결하는 방법을 지정합니다. 다음 색 연결 모드 중에서 하나를 선택할 수 있습니다:

- 전체 — 색 배열의 첫 번째 인덱스에 있는 색을 메시의 모든 원형에 연결합니다.
- Per Primitive—메시의 각 원형마다 다른 색이 연결됩니다. 메시의 **그리기 모드**가 원형을 정의합니다. 예를 들어 이 메소드는 색 배열의 첫 번째 인덱스에 있는 색을 메시에 정의된 첫 번째 원형에 연결하는 식입니다.
- 정점 당—색 배열의 색을 정의하는 **인덱스**에 따라 메시의 정점에 연결합니다. 입체 다각형 또는 다각형을 제공하는 그리기 모드를 선택하는 경우, 두 정점 사이에서 움직임에 따라 한 정점에서의 색이 다음 정점에서의 색과 혼합됩니다.
- Binding Off—메시에 어떠한 색도 연결되지 않습니다. LabVIEW는 색 배열을 무시합니다.

색 배열

메시의 색 배열은 인덱스 배열의 각 인덱스에 적용할 RGBA 색 값을 정의합니다. 지정하는 **색 연결 모드**에 따라 LabVIEW가 이 배열에 색을 어떻게 적용할지 결정됩니다. 색 배열은 클러스터의 배열이며, 이때 각 클러스터에는 빨간색, 녹색, 파랑색 및 한 가지 색을 정의하는 알파값이 포함되어 있습니다.

법선 연결 모드

선택하는 법선 연결 모드는 LabVIEW가 **법선 배열**의 원소를 메시에 연결하는 방법을 지정합니다. 법선 연결 모드는 빛이 메시의 표면에 반사되는 방법을 결정합니다. 다음 법선 연결 모드 중에서 하나를 선택할 수 있습니다:

- 전체—법선 배열의 첫 번째 인덱스에 있는 법선을 메시의 모든 원형에 연결합니다.
- 원형 당—메시의 각 원형마다 다른 법선이 연결됩니다. 메시의 그리기 모드가 원형을 정의합니다. 예를 들어 이 메소드는 법선 배열의 첫 번째 인덱스에 있는 법선을 메시에 정의된 첫 번째 원형에 연결하는 식입니다.
- 정점 당—법선 배열의 법선을 정의하는 **인덱스**에 따라 메시의 정점에 연결합니다. 메시의 각 정점에 법선을 정의해야 합니다.
- 연결하지 않음—법선을 연결하지 않습니다. LabVIEW는 법선 배열을 무시합니다.

법선 배열

메시의 법선 배열은 메시에서 각 법선의 원점을 정의합니다. 배열의 각 원소는 메시에서 하나의 법선을 나타냅니다. 지정하는 **법선 연결 모드**에 따라 LabVIEW가 이 배열에 법선을 어떻게 적용할지 결정됩니다. 법선 배열은 클러스터의 배열입니다. 이 때 각 클러스터에는 한 법선의 원점을 정의하는 x, y, z 값이 포함되어 있습니다.

텍스처 좌표 배열

메시의 텍스처 좌표 배열은 **텍스처**를 메시의 표면에 맵핑하는 방법을 정의합니다. 배열의 각 원소는 하나의 텍스처 좌표를 나타냅니다. 메시의 각 정점에 텍스처 좌표를 정의해야 합니다. 텍스처 좌표 배열은 클러스터의 배열입니다. 이 때 각 클러스터에는 텍스처의 한 좌표를 정의하는 s와 t 값이 포함되어 있습니다. 텍스처링을 활성화하지 않는 경우, LabVIEW는 이 배열을 무시합니다. 장면 객체 프로퍼티 특수 기능: 텍스처링을 사용하여 3D 장면의 객체에 텍스처링을 활성화합니다.

메시의 속성 수정이 메시의 모양에 어떠한 영향을 끼치는지에 대한 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 Using Meshes VI를 참조하십시오.

LabVIEW의 파일 I/O 작업

파일 I/O

파일 I/O는 데이터를 파일 안팎으로 전달합니다. 파일 I/O 팔레트의 파일 I/O VI와 함수를 사용하여 다음을 포함한 모든 파일 I/O 기능을 다룹니다:

- 데이터 파일의 열기와 닫기
- 파일에서 데이터 읽기와 데이터 쓰기
- 스프레드시트 형식 파일 읽기와 쓰기
- 파일 및 디렉토리 이동하기와 이름 바꾸기
- 파일 특성 변경하기
- 설정 파일 만들기, 수정하기, 읽기

하나의 VI 또는 함수를 사용하여 파일을 열거나, 읽거나, 쓰거나, 닫을 수 있습니다. 또한, 함수를 사용하여 프로세스의 각 단계를 컨트롤할 수 있습니다.

파일 I/O 작업의 예제는 labview\examples\File IO를 참조하십시오.

관련 예제 찾기

파일 I/O의 기초

일반적인 파일 I/O는 다음과 같은 과정으로 진행됩니다.

1. 파일을 생성하거나 엽니다. 파일을 열고나면, 참조 번호라는 고유한 식별값으로 그 파일을 표시하게 됩니다.
2. 파일 I/O 팔레트의 VI나 함수를 사용하여 파일의 읽기나 쓰기를 수행합니다.
3. 파일을 닫습니다.

파일 I/O VI와 [텍스트 파일에서 읽기] 및 [텍스트 파일에 쓰기] 함수와 같은 일부 파일 I/O 함수는 일반적인 파일 I/O 작업의 세 단계를 모두 수행할 수 있습니다. 여러가지 작업을 하도록 디자인된 VI와 함수는 한가지 작업을 수행하는 함수보다 효율성이 떨어질 수도 있습니다.



관련 정보

[참조 번호 컨트롤과 인디케이터](#) (87 페이지 참조)

[디스크 스트리밍을 사용하여 메모리 절약하기](#) (479 페이지 참조)

[2진 파일 I/O 함수에 대한 파일 버퍼링을 비활성화하여 효율성 향상시키기](#) (480 페이지 참조)

[어떤 파일 포맷을 사용할지 결정하기](#) (481 페이지 참조)

[파일 경로](#) (482 페이지 참조)

DataPlugin을 사용하여 데이터 파일 지원 확장하기 (Windows)

DataPlugin은 통합 메소드를 제공하므로, 파일 포맷에 관계없이 LabVIEW로 데이터에 접근할 수 있습니다. DataPlugin은 특정 파일 포맷에 들어 있는 데이터를 TMD 데이터 모델로 맵핑하여 읽고 해석하는 캡슐화된 코드입니다. TMD 데이터 모델은 데이터를 파일, 채널 그룹 및 채널이라는 3개의 계층구조 레벨로 배치합니다.

DataPlugin 다운로드하기 및 등록하기

National Instruments는 일부 데이터 포맷에 대한 DataPlugin을 제공합니다. National Instruments 웹사이트의 ni.com/dataplugins에서 DataPlugin을 보고 다운로드할 수 있습니다.

DataPlugin을 다운로드한 후, 데이터 파일에 접근하기 전 반드시 로컬 컴퓨터에 등록을 해야 합니다. DataPlugin 파일에서 마우스 오른쪽 버튼을 클릭하여 DataPlugin을 직접 등록합니다. 또한 [DataPlugin 등록] VI를 사용하여 프로그램적으로 DataPlugin을 등록할 수 있습니다.

DataPlugin 생성하기

특정 데이터 포맷에 대한 DataPlugin을 찾지 못하는 경우, LabVIEW DataPlugin SDK를 사용하여 DataPlugin을 생성할 수 있습니다. 또한 SDK를 사용하여 모든 Windows 컴퓨터에 새로운 DataPlugin을 설치하는 설치 프로그램을 생성할 수 있습니다. 다음 정보는 LabVIEW DataPlugin SDK 사용자 매뉴얼을 참조하십시오:

- LabVIEW DataPlugin 타입
- LabVIEW에서 DataPlugin 생성하기
- 채널 길이 제한
- 지원되는 데이터 타입 및 변환
- 이름 규약
- 지원되는 LabVIEW 버전

DataPlugin을 사용하여 ATML 2.0 파일에서 정보 읽기에 대한 예제는 `labview\examples\File IO\DataPlugins\ATML\DataPlugins - ATML.lvproj` 파일의 Read ATML File VI를 참조하십시오.

디스크 스트리밍을 사용하여 메모리 절약하기

디스크 스트리밍을 사용하면 함수가 파일을 열고 닫기 위해 OS와 상호작용하는 횟수를 줄여 메모리 리소스를 절약합니다. 디스크 스트리밍이란 사용자가 하나의 루프 내에서 여러 개의 읽기나 쓰기 작업을 수행하는 동안 파일을 계속해서 열어두는 것을 말합니다.

[텍스트 파일에 쓰기] 함수나 [2진 파일로부터 읽기] 함수와 같이 파일에서 데이터를 읽거나 파일에 데이터를 쓰는 함수에 컨트롤이나 상수를 연결하지 말아야 합니다. 만약 그렇게 하는 경우 해당 함수 또는 VI가 실행될 때마다 파일을 열고 닫는 오버헤드가 추가됩니다. 대신 디스크 스트리밍 작업을 구현하여 이러한 오버헤드를 제거하십시오.

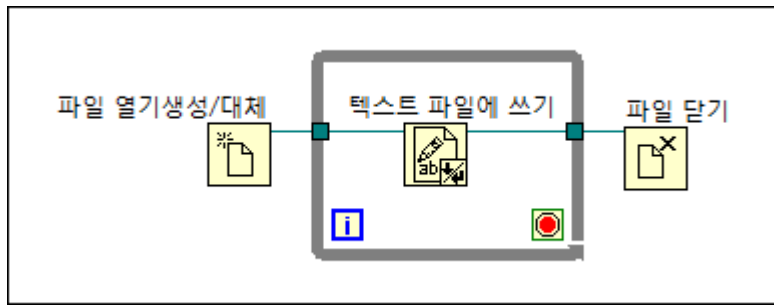
디스크 스트리밍을 사용해야 하는 경우

어플리케이션에 디스크 스트리밍 작업이 적절한지 결정할 때에는 다음 정보를 고려하십시오:

- 디스크 스트리밍은 속도에 민감한 장시간 데이터 수집 수행시 이상적입니다.
- 데이터를 수집하며 동시에 파일에 데이터를 연속적으로 기록하려면 디스크 스트리밍을 사용하십시오.

디스크 스트리밍 작업 구현하기

일반적인 디스크 스트리밍 작업을 생성하려면, 다음 블록 다이어그램과 같이 루프 바깥쪽에 [파일 열기/생성/대체] 함수 및 [파일 닫기] 함수를 놓습니다.



디스크 스트리밍 패턴에서 VI는 매 반복마다 파일을 열고 닫는 것과 관련된 오버헤드 없이 루프 내부에서 연속적으로 파일에 데이터를 작성할 수 있습니다.



팁

최상의 결과를 위해, 데이터 수집이 완료될 때까지 분석 VI와 함수 같은 다른 VI와 함수를 함께 실행하지 않습니다.

2진 파일 I/O 함수에 대한 파일 버퍼링을 비활성화하여 효율성 향상시키기

각 파일 I/O 작업에서, LabVIEW가 OS를 호출하고 디스크에 있는 파일에서 및 파일로 데이터 전송을 요청하는데에는 몇 ms의 시간이 걸립니다. 이러한 ms 단위의 시간은 LabVIEW가 OS를 호출할 때마다 누적되게 됩니다. 이렇게 반복적으로 시간이 소요되는 것을 방지하기 위하여, 대부분의 OS 파일 시스템은 버퍼를 제공하여 일시적으로 파일 읽기 또는 파일 쓰기 대기 상태인 각 데이터 조각을 보관합니다. 버퍼가 가득차면, OS는 하나의 파일 I/O 작업을 수행합니다.

위에서 설명한 프로세스를 '버퍼링'이라고 하며, LabVIEW는 기본으로 버퍼링을 활성화합니다. 버퍼링은 OS가 디스크를 사용하는 횟수 및 이에 소요되는 처리 시간을 줄여줍니다.

버퍼링을 비활성화해야 하는 경우

일부 경우에서 버퍼링을 비활성화하여 데이터 스트리밍 속도를 향상시킬 수 있습니다. 예를 들어, RAID(Redundant Array of Independent Disks)를 사용하여 프로세스 속도를 높일 수 있습니다. RAID는 단일 디스크처럼 동작하는 여러 하드 디스크의 모음으로, OS가 동시에 사용할 수 있어서 데이터를 읽고 쓰는데 소요되는 시간이 적습니다. 버퍼링이 활성화된 상태에서 RAID를 사용하면, OS가 실제로 디스크에 데이터를 쓰는데 소요되는 시간보다도 LabVIEW가 OS에 데이터를 복사하는데 시간이 더 많이 필요할 수도 있습니다. [파일 열기/생성/대체] 함수의 버퍼링 비활성화 입력에서 버퍼링을 비활성화하여, 이렇게 데이터를 복사하지 않고 강제적으로 OS가 디스크에 직접 데이터를 전송하도록 만들 수 있습니다.

하지만, 버퍼링을 비활성화하는 경우, 다음의 조건이 참이어야 합니다.

- 파일에 있는 데이터의 크기가 파일을 포함하고 있거나 앞으로 포함하게 될 디스크의 섹터 크기의 배수가 되어야 합니다. 이때, 디스크의 섹터 크기는 바이트로 표기됩니다. 섹터는 디스크 공간의 일부로서 주로 512 바이트의 데이터, 즉 지정된 양의 데이터를 저장합니다. [볼륨 정보 얻기] 함수의 섹터 크기(바이트) 출력을 사용하여 디스크의 섹터 크기를 확인할 수 있습니다. LabVIEW가 디스크에 데이터를

저장할 때, 데이터가 여러 섹터를 차지할 수 있지만 각각의 섹터가 완전히 채워져야 합니다. 512 바이트 섹터는 512 바이트의 데이터를 필요로 합니다. 데이터 크기가 섹터 크기의 배수가 아닌 경우, 데이터를 채우기용 데이터로 늘린 후 LabVIEW가 해당 파일을 읽기 전에 채우기용 데이터를 제거해야 합니다.

- 파일에 있는 데이터는 디스크가 요구하는 정렬 기준의 배수에 맞춰서 정렬되어야 합니다. LabVIEW가 데이터를 정렬하며, 사용자가 정렬 기준을 변경할 수 없습니다. 데이터가 정렬 기준을 충족하지 못하는 경우, LabVIEW는 에러를 반환하며 사용자가 버퍼링을 활성화하여 파일을 다시 열어야 합니다.

관련 정보

데이터 버퍼링하기

어떤 파일 포맷을 사용할지 결정하기

파일 I/O 팔레트의 VI들은 파일의 포맷에 따라 사용합니다. 파일 읽기 또는 파일에 쓰기는 다음 포맷에서 가능합니다:

- 텍스트—Microsoft 메모장과 같은 다른 어플리케이션에서 사용할 수 있는 데이터를 만들려면, 가장 일반적이며 통용 가능한 텍스트 파일을 사용합니다. LabVIEW에서는 다음 타입의 파일이 텍스트 파일에 속합니다:
 - 설정 파일
 - Excel 파일

노트

Excel 파일 .xlsx는 텍스트 편집기에서 직접 열 수는 없지만, 여러 개의 XML 파일로 이루어진 압축(Zipped) 파일입니다. 각 XML 파일은 텍스트 파일 포맷입니다.
- LabVIEW 측정 파일
- 스프레드시트 파일
- XML 파일
- 2진—무작위 접근 파일 읽기나 쓰기를 수행해야 하거나 수행 속도 및 디스크 공간이 중요한 경우, 텍스트 파일에 비해 디스크 공간이나 속도에서 더 효율적인 2진 파일을 선택합니다. LabVIEW에서는 다음 타입의 파일이 2진 파일에 속합니다:
 - TDM/TDMS 파일
 - 웨이브폼
 - 압축(Zip) 파일
- 데이터로그—LabVIEW에서 복잡한 데이터 레코드 또는 서로 다른 데이터 타입을 조작하려면, 데이터로그 파일 형식을 사용합니다. 데이터로그 파일은 LabVIEW로만 데이터에 접근하고, 복잡한 데이터 구조를 저장해야 하는 경우 가장 좋은 방법입니다.

(Windows) LabVIEW를 사용하여 위에 나열된 파일 포맷 이외의 포맷을 갖는 데이터에 접근하려면 DataPlugin을 사용하십시오. DataPlugin은 통합 메소드를 제공하므로, 파일 포맷에 관계없이 LabVIEW로 데이터에 접근할 수 있습니다.



- 관련 정보
- [텍스트 파일](#) (490 페이지 참조)
 - [설정 파일](#) (484 페이지 참조)
 - [스프레드시트 파일](#) (487 페이지 참조)
 - [LabVIEW 측정 파일](#) (836 페이지 참조)
 - [스프레드시트 파일](#) (487 페이지 참조)
 - [XML 파일](#) (492 페이지 참조)
 - [2진 파일](#) (483 페이지 참조)
 - [TDM/TDMS 파일](#) (841 페이지 참조)
 - [웨이브폼](#) (515 페이지 참조)
 - [Zip 파일](#) (498 페이지 참조)
 - [데이터로그 파일](#) (485 페이지 참조)
 - [DataPlugin을 사용하여 데이터 파일 지원 확장하기\(Windows\)](#) (478 페이지 참조)

파일 경로

파일 경로 컨트롤은 다음과 같이 디스크 상의 파일의 위치를 식별하는 LabVIEW 데이터 타입입니다.



파일 경로는 파일의 크기, 파일 시스템의 최상위 레벨에서부터 현재 파일까지의 디렉토리 및 파일명을 설명합니다. 현재 플랫폼의 표준 구문으로 경로 컨트롤 또는 인디케이터에 파일 경로를 입력하거나 디스플레이합니다.

다음 테이블은 LabVIEW에서 사용할 수 있는 여러 파일 경로 타입에 대한 정보를 제공합니다.

타입	설명	사용
절대 경로	절대 경로는 파일 시스템의 최상위 레벨에서 시작하여 파일 또는 디렉토리의 위치를 나타냅니다. 예: E:\Temp\test.txt.	절대 경로를 사용하여 현재 작업 디렉토리와 관계없이 파일 시스템에서 동일한 위치를 가리킵니다.

타입	설명	사용
상대 경로	상대 경로는 파일 시스템의 임의의 위치를 기준으로 파일 또는 디렉토리의 위치를 나타냅니다. 예: ..\test.txt.	상대 경로를 사용하여 현재 작업 디렉토리와 함께 변경될 수 있는 위치를 가리킵니다. 상대 경로를 사용하면 다른 컴퓨터에서 VI를 실행하거나 어플리케이션을 빌드할 때 경로에 대해 재작업을 하는 것을 피할 수 있습니다.
기호형 경로 또는 가상 경로	기호형 경로 또는 가상 경로는 LabVIEW에서 사용할 수 있는 경로의 미리 정의된 짧은 경로입니다. 예를 들어 <vilib> 기호형 경로는 labview 디렉토리에서 <vi.lib> 디렉토리를 나타냅니다. 런타임 때 LabVIEW는 설치된 위치에 따라 기호형 경로 <vilib>를 C:\Program Files (x86)\National Instruments\LabVIEW 2009\vi.lib 또는 D:\Program Files\National Instruments\LabVIEW 2009\vi.lib와 같이 절대 경로로 해석합니다.	기호형 경로를 사용하여 LabVIEW가 설치된 디스크 또는 로그인한 사용자 등과 같은 실행 환경에 따라 변경되는 위치를 가리킬 수 있습니다.

2진 파일

정수와 같은 2진 데이터로 저장하면 디스크에서 고정된 크기의 바이트 수를 사용합니다. 예를 들어, 1, 1,000, 또는 1,000,000과 같이 0에서 40억 사이의 숫자를 2진 포맷으로 저장하면 각각의 숫자는 4 바이트를 차지합니다.

숫자 데이터를 파일에 저장한 후, 해당 파일에서 특정 숫자 또는 임의의 숫자에 접근하려면 2진 파일을 사용합니다. 2진 파일은 사람이 판독 가능한 텍스트 파일과는 달리 머신에서만 판독 가능합니다. 2진 파일은 데이터를 저장하기에 가장 간결하고 빠른 포맷입니다. 2진 파일에 여러 개의 데이터 타입을 저장할 수 있지만, 그러한 경우의 빈도는 높지 않습니다.

2진 파일은 데이터를 저장하고 불러올 때 데이터를 텍스트 형으로 변환할 필요가 없고 디스크 공간도 덜 차지하므로 더 효율적입니다. 2진 파일은 1 바이트의 디스크 공간으로 256개의 값을 표현할 수 있습니다. 일반적으로 2진 파일은, 확장형 실수나 복소수 값의 경우를 제외하고, 데이터의 바이트 단위 이미지를 포함한 채로 **메모리에 저장**됩니다. 데이터의 바이트 단위 이미지를 포함한 파일이 메모리에 저장되어 있는 경우, 파일 데이터의 형변환이 필요 없으므로 파일을 읽는 속도가 더 빨라집니다.



노트

텍스트와 2진 파일은 바이트 스트림 파일이므로, 데이터가 순차적으로 문자 또는 바이트의 형태로 저장됩니다.

파일 I/O VI와 함수를 사용하여 2진 파일에서 **읽기** 및 **쓰기**를 수행합니다. 파일에서 숫자 데이터를 읽거나 파일에 숫자 데이터를 쓰려는 경우, 또는 여러 OS에서 사용할 수 있는 텍스트 파일을 생성하려는 경우, 2진 파일 함수 사용을 고려하십시오.

배정도 부동소수 값의 배열을 2진 파일로부터 읽고 2진 파일에 쓰는 예제는 각각 labview\examples\File IO\Binary\Simple Binary 디렉토리에 있는 Simple Binary File.lvproj를 참조하십시오.

관련 정보

2진 파일 생성하기

2진 파일로부터 읽기

2진 파일에 쓰기

2진 파일 생성하기

[2진 파일에 쓰기] 함수를 사용하여 2진 파일을 생성합니다. 어떤 데이터 타입도 [2진 파일에 쓰기] 함수의 데이터 입력에 연결할 수 있습니다. [2진 파일로부터 읽기] 함수를 사용하여 읽으려는 파일 데이터의 데이터 타입을 지정합니다. 해당 데이터 타입의 컨트롤 또는 상수를 [2진 파일로부터 읽기] 함수의 데이터 타입 입력에 연결합니다. [2진 파일에 쓰기]와 [2진 파일로부터 읽기] 함수를 사용하여 다른 OS에서 생성된 텍스트 파일을 읽고 쓸 수 있습니다.

[2진 파일로부터 읽기] 함수 및 [2진 파일에 쓰기] 함수 사용의 예제는 labview\examples\File IO\Binary\Simple Binary 디렉토리에 있는 Simple Binary File.lvproj를 참조하십시오.

설정 파일

설정 파일 VI를 사용하여 Windows 설정 셋팅 .ini 파일을 읽고 생성하며, 경로와 같은 플랫폼 특성의 데이터를 플랫폼에 독립적인 포맷으로 써서 이 VI가 생성하는 파일을 여러 플랫폼에서 사용할 수 있도록 합니다. 설정 파일 VI는 **설정 셋팅 파일 포맷**을 사용합니다. 모든 플랫폼에서 [설정 파일] VI가 생성한 파일을 읽고 쓸 수 있지만, [설정 파일] VI를 사용하여 macOS 또는 Linux 포맷의 설정 파일을 생성하거나 수정할 수는 없습니다.



노트

ANSI 포맷인 경우에만 Windows 설정 셋팅 파일에 설정 파일 VI를 사용할 수 있습니다.

설정 파일 VI를 사용하는 예제는 labview\examples\File IO\Configuration (INI)\Configuration (INI) File.lvproj를 참조하십시오.



노트 Windows **설정 셋팅** 파일의 표준 확장자는 .ini이지만 [설정 파일] VI는 어떠한 확장자를 갖더라도 포맷만 정확하면 호환됩니다.

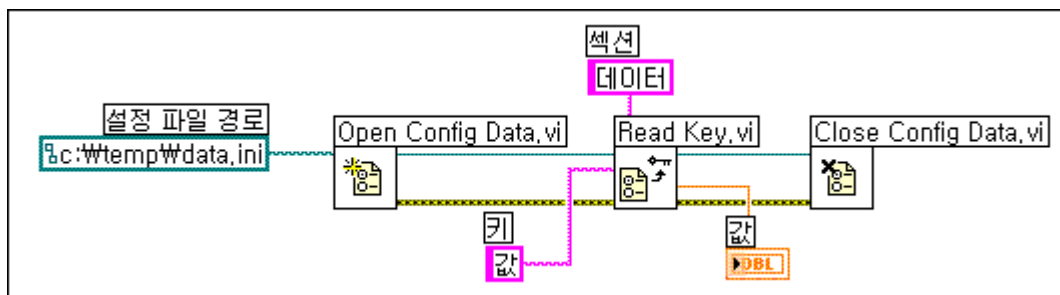
표준 Windows 설정 셋팅 파일은 텍스트 파일에 데이터를 저장하기 위한 특정한 포맷입니다. .ini 파일은 특정한 포맷을 따르므로 데이터에 프로그램적으로 쉽게 접근할 수 있습니다.

예를 들어, 다음 내용의 설정 셋팅 파일을 고려해봅시다:

[Data]

Value=7.2

다음 블록 다이어그램과 같이, 이 데이터를 읽기 위해 [설정 파일] VI를 사용할 수 있습니다. 이 VI는 [키 읽기] VI를 사용하여 데이터라는 섹션에서 값이라는 키를 읽습니다. 파일이 Windows 설정 셋팅 파일 포맷을 유지한다면, 이 VI는 파일의 변화에 무관하게 동작합니다.



관련 정보

[Windows 설정 셋팅 파일 포맷](#)

[설정 파일 VI 사용하기](#)

데이터로그 파일

데이터로그 파일을 사용하여 LabVIEW에서만 데이터에 접근하고 사용하며 복잡한 데이터 구조를 빠르고 쉽게 저장합니다.

데이터로그 파일은 각 행이 하나의 레코드를 나타내는 스프레드시트와 유사하게 데이터를 구조화된 레코드의 시퀀스로 저장합니다. 데이터로그 파일에서 각 레코드는 연관된 동일한 데이터 타입을 유지해야 합니다. LabVIEW는 저장할 데이터를 포함한 클러스터로 각 레코드를 파일에 기록합니다. 반면, 데이터로그 레코드의 구성요소는 파일 생성 시 임의의 데이터 타입으로 결정하여 저장할 수 있습니다.

예를 들어, 레코드 데이터 타입이 문자열과 숫자의 클러스터인 데이터로그를 생성할 수 있습니다. 그러면, 각 데이터로그의 레코드는 문자열과 숫자의 클러스터입니다. 그러나, 첫 번째 레코드가 ("abc",1)일 때, 두 번째 레코드는 ("xyz",7)이 될 수 있습니다.

결국 데이터로그 파일의 레코드 데이터 타입을 변경하기로 결정하게 될 수도 있습니다. 그렇게 하려면, 관련 레코드를 조작하기 위해 생성했던 VI는 모두 새로운 데이터 타입을

읽을 수 있도록 업데이트해야 합니다. 그러나, 일단 VI가 업데이트되면, 이전 레코드 데이터 타입으로 생성된 파일을 더이상 읽을 수 없게 됩니다.

데이터로그 파일을 읽거나 쓸 때, 데이터 변환이 거의 필요하지 않으므로 처리 속도가 매우 빨라집니다. 파일의 선행 레코드의 모든 레코드를 읽지 않아도 원래 데이터 블록을 레코드로 읽을 수 있으므로 데이터 검색이 빠르고 간단해집니다. 데이터로그 파일에서는 레코드에 접근하려면 레코드 번호만 알면 되므로 무작위 접근이 빠르고 쉽습니다. LabVIEW는 데이터로그 파일 생성시 각각의 레코드마다 순차적으로 레코드 번호를 할당합니다.

프런트패널과 블록다이어그램에서 **데이터로그 파일에 접근**할 수 있습니다.

LabVIEW는 데이터로그와 관련된 VI가 실행할 때마다 데이터로그 파일에 레코드를 기록합니다. LabVIEW가 데이터로그 파일에 레코드를 로그한 후에는 레코드를 덮어쓸 수 없습니다. 데이터로그 파일을 읽을 때, 한 번에 하나 또는 그 이상의 레코드를 읽을 수 있습니다.

데이터로그 파일을 생성하는 다른 방법은 다른 VI와 리포트에서 사용할 데이터를 기록하는 **프런트패널 데이터 로깅**을 사용하는 것입니다.

데이터로그 파일을 읽고 쓰는 예제는 labview\examples\File IO\Datalog\Datalog File.lvproj를 참조하십시오.

관련 정보

데이터로그 파일 생성하기

데이터로그 파일로부터 읽기

데이터로그 파일에 쓰기

프런트패널 데이터 로깅하기

데이터로그 파일 생성하기

데이터로그 파일을 생성하고 읽으려면, **프런트패널 데이터 로깅**을 활성화하거나 [데이터로그] 함수를 사용하여 데이터를 수집하고 파일에 데이터를 씁니다.

데이터로그 파일에서는 데이터를 포맷할 필요가 없습니다. 그러나 데이터로그 파일을 **읽거나 쓸** 때, 반드시 데이터 타입을 지정해야 합니다. 예를 들어, 시간과 날짜를 온도 데이터와 함께 기록하고자 한다면, 데이터로그 파일에 데이터를 쓰고 한 개의 숫자와 두 개의 문자열을 가진 클러스터로 이 데이터를 지정합니다.

온도, 시간, 날짜가 함께 존재하는 파일을 읽는다면, 하나의 숫자와 두 개의 문자열이 있는 클러스터로 데이터 타입을 지정합니다.

데이터로그 타입

데이터로그 파일에서 각 레코드의 데이터 타입은 모든 타입이 될 수 있습니다. 그러나, 데이터 타입은 데이터로그 파일에 데이터를 로그하는 방법에 따라 변합니다.

[데이터로그 쓰기] 함수를 이용하여 생성한 데이터로그 파일에서, 데이터 타입은 데이터로그 파일에 LabVIEW가 쓴 데이터와 일치합니다.

프런트패널 데이터 로깅을 이용하여 생성한 데이터로그 파일에서, 데이터 타입은 두 개의 클러스터를 가지는 클러스터입니다. 첫 번째 클러스터는 타임스탬프를 가지고 있고, 두 번째 클러스터는 프런트패널 데이터를 가지고 있습니다. 타임스탬프는 초를 나타내는 32비트의 부호없는 정수형과 LabVIEW 시스템 시간에서 경과된 밀리초를 나타내는 16비트의 부호없는 정수형을 가지고 있습니다. 프런트패널 데이터 클러스터는 각각의 탭 순서를 가진 모든 프런트패널 컨트롤과 인디케이터에 대응하는 데이터 타입을 가지고 있습니다.

스프레드시트 파일

스프레드시트 파일은 데이터의 1D 또는 2D 배열을 표의 형식, 또는 열과 행으로 구성합니다. 테이블의 각 셀은 1D 또는 2D 배열의 원소를 포함합니다.

텍스트 기반 스프레드시트 파일

텍스트 기반 스프레드시트 파일은 텍스트 파일의 한 부분입니다. 데이터를 스프레드시트에 쓰려면 데이터를 스프레드시트 문자열로 포맷해야 하는데, 이는 탭이나 쉼표와 같은 구분 문자를 포함한 문자열입니다. 스프레드시트에서 데이터를 읽은 후 데이터는 스프레드시트 문자열의 세트가 됩니다. 해당 문자열을 스프레드시트 파일이 원하는 데이터 타입으로 포맷해야 합니다. 파일 I/O VI 및 함수는 구분된 문자열이 있는 텍스트 기반 스프레드시트 파일만을 지원합니다.

2진 기반 스프레드시트 파일

2진 기반 스프레드시트 파일에는 텍스트가 아닌 2진 데이터가 포함되어 있습니다. 2진 스프레드시트 파일 포맷은 다양합니다. LabVIEW에서 타사 스프레드시트 파일의 2진 데이터를 읽으려면, DataPlugin을 사용하십시오.

[측정 파일에 쓰기] 익스프레스 VI를 사용하여 수집한 다이나믹 데이터를 Microsoft Excel 파일 .xlsx에 저장할 수 있습니다. 이 익스프레스 VI는 국제 표준 ISO/IEC 29500:2008을 따르는 Office Open XML 파일 포맷을 사용하여 Excel 파일을 생성합니다. ISO/IEC 29500:2008 표준에 대한 정보는 국제 표준화 기구(ISO) 웹사이트인 www.iso.org를 참조하십시오.



노트

어플리케이션이 Excel 파일 포맷을 사용하는 경우, 어플리케이션을 이전 버전의 LabVIEW에 대해 저장할 수 없습니다. 오래된 버전에 대해 저장하기 전, 반드시 다른 파일 포맷을 사용하도록 어플리케이션을 수정하여야 합니다.

Excel 파일로 데이터를 쓰기 위해 Microsoft Excel을 설치할 필요는 없습니다. 데이터를 Excel 파일로 보려면, 반드시 Office Open XML 파일 포맷을 읽을 수 있는 스프레드시트 뷰어를 사용해야 합니다.

각 스프레드시트 뷰어에는 볼 수 있는 데이터의 행과 열의 개수에 대한 최대값이 정해져 있습니다. Excel 파일에 쓰는 데이터의 양이 최대값을 초과하는 경우 에러가 발생할 수 있습니다. Excel 파일에서 볼 수 있는 최대값에 관한 정보는 각 스프레드시트 뷰어의 문서를 참조하십시오.

Excel 파일에 데이터를 쓴 후, 해당 Excel 파일을 DIAdem 2010 또는 이후 버전과 같은 다른 National Instruments 소프트웨어 어플리케이션에 반입할 수 있습니다. Excel 파일 반입에 관한 정보는 DIAdem 문서를 참조하십시오.

[측정 파일에 쓰기] 익스프레스 VI를 사용하여 데이터를 Excel 파일에 쓰는 예제는 labview\examples\File I/O\Spreadsheet\Excel 폴더의 Write Excel File VI를 참조하십시오.

관련 정보

DataPlugin

스프레드시트 파일 생성하기

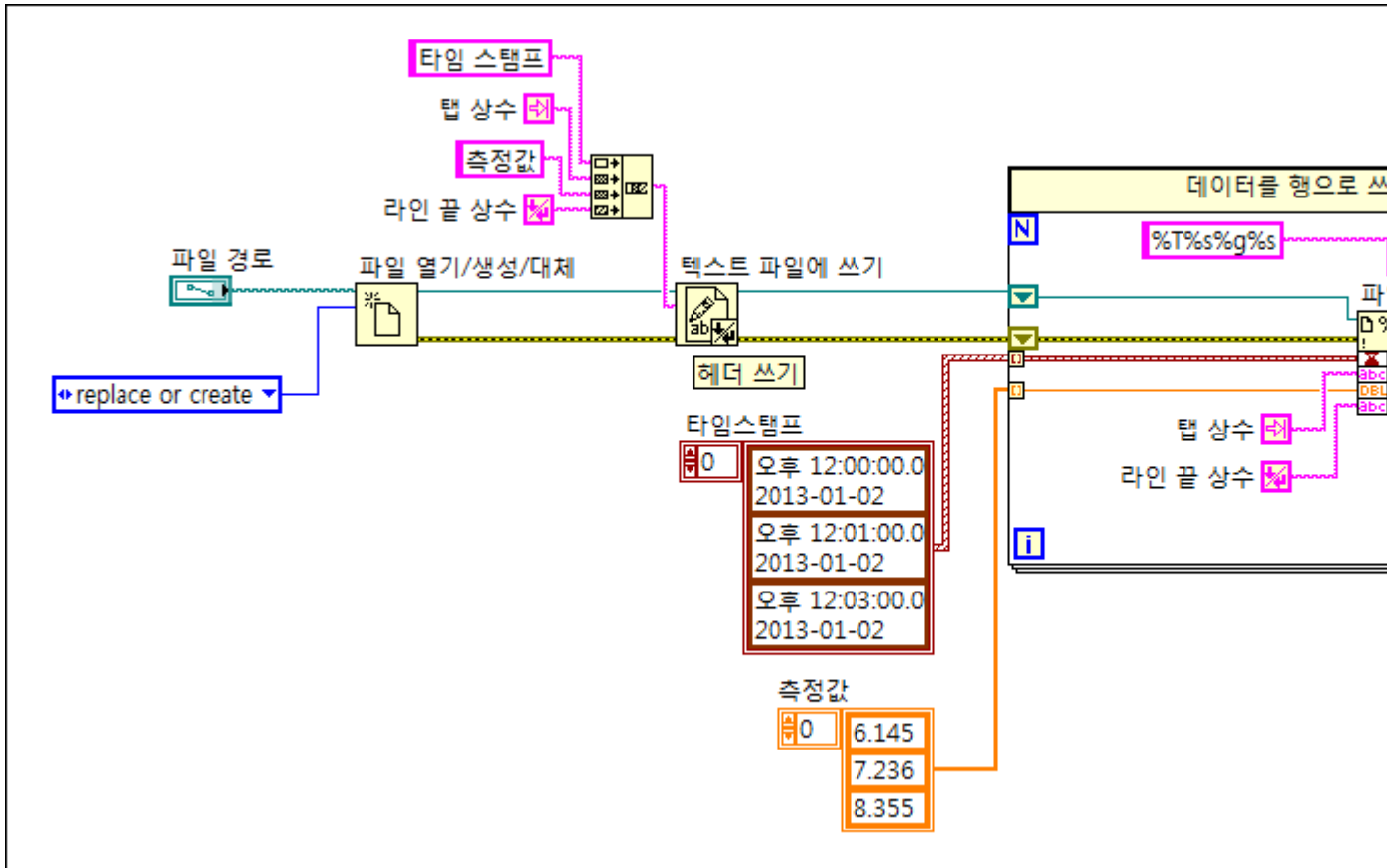
스프레드시트 파일로부터 읽기

스프레드시트 파일에 쓰기

스프레드시트 파일 생성하기

텍스트 기반 스프레드시트 파일은 텍스트 파일의 한 부분입니다. 데이터를 스프레드시트에 쓰려면 문자열 데이터를 스프레드시트 문자열로 포맷해야 하며, 이는 탭과 같은 구분 문자를 포함한 문자열입니다.

다음 VI는 여러 다양한 데이터 타입을 텍스트 파일에 작성합니다. [텍스트 파일에 쓰기] 함수는 문자열 데이터와 구분 문자를 포함한 헤더를 작성합니다. [파일로 포맷] 함수는 타임스탬프 데이터, 숫자형 데이터, 구분 문자를 스프레드시트 문자열로 포맷하고 해당 문자열을 파일에 작성합니다.



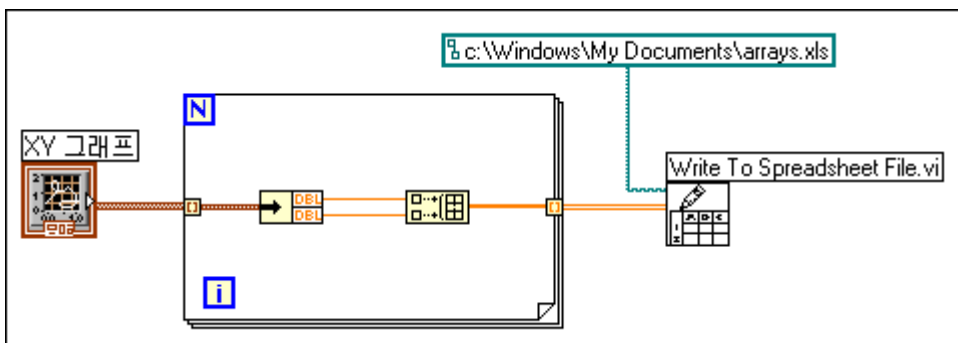
그래프나 차트에 나타난 숫자 세트나 수집한 데이터를 스프레드시트 문자열로 변환하기 위해서는 [구분된 스프레드시트 쓰기] VI 또는 [배열을 스프레드시트 문자열로] 함수를 사용합니다.



노트

Microsoft Excel 파일(.xlsx)의 경우, [측정 파일에 쓰기] 익스프레스 VI를 사용하여 수집한 다이내믹 데이터를 Excel 파일로 저장합니다.

다음 VI는 그래프로부터 데이터를 수집하고 이를 스프레드시트 파일에 씁니다.



워드 프로세싱 어플리케이션에서 생성한 텍스트를 읽다가 에러가 발생하는 경우는 파일 I/O VI가 처리할 수 없는 폰트, 색상, 스타일, 그리고 사이즈로 텍스트가 포맷되었기 때문입니다.

숫자와 문자를 스프레드시트 또는 워드 프로세싱 어플리케이션에 쓰고자 할 경우, 문자열 함수 또는 배열 함수로 데이터를 포맷하고 문자열끼리 조합합니다. 그 후 파일에 데이터를 기록합니다.



관련 정보

[텍스트 파일](#) (490 페이지 참조)

[문자열을 이용한 데이터의 그룹화](#) (410 페이지 참조)

[스프레드시트 파일](#) (487 페이지 참조)

[문자열을 이용한 데이터의 그룹화](#) (410 페이지 참조)

[배열과 클러스터를 이용한 데이터의 그룹화](#) (417 페이지 참조)

[배열](#) (417 페이지 참조)

[클러스터](#) (422 페이지 참조)

텍스트 파일

파일 I/O 속도와 디스크 사용 공간이 문제되지 않는 경우, 무작위 접근 읽기 또는 쓰기를 수행하지 않아도 되는 경우, 그리고 숫자 정밀도가 중요하지 않은 경우에는 텍스트 포맷 파일을 사용하여 다른 사용자나 어플리케이션과 공유합니다.

텍스트 파일은 사용과 공유가 가장 쉬운 포맷입니다. 대부분의 컴퓨터는 텍스트 파일을 읽고 쓸 수 있습니다. 텍스트 기반 파일을 읽을 수 있는 텍스트 기반 프로그램은 다양합니다. 대부분의 인스트루먼트 컨트롤 어플리케이션들은 텍스트 문자열을 사용합니다.

워드 프로세싱이나 스프레드시트와 같은 다른 어플리케이션에서 데이터에 접근하려면 텍스트 파일로 데이터를 저장합니다. 데이터를 텍스트 포맷으로 저장하려면, 모든 데이터를 텍스트 문자열로 변환하는 [문자열] 함수를 사용합니다. 텍스트 파일은 여러 다른 데이터 타입으로 구성된 정보를 포함할 수 있습니다.

그래프나 차트 데이터처럼 원래의 형식이 텍스트가 아닐 때, 텍스트 파일은 2진이나 데이터로그 파일보다 더 큰 메모리를 차지합니다. 데이터를 ASCII 형으로 나타내면 일반적으로 데이터 자체보다 더 크기 때문입니다. 예를 들어, -123.4567는 단정도 부동소수로 4 바이트에 저장할 수 있습니다. 그러나 ASCII 형은 각 문자마다 하나의 바이트, 즉 9 바이트를 차지합니다.

또한 텍스트 파일에서는 숫자형 데이터에 무작위로 접근하는 것이 어렵습니다. 문자열에서 각각의 문자는 정확히 1 바이트씩 차지하지만, 숫자를 텍스트로 표현하는 데 필요한 공간은 고정적이지 않습니다. 텍스트 파일에서 9번째 숫자를 찾으려면, LabVIEW는 먼저 앞의 8개 숫자를 읽고 변환해야 합니다.

숫자 데이터를 텍스트 파일로 저장하는 경우, 정밀도가 떨어질 수 있습니다. 컴퓨터는 숫자 데이터를 2진 데이터로 저장하고, 사용자는 일반적으로 숫자형 데이터를 10진 표기법으로 텍스트 파일에 씁니다. 텍스트 파일에 데이터를 쓸 때, 정밀도가 떨어질 수 있습니다. 2진 파일에서는 정밀도 손실의 문제가 없습니다.

파일 I/O VI와 함수를 사용하여 텍스트 파일에서 **읽기** 또는 **쓰기**를 수행하거나 스프레드시트 파일에서 **읽기** 또는 **쓰기**를 수행합니다.

텍스트 파일을 가지고 파일 I/O를 사용하는 예제에 대해 다음 VI를 참조하십시오:

- labview\examples\File I/O\Text (ASCII)\Text (ASCII) Files.lvproj
- labview\examples\File I/O\Spreadsheet\Tab-Delimited Data\Tab-Delimited Data.lvproj

관련 정보

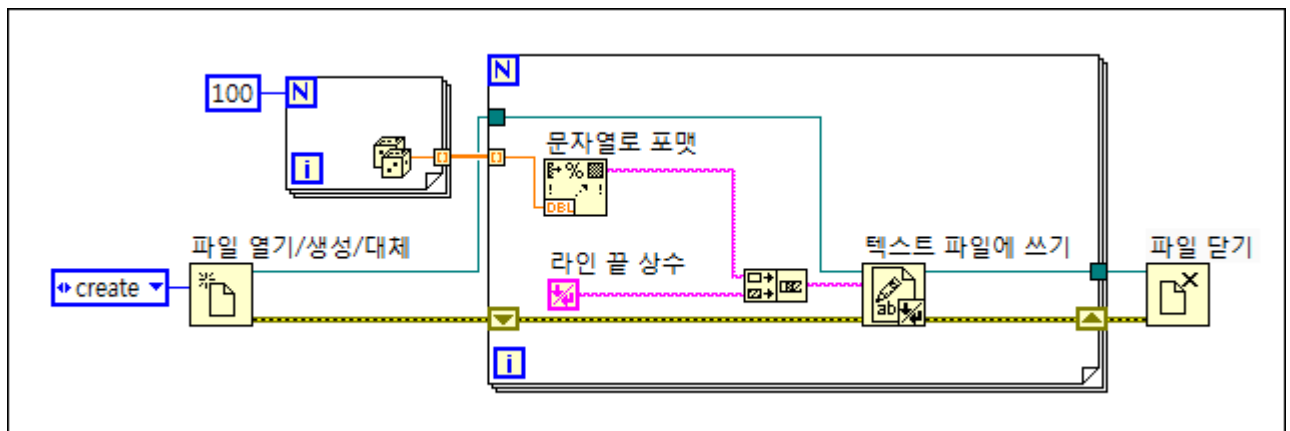
텍스트 파일 생성하기

텍스트 파일에서 읽기

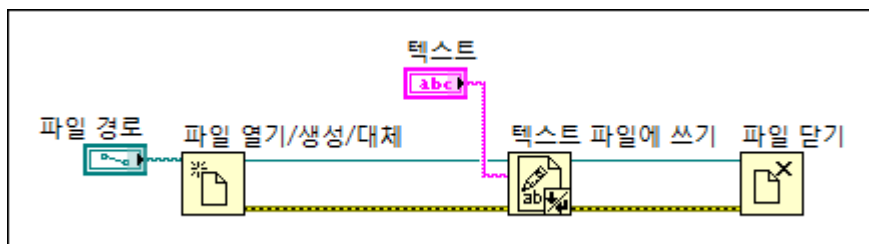
텍스트 파일에 쓰기

텍스트 파일 생성하기

텍스트 파일에 데이터를 쓰려면, 반드시 데이터를 문자열 데이터 타입으로 변환해야 합니다. 문자열 함수를 사용하여 데이터를 문자열로 변환합니다. 다음 VI는 임의로 생성한 수의 배열을 문자열로 변환하고, 해당 문자열을 텍스트 파일에 작성합니다.



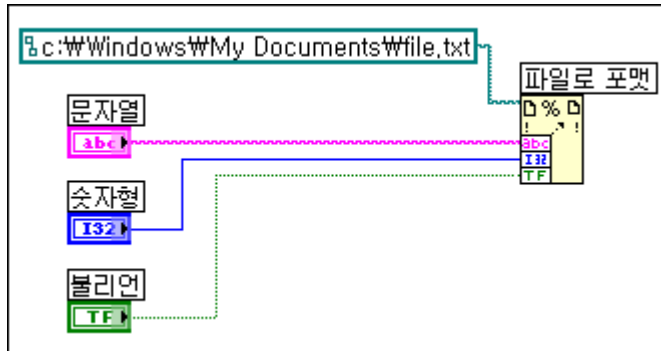
대부분의 워드 프로세싱 어플리케이션들은 텍스트를 읽기 위해 포맷된 텍스트를 요구하지 않기 때문에 포맷하지 않고 텍스트 파일에 텍스트를 씁니다. 텍스트 파일에 텍스트 문자열을 쓰려면 다음 VI와 같이 파일을 자동으로 열고 닫는 [텍스트 파일에 쓰기] 함수를 사용합니다.



데이터를 파일에 기록하기와 포맷하기

[파일로 포맷] 함수를 사용하여 문자열, 숫자, 경로 및 불리언 데이터를 텍스트로 포맷하고 포맷된 텍스트를 파일에 씁니다.

다음 vi는 여러 데이터 타입을 수집하고 이를 파일에 씁니다.



[문자열로 포맷] 함수로 문자열을 포맷하고 [텍스트 파일에 쓰기] 함수로 결과 문자열을 쓰는 대신 이 함수를 자주 사용할 수 있습니다.

파일에서 데이터를 스캔하기

파일에서 문자열, 숫자, 경로, 불리언 값 등의 원하는 텍스트를 찾으려면 [파일로부터 스캔] 함수를 사용하고 텍스트를 데이터 타입으로 변환합니다. [2진 파일로부터 읽기] 함수 또는 [텍스트 파일에서 읽기] 함수로 파일에서 데이터를 읽고 [문자열로부터 스캔] 함수로 결과 문자열을 스캔하는 대신 이 함수를 자주 사용할 수 있습니다.



관련 정보

[문자열을 이용한 데이터의 그룹화](#) (410 페이지 참조)

XML 파일

XML(Extensible Markup Language)은 태그를 이용하여 데이터를 표현하는 포매팅 표준의 하나입니다. HTML과는 다르게, XML 태그는 어떻게 데이터 부분을 포맷할 것인지 브라우저에게 지시하지 않습니다. 대신에, XML 태그는 데이터 부분을 식별합니다.

예를 들어, 당신이 웹에서 책을 판매하는 책 판매원이라고 가정합니다. 도서관에서 다음과 같은 기준으로 책을 분류하기를 원합니다:

- 책의 타입(소설 또는 비소설)
- 제목
- 저자
- 출판사
- 가격
- 장르

- 줄거리
- 페이지 수

개별 책에 대한 XML 파일을 만들 수 있습니다. 독일의 대성당 여행이라는 제목을 가진 책의 XML 파일은 다음과 비슷하게 될 것입니다:

```
<nonfiction>
```

```
<Title>독일의 대성당 여행<Title>
```

```
<Author>Tony Walters<Author>
```

```
<Publisher>Douglas Drive Publishing<Publisher>
```

```
<PriceUS>$29.99<PriceUS>
```

```
<Genre>여행<Genre>
```

```
<Genre>건축<Genre>
```

```
<Genre>역사<Genre>
```

```
<Synopsis>이 책은 독일에서 가장 감동적인 12곳의 대성당을 풀 컬러 사진, 축  
척 단면, 공사 일정 등으로 충분히 설명합니다.<Synopsis>
```

```
<Pages>224<Pages>
```

```
</nonfiction>
```

같은 방법으로 LabVIEW 데이터를 이름, 값, 그리고 타입으로 분류할 수 있습니다. 다음과 같이 사용자 이름에 대한 문자열 컨트롤을 XML로 나타낼 수 있습니다:

```
<String>
```

```
<Name>사용자 이름<Name>
```

```
<Value>Reggie Harmon<Value>
```

```
</String>
```

관련 정보

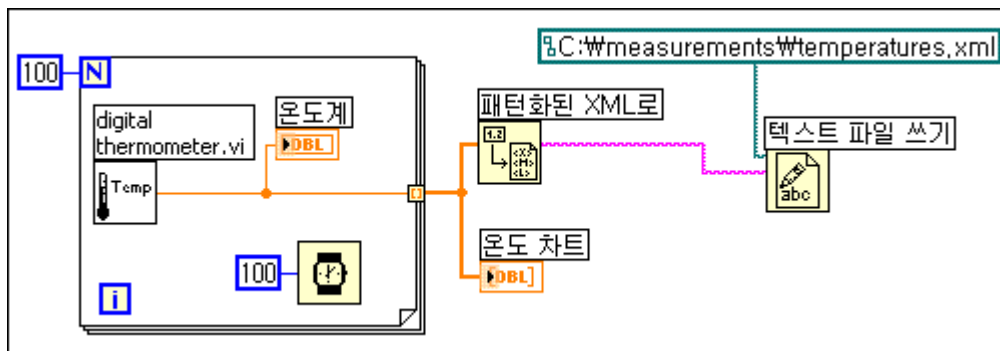
[데이터를 XML로 또는 그 반대로 변환하기](#)

[LabVIEW에서 XML 파일 분석하기](#)

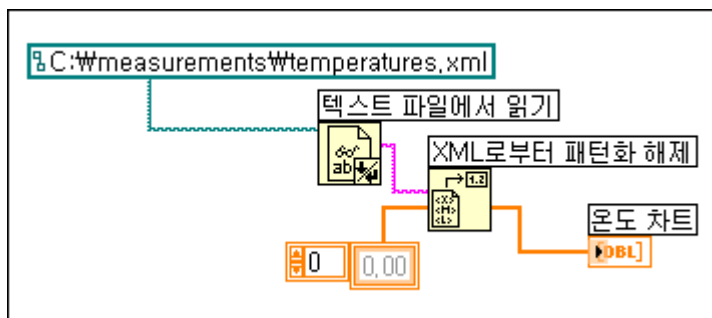
데이터를 XML로 또는 그 반대로 변환하기

LabVIEW 데이터를 XML로 변환하여 데이터를 포맷하면 데이터를 파일로 저장할 때, 데이터를 설명하는 태그로부터 데이터의 값, 이름, 타입을 쉽게 알 수 있습니다. 예를 들어, 온도값의 배열을 XML로 변환하여 텍스트 파일로 저장하면 개별 온도를 구분하는 `<Value>` 태그를 찾아 쉽게 온도값을 식별할 수 있습니다.

[패턴화된 XML로] 함수를 사용하여 LabVIEW 데이터 타입을 XML 포맷으로 변환합니다. 다음의 예제는 100개의 시뮬레이션된 온도를 생성하고, 이 온도 배열을 차트에 플롯하며, 다시 이 숫자 배열을 XML 포맷으로 변환하고, XML 데이터를 `temperature.xml` 파일로 저장합니다.



[XML로부터 패턴화 해제] 함수를 사용하여 XML 포맷의 데이터 타입을 LabVIEW 데이터로 변환합니다. 다음의 예제는 100개의 온도를 `temperatures.xml` 파일에서 읽고 이 온도 배열을 차트에 플롯합니다.



노트

LabVIEW 배리언트 데이터를 XML로 패턴화 또는 패턴화 해제할 수 있지만, LabVIEW는 배리언트 데이터가 클러스터의 원소이거나, 배리언트의 배열인 경우, 에러를 반환합니다.

XML 포맷에서 변환 및 XML 포맷으로 변환하는 예제는 `labview\examples\File IO\XML\Flatten and Unflatten XML\Flatten and Unflatten XML.lvproj`를 참조하십시오.

LabVIEW XML 스키마

LabVIEW는 데이터를 기존의 XML 스키마로 변환합니다. 현재, 사용자 정의 스키마를 만들 수 없으며, LabVIEW가 각 데이터 부분에 태그하는 방법을 컨트롤할 수 없습니다. 또한, 모든 VI나 함수가 XML로 변환되는 것은 아닙니다.

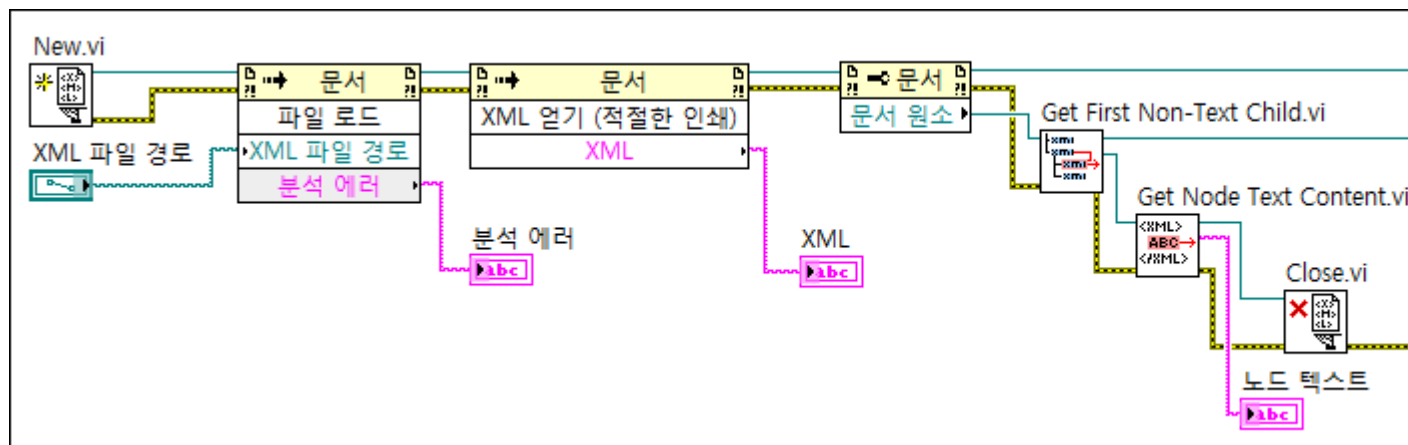
LabVIEW가 사용하는 미리 정의된 XML 스키마는 labview\vi.lib\Utility 디렉토리에 위치한 LVXMLSchema.xsd입니다. 텍스트 편집기에서 파일을 열어서 스키마를 읽을 수 있습니다.

LabVIEW에서 XML 파일 분석하기

XML(Extensible Markup Language)는 플랫폼에 독립적인 SGML(Standard Generalized Markup Language)의 일부로서, 정보를 저장하고 교환하는데 사용할 수 있습니다. XML 문서에 대한 작업을 수행하고 있을 때, 파서를 사용하여 XML 포맷을 직접 변환하지 않고도 데이터를 추출하고 조작할 수 있습니다. 예를 들어, 문서 객체 모델(DOM) Core 스펙은 XML 문서를 생성하고, 읽고, 조작하는 프로그래밍 인터페이스를 정의합니다. DOM Core 스펙은 XML 파서가 지원해야 하는 프로퍼티와 메소드를 정의합니다. LabVIEW의 XML 파서는 문서 객체 모델(DOM)을 사용하는 Xerces 2.7 파서입니다.

XML 파서 및 XML 파서 관련 VI와 노드를 사용하여 XML 문서에 대한 읽기, 쓰기, 변경 작업을 수행합니다.

다음 블록다이어그램에서 보이는 것처럼, XML 파서 VI를 통해 LabVIEW에서 플랫폼에 독립적인 XML 파서를 사용할 수 있습니다.



XML 파서 VI를 사용하는 예제는 labview\examples\File IO\XML\Parse XML 디렉토리를 참조하십시오.

관련 예제 찾기

새 VI는 XML 파서 세션을 열고 XML 문서 참조를 반환합니다. XML 문서를 생성하고, 기존 문서를 XML 파서에 로드하고, 문서 셋팅을 설정할 수 있습니다.

기본 파일을 변경하려면, 파일 로드 메소드를 사용하여 메모리에 XML 문서를 로드합니다. XML 파서가 문서 안에서 에러를 발견하여 해당 문서를 분석하지 못하는 경우를 제외

하면, LabVIEW는 XML 문서에 접근할 수 있습니다. 분석 에러 출력은 이렇게 XML과 관련된 에러를 디스플레이합니다.

XML 읽기 (적절한 인쇄) 메소드를 사용하여 XML 문서의 내용을 문자열로 적절하게 인쇄합니다. 적절한 인쇄는 XML 데이터에 화이트스페이스와 라인 끝 (EOL:end-of-line) 문자를 추가해서 XML 데이터를 더욱 읽기 쉽게 만들어줍니다. XML 데이터를 적절하게 인쇄할 필요가 없는 경우에는 XML 읽기 메소드를 사용하십시오. 이전 블록다이어그램에서는, XML 출력이 XML 문서의 내용을 디스플레이합니다.

[달기] VI는 XML 문서 참조를 달고 VI가 발견하는 모든 에러를 반환합니다.



노트

XML 파서는 검증 작업을 수행할 때에만 화이트스페이스를 무시합니다. 그렇지 않은 경우, 파서는 화이트스페이스를 직하위 노드로 포함시킵니다.

XML 문서 검증하기

XML 파서를 설정하여 특정 XML 문서가 유효한지 구분할 수 있습니다. 문서가 외부 언어를 따르는 경우, 해당 문서는 유효합니다. LabVIEW 파서에서, 외부 언어는 문서 타입 정의 (DTD:Document Type Definition) 또는 스키마가 될 수 있습니다.

일부 파서는 오직 XML을 분석하고, 파일을 로드하기 전에 XML을 검증하지 않습니다. LabVIEW에 있는 파서는 검증용 파서입니다. 검증용 파서는 DTD 또는 스키마와 비교하여 XML 문서를 검증하고, 발견되는 유효하지 않은 아이템을 보고합니다. 지정된 문서의 형식과 타입이 기대한 것과 같도록 확인해야 합니다. 검증용 파서를 사용하면, 각각의 문서 타입에 따라 사용자 정의 검증 코드를 생성할 필요가 없습니다.

XML 파서는 파일 로드 메소드의 분석 에러 출력에 검증 에러를 보고합니다.



노트 XML 파서는 LabVIEW가 문서 또는 XML 문자열을 로드할 때 해당 문서 또는 XML 문자열을 검증합니다. 문서 또는 문자열을 변경한 후 변경 사항이 반영된 문서 또는 문자열을 검증하고 싶은 경우, 파일 로드 또는 문자열 로드 메소드를 사용하여 해당 문서 또는 문자열을 다시 로드합니다. 이후 파서는 다시 그 내용을 검증하게 됩니다.

파일 로드 메소드를 사용하는 예제는 labview\examples\File IO\XML\Parse XML 디렉토리의 Load XML with Parser API VI를 참조하십시오.

XML 파서 설정하기

사용 가능한 기능을 모두 활성화하려는 경우, XML 파서의 모든 인스턴스를 설정해야 합니다. 문서 클래스에서 프로퍼티를 설정하여 파서를 설정할 수 있습니다.

블록다이어그램에 프로퍼티 노드(XML)를 놓고 사용할 수 있는 프로퍼티를 살펴봅니다. 프로퍼티 노드(XML)은 다음의 프로퍼티와 메소드를 포함합니다:

- namespace 처리—사용자가 XML 파서 namespace 처리를 활성화 또는 비활성화할 수 있습니다. 기본은 참이며, XML 파서가 자체적인 namespace 스펙 제한 및 규칙을 따르게 됩니다.
- 스키마 처리—사용자가 XML 파서 스키마 처리를 활성화 또는 비활성화할 수 있습니다. 기본은 거짓이며, XML 파서는 어떤 스키마도 처리하지 않습니다. 이 프로퍼티를 참으로 설정하는 경우, 반드시 namespace 처리를 활성화해야 합니다.
- 외부 DTD 로드—사용자가 외부 DTD 로드를 활성화 또는 비활성화할 수 있습니다. 기본은 참이며, XML 파서에서 사용자가 외부 DTD를 로드할 수 있습니다. 로드에서 확인 프로퍼티를 항상이나 자동으로 설정한 경우, 파서는 이 프로퍼티를 무시하고 항상 DTD를 로드합니다.
- 화이트스페이스 유지—텍스트 노드로서 무시할 수 있는 화이트스페이스를 검증 파서가 포함할지 여부를 사용자가 지정할 수 있습니다. 기본은 참이며, 무시할 수 있는 화이트스페이스를 텍스트 노드로 DOM 트리에 추가합니다. 이 프로퍼티를 거짓으로 설정하는 경우, XML 파서는 무시할 수 있는 화이트스페이스를 모두 버리고 DOM 트리에 텍스트 노드를 추가하지 않습니다.

노트 XML 파서는 검증 작업을 수행할 때에만 화이트스페이스를 무시합니다. 그렇지 않은 경우, 파서는 화이트스페이스를 직하위 노드로 포함시킵니다. 예를 들어, 태그의 첫 번째 직하위는 그 다음 원소가 아닌 화이트스페이스일 가능성이 높습니다.

- 스키마 전체 확인—사용자가 전체 스키마 제약 확인을 설정할 수 있습니다. 로드 시 검증 프로퍼티가 항상 또는 자동으로 설정된 경우에만 이 프로퍼티가 적용됩니다. 기본은 거짓이며, 부분적인 제약 확인을 실행합니다. 전체 스키마 제약 확인으로 인해 시간과 메모리를 많이 소요하게 될 수도 있습니다.
- 로드 시 검증—XML 파서가 사용하는 검증 방식을 사용자가 지정할 수 있습니다. 다음 열거형 값 중에서 하나를 선택할 수 있습니다:
 - 자동—(기본) 파서가 내부 또는 외부 DTD 서브셋을 감지하는 경우 검증을 On 상태로 만듭니다.
 - 없음—검증을 Off 상태로 만듭니다.

노트 이 프로퍼티를 없음으로 설정하고 외부 DTD 로드 프로퍼티를 참으로 설정한 경우, LabVIEW는 문서를 분석하지만 검증 에러를 반환하지는 않습니다.

- 항상—검증을 On 상태로 만듭니다.

XML 파서 VI를 사용하는 예제는 labview\examples\File IO\XML\Parse XML 디렉토리를 참조하십시오.

관련 예제 찾기

XML 문서 쿼리하기

XML Path Language (XPath)는 원소, 속성, 텍스트 등의 XML 노드의 위치를 찾을 때 사용하는 언어입니다. [첫 번째 일치된 노드 얻기] VI를 사용하여 사용자가 지정하는 XPath 표현식과 일치하는 첫 번째 노드의 위치를 찾습니다. [모든 일치된 노드 얻기] VI를 사용하여 사용자가 지정하는 XPath 표현식과 일치하는 모든 노드의 위치를 찾습니다. 이러한 VI는 사용자가 지정하는 내용 노드를 사용하여 노드의 위치를 찾습니다. 내용 노드는 LabVIEW가 XPath 쿼리를 수행하는 XML 문서 내의 위치를 설명합니다. 따라서, 사용자가 상대 또는 절대적인 XPath 표현식을 지정하여 XML 문서를 쿼리할 수 있습니다.

LabVIEW는 XPath version 1.0을 지원하는데, 이는 World Wide Web Consortium (W3C)에 의해 정의되었습니다.

[모든 일치된 노드 얻기] VI 사용에 대한 예제는 labview\examples\File IO\XML\Parse XML 디렉토리에 있는 Parse XML String for Multiple Nodes VI를 참조하십시오.

Zip 파일

압축(Zip) 파일 포맷을 사용하면 데이터를 압축하여 기록할 수 있습니다. LabVIEW는 디렉토리 구조를 유지하며 하나 또는 그 이상의 파일을 프로그램적으로 압축(Zip) 파일로 압축할 수 있습니다.



노트

압축(Zip) 파일은 RAR, TAR, 7Z, GZ 등과 같은 다른 파일 포맷이 포함되는 기록 파일의 일부입니다. LabVIEW는 Zip 파일 포맷만을 지원합니다.

압축(Zip) VI를 사용하여 압축(Zip) 파일을 만드는 예제는 labview\examples\File IO\Zip\Zip Files.lvproj를 참조하십시오.

Windows 설정 셋팅 파일 포맷

Windows 설정 셋팅 파일은 여러 개의 섹션으로 나누어진 텍스트 파일입니다. 괄호 안에는 각 섹션 명칭이 주어집니다. 파일 내의 모든 섹션 명칭은 고유해야 합니다. 섹션은 키/값의 쌍으로 이루어지며 등호 부호(=)로 구분됩니다. 각 섹션 내의 모든 키 명칭은 고유해야 합니다. 키 이름은 설정 환경을 나타내고, 값 이름은 해당 환경에서의 설정을 나타냅니다. 다음 예제는 파일의 배치를 보여줍니다:

```
[Section 1]
```

```
key1=value
```

```
key2=value
```

```
[Section 2]
```

```
key1=value
```

```
key2=value
```

키 파라미터의 값에는 다음의 데이터 타입과 [설정 파일] VI를 사용합니다:

- 문자열
- 경로
- 불리언

- 64비트 배정도 부동소수 숫자
- 32비트 부호 있는 정수
- 32비트 부호 없는 정수



노트

LabVIEW에서 불리언 값은 참과 거짓으로 기본 설정되며, 영어인 경우 대소문자를 구분하지 않습니다. 예를 들어, False, false, 0을 모두 FALSE(거짓) 불리언 값으로 사용할 수 있습니다.

설정 파일 VI를 사용하려면, .ini 파일에 있는 텍스트에서 각각의 라인을 정확하게 포맷해야 합니다. 라인은 비어있거나, 키-값 쌍과 같은 섹션 이름이나 주석을 포함해야 합니다.



노트 ANSI 포맷인 경우에만 Windows 설정 셋팅 파일에 설정 파일 VI를 사용할 수 있습니다.

섹션 이름을 포함하는 라인을 포맷할 때, 다음의 가이드라인을 사용하십시오:

- 최소한 1 개의 문자를 포함해야 합니다.
- 이름의 일부로 닫힌 괄호를 사용하지 않습니다.
- 인쇄할 수 없는 문자를 포함시키지 않습니다.
- 화이트스페이스가 아닌 첫 문자에 열린 괄호를 사용합니다.
- 텍스트의 라인 끝에 닫힌 괄호를 사용합니다.
- 모든 문자를 한 라인 상에 놓습니다.

키 값 인덱스를 포함하는 라인을 포맷할 때, 다음의 가이드라인을 사용하십시오:

- 섹션 라인 다음에 오는 라인에 키-값 인덱스를 포함합니다.
- 키 이름 다음에 등호 부호를 놓습니다.
- 모든 문자를 한 라인 상에 놓습니다.
- 유효한 키 이름을 사용합니다.
 - 최소한 1 개의 문자를 포함해야 합니다.
 - 세미콜론으로 시작하지 않습니다.
 - 등호 부호를 포함시키지 않습니다.
 - 열린 괄호로 시작하지 않습니다.
 - 화이트스페이스 문자로 시작하지 않습니다.
 - 화이트스페이스 문자로 끝내지 않습니다.
 - 인쇄할 수 없는 문자를 포함시키지 않습니다.
- 유효한 값 이름을 사용합니다.
 - 데이터 타입을 일관성 있게 사용합니다.
 - 화이트스페이스 문자로 시작하지 않습니다.

- 화이트스페이스 문자로 끝나지 않습니다.



노트 값이 화이트스페이스 문자로 시작하거나 끝나면 따옴표 안에 값을 놓아야 합니다. LabVIEW는 VI가 값을 반환할 때 따옴표를 제거합니다.

설정 파일 vi는 읽기 작업을 진행하는 동안 섹션 이름이나 값-키 쌍이 없는 라인을 무시하고, 쓰기 작업을 진행하는 동안에는 섹션 이름이나 값-키 쌍이 없는 라인을 지킵니다.

vi는 설정 파일에 쓸 때, 문자열이나 경로 데이터 주위에 인용 표시를 합니다. 또한, LabVIEW에서는 .ini 파일에 있는 값 주위에 작은 따옴표를 사용할 수 있습니다.

설정 파일에 대한 주석을 표기할 때 세미콜론을 사용하는데, 세미콜론은 뒤에 오는 텍스트가 주석이라는 것을 표시합니다. 라인 시작 부분에 있는 세미콜론은 주석을 표시합니다. 섹션 이름, 숫자형 값, 불리언 값에 있는 세미콜론 또한 주석을 표시합니다. 하지만, 문자열 값 이름에 있는 세미콜론은 주석을 표시하지 않습니다.

키 이름의 첫 번째 문자로 세미콜론을 입력하면, 설정 파일이 세미콜론을 키 이름의 첫 번째 문자로 지원하지 않기 때문에 LabVIEW는 키 이름이나 값을 반환하지 않습니다. 키 이름에서 첫 번째 문자가 아닌 다른 자리에 세미콜론을 포함시키면, LabVIEW는 세미콜론과 함께 키 이름을 반환합니다.

다음 테이블은 설정 파일 vi가 섹션 이름을 .ini 파일에서 읽는 방법의 예를 보여줍니다.

.ini 파일 내 예제 텍스트	섹션 이름
[sec1]	sec1
[[sec2]]	[sec2
[sec with spaces]	sec with spaces
[seccom] ;comments	seccom
[sectext] sometext	sectext

다음 테이블은 설정 파일 키-값의 쌍을 .ini 파일에서 읽는 방법의 예를 보여줍니다.

.ini 파일 내 예제 텍스트	키 이름	값
keyname='mystring'	keyname	mystring
;keyname="mystring"	—	—
keyname="my;string"	keyname	my;string
keyname = 12.3 ;comm	keyname	12.3
key;name="mystring"	key;name	mystring

.ini 파일 내 예제 텍스트	키 이름	값
key\;name="my;string";more	key\;name	"my;string";more
key name6=" mystring6"	key name6	mystring6
keyname8 = mystring8	keyname8	mystring8
keyname9 = " mystring9 "	keyname9	mystring9
keyname12==mystring12	keyname12	=mystring12
#keyname13 = mystring13	#keyname13	mystring13
abc=	abc	—
abc	—	—



노트 [키 읽기] VI는 다형성 VI이므로 특정 인스턴스에서는 다르게 동작합니다. 예를 들어, `keyname = 12.3 ;comm` 키-값의 쌍에서, 문자열 인스턴스는 `12.3;comm`이라는 문자열 값을 반환하고 배정도 인스턴스는 값 출력에서 `12.3`이라는 숫자형을 반환합니다.

[설정 파일] VI는 원시 데이터 또는 이스케이프 문자열 데이터를 읽고 쓸 수 있습니다. 이 VI는 데이터를 ASCII 형식으로 변환하지 않고, 바이트 단위로 원시 데이터를 읽고 씁니다. 변환 또는 이스케이프 문자열의 경우, LabVIEW는 설정 셋팅 파일에 디스플레이되지 않는 텍스트 문자와 이를 나타내는 16진수 이스케이프 코드, 예를 들면, 캐리지 리턴을 발생하는 `\0D`를 함께 저장합니다. 또한, LabVIEW는 설정 셋팅 파일의 백슬래시 문자 `\`를 더블 백슬래시 문자 `\\`로 저장합니다. [설정 파일] VI의 원시 문자열 읽기? 또는 원시 문자열 쓰기? 입력을 원시 데이터의 경우 참, 이스케이프 데이터인 경우 거짓으로 설정합니다.

설정 셋팅 파일에서, LabVIEW는 경로의 표준 Linux 포맷으로 경로 데이터를 저장합니다. VI는 설정 셋팅 파일에 저장된 절대 경로 `/c/temp/data.dat`를 아래와 같이 해석합니다:

- (Windows) `c:\temp\data.dat`
- (macOS 32비트) `c:temp:data.dat`
- (macOS 64비트 및 Linux) `/c/temp/data.dat`

VI는 상대 경로 `temp/data.dat`를 다음과 같이 해석합니다:

- (Windows) `temp\data.dat`
- (macOS 32비트) `:temp:data.dat`
- (macOS 64비트 및 Linux) `temp/data.dat`

재사용을 위해 데이터를 기록하기

다른 VI와 리포트에서 사용할 데이터를 저장하기 위해 프런트패널 데이터 로깅을 사용합니다. 예를 들어, 그래프의 데이터를 로그하고 해당 데이터를 다른 VI의 그래프에서 사용할 수 있습니다.



노트

LabVIEW는 리모트 패널을 통해 프런트패널 데이터 로깅을 지원하지 않습니다.

VI를 실행할 때마다 프런트패널 데이터 로깅은 프런트패널 데이터를 2진 포맷인 **별도의 데이터로그 파일**에 저장합니다.

데이터로그 파일은 VI가 실행될 때마다 발생하는 데이터와 타임스탬프를 포함하는 기록을 가집니다. 데이터로그 파일에 접근할 때, 검색 모드로 VI를 실행하여 원하는 데이터를 선택하고 프런트패널의 컨트롤을 사용하여 데이터를 봅니다. 검색 모드에서 VI를 실행할 때, 다음과 같이 프런트패널의 상단에 숫자 컨트롤이 나타나므로 레코드를 탐색할 수 있습니다.



자동 대화식 프런트패널 데이터 로깅

자동 로깅을 활성화하려면 수행>>완료시 로그를 선택합니다. 처음 VI의 프런트패널 데이터를 로그하면, LabVIEW는 데이터로그 파일의 이름을 입력요청합니다. 데이터로그 파일에 대해 .dat나 .txt처럼 어떤 파일 확장자이든 사용할 수 있습니다. LabVIEW는 VI를 실행할 때마다 데이터를 로그하며 VI를 실행하는 추가적인 시간마다 데이터로그 파일에 새로운 레코드를 추가합니다. LabVIEW가 데이터로그 파일에 레코드를 로그한 후에는 레코드를 덮어쓸 수 없습니다.

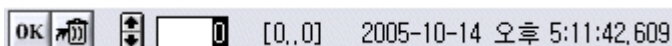
데이터를 대화식으로 저장하려면, 수행>>데이터 로깅>>로그를 선택합니다. LabVIEW는 바로 데이터로그 파일에 데이터를 추가합니다. 대화식으로 데이터를 로그할 경우 데이터를 언제 로그할 것인지 선택할 수 있습니다. 자동으로 데이터를 로깅하도록 하면 VI를 실행할 때마다 데이터가 로그됩니다.



노트 프런트패널 데이터 로깅에서 웨이브폼 차트는 한 번에 한 개의 데이터만 로그합니다. 차트 인디케이터에 배열을 연결할 경우, 데이터로그 파일은 차트가 디스플레이하는 배열의 부분을 포함합니다.

대화식으로 로그된 프런트패널 데이터 보기

데이터 로그 후, 수행>>데이터 로깅>>가져오기를 선택하면 로그된 값을 대화식으로 볼 수 있습니다. 데이터 검색 도구 모음이 다음처럼 나타납니다.



하이라이트된 숫자는 보고 있는 데이터 레코드를 나타냅니다. 대괄호 안의 숫자는 현재 VI에 로그한 레코드의 범위를 나타냅니다. VI를 실행할 때마다 레코드를 로그합니다. 날짜와 시간은 언제 선택된 레코드를 로그했는지 나타냅니다. 증가 또는 감소 화살표를 누르면 다음 또는 이전의 레코드를 볼 수 있습니다. 또한, 키보드의 위, 아래 화살표도 사용할 수 있습니다.

데이터 검색 도구 모음과 함께, 프런트패널의 모양은 도구 모음에서 선택한 레코드에 따라서 변경됩니다. 예를 들어, 증가 화살표를 클릭하여 다른 레코드로 보내면, 컨트롤과 인디케이터는 데이터를 로그했을 때의 특정한 레코드에 대한 데이터를 디스플레이합니다. 확인 버튼을 누르면 검색 모드를 빠져나와 데이터로그 파일을 보고 있던 VI로 돌아옵니다.

레코드 삭제하기

검색 모드에 있는 동안, 특정 레코드를 삭제할 수 있습니다. 검색 모드에서 레코드를 보고 버리기 버튼을 클릭하면 삭제할 개별 레코드를 표시할 수 있습니다. 버리기 버튼을 다시 한번 클릭하면, 그 레코드는 삭제 대상에서 제외됩니다.

검색 모드에서 수행»데이터 로깅»데이터 삭제를 선택하면 삭제 표시했던 모든 레코드가 삭제됩니다.

확인 버튼을 클릭하기 전에 표시된 레코드가 지워지지 않은 경우, LabVIEW는 표시된 레코드를 지워야 하는지 사용자에게 묻습니다.

로그 파일 연결 제거하기

프런트패널 데이터를 가져오거나 로깅할 때 필요한 VI와 데이터로그 파일을 연동하기 위해 로그 파일 연결을 사용합니다. 하나의 VI와 연동된 두 개 또는 더 많은 데이터로그 파일을 가질 수 있습니다. 이것은 VI 데이터를 테스트하거나 비교할 때 도움이 됩니다. 예를 들어, VI를 처음 실행할 때 로그된 데이터를 VI를 두 번째 실행할 때 로그된 데이터와 비교할 수 있습니다. VI에서 여러 개의 데이터로그 파일을 연동하려면, 수행»데이터 로깅»로그 파일 연결 지우기를 선택하여 반드시 로그 파일 연결을 삭제합니다. LabVIEW는 다음에 VI를 실행할 때 데이터로그 파일을 지정하기 위해서 자동으로 로깅을 활성화하거나 또는 대화식으로 로그 데이터를 선택할지를 물어봅니다.

로그 파일 연결 변경

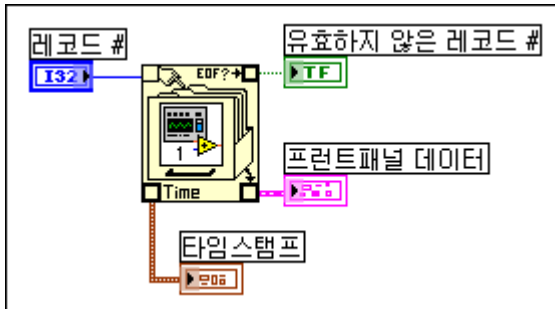
수행»데이터 로깅»로그 파일 연결 변경을 선택하여 다른 로그 파일에서 프런트패널 데이터를 가져오거나 다른 로그 파일에 프런트패널 데이터를 로그하도록 로그 파일 연결을 변경합니다. LabVIEW는 다른 로그 파일을 선택하거나 새로운 로그 파일을 생성하도록 입력요청합니다. VI에 다른 데이터를 가져오거나 VI의 데이터를 다른 데이터로그 파일에 연결할 때 로그 파일 연결을 변경할 수 있습니다.

프로그램적으로 프런트패널 데이터를 가져오기

또한, SubVI 또는 파일 I/O VI와 함수를 사용하여 로그된 데이터를 가져올 수 있습니다.

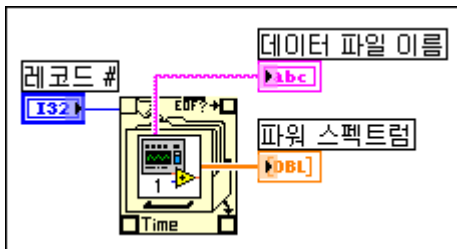
SubVI를 사용하여 프런트패널 데이터를 가져오기

SubVI를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 데이터베이스 접근 활성화를 선택하면 다음 블록다이어그램과 같이, SubVI 주위에 노란 상자가 나타납니다.



파일 캐비닛처럼 보이는 노란 상자는 데이터로그 파일에서 데이터에 접근하기 위한 터미널을 가지고 있습니다. 데이터 접근을 활성화할 때, SubVI의 입력과 출력은 실제로 출력으로 작용하여 현재 VI의 프런트패널에 로그된 데이터를 반환합니다. 레코드 #는 가져올 레코드를 지정하고, 유효하지 않은 레코드 #는 레코드 번호의 존재 여부를 나타내며, 타임스탬프는 레코드 생성 시간이고, 프런트패널 데이터는 프런트패널 객체의 클러스터입니다. [폴기] 함수를 프런트패널 데이터 클러스터에 연결하여 프런트패널 객체의 데이터를 얻을 수 있습니다.

다음 블록다이어그램과 같이, 대응하는 SubVI의 터미널에 직접적으로 연결하여 특정한 입출력 값을 가져올 수 있습니다.



VI를 실행하면, SubVI는 실행되지 않습니다. 대신, 프런트패널에서 로그된 데이터를 VI 프런트패널의 클러스터로 반환합니다.



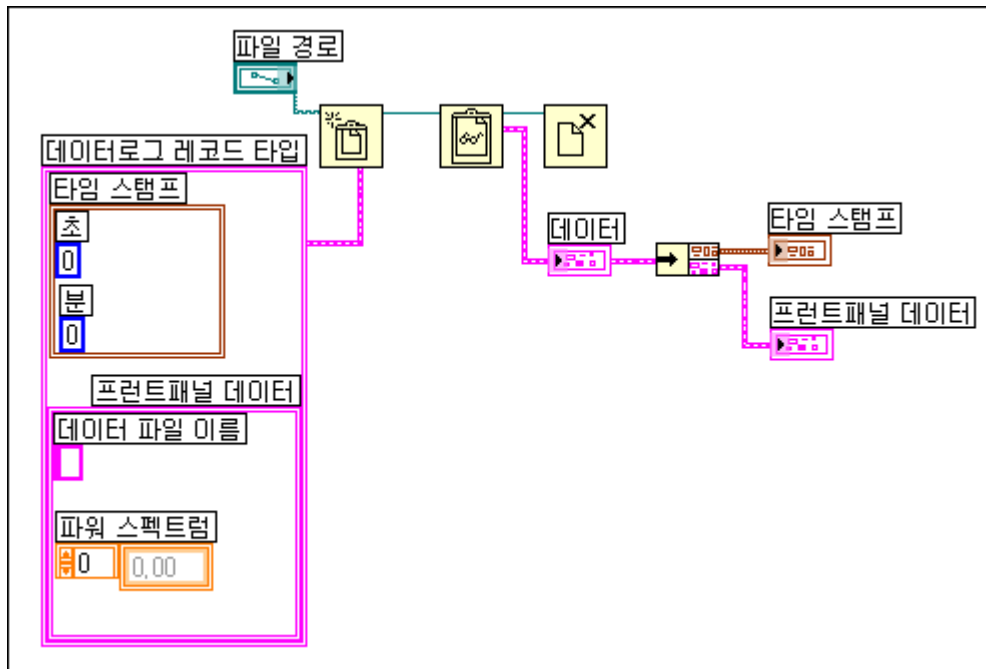
노트 SubVI 또는 익스프레스 VI를 확장 가능 노드로 디스플레이하는 경우, 해당 노드에 대한 데이터베이스 접근을 활성화할 수 없습니다.

레코드 지정하기

SubVI는 n 개의 로그된 레코드를 가지고 있으면, SubVI의 레코드 # 터미널에 $-n$ 부터 $n-1$ 까지 임의의 숫자를 연결할 수 있습니다. 음수가 아닌 레코드 번호를 사용하여 첫 번째 로그된 레코드를 기준으로 한 레코드에 접근할 수 있습니다. 0은 첫 번째 레코드를 나타내고, 1은 두 번째 레코드를 나타내며, 이런 방식으로 진행되어 $n-1$ 은 마지막 레코드를 나타냅니다.

음수 레코드 수를 사용하여 최종적으로 로그된 레코드와 상대적인 레코드에 접근할 수 있습니다. -1은 마지막 레코드를 나타내며 -2는 마지막에서 두 번째를 나타내는 식으로

진행되어 -n은 첫 번째 레코드를 나타냅니다. -n에서 n-1 범위 밖의 숫자를 레코드 # 터미널에 연결하는 경우, 유효하지 않은 레코드 # 출력은 참이고, SubVI는 데이터를 가져오지 않습니다.



LabVIEW 데이터 디렉토리

.1vm 또는 .txt와 같은 형식으로 LabVIEW가 생성할 수 있는 데이터 파일로 저장하려면, 기본 LabVIEW Data 디렉토리를 사용합니다. LabVIEW는 LabVIEW Data 디렉토리를 OS의 기본 파일 디렉토리에 설치하여 LabVIEW에서 생성한 데이터 파일을 찾기 쉽도록 체계화합니다. 기본적으로, [측정 파일에 쓰기] 익스프레스 VI는 이 디렉토리에서 생성하는 .1vm 또는 .tdm 파일을 저장하고, [측정 파일로부터 읽기] 익스프레스 VI는 이 디렉토리에서 파일을 읽습니다. 기본 데이터 디렉토리 상수와 어플리케이션:기본:데이터 디렉토리 프로퍼티는 기본으로 LabVIEW 데이터 디렉토리를 반환합니다.

도구>> 옵션을 선택하고 항목 리스트에서 경로를 선택하여 다른 기본 데이터 디렉토리를 지정할 수 있습니다. 기본 데이터 디렉토리는 사용자가 새 VI, 사용자 컨트롤, VI 템플릿 또는 다른 LabVIEW 문서를 생성할 때 지정하는 기본 디렉토리와는 다릅니다.

텍스트 기반 측정 파일

텍스트 기반의 측정 파일 .1vm은 [측정 파일에 쓰기] 익스프레스 VI가 생성한 데이터를 가지고 있습니다. 텍스트 기반 측정 파일은 스프레드시트 또는 텍스트 편집 어플리케이션에서 열 수 있도록 탭으로 자리를 구분한 텍스트 파일입니다. .1vm 파일은 익스프레스 VI가 생성한 데이터 이외에, 데이터가 만들어진 날짜와 시간과 같은 데이터의 정보도 포함합니다. LabVIEW는 데이터를 소수점 여섯 자리까지 .1vm 파일에 저장합니다.

[측정 파일로부터 읽기] 익스프레스 VI를 사용하여 .1vm 파일에서 **데이터 읽기**를 수행합니다.

.lvb 파일은 쉼표를 숫자 간의 구분문자로 사용합니다. .lvb 파일의 데이터를 문자열에서 숫자로 변환하려면 **현지화 코드 포맷 지정자**를 사용하여 소수점을 마침표로 지정합니다.

.lvb 파일에 대한 더 자세한 정보는 ni.com에서 다음 관련 문서를 참조하십시오.

2진 측정 파일

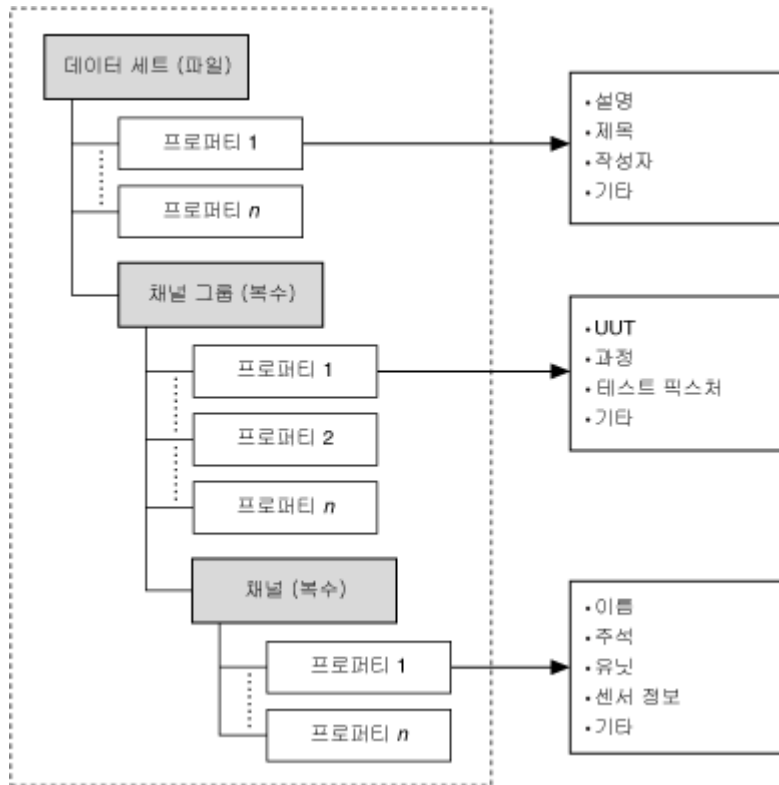
2진 측정 파일 .tdm 또는 .tdms는 [측정 파일에 쓰기] 익스프레스 VI, 스토리지/DataPlugin VI, TDM 스트리밍 VI와 함수가 생성하는 데이터를 포함합니다. .tdm 파일은 XML 기반 포맷을 사용하여 웨이브폼 프로퍼티를 저장하고 웨이브폼 데이터를 포함하는 2진 파일에 링크합니다. .tdms 파일은 웨이브폼 프로퍼티를 저장하며, 웨이브폼 데이터를 포함하는 2진 파일입니다. 2진 측정 파일은 **LabVIEW 측정 데이터 파일 (.lvb)**에 비해 정확도가 높고, 디스크 공간을 덜 차지하고, 실행 속도가 빠릅니다.

2진 측정 파일을 사용하여 LabVIEW와 DIAdem과 같은 NI 소프트웨어 간의 데이터를 교환할 수 있습니다.

(Windows) [측정 파일로부터 읽기] 익스프레스 VI와 [측정 파일에 쓰기] 익스프레스 VI를 사용하여 .tdm 파일에서 **데이터 읽기** 및 **데이터 쓰기**를 수행할 수 있습니다. 스토리지/DataPlugin VI를 사용하여 .tdm 파일에서 데이터 읽기 및 데이터 쓰기를 수행할 수 있습니다. TDM 스트리밍 VI 및 함수를 사용하여 .tdms 파일에서 데이터 읽기 및 데이터 쓰기를 수행할 수 있습니다.

TDM 데이터 모델

TDM 데이터 모델은 데이터를 파일, 채널 그룹 및 채널이라는 3개의 계층적 레벨로 정돈합니다. 다음 그림은 TDM 데이터 모델을 설명합니다.



이전 그림에서, 파일 레벨에는 무한한 개수의 채널 그룹이 있을 수 있으며, 각 그룹에는 무한한 개수의 채널이 있을 수 있습니다. 각 채널은 데이터 값의 1D 배열을 포함합니다.

TDM 데이터 모델의 장점

이 데이터 모델을 사용하면 데이터를 유연하게 구성하고, 이해하기 쉽게 만들 수 있습니다. 예를 들어, 하나의 파일에 원시 데이터라는 한 그룹과 분석 데이터라는 또 다른 그룹을 둘 수 있습니다. 또한 센서 타입이나 위치에 따라 여러 개의 그룹을 가질 수 있습니다.

사용자는 각 계층 레벨에 무제한 개수의 사용자 스칼라 프로퍼티를 저장할 수 있습니다. 각 레벨은 무제한 개수의 사용자 프로퍼티를 허용하여 문서화가 잘 되어 있고 검색이 가능한 데이터 파일을 완성합니다. 파일 내부에 파일에 대한 설명 정보를 둘 수 있다는 것은 이 데이터 모델의 중요한 장점이며, 헤더 구조를 설계할 필요없이 데이터를 문서화할 수 있어 편리합니다. 후에 문서에 대한 요구사항이 더 많아지는 경우에도 어플리케이션을 다시 설계할 필요가 없이, 현재 모델을 확장하여 특정 요구사항을 충족시킬 수 있습니다. 측정 데이터 문서화에 사용자 프로퍼티를 더 많이 사용할수록, 이후 데이터를 더욱 쉽게 찾을 수 있습니다.

관련 정보

TDM과 TDMS

TDM 파일과 상호작용하기

TDMS 파일과 상호작용하기

TDM과 TDMS

TDM 데이터 모델은 다음 2개의 파일 포맷을 지원합니다:

- (Windows) TDM – TDM 파일 포맷은 측정 하드웨어가 수집한 원시 데이터와 메타 데이터를 별도의 파일에 저장합니다. 메타 데이터는 측정 채널의 이름 및 단위, 테스트 엔지니어 및 그 외 다른 정보가 될 수 있습니다. 메타 데이터는 XML 포맷에 .tdm 확장자를 갖습니다. 원시 데이터는 컴팩트 2진 포맷에 .tdx 확장자를 갖습니다.
- TDM 스트리밍 (TDMS)—TDMS 파일 포맷은 원시 데이터와 메타 데이터를 모두 2진 포맷에 .tdms 확장자를 갖는 하나의 파일로 저장합니다.



노트

.tdms 파일을 생성하거나 열면 LabVIEW는 자동으로 .tdms_index 파일을 생성할 수 있습니다. LabVIEW는 .tdms_index 파일을 사용하여 .tdms 파일에 대한 임의의 접근 속도를 높입니다. TDMS 어플리케이션에서 .tdms_index 파일은 옵션입니다. 디스크 공간의 제약이 있는 경우, [TDMS 열기] 함수를 사용하여 이 파일의 생성을 중지합니다.

두 포맷 중 하나를 사용하여 측정 데이터를 저장할 수 있습니다. 다음 테이블은 어플리케이션에 더 적합한 파일 포맷을 결정하는데 도움이 됩니다.

TDM	TDMS	노트	
크로스 플랫폼 지원	⊗	✓	TDM 파일은 Windows만 지원합니다. TDMS 파일 포맷은 Windows, macOS, Linux 및 Phar Lap ETS, VxWorks, NI Linux Real-Time과 같은 Real-Time OS를 지원합니다.
DAQmx 지원	⊗	✓	DAQmx 9.0 또는 이후 버전이 필요합니다.
DIAdem 지원	✓	✓	DIAdem 10.0 또는 이후 버전이 필요합니다.
LabWindows™/CVI™ 지원	✓	✓	LabWindows/CVI 8.1 또는 이후 버전이 필요합니다.
Measurement Studio 지원	⊗	✓	Measurement Studio 8.6 또는 이후 버전이 필요합니다.
LabVIEW 툴킷 지원	✓	✓	TDM 및 TDMS 파일 포맷은 DataFinder Connectivity VI와 함께 작동합니다. 또한 TDMS 파일 포맷은 다음의 LabVIEW 툴킷 및 모듈과 작동합니다. • LTE Measurement Studio 1.0 또는 이후 버전 • Simulation Interface Toolkit 4.0 또는 이후 버전 • Statechart Module 2.0 또는 이후 버전 • Sound and Vibration Measurement Suite 7.0 또는 이후 버전

TDM	TDMS	노트	
복소수 데이터 지원	⊘	✓	TDMS 파일 포맷은 TDM 파일 포맷이 지원하는 모든 데이터 타입을 지원하는 것 외에도, 또한 단정도 및 배정도 복소수 부동소수와 같은 다른 데이터 타입도 지원합니다.
고속 데이터 스트리밍 지원	⊘	✓	LabVIEW에는 고속으로 TDMS 데이터를 스트리밍할 수 있는 TDM 스트리밍 VI 및 함수가 포함되어 있습니다. 그러나, LabVIEW는 TDM 파일 포맷에 대해 그러한 VI 및 함수를 제공하지 않습니다.
고급 파일 작업 지원	⊘	✓	또한 LabVIEW는 표준 TDM 스트리밍 VI 및 함수 외에도 고급 VI 및 함수를 제공하며, 이를 사용하여 TDMS 데이터를 동기적 또는 비동기적으로 스트리밍할 수 있습니다. 그러나, LabVIEW는 TDM 파일 포맷에 대해 그러한 VI 및 함수를 제공하지 않습니다.
데이터 부분 제거 지원	✓	✓	TDM 파일의 경우, LabVIEW에서 [데이터 삭제] VI를 사용하여 채널 그룹 또는 채널을 제거할 수 있습니다. TDMS 파일의 경우, LabVIEW에서 [TDM 데이터 삭제] 함수를 사용하여 그룹에서 채널 또는 여러 채널로부터의 데이터를 삭제할 수 있습니다.

관련 정보

TDM 데이터 모델

TDM 파일과 상호작용하기

TDMS 파일과 상호작용하기

TDM 파일과 상호작용하기 (Windows)

LabVIEW에서 다음 옵션을 사용하여 TDM 파일을 읽거나 쓸 수 있습니다:

- 익스프레스 VI—[측정 파일로부터 읽기] 및 [측정 파일에 쓰기] 익스프레스 VI를 사용하면 TDM 파일에서 데이터를 읽거나 TDM 파일에 데이터를 쓸 수 있습니다.
- 스토리지/DataPlugin VI—스토리지/DataPlugin VI를 사용하면 익스프레스 VI를 사용할 때 보다 더 많은 작업을 수행할 수 있습니다. 데이터 읽기나 쓰기 외에도, TDM 파일, 채널 그룹 또는 채널의 프로퍼티를 얻거나 설정할 수 있습니다. 또한 TDM 파일의 데이터를 테이블 또는 그래프로 디스플레이하여 볼 수 있습니다.

채널 또는 채널 그룹에서 데이터를 읽거나 쓰는 예제는 각각 다음의 LabVIEW 프로젝트를 참조하십시오:

- labview\examples\File IO\DataPlugins\TDM\TDM Event Data\TDM Event Data.lvproj

- labview\examples\File IO\DataPlugins\TDM\TDM Channel Groups\TDM Channel Groups.lvproj

관련 정보

TDM 데이터 모델

TDM과 TDMS

TDMS 파일과 상호작용하기

TDMS 파일과 상호작용하기

LabVIEW에서 다음 옵션을 사용하여 TDMS 파일을 읽거나 쓸 수 있습니다:

- 익스프레스 VI—[측정 파일로부터 읽기] 및 [측정 파일에 쓰기] 익스프레스 VI를 사용하면 TDM 파일에서 데이터를 읽거나 TDMS 파일에 데이터를 쓸 수 있습니다. 이 익스프레스 VI는 대화식 설정의 편리함을 제공합니다; 그러나 성능이 희생되므로 고속 스트리밍 또는 Real-Time 어플리케이션에는 적합하지 않습니다.
- (Windows) 스토리지/DataPlugin VI — 파일로부터 데이터를 읽거나 파일에 데이터를 쓰기 전, [데이터 스토리지 열기] 익스프레스 VI를 사용하여 TDMS 파일을 열 수 있습니다. TDMS 파일, 채널 그룹 또는 채널의 프로퍼티를 얻거나 설정할 수 있습니다. 하지만 이 VI를 사용하면 또한 성능이 희생되므로, 고속 스트리밍 또는 Real-Time 어플리케이션에는 적합하지 않습니다.
- TDM 스트리밍 VI 및 함수—보다 높은 유연성 및 최상의 성능을 얻으려면, TDM 스트리밍 VI 및 함수를 사용하십시오. 이 VI와 함수를 사용하면, TDMS 파일 및 프로퍼티를 가장 효율적으로 읽거나 쓸 수 있습니다. 이 VI 및 함수는 또한 고속 스트리밍 또는 Real-Time 어플리케이션에 사용할 수 있습니다.

TDMS 데이터를 읽거나 쓰는 예제는 다음의 LabVIEW 프로젝트를 참조하십시오:

- labview\examples\File IO\TDMS\Standard Read and Write\TDMS Standard Read and Write.lvproj
- labview\examples\File IO\TDMS\Advanced Read and Write\Asynchronous Read and Write\TDMS Advanced Asynchronous Read and Write.lvproj

관련 정보

TDM 데이터 모델

TDM과 TDMS

TDM 파일과 상호작용하기

표준 및 고급 TDMS 함수

TDM 스트리밍 팔레트는 표준 TDMS VI 및 함수와 고급 TDMS VI 및 함수를 모두 포함하고 있습니다. 고급 TDMS VI 및 함수를 사용하여 .tdms 파일에서 비동기화 읽기와 쓰기 작업과 같은 고급 파일 I/O 작업을 수행할 수 있습니다.



노트

고급 TDMS VI와 함수의 기능이 꼭 필요하지 않은 경우, 표준 TDMS VI와 함수를 사용합니다. 고급 TDMS VI와 함수를 잘못 사용하면 .tdms 파일이 손상될 수 있기 때문입니다. 고급 TDMS VI와 함수를 사용할 때 버퍼를 비활성화할 경우, 읽거나 쓰는 데이터의 크기는 하드 디스크의 섹터 크기의 배수가 되어야 합니다.

표준 TDMS VI와 함수와 비교하면 고급 TDMS VI와 함수는 다음과 같은 장점을 가지고 있습니다.

- 메타 데이터와 원시 데이터 따로 쓰기-표준 TDMS VI와 함수는 메타 데이터와 원시 데이터를 동시에 씁니다. 고급 TDMS VI와 함수를 사용하면 메타 데이터와 원시 데이터를 따로 쓸 수 있습니다. [TDMS 채널 정보 설정] 함수를 사용해서 .tdms 파일에 메타 데이터를 쓰고, 그 후 [TDMS 고급 비동기화 쓰기] 함수를 사용해서 .tdms에 원시 데이터를 쓸 수 있습니다.

노트 .tdms 파일에 데이터를 쓸 때 데이터는 메타 데이터와 원시 데이터를 모두 포함합니다. .tdms 파일의 메타 데이터와 원시 데이터에 대한 자세한 정보는 ni.com의 다음 관련 문서를 참조하십시오.

- (Windows) 데이터를 비동기적으로 읽고 쓰기-표준 TDMS VI와 함수는 데이터를 동기적으로만 읽고 쓸 수 있습니다. 이 경우, 크기가 큰 데이터를 읽거나 쓸 때 버퍼의 크기가 작으면 성능이 저하될 수 있습니다. 고급 TDMS VI와 함수를 사용하면 데이터를 비동기적으로 읽거나 쓸 수 있습니다. [TDMS 비동기화 읽기 설정]이나 [TDMS 비동기화 쓰기 설정] 함수를 사용해서 버퍼를 할당하고, 백그라운드에서 다수의 비동기화 읽기 또는 쓰기를 시작합니다.
- (Windows) 외부 데이터에 직접 접근하여 최대 데이터 스트리밍을 최적화합니다. 표준 TDMS VI 및 함수와는 달리, 고급 TDMS VI 및 함수를 사용하여 데이터 스트리밍 디바이스를 제어하는 디바이스 드라이버의 메모리 직접 참조(DMA: direct memory access) 버퍼와 같이, LabVIEW 밖의 구성요소가 소유하는 데이터와 직접 상호 작용할 수 있습니다. [DMA 버퍼에서 .tdms 파일로 직접 비동기적으로 데이터를 쓰거나](#), .tdms 파일로부터 직접 DMA 버퍼로 데이터를 읽을 수 있습니다. DMA 버퍼의 외부 데이터에 직접 접근하면 큰 데이터 세트의 복사를 줄일 수 있고, 데이터 스트리밍을 향상할 수 있습니다.

노트 이 기능을 사용하려면 데이터 스트리밍 디바이스의 드라이버 API로부터 반환된 [외부 데이터 값 참조](#)가 필요합니다. 외부 데이터 값 참조를 [TDMS 고급 비동기화 읽기(데이터 참조)] 또는 [TDMS 고급 비동기화 쓰기(데이터 참조)] 함수의 데이터 참조에 연결하여 외부 데이터와 상호 작용합니다.

- 기존 데이터 덮어쓰기-표준 TDMS VI 및 함수와 달리, 고급 TDMS VI와 함수를 사용하면 .tdms 파일에서 데이터의 일부를 덮어쓸 수 있습니다. [TDMS 다음 쓰기 위치 설정] 함수를 사용해서 기존의 데이터를 덮어쓸 오프셋 위치를 지정합니다.
- 데이터를 쓰기 전에 파일 크기 예약하기-표준 TDMS VI 및 함수와 달리, 고급 TDMS VI와 함수를 사용하면 데이터를 쓰기 전에 미리 디스크 공간을 할당할 수 있

어 파일 시스템 레벨에서 분할을 방지할 수 있습니다. [TDMS 파일 크기 예약] 함수를 사용해서 .tdms 파일에 데이터를 쓰기 전에 미리 디스크 공간을 할당합니다.

파일 포맷 버전

TDM 스트리밍 (TDMS) 파일 포맷은 1.0과 2.0의 두 가지 버전이 있습니다. 파일 포맷 버전 1.0은 데이터를 TDMS 파일에서 읽고 쓰기, 프로퍼티를 얻고 설정하기 등과 같은 기본 파일 작업 기능을 제공합니다. 파일 포맷 2.0 버전에는 1.0 버전의 모든 기능 및 다음과 같은 추가적인 기능이 포함되어 있습니다:

- 끼워넣은 데이터를 TDMS 파일에 쓸 수 있습니다.
- TDMS 데이터를 다른 바이트 순서나 엔디언 포맷으로 쓸 수 있습니다.
- OS 버퍼를 사용하지 않고도 TDMS 데이터를 쓸 수 있습니다. 이렇게 하면 특히 RAID(Redundant Array of Independent Disks) 시스템에서 성능이 크게 향상됩니다.
- NI-DAQmx를 사용하여 원시 데이터를 스케일링 정보와 함께 TDMS 파일에 쓸 수 있습니다.
- 파일 포맷 버전 2.0은 여러 덩어리의 인접한 데이터에 단일 헤더를 사용하여 인접한 데이터 수집의 쓰기 성능을 최적화합니다. 이렇게 하면 단일 값 수집의 성능이 향상됩니다.
- 파일 포맷 버전 2.0은 TDMS 파일에 비동기화 쓰기를 지원합니다. 이렇게 되면 어플리케이션을 쓰기 함수가 끝날 때까지 기다리지 않고 파일에 데이터를 쓰는 동안 메모리에 있는 데이터를 처리할 수 있습니다.

동기화 및 비동기화 파일 작업

LabVIEW에서 TDMS 데이터를 동기적 또는 비동기적으로 읽고 쓸 수 있습니다. 여기서 동기적 및 비동기적이라는 용어는 하드웨어 연결이 공통 클럭 또는 타임베이스를 공유하는지를 말하는 것이 아닙니다. 이 용어는 TDM 스트리밍 VI 및 함수가 호출 스레드에 따라 어떻게 동작하는지를 말합니다.

- 동기화 I/O—이 모드에서 TDMS 어플리케이션은 파일 I/O 작업을 시작한 후, 즉시 I/O 요청이 완료될 때까지 상태 기다림을 입력합니다. 어플리케이션은 I/O 요청이 완료될 때까지 새로운 I/O 작업을 시작하거나 남은 코드를 실행할 수 없습니다. 어플리케이션이 여러 개의 I/O 작업을 수행하는 경우, 프로세서는 대부분의 시간을 I/O 작업 완료를 기다리며 보내게 됩니다.
- (Windows) 비동기화 I/O — 이 모드에서 TDMS 어플리케이션은 여러 개의 파일 I/O 작업을 백그라운드에서 동시에 시작하고 실행할 수 있습니다. 이 어플리케이션은 새 I/O 작업을 시작하기 위해 이전 I/O 작업이 완료되기를 기다릴 필요가 없습니다. 이 모드를 사용하면 TDMS 어플리케이션의 작업량, 지연 또는 응답을 향상시킬 수 있습니다.

관련 정보

TDM 데이터 모델

TDM과 TDMS

TDMS 파일 사용 시 파일 버퍼링

TDMS 파일 I/O 작업을 하는 동안, LabVIEW가 TDMS 파일에 작성한 데이터는 버퍼가 다 찰 때까지 또는 사용자가 파일을 닫을 때까지 버퍼에 있게 됩니다. 버퍼링은 OS 파일 시스템의 채널 레이어나 TDMS 디스크 캐시에서 발생할 수 있습니다.

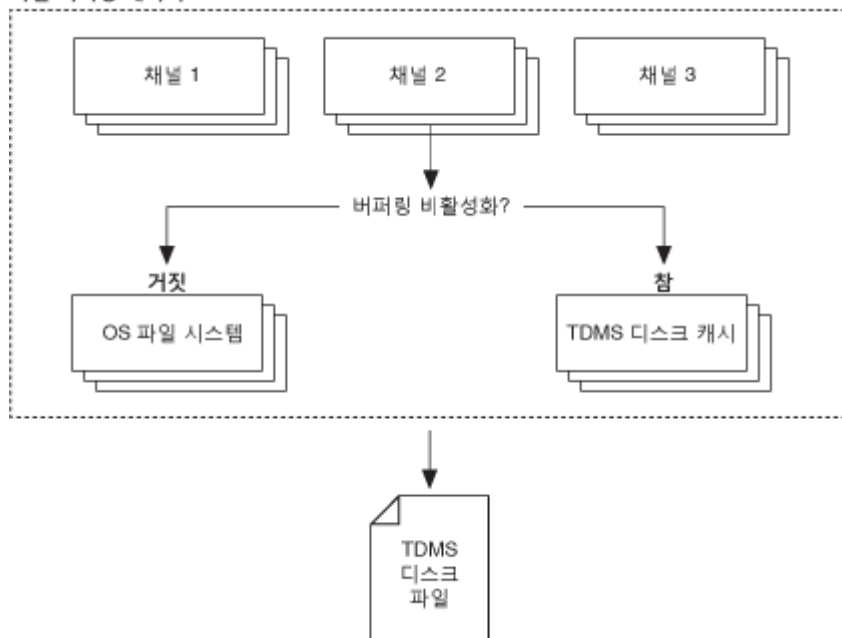
다음 그림은 표준 TDMS VI에서 TDMS 파일에 발생하는 버퍼링을 보여줍니다.



노트

고급 TDMS VI와 함수의 경우, OS 파일 시스템에서만 버퍼가 발생합니다. 버퍼링 비활성화? 입력이 참인 경우, 어떤 레이어에서도 버퍼링이 발생하지 않습니다.

파일 버퍼링 레이어



버퍼링이 발생하는 레이어는 [TDMS 프로퍼티 설정] 함수를 사용하여 설정한 프로퍼티에 따라 달라질 수 있습니다. 예를 들어, NI_MinimumBufferSize 프로퍼티는 채널 레이어에서 버퍼링할 데이터 샘플의 개수를 결정합니다. TDMS 함수의 버퍼링 비활성화? 입력은 LabVIEW가 OS에 발생하는 시스템 버퍼링을 비활성화할지 여부를 결정합니다. 버퍼링 비활성화? 입력이 참이면, LabVIEW의 시스템 버퍼링이 비활성화되고 TDMS 디스크 캐시에서 버퍼링이 발생합니다. NI_DiskCacheSize 프로퍼티는 데이터에 할당할 TDMS 디스크 캐시에서 캐시 크기를 결정합니다. 이 프로퍼티는 버퍼링 비활성화? 입력이 참인 경우에만 유효합니다.

모든 버퍼를 TDMS 파일에 직접 쓰려면, [TDMS 비우기] 함수를 사용하십시오.



노트 NI_MinimumBufferSize 프로퍼티 및 NI_DiskCacheSize 프로퍼티는 고급 TDMS VI 및 함수에는 적용되지 않습니다.

버퍼링을 비활성화해야 하는 경우

다음 상황에서 버퍼링을 비활성화시켜 데이터 전송의 속도를 높일 수 있습니다:

- 크기가 큰 샘플을 읽거나 작성하려는 경우.
- 서로 다른 샘플에서 많은 양의 데이터를 읽어나 작성하려는 경우.
- RAID(Redundant Array of Independent Disks)를 사용하여 프로세스 속도를 높이려는 경우. RAID는 단일 디스크처럼 동작하는 여러 하드 디스크의 모음으로, OS가 동시에 사용할 수 있어서 데이터를 읽고 쓰는데 소요되는 시간이 적습니다. 버퍼링이 활성화된 상태에서 RAID를 사용하면, OS가 실제로 디스크에 데이터를 쓰는데 소요되는 시간보다도 LabVIEW가 OS에 데이터를 복사하는데 시간이 더 많이 필요할 수도 있습니다.

관련 정보

데이터 버퍼링하기

파일 버퍼링

표준 및 고급 TDMS 함수

TDMS 데이터 비동기화 읽기

다음 가이드라인은 .tdms 파일에서 데이터를 비동기적으로 읽는 방법을 설명합니다.

1. [TDMS 고급 열기] 함수를 사용하여 .tdms 파일을 엽니다. 비동기화 활성화?의 값이 참인지 확인합니다.
2. [TDMS 비동기화 읽기 설정] 함수를 사용하여 셋팅을 설정하고 비동기화 읽기에 대해 버퍼를 할당합니다.
3. (옵션) [TDMS 다음 읽기 위치 설정] 함수를 사용하여 파일에서 데이터를 읽기 시작할 부분을 설정합니다.
4. [TDMS 비동기화 읽기 시작] 함수를 사용하여 데이터를 버퍼에 읽습니다.
5. (옵션) [TDMS 비동기화 읽기 상태 열기] 함수를 사용하여 사용 가능한 데이터가 포함된 버퍼의 개수를 알아냅니다.
6. [TDMS 고급 비동기화 읽기] 함수를 사용하여 버퍼로부터 데이터를 읽습니다.
7. (옵션) [TDMS 비동기화 읽기 정지] 함수를 사용하여 데이터 읽기를 정지합니다.
8. [TDMS 고급 닫기] 함수를 사용하여 비동기화 읽기가 끝나면 .tdms 파일을 닫습니다.

TDMS 데이터 비동기화 쓰기

다음 가이드라인은 .tdms 파일에 데이터를 비동기적으로 쓰는 방법을 설명합니다.

1. [TDMS 고급 열기] 함수를 사용하여 .tdms 파일을 엽니다. 비동기화 활성화?의 값이 참인지 확인합니다.
2. [TDMS 채널 정보 설정] 함수를 사용하여 원시 데이터의 채널 정보를 설정합니다.
3. (권장) [TDMS 파일 크기 예약] 함수를 사용하여 쓰기 작업의 디스크 공간을 미리 할당합니다.
4. [TDMS 비동기화 쓰기 설정] 함수를 사용하여 셋팅을 설정하고, 비동기화 쓰기에 대해 버퍼를 할당합니다.
5. [TDMS 고급 비동기화 쓰기] 함수를 사용하여 .tdms 파일에 수집한 데이터를 스트리밍합니다.
6. [TDMS 고급 닫기] 함수를 사용하여 비동기화 쓰기가 끝나면 .tdms 파일을 닫습니다.

웨이브폼

웨이브폼에는 타이밍 정보 및 그에 대응하는 값이 포함됩니다. 웨이브폼 신호를 수집한 후, 신호를 스프레드시트, 텍스트 또는 데이터로그 파일로 쓸 수 있습니다.



관련 정보

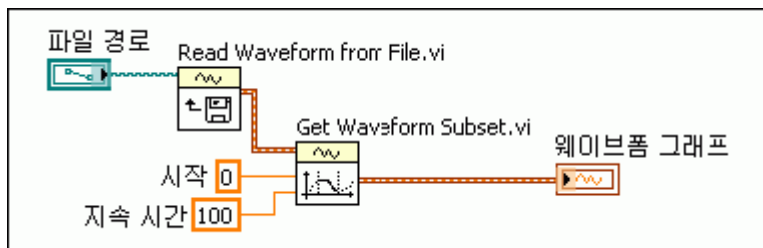
[파일에서 웨이브폼 읽어오기](#) (515 페이지 참조)

[파일에 웨이브폼 쓰기](#) (516 페이지 참조)

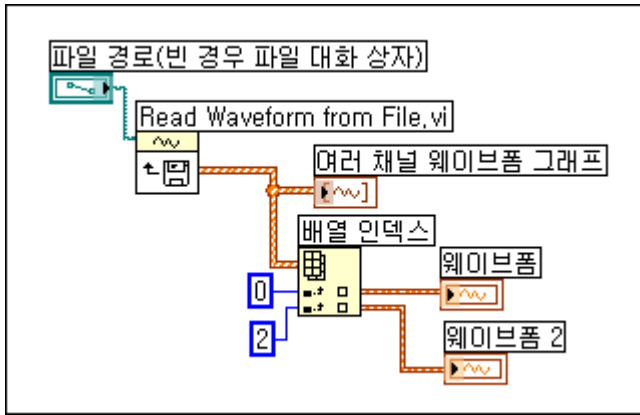
파일에서 웨이브폼 읽어오기

[파일로부터 웨이브폼 읽기] VI를 사용하여 파일에서 웨이브폼을 읽습니다. 단일 웨이브폼을 읽은 후, [웨이브폼 부분 얻기] VI를 사용하여 특정 시간 또는 인덱스에서 웨이브폼의 일부분을 가져오거나, [웨이브폼 속성 얻기] 함수를 사용하여 웨이브폼 속성을 추출할 수 있습니다.

다음 VI는 파일에서 웨이브폼을 읽고, 웨이브폼의 처음 100개 원소를 반환하고, 웨이브폼의 부분을 그래프에 플롯합니다.



[파일로부터 웨이브폼 읽기] VI는 또한 파일에서 여러 개의 웨이브폼을 읽습니다. 이 VI는 멀티플롯 그래프에서 디스플레이할 수 있는 웨이브폼 데이터의 배열을 반환합니다. 파일에서 하나의 웨이브폼에 접근하려면, 반드시 다음 블록다이어그램과 같이 웨이브폼 데이터의 배열을 인덱싱해야 합니다. 이 VI는 여러 개의 웨이브폼을 포함하는 파일에 접근할 수 있습니다. [배열 인덱스] 함수는 파일에서 첫 번째와 세 번째 웨이브폼을 읽고, 두 개의 웨이브폼 그래프에서 이를 각각 그립니다.



또한 스토리지/DataPlugin VI나 [측정 파일로부터 읽기] 익스프레스 VI를 사용하여 파일에서 웨이브폼을 읽을 수 있습니다.



관련 정보

[그래프와 차트의 종류](#) (433 페이지 참조)

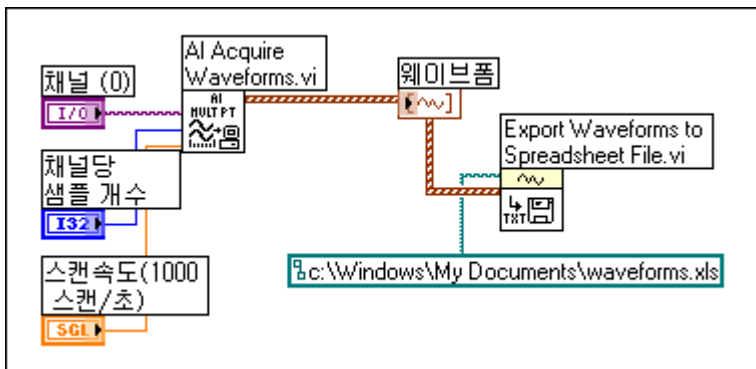
[배열과 클러스터를 이용한 데이터의 그룹화](#) (417 페이지 참조)

파일에 웨이브폼 쓰기

파일에 웨이브폼을 보내기 위해서는 [웨이브폼을 파일에 쓰기]와 [웨이브폼을 스프레드시트 파일로 반출] VI를 사용합니다. 웨이브폼은 스프레드시트, 텍스트, 또는 데이터로그 파일에 쓸 수 있습니다.

생성한 웨이브폼을 VI에서만 사용하려는 경우, 웨이브폼을 **데이터로그 파일**(.log)로 저장합니다.

다음 VI는 여러 개의 웨이브폼을 수집하고, 그래프에 디스플레이하며, 스프레드시트 파일에 씁니다.



또한 스토리지/DataPlugin VI나 [측정 파일에 쓰기] 익스프레스 VI를 사용하여 웨이브폼을 파일에 쓸 수도 있습니다.

LabVIEW의 클래스와 객체

LabVIEW 객체 지향 프로그래밍

LabVIEW 객체 지향 프로그래밍은 캡슐화와 상속 등 C++ 및 Java와 같은 다른 객체 지향 프로그래밍 언어의 개념을 사용합니다. 이러한 개념을 사용하여 어플리케이션 내의 다른 코드 섹션에 영향을 미치지 않고도 관리하고 수정하기 쉬운 코드를 생성할 수 있습니다. LabVIEW의 객체 지향 프로그래밍을 사용하여 사용자 정의 데이터 타입을 생성할 수 있습니다.

LabVIEW 도움말에 있는 객체 지향 프로그래밍에 대한 개념 정보는 객체 지향 프로그래밍 원리를 이해하는 사용자를 대상으로 만들어졌습니다. 다음의 참고자료는 아직 객체 지향 프로그래밍이 익숙하지 않은 사용자가 LabVIEW의 객체 지향 프로그래밍을 사용하는 경우 도움이 될 수 있습니다.

Budd, Timothy. An Introduction to Object-Oriented Programming. Redwood City, Calif.: Addison-Wesley, 2001.

Coad, Peter, and Jill Nicola. Object-Oriented Programming. Upper Saddle River, N.J.: Pearson Education, 1993.

Conway, Jon, and Steve Watts. A Software Engineering Approach to LabVIEW. Upper Saddle River, N.J.: Prentice Hall, 2003.

Weisfeld, Matt. The Object-Oriented Thought Process. Indianapolis, Ind.: Sam's Publishing, 2004.



노트

National Instruments는 객체 지향 프로그래밍 입문용으로 The Object-Oriented Thought Process를 먼저 읽어볼 것을 권장합니다.

추가적인 리소스를 포함하고 있는 LabVIEW object-oriented programming FAQ는 ni.com의 관련 문서를 참조하십시오.

객체 지향 프로그래밍 관련 예제는 `labview\examples\lloop` 디렉토리를 참조하십시오.

관련 예제 찾기

관련 개념

클래스와 객체

LabVIEW 클래스 생성하기

LabVIEW 클래스 개발하기

LabVIEW 클래스 객체 비교하기

LabVIEW 인터페이스

어플리케이션에서 LabVIEW 클래스 및 인터페이스 사용하기

관련 사용법

LabVIEW 클래스 또는 인터페이스 생성하기

LabVIEW 클래스 또는 인터페이스에서 멤버 VI 생성하기

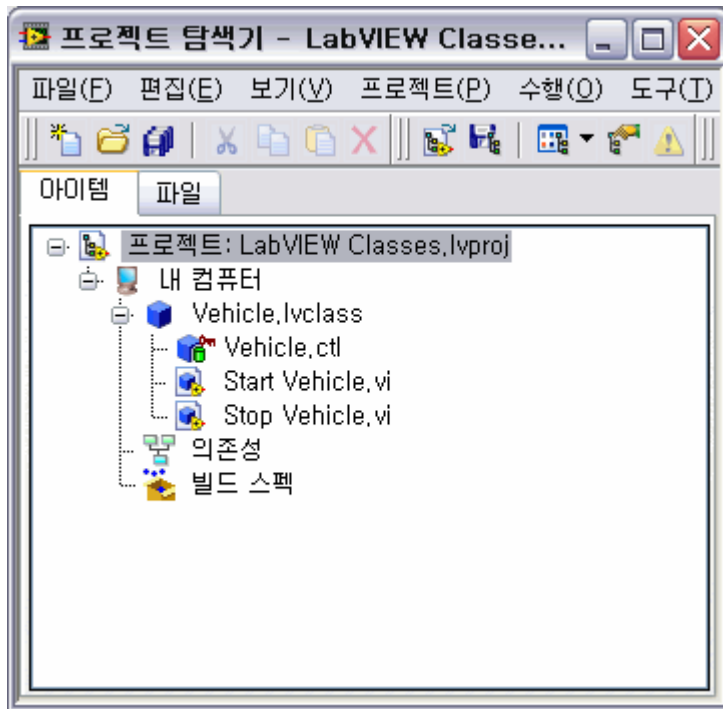
LabVIEW 클래스 또는 인터페이스에 아이템 추가하기

LabVIEW 클래스 또는 인터페이스의 와이어 모양 변경하기

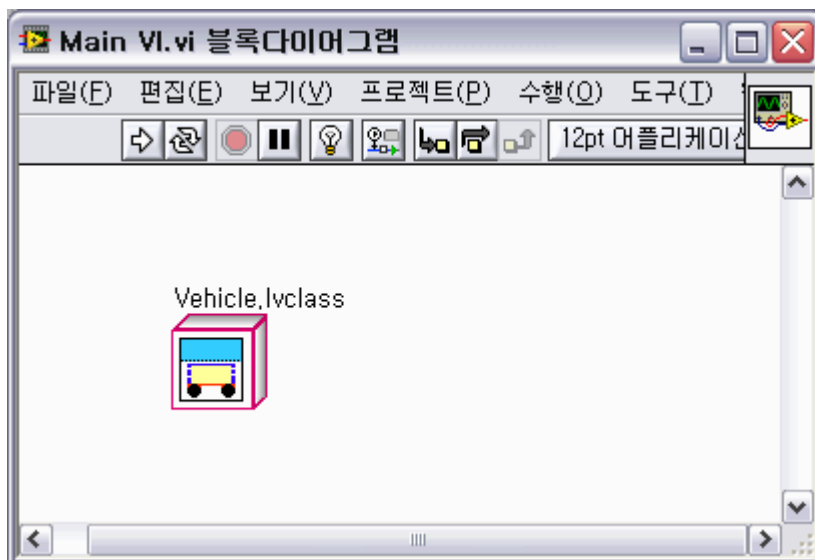
클래스와 객체

객체 지향 프로그래밍에서 클래스는 아이템이 공통적으로 갖는 일반적인 특성을 나타냅니다. 클래스를 운송 수단(Vehicle)에 비유해서 생각해볼 수 있습니다. 운송 수단이 일반적으로 갖는 특성이 있고, 클래스는 이러한 특성들을 정의합니다. 객체는 한 클래스의 특정 인스턴스입니다. 사용자가 운전하고 있는 특정 승용차가 운송 수단(vehicle) 클래스의 객체가 될 수 있습니다. 클래스 정의는 이 차, 즉 객체가 어떻게 작동하는지 결정합니다.

클래스는 클래스 객체와 관련된 데이터와 메소드를 정의합니다. 계속해서 운송 수단을 예로 들어 설명을 하겠습니다. 수많은 운송 수단이 세상에 존재합니다. 일반적으로 모든 타입의 승용차, 트럭, 버스를 운송 수단으로 분류할 수 있습니다. 운송 수단은 차체에는 문을, 변속기에는 기어를 갖고 있습니다. 차문과 변속기 기어의 개수에 대한 정보는 운송 수단에 대한 데이터라고 할 수 있습니다. 운송 수단에서 속도를 높일 수 있고 브레이크를 걸 수도 있습니다. 가속 및 브레이크 걸기는 운송 수단의 동작, 혹은 메소드입니다. 운송 수단과 관련된 데이터와 메소드는 Vehicle 클래스를 정의합니다. 사용자는 객체의 데이터와 메소드를 정의하기 위해 클래스를 생성합니다. 다음 이미지는 Vehicle 클래스를 나타내는 LabVIEW 클래스를 보여줍니다. LabVIEW는 Vehicle 클래스의 데이터를 Vehicle.cti에 저장하고 운송 수단의 메소드를 Start Vehicle VI 및 Stop Vehicle VI라는 이름의 VI에 저장합니다.



객체는 한 클래스의 특정 인스턴스입니다. 사용자가 운전 중인 운송 수단은 Vehicle 클래스의 특정 구현체 또는 Vehicle 클래스의 객체입니다. 다음의 Main VI 블록 다이어그램은 Vehicle 클래스의 객체를 보여줍니다. 객체는 클래스에 의해 정의된 데이터와 메소드를 갖습니다. LabVIEW는 컨트롤에 클래스 데이터를 저장하고, 사용자는 LabVIEW 클래스의 메소드인 멤버 VI를 생성합니다.



LabVIEW 객체 지향 프로그래밍의 용도를 개념화하기 위해, LabVIEW 클래스 사용자를 다음과 같은 방법으로 구분할 수 있습니다:

- LabVIEW 클래스 개발자—개발자와 프로그래머가 사용할 수 있도록 LabVIEW 클래스를 개발합니다. LabVIEW 클래스 개발자는 객체 지향 프로그래밍에 대한 사전 경험이 있습니다. LabVIEW 클래스 개발자는 LabVIEW 클래스와 클래스 구조를 이해해야 합니다.

- LabVIEW 클래스 사용자—LabVIEW 클래스 개발자가 생성한 LabVIEW 클래스를 사용합니다. LabVIEW 클래스 사용자는 객체 지향 프로그래밍의 혜택을 누릴 수 있지만, 어플리케이션에서 이를 사용하기 위해 어떻게 클래스가 운영되는지 그 구조를 반드시 이해하지 않아도 되고 객체 지향 프로그래밍에 대한 사전 경험이 전무한 경우도 가능합니다. LabVIEW 클래스 개발자가 클래스를 배포한 후, LabVIEW 클래스 사용자는 클래스 내부 작업에 접근하지 못할 수 있습니다. LabVIEW 클래스 개발자가 LabVIEW 클래스를 변경하더라도 LabVIEW 클래스 사용자가 개발한 어플리케이션에는 최소한의 영향이 미쳐야 합니다.

LabVIEW 클래스 사용자가 LabVIEW 클래스를 어떻게 생성하는지를 알고 있을 필요는 없지만, 어플리케이션에서 클래스가 정의하는 데이터 타입을 어떻게 사용하는지와 LabVIEW 클래스를 사용하는 코드를 디버깅할 때 어떤 정보를 이용할 수 있는지, 그리고 새로운 버전의 LabVIEW 클래스가 프로그래머가 만든 어플리케이션에 어떤 영향을 미칠 수 있는지를 이해해야 합니다. LabVIEW 클래스를 개발하지 않고 사용하기만 하는 경우, 다른 개발자가 제공하는 LabVIEW 클래스 사용에 대한 구체적인 정보는 [어플리케이션에서 LabVIEW 클래스 및 인터페이스 사용하기](#) 토픽을 참조하십시오.

LabVIEW 클래스 생성하기

[LabVIEW 클래스 생성하기](#)를 통해 사용자 정의 데이터 타입을 LabVIEW에 생성할 수 있습니다. LabVIEW 클래스는 객체와 관련된 데이터뿐만 아니라 데이터에 실행할 수 있는 동작을 정의하는 메소드를 정의합니다. [캡슐화](#)와 [상속](#)을 통해 사용자가 어플리케이션 상에서 코드에 영향을 미치지 않고 변경하기 용이한 모듈형 코드를 생성할 수 있습니다.

LabVIEW에서, 클래스의 데이터는 프라이빗으로서 클래스의 멤버인 vi만이 데이터에 접근할 수 있습니다. 프라이빗 데이터 컨트롤에서 클래스의 데이터를 정의합니다. LabVIEW 클래스를 생성하고 저장할 때, LabVIEW는 새로운 데이터 타입을 정의하는 클래스 라이브러리 파일 .lvclass를 생성합니다. 클래스 라이브러리 파일을 프라이빗 데이터 컨트롤과 vi 리스트 및 다양한 vi 프로퍼티 등의 사용자가 생성한 모든 멤버 vi의 정보를 기록합니다. 클래스 라이브러리는 [프로젝트 라이브러리](#) .lvlib와 유사합니다. 그러나, 클래스 라이브러리는 새로운 데이터 타입을 정의합니다.

프라이빗 데이터 컨트롤은 고유한 클래스 라이브러리 파일로, 새로운 데이터 타입에 대한 데이터 클러스터를 정의하며 클래스 와이어 상의 데이터이기도 합니다. LabVIEW는 디스크에 프라이빗 데이터 컨트롤을 저장하지 않습니다. 대신, LabVIEW는 클래스 라이브러리 파일 안에 이를 저장합니다. 프라이빗 데이터 컨트롤을 클래스 라이브러리 파일 내에 저장하기 때문에 LabVIEW 사용자가 클래스 정의에 부합하지 않는 잘못된 프라이빗 데이터를 사용하는 경우는 없을 것입니다.



팁

LabVIEW 클래스와 동일한 이름의 디렉토리를 디스크에 생성하여 클래스 라이브러리 파일, [클래스 멤버 vi](#) 및 [클래스 사용자 정의 기본 프로브](#)를 저장할 수 있습니다. 디렉토리에 클래스 라이브러리가 소유하는 파일을 포함할 수 있습니다. 만일 하나 이상의 클래스 라이브러리를 동일한 디렉토리에 포함한다면, 같은 이름의 vi를 다른 클래스 라이브러리에 포함하려는 경우 충돌이 발생할 수 있습니다. 개발 프로세스에서 [다이나믹 멤버 vi](#)를 덮어쓰는 경우 이름과 관련된 충돌이 발생할 수 있습니다.

캡슐화

각 LabVIEW 클래스는 데이터 클러스터와 해당 클러스터를 읽고 쓰는 메소드로 구성되어 있습니다. LabVIEW 클래스 데이터는 항상 프라이빗이거나 클래스 멤버가 아닌 VI로부터 숨겨진 상태입니다. 프라이빗 데이터에 접근하려면, 클래스 멤버 VI 형태로 메소드를 생성하여 클래스 프라이빗 데이터에 함수를 사용합니다. 캡슐화는 오직 클래스 멤버 VI를 통해서만 데이터에 접근할 수 있는 클래스 안에 데이터와 메소드를 통합하는 것입니다. 캡슐화를 통해 사용자는 어플리케이션 내의 다른 코드 섹션에 영향을 미치지 않고 쉽게 업데이트하거나 변경할 수 있는 코드 모듈러 블록을 생성할 수 있습니다.

클래스 데이터가 항상 프라이빗이지만, 다양한 수준에서 멤버 VI를 사용자에게 공개할 수 있습니다. 다음의 옵션을 사용하여 메소드의 **접근 영역을 설정**할 수 있습니다:

- 퍼블릭—모든 VI가 멤버 VI를 SubVI로 호출할 수 있습니다.
- 공동 영역—같은 클래스에 있는 VI, 클래스의 friend, 클래스의 friend 라이브러리에 있는 VI만이 멤버 VI를 호출할 수 있습니다. 공동 영역 멤버 VI는 프로젝트 탐색기 윈도우에서 짙은 파란색 열쇠 문양으로 표시됩니다.
- 보호됨—오직 동일한 클래스 혹은 하위 클래스에 속한 VI만 멤버 VI를 호출할 수 있습니다. 보호된 멤버 VI는 짙은 노란색 키 문양을 프로젝트 탐색기 윈도우에 디스플레이합니다.
- 프라이빗—동일한 클래스에 속한 VI만 멤버 VI를 호출할 수 있습니다. 프라이빗 멤버 VI는 붉은색 키 문양을 프로젝트 탐색기 윈도우에 디스플레이합니다.
- 미지정—이 옵션은 폴더를 선택할 때에만 나타납니다. 폴더에 지정된 접근 아이템이 없습니다. 접근은 퍼블릭입니다. 기본 설정으로, 클래스의 폴더는 지정된 접근을 가지고 있지 않으므로 폴더는 퍼블릭으로 접근이 가능합니다.



노트

폴더에 대해 접근 옵션을 지정하는 경우, 접근 셋팅은 폴더의 모든 아이템에 적용되고 폴더의 개별 아이템에 대한 접근 옵션을 덮어씁니다.



노트 공동 영역으로 설정하면 **다이나믹 디스패치 VI**는 깨집니다. 보호된 영역에 있는 다이나믹 디스패치 VI에 friend 접근 권한을 부여하려면, friend가 호출하는 정적 디스패치 래퍼 VI를 생성하고 이 래퍼 VI를 공동 영역으로 설정합니다.

라이브러리의 Friend 지정하기

VI가 공동 영역에 있는 라이브러리의 모든 멤버를 호출할 수 있게하려면, VI를 라이브러리의 friend로 지정합니다. 또한 다른 라이브러리를 라이브러리의 friend로 지정할 수 있습니다.

Friend 속성(friendship)은 전달(transitive)되지 않습니다. 예를 들어, 첫번째 라이브러리가 두번째 라이브러리를 friend로 지정하고, 이 두번째 라이브러리가 세번째 라이브러리를 friend로 명명한 경우, 세번째 라이브러리는 첫번째 라이브러리의 friend로 간주되지 않습니다. 첫번째 라이브러리가 세번째 라이브러리를 friend로 할당하지 않으면, 세번째 라이브러리는 첫번째 라이브러리의 VI에 접근할 수 없습니다. 공동 영역에 있는 라이브러리가 클래스를 friend로 지정하면 이 클래스의 멤버 VI는 이 라이브러리의 VI에 접근할 수 있습니다. 그러나 이 friend 속성(friendship)은 클래스의 하위로 확대되지 않습니다.

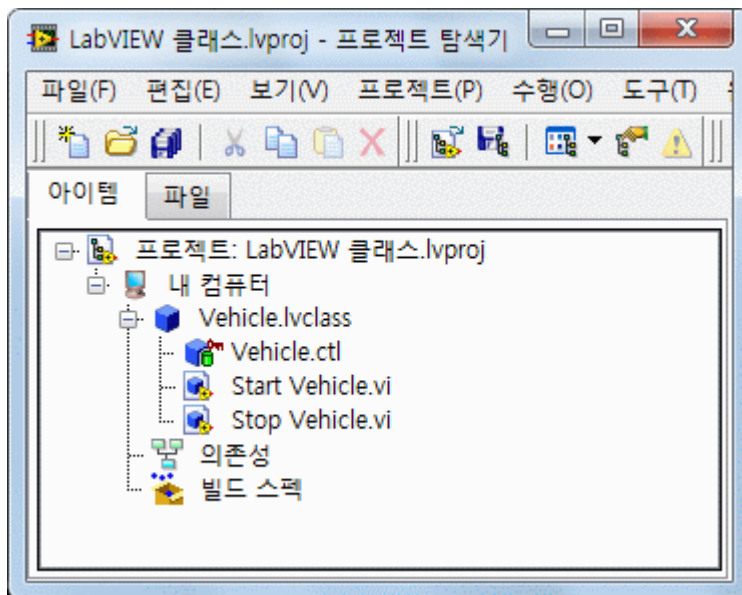
LabVIEW 클래스 밖에서 VI를 생성하고 블록다이어그램에서 퍼블릭 멤버 VI를 SubVI로 사용할 수 있습니다. 퍼블릭 멤버 VI를 사용하면 클래스의 프라이빗 데이터를 조작할 수 있습니다. 멤버 VI의 블록다이어그램에서 프라이빗 및 보호된 멤버 VI를 사용하여 LabVIEW 클래스 사용자에게 공개되지 않은 클래스의 프라이빗 데이터를 조작할 수 있습니다. 클래스 밖의 VI를 클래스의 friend로 지정하여, friend VI에게 클래스의 공동 영역 멤버를 호출하는 권한을 부여할 수 있습니다. 클래스의 출입 포인트를 제한하면 데이터에 오류가 발생할 확률이 줄어들기 때문에 코드 디버깅을 용이하게 할 수 있습니다.

LabVIEW 클래스에서의 캡슐화 및 접근 영역 예제는 다음의 프로젝트를 참조합니다.

- labview\examples\Object-Oriented Programming\Data Encapsulation\Data Encapsulation with LabVIEW Classes.lvproj
- labview\examples\Object-Oriented Programming\Access Scope\Access Scope with LabVIEW Classes.lvproj

프라이빗 데이터 컨트롤 정의하기

LabVIEW는 사용자가 **LabVIEW 클래스를 생성**할 때 자동으로 클래스의 프라이빗 데이터 컨트롤을 생성합니다. 다음의 **프로젝트 탐색기** 윈도우에서 LabVIEW 클래스 아이콘이 유색 큐브임을 확인하십시오. 이 큐브는 LabVIEW 클래스를 나타냅니다. 프라이빗 데이터 컨트롤 아이콘은 초록색 원기둥이 있는 유색 큐브입니다. 이 원기둥은 데이터 스토리지를 나타냅니다. 프라이빗 데이터 컨트롤 아이콘 또한 컨트롤이 프라이빗임을 나타내는 붉은색 키 문양을 디스플레이합니다.

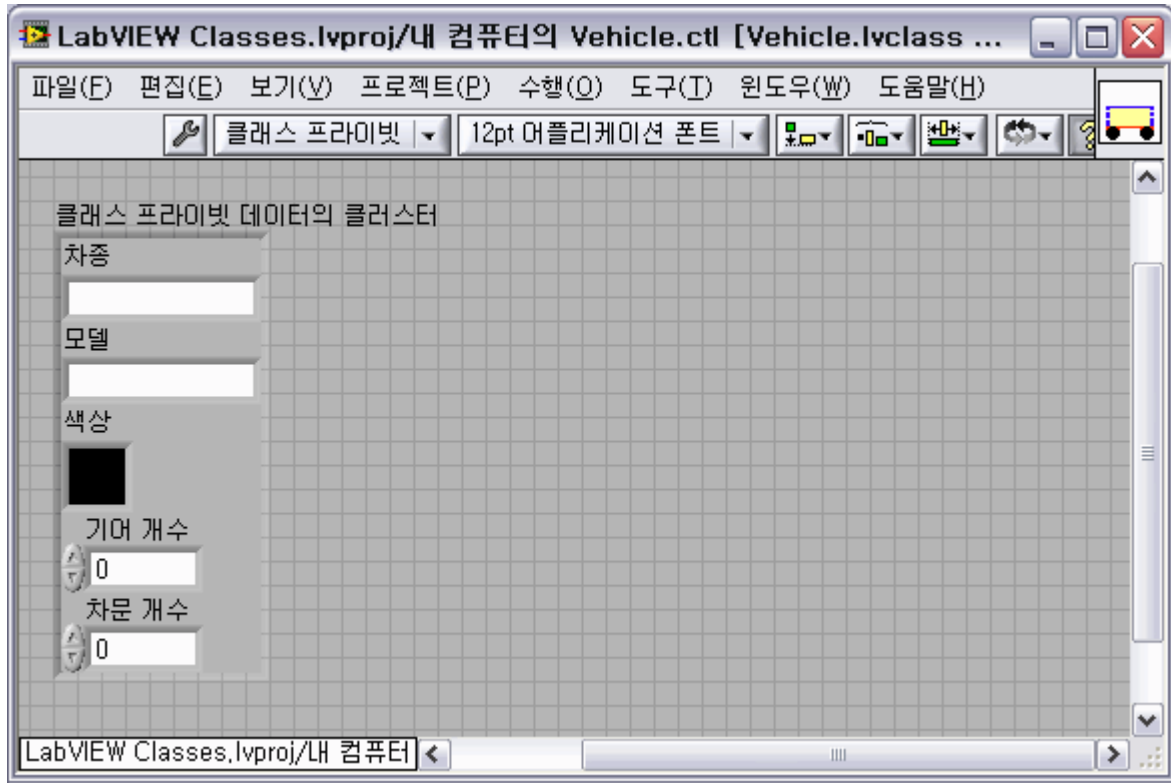


컨트롤 편집기 윈도우를 사용하여 클래스의 프라이빗 데이터 컨트롤을 사용자 정의할 수 있습니다. **프로젝트 탐색기** 윈도우 클래스의 프라이빗 데이터 컨트롤을 더블 클릭하면 LabVIEW는 컨트롤 편집기 윈도우를 디스플레이합니다. 클래스 프라이빗 데이터의 클러스터 안에 컨트롤과 인디케이터를 놓으면 LabVIEW 클래스의 **프라이빗 데이터 타입을 정의**할 수 있습니다. 클래스 프라이빗 데이터의 클러스터 안의 컨트롤에 설정한 기본값은 클래스의 기본값입니다.



노트 프라이빗 데이터 컨트롤은 **x컨트롤**을 포함할 수 없습니다.

다음 예에서, Vehicle 클래스의 데이터 타입은 기어 개수, 차문 개수, 색이라는 세 숫자형과 차종과 모델이라는 두 문자열을 포함합니다.



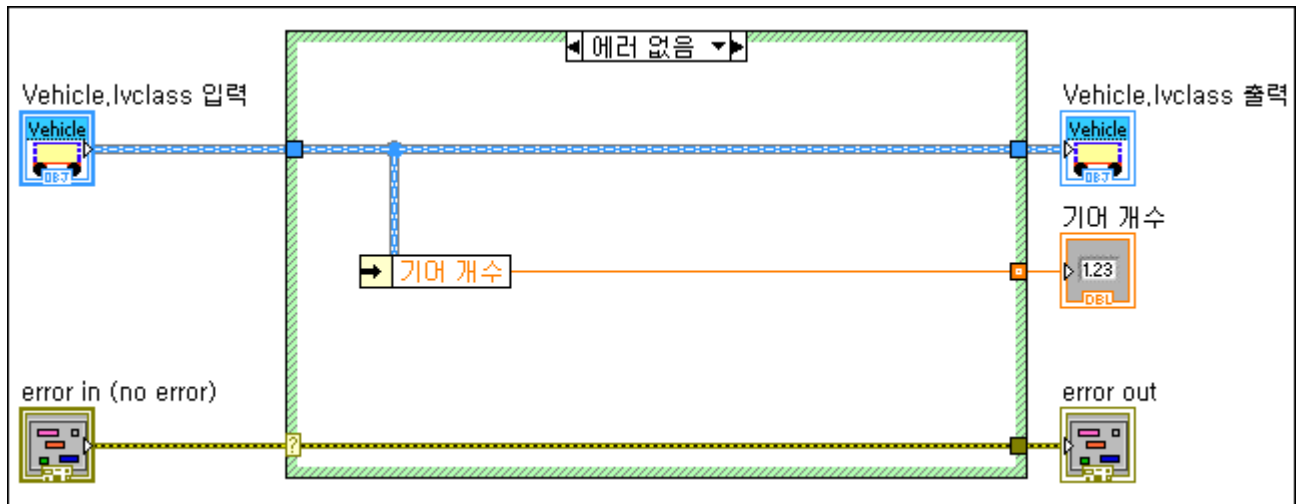
노트 클래스가 프라이빗 데이터를 필요로 하지 않는 경우 클래스 프라이빗 데이터의 클러스터를 빈 상태로 남겨놓을 수 있습니다.

블록 다이어그램이나 프런트패널에서 클래스의 객체를 시각적으로 나타내는 **아이콘**을 생성할 수 있습니다. 클래스 프로퍼티 대화 상자의 일반 셋팅 페이지에 있는 편집 버튼을 클릭하여 아이콘 편집기 대화 상자를 시작합니다. **클래스 아이콘을 생성**합니다. 그러면 LabVIEW는 이 아이콘을 **클래스에 있는 모든 개체에 적용**합니다. 그런 다음에 클래스에 있는 각각의 개체에 대해 **개별적으로 아이콘을 변경**합니다.

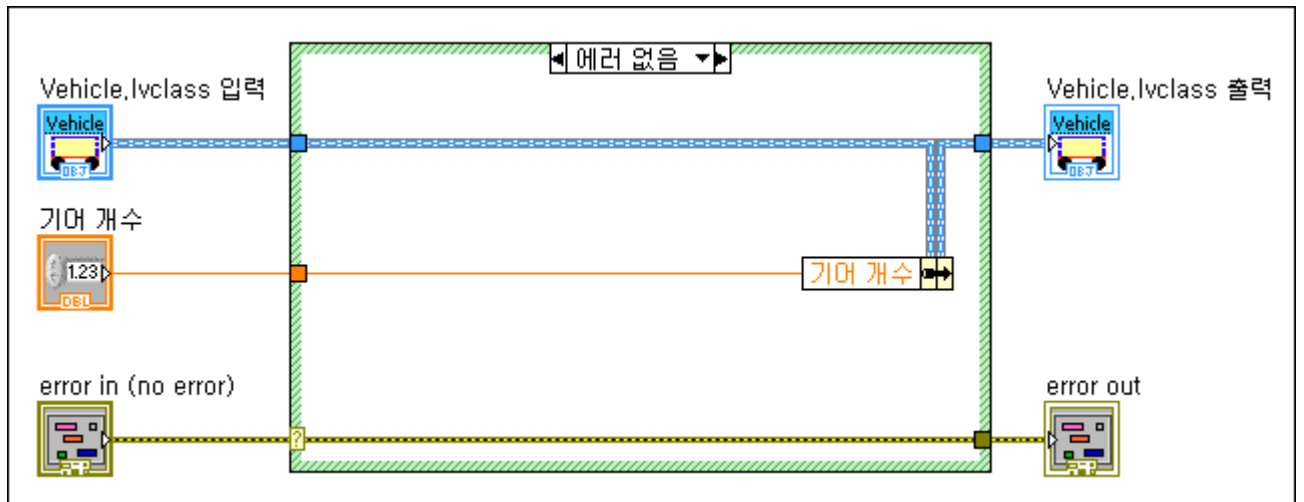
멤버 VI 생성하기

멤버 VI를 생성하여 클래스의 프라이빗 데이터에 작업을 수행합니다. 멤버 VI는 LabVIEW 클래스의 메소드입니다. 멤버 VI는 멤버 VI를 생성한 LabVIEW 클래스의 멤버로, 클래스의 프라이빗 데이터 컨트롤 밑의 프로젝트 탐색기 윈도우에 나타납니다.

LabVIEW는 클래스의 데이터를 클러스터로 정의합니다. 모든 멤버 VI는 클래스 데이터의 클러스터를 읽고 쓸 수 있습니다. LabVIEW는 클러스터의 개별 원소에 **접근하는 VI를 생성**하는 바로 가기를 제공합니다. 이러한 접근자 VI는 LabVIEW 클래스의 멤버이며 클래스 데이터를 읽고 쓸 수 있습니다. 접근자 VI를 생성하여 클래스 데이터를 읽는 경우, LabVIEW는 다음 이미지와 같이 클래스 데이터를 줍니다.



접근자 vi를 생성하여 클래스 데이터에 쓰는 경우, LabVIEW는 다음 이미지와 같이 새 값을 클래스 데이터에 씁니다.



또한 [풀기] 및 [이름으로 풀기] 함수를 사용하여 멤버 vi의 블록 다이어그램에 있는 클래스의 프라이빗 데이터를 풀 수 있습니다. [묶기] 및 [이름으로 묶기] 함수를 사용하여 사용자가 접근하고 조작한 프라이빗 데이터의 클러스터를 다시 묶습니다. 클래스의 데이터가 프라이빗이기 때문에, 멤버가 아닌 vi의 블록 다이어그램에 있는 클래스 데이터와 함께 사용하려고 하면 [묶기]와 [풀기] 노드는 깨집니다.



노트 새 원소를 프라이빗 데이터에 끼워 넣는 경우, 되도록이면 [묶기] 및 [풀기] 함수 대신 [이름으로 묶기] 및 [이름으로 풀기] 함수를 사용하여 vi가 깨지는 것을 방지하도록 합니다.

어떤 값을 풀고, 변경한 후, 다시 객체로 묶는 멤버 vi를 작성하는 경우에는, 구조 경계에 한 쌍의 [풀기]와 [묶기] 노드가 있는 In Place 원소 구조를 사용하는 것이 더 효율적일 수 있습니다. 구조를 사용하면, LabVIEW는 특정 메모리 최적화 전략을 적용하게 됩니다. 일반 [풀기] 및 [묶기] 노드를 사용할 때 이와 똑같은 메모리 최적화 전략이 적용될 수도 있습니다. 하지만, 복잡한 vi를 사용하는 일부 상황에서는 LabVIEW 컴파일러가 관련 최적화 전략의 안전성을 신뢰하지 못해서 이를 적용하지 않고, 결과적으로 메모리 성능을 저해할 가능성이 있습니다. 구조 노드는 이러한 최적화 전략의 안정성을 보장하기 때문에, 반드시 vi에 대한 최적화 전략이 적용됩니다.

멤버 VI는 여러 가지 방법으로 생성할 수 있습니다. 클래스에서 마우스 오른쪽 버튼을 클릭하고 다음의 바로 가기 메뉴 아이템에서 선택하십시오:

- **새로 만들기»VI**—새 멤버 VI를 엽니다.
- **새로 만들기»프로퍼티 정의 폴더**—새 멤버 VI를 만들거나, 기존 멤버 VI를 추가할 수 있는 프로퍼티 정의 폴더를 생성합니다. LabVIEW 클래스에 프로퍼티 정의 폴더가 있을 경우, LabVIEW 클래스를 프로퍼티 노드에 연결해서 프라이빗 데이터에 접근할 수 있습니다.
- **새로 만들기»다이나믹 디스패치 템플릿으로부터 VI**—LabVIEW는 여러 입력 및 여러 출력 클러스터, 여러 핸들링을 위한 케이스 구조, 입력 LabVIEW 클래스, 출력 LabVIEW 클래스로 새 멤버 VI를 채웁니다. LabVIEW는 입력 및 출력 터미널 모두를 **VI 커넥터 팬에서 다이나믹한 것으로** 설정합니다.
- **새로 만들기»정적 디스패치 템플릿으로부터 VI**—LabVIEW는 여러 입력 및 여러 출력 클러스터, 여러 핸들링을 위한 케이스 구조, 입력 LabVIEW 클래스, 출력 LabVIEW 클래스로 새 멤버 VI를 채웁니다. 다이나믹 디스패치 VI와 달리, LabVIEW는 정적 디스패치 VI의 커넥터 팬에서 입력과 출력 터미널을 다이나믹으로 설정하지 않습니다.
- **새로 만들기»데이터 멤버 접근을 위한 VI**—접근자 생성 대화 상자를 엽니다. 이 대화 상자를 사용하여 LabVIEW 클래스 데이터에 접근할 수 있는 멤버 VI를 신속하게 생성합니다.

노트 이 옵션을 사용하기 전에 새 LabVIEW 클래스를 저장해야 합니다. 새 클래스를 저장하지 않으면 LabVIEW는 데이터 멤버 접근을 위한 VI 옵션을 비활성시킵니다.

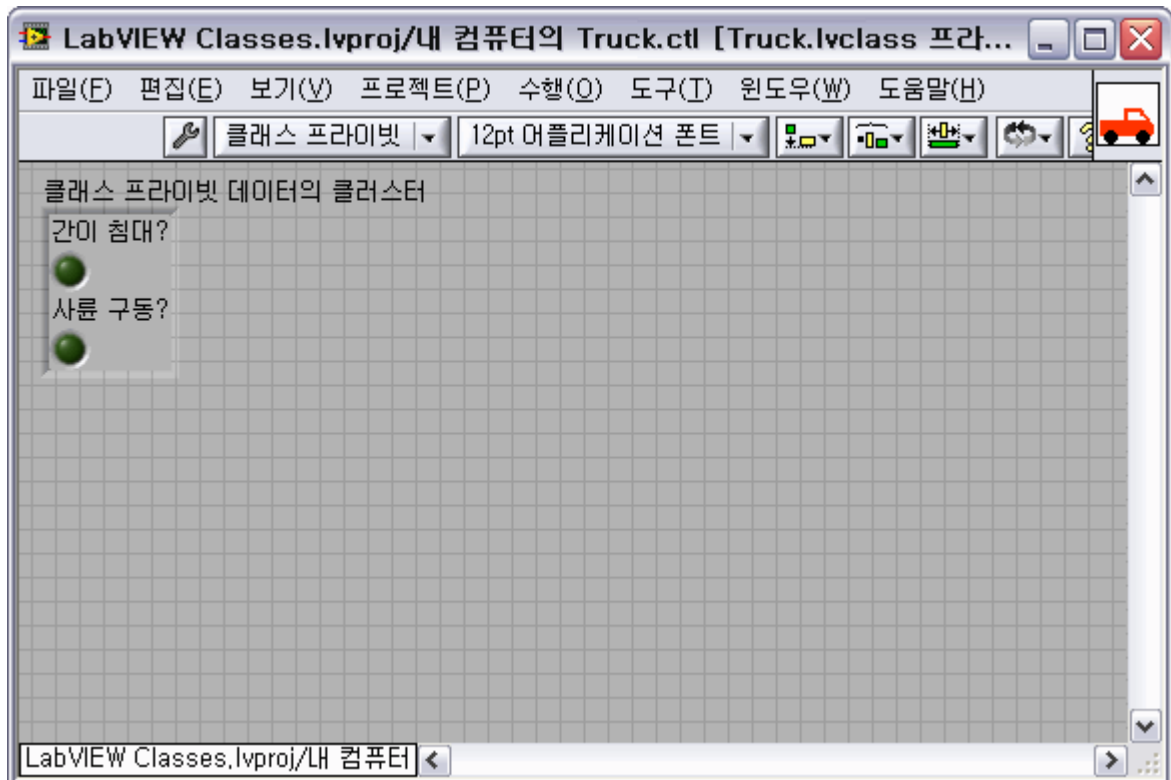
- **새로 만들기»덮어쓰기 위한 VI**—상위 멤버 VI를 덮어쓰는 멤버 VI를 생성합니다. LabVIEW는 상위 VI 아이콘을 사용하며 직하위 클래스의 아이콘 마스크를 덮어써서 새 VI의 아이콘을 생성합니다.

노트 덮어쓸 수 있는 유효한 멤버 VI가 없는 경우 LabVIEW는 새로 만들기»덮어쓰기 위한 VI 옵션을 비활성화합니다. [상속](#) 섹션에서 다이나믹 VI와 덮어쓰기에 대한 추가 정보를 참조하십시오.

프론트패널 또는 블록 다이어그램의 클래스 상수, 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 클래스 라이브러리 보기를 선택하여 프로젝트 탐색기 윈도우의 클래스를 하이라이트합니다. 작업하는 클래스가 LabVIEW 프로젝트에 없으면, LabVIEW는 클래스 윈도우를 열어 클래스를 디스플레이합니다.

상속

상속은 기존의 클래스를 새 클래스의 시작 포인트로 사용할 수 있도록 합니다. 새로 LabVIEW 클래스를 생성하고 다른 클래스로부터 **데이터와 멤버 VI를 상속받도록 설정**하면, 새 클래스는 상속하는 클래스의 퍼블릭 및 보호된 멤버 VI를 사용할 수 있습니다. 또한 자신의 데이터와 멤버 VI를 추가하여 기능을 확장합니다. 예를 들어, Vehicle 클래스의 프라이빗 데이터는 기어 개수와 차문 개수, 색, 차종, 모델을 포함합니다. 만일 Truck이라는 이름을 가진 새로운 클래스를 생성한다면, Truck 클래스가 Vehicle 클래스의 데이터를 상속하도록 설정할 뿐만 아니라, 간이 침대?와 사륜 구동?이라는 볼리언 데이터를 추가할 수 있습니다. 하지만, 다음 그림에서 알 수 있듯이, LabVIEW 클래스를 뭉거나 풀 때 노드는 오직 현재 클래스의 프라이빗 데이터의 터미널만을 보여줄 뿐, 상위 클래스에서 상속한 클래스 데이터의 터미널은 모두 나타내지 않습니다.



상위 데이터는 프라이빗이고, 멤버 VI를 통해 상위 클래스가 제공하는 함수를 사용해서 이를 수정할 수 있습니다. 하위 클래스의 멤버 VI는 LabVIEW 상의 어느 VI처럼 어떤 퍼블릭 멤버 VI든지 호출할 수 있습니다. 하지만 하위 클래스 멤버 VI 또한 상위 클래스의 보호된 멤버 VI를 호출할 수 있습니다. 상위 멤버 VI를 보호된 VI로 지정할 때, 어떤 직하위 클래스의 멤버 VI든지 메소드를 호출할 수 있지만 상속 계층구조 밖에 있는 VI는 이와 같이 호출할 수 없습니다. Truck 클래스에서 Vehicle 클래스의 기어 개수에 접근하고자 한다면, Vehicle 클래스에 Get Gears.vi라고 불리는 퍼블릭 혹은 보호된 멤버 VI를 생성할 수 있습니다. Get Gears.vi의 블록 다이어그램에서 Vehicle 클래스 풀기를 실행하여 기어 개수를 나타낼 수 있습니다. 그 후 기어 개수를 커넥터 팬의 출력 터미널에 지정하면, 이를 통해 Truck 클래스와 같은 하위 클래스에서 Vehicle 클래스의 특정 프라이빗 데이터에 접근할 수 있습니다.

Vehicle 클래스의 **데이터 멤버에 접근하도록 멤버 VI를 생성**해서 기어 갯수에 접근할 수 있습니다. 멤버 VI를 생성할 때, 접근자 생성 대화 상자에서 프로퍼티 노드에서 이용 가능 확인란에 확인 표시를 합니다. 그 후 Truck 클래스를 프로퍼티 노드에 연결하고, 프로퍼티 터미널에서 마우스 오른쪽 버튼을 클릭해서 프로퍼티 선택>> 기어 갯수를 선택할 수 있습니다.



노트 LabVIEW 클래스는 비록 직상위 클래스더라도 다른 LabVIEW 클래스의 프라이빗 멤버 VI를 호출할 수 없습니다. 동일한 클래스에 있는 다른 멤버 VI의 블록 다이어그램에 있는 프라이빗 멤버 VI만을 사용할 수 있습니다.

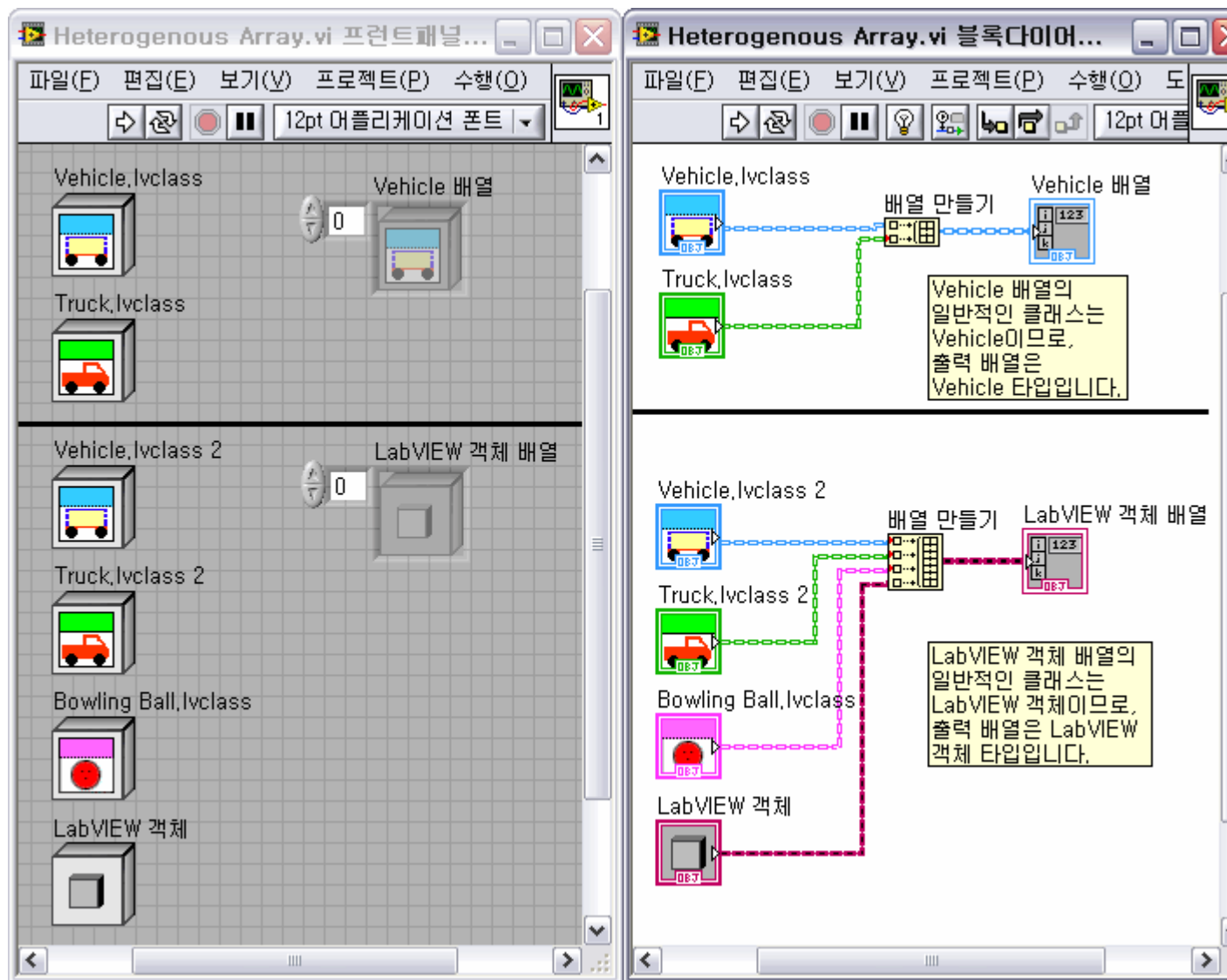


노트(FPGA Module) FPGA VI를 생성할 때 특정 제한이 있는 상속을 사용할 수 있습니다.

LabVIEW 객체

LabVIEW 객체라는 구문은 특정 클래스의 이름입니다. **LabVIEW 객체**는 LabVIEW의 객체 지향 프로그래밍의 **상속 트리**에서 최상위입니다. 기본적으로, 모든 LabVIEW 클래스는 LabVIEW 객체로부터 상속됩니다. LabVIEW 객체를 사용하여 여러 LabVIEW 클래스에서 일반적인 작업을 실행하는 VI를 생성할 수 있습니다. 예를 들어, LabVIEW 클래스 배열을 생성한 경우, 배열의 클래스 타입 또는 어떤 하위 클래스의 원소를 포함할 수 있기 때문에 배열의 데이터는 서로 다릅니다. 배열이 LabVIEW 객체 타입인 경우, 운송 수단 (Vehicle), 트럭(Truck), 볼링 볼(Bowling Ball) 등을 포함할 수 있습니다. Bowling Ball 클래스는 Vehicle이나 Truck 클래스에서 상속되지 않았기 때문에, 이 경우에 LabVIEW는 가장 일반적인 상위 기본 클래스인 LabVIEW 객체의 배열을 생성합니다.

다음 이미지는 Vehicle 및 Truck 클래스를 포함하는 Vehicle 배열을 보여줍니다. Truck이 Vehicle로부터 상속했기 때문에, 이 배열의 일반 상위 기본 클래스는 Vehicle 타입입니다. 이미지는 또한 LabVIEW Object, Vehicle, Truck, Bowling Ball 클래스를 포함하는 LabVIEW 객체 배열을 디스플레이합니다. Bowling Ball은 Vehicle이나 Truck로부터 상속받지 않으나, 세 클래스 모두 궁극적으로 LabVIEW Object로부터 상속받습니다. 따라서, LabVIEW 객체 배열은 LabVIEW Object 타입입니다.



상속 설정하기

모든 LabVIEW 클래스는 기본적으로 LabVIEW 객체로부터 상속됩니다. 다른 클래스에게 상속하는 클래스를 변경하고자 하는 경우, 관련된 클래스를 생성한 후에 **상속 변경**을 해야 합니다. 클래스 프로퍼티 대화 상자에서 상속 및 다른 클래스 옵션을 설정할 수 있습니다. LabVIEW 클래스 계층구조 윈도우에서 **LabVIEW 클래스 계층구조 보기**를 할 수 있습니다. 클래스 상속 계층구조는 다음 타입의 클래스를 포함할 수 있습니다:

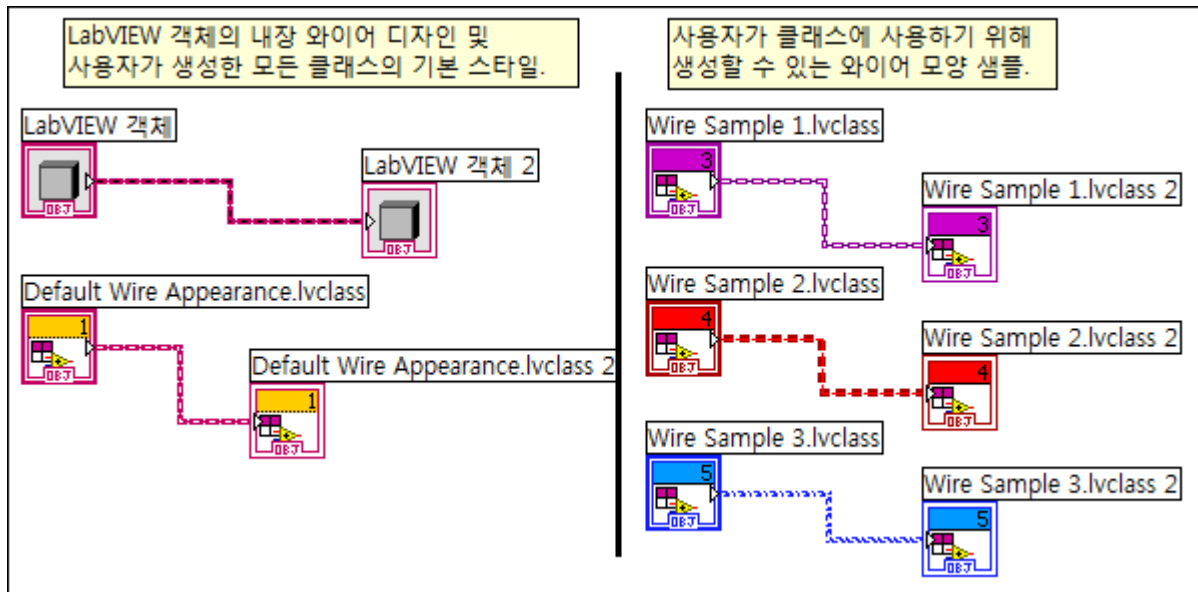
- 직상위 클래스(Parent class)— 다른 LabVIEW 클래스에게 데이터, 퍼블릭 및 보호된 멤버 VI를 상속하는 LabVIEW 클래스.
- 직하위 클래스(Child class)— 직상위 클래스의 퍼블릭 및 보호된 멤버 VI를 상속받는 LabVIEW 클래스. 직상위 클래스가 접근자 VI를 제공하지 않으면, 직하위 클래스는 프라이빗 데이터를 상속하지 않습니다.
- 등위 클래스(Sibling class)— 다른 LabVIEW 클래스와 동일한 직상위 클래스를 공유하는 LabVIEW 클래스.
- 상위 클래스(Ancessor class)— 다른 LabVIEW 클래스의 직상위(parent), 직상위의 상위(grandparent), 직상위 상위의 상위(great-grandparent), 그리고 어떤 형식으로든 상위인 LabVIEW 클래스. LabVIEW 객체는 모든 LabVIEW 클래스의 궁극적인 최상위입니다.
- 하위 클래스(Descendant class)— 다른 LabVIEW 클래스의 직하위(child), 직하위의 하위(grandchild), 직하위 하위의 하위(great-grandchild), 그리고 어떤 형식으로든 하위인 LabVIEW 클래스.



노트 직상위 클래스의 프로퍼티 접근자 VI를 덮어쓰는 VI를 생성하는 경우, 직하위 및 직상위 클래스의 프로퍼티 정의 폴더는 동일한 이름 및 폴더 경로를 가져야 합니다.

와이어 모양

클래스는 새로운 데이터 타입을 정의합니다. 이런 클래스 타입의 와이어는 블록다이어그램에서 기본 LabVIEW 클래스 와이어로 나타나거나 직상위 클래스의 와이어 모양을 상속합니다. 클래스 프로퍼티 대화 상자에서 LabVIEW 클래스의 **와이어 모양을 변경**할 수 있습니다. 읽기 쉬운 블록다이어그램을 생성하기 위해, 적절한 경우에 서로 다른 LabVIEW 클래스의 와이어 모양을 변경할 수 있습니다. 지나치게 많은 와이어 색과 패턴은 블록다이어그램을 읽는 것을 어렵게 할 수 있습니다. 다음 이미지를 보면 왼쪽에서 LabVIEW에 내장된 와이어 모양을 그리고 오른쪽에서 사용자 정의된 와이어 모양의 예를 확인할 수 있습니다.



LabVIEW에서 지나치게 복잡한 와이어 패턴 또는 색을 사용하지 않기 위한 추가적인 팁은 [LabVIEW Style Checklist](#)를 참조하십시오.

다이나믹 및 정적 디스패치 멤버 VI

메소드란 객체에 대해 수행하거나 객체에 의해 수행되는 동작을 지칭합니다. LabVIEW 객체 지향 프로그래밍에서, 메소드는 사용자가 생성하는 멤버 VI입니다. 멤버 VI는 LabVIEW 클래스의 데이터에서 동작합니다. 단일 VI에서 일부 메소드를 정의할 수 있습니다. LabVIEW가 항상 동일한 VI를 호출하기 때문에, 이러한 메소드는 정적 디스패치 메소드라고 합니다. 또한 클래스 계층구조에서 동일한 이름의 여러 VI를 통해 메소드를 정의할 수 있습니다. 런타임 전에는 LabVIEW가 VI 세트에서 정확히 어떤 VI를 호출하는지 알 수 없기 때문에, 이는 다이나믹 디스패치 메소드라고 합니다. 다이나믹 디스패치 메소드는 다형성 VI와 유사합니다. 다형성 VI는 사용자가 연결하는 데이터 타입에 따라 어떤 VI를 호출할지 구분하는 반면에, 다이나믹 디스패치 메소드는 런타임이 시작될 때까지 대기한 후 입력 터미널에 도착하는 데이터에 따라 클래스 계층구조에서 호출할 멤버 VI를 결정합니다.

멤버 VI가 정적일지 또는 다이나믹일지의 여부는 멤버 VI의 커넥터 팬에서 사용자가 지정합니다. 커넥터 팬이 다이나믹 디스패치 입력 터미널을 포함하는 경우, 이 멤버 VI는 다이나믹 디스패치 메소드의 일부입니다. 다이나믹 디스패치 입력 터미널이 없는 경우, 이 멤버 VI는 정적 디스패치 메소드를 정의합니다.

하나의 LabVIEW 클래스가 다른 LabVIEW 클래스로부터 상속하는 경우, 직하위 클래스는 직상위 클래스에 정의된 모든 퍼블릭 및 보호된 메소드를 상속합니다. 직하위 클래스의 멤버 VI를 직상위 클래스의 멤버 VI와 완전히 동일한 이름으로 지정함으로써 직하위에서 메소드가 어떻게 구현될지도 정의됩니다.

LabVIEW의 정적 디스패치 메소드는 단일 VI에 의해 정의되는 것이기 때문에, 상위 클래스의 정적 디스패치 멤버 VI와 동일한 이름으로 직하위 클래스 멤버 VI의 이름을 지정하지 마십시오. 예를 들어, 만약 직상위인 Vehicle 클래스에 Open Door VI라는 정적 디스패치 멤버 VI가 포함되어 있다면, 직하위인 Truck 클래스는 Open Door VI라는 멤버 VI를 가질 수 없습니다. Truck은 Vehicle에서 멤버 VI를 상속하기 때문에 Open Door VI는 Truck에서 이미 정의되어 있습니다. 정적 디스패치 멤버 VI를 블록 다이어그램 상에서 SubVI처럼 놓는다면, 모든 점에서 일반 SubVI 호출처럼 동작합니다.

여러 개의 다이나믹 디스패치 VI 메소드를 상속 계층구조의 각 레벨마다 하나씩 정의할 수 있습니다. 다이나믹 디스패치 VI가 직상위 클래스에서 정의되었고 사용자 또한 이를 직하위 클래스에서 정의하는 경우, 직하위 클래스에서의 구현이 직상위의 구현을 덮어쓰거나 확장시킵니다. 다음의 예에서, Vehicle 클래스와 Truck 클래스 모두 다이나믹 디스패치 메소드인 Set Make VI의 실행을 정의합니다. 블록다이어그램에 다이나믹 디스패치 VI를 SubVI로 놓는다면, 블록다이어그램의 노드는 LabVIEW가 편집 모드에 있을 때 호출하는 일반 SubVI처럼 작동합니다. 그러나 VI를 실행한다면, 다이나믹 디스패치 입력 터미널에 들어오는 데이터가 LabVIEW가 호출하는 클래스 계층구조 내 다이나믹 멤버 VI의 실행을 결정합니다. LabVIEW 클래스 와이어가 자신의 타입 혹은 직하위 타입 데이터를 전달할 수 있기 때문에, 노드는 사용자가 클래스 데이터로 정의한 모든 다이나믹 디스패치 VI를 실행합니다. 다음 예제를 다시 한 번 검토하십시오. Set Make VI로 구현한 Vehicle 클래스만 Main VI의 블록다이어그램에 있습니다. 루프의 첫번째 반복에서 LabVIEW는 Vehicle 클래스의 데이터가 클래스 와이어에 있기 때문에 Set Make VI로 구현한 Vehicle 클래스를 실행합니다. 루프의 두번째 반복에서 LabVIEW는 트럭 클래스의 데이터가 클래스 와이어에 있기 때문에 차종 설정 VI의 트럭 클래스를 실행합니다.

직상위 클래스가 다이나믹 디스패치 VI를 정의하지만 그 VI를 구현하지 않는 경우, 각 직하위 클래스가 직상위 VI를 덮어써야 합니다. 많은 경우, 사용자가 직상위 클래스 VI에서 의미있는 동작을 제공하는 것이 불가능합니다. 예를 들어, Area VI를 정의하는 Shape이라는 클래스가 있다고 가정해봅시다. Area VI는 Shape 객체에서 모양의 넓이를 반환합니다. 모든 모양, 심지어는 대다수 모양의 넓이 연산할 수 있는 단 하나의 수식은 존재하지 않습니다. 따라서, 각 하위 클래스는 모양에 대응하는 수식으로 Area VI를 덮어써야 합니다. 그 결과, Circle이라는 직하위 클래스를 생성하는 경우, Circle 클래스는 $\pi \times \text{반지름} \times \text{반지름}$ 을 계산하는 Area VI를 제공해야 합니다.

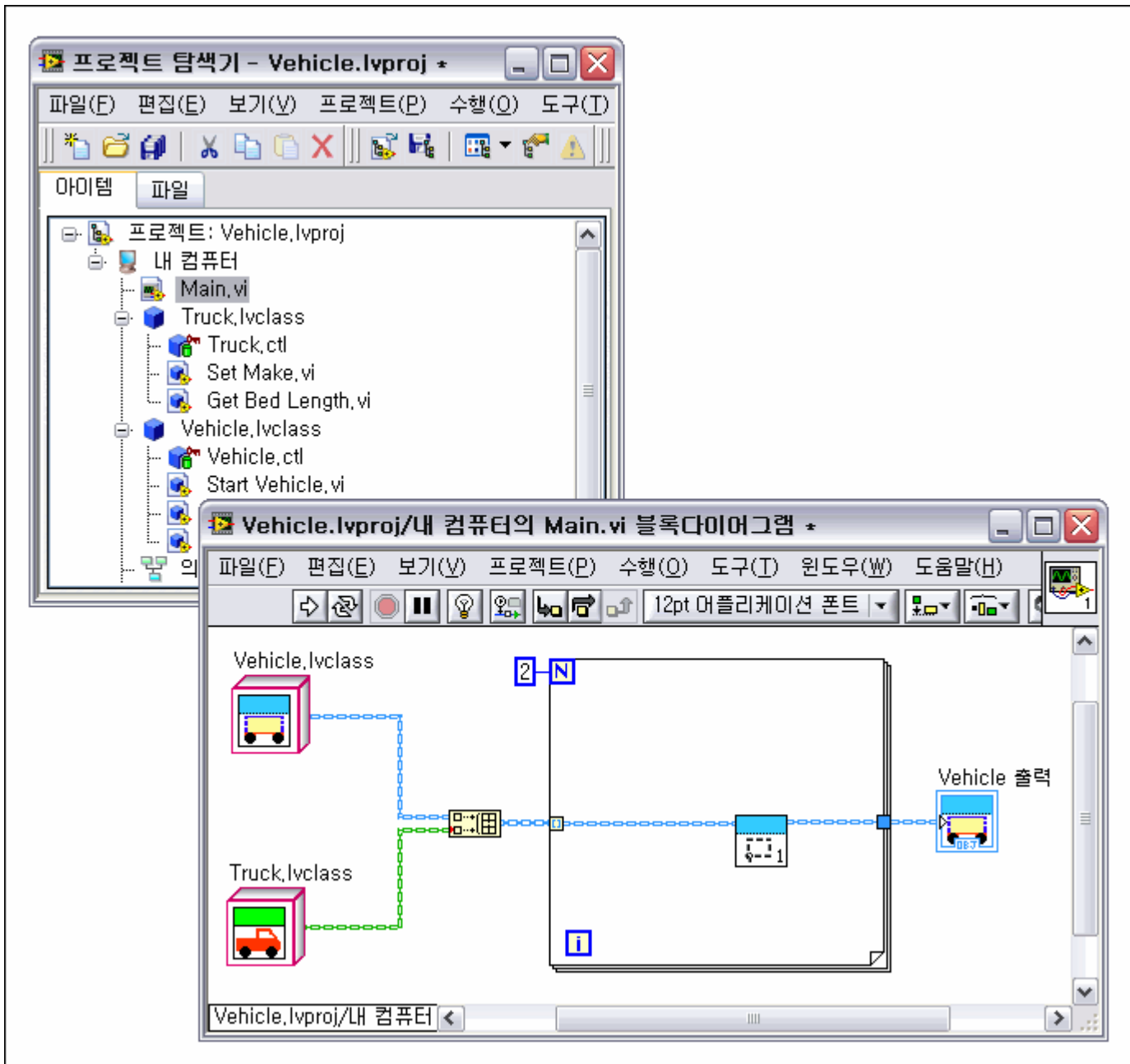
직상위 클래스 VI의 구현을 정의하지 않은 경우, 직상위 VI는 단순히 커넥터 팬을 정의하고 직상위 VI를 덮어써야 하는 다른 VI가 모두 일치시켜야 할 VI 프로퍼티를 정의할 뿐입니다. 각 직하위 클래스는 이를 덮어쓰는 VI를 제공해야 합니다. LabVIEW가 이 요구사항을 적용하도록 하려면, 직상위 클래스에 있는 **VI를 표시**해서 각 직하위 클래스가 덮어써야 하는 VI로 설정하고 이 요구조건을 LabVIEW가 적용하도록 만듭니다.

일부 직하위 클래스는 멤버 VI의 기능을 덮어쓰지 못할 수도 있습니다. 예를 들어, Shape(모양) 클래스의 직하위 클래스로 Quadrilateral(사각형)이 있는 경우, 이 직하위 클래스에 속하는 사각형이 어떤 종류인지 모르면 그 넓이를 계산할 정확한 수식을 제공할 수 없습니다. Quadrilateral 클래스 안에서 Area VI가 구현될 때 값을 생성하지 못하는 경우가 없도록, Quadrilateral 클래스가 직하위 클래스에 덮어쓰기 요청을 전송하도록 지시할 수 있습니다. 이를 위해, 직하위 클래스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 클래스 프로퍼티 대화 상자를 디스플레이합니다. 상속 페이지에서 모든 덮어쓰기 요청을 하위 클래스에 전송 확인란에 확인 표시를 합니다. 그러면 LabVIEW에서 Trapezoid, Rectangle과 같은 Quadrilateral 클래스의 직하위 클래스들이 Area VI를 덮어쓰게 됩니다.

하위 클래스가 직상위 클래스 멤버 VI를 덮어쓰도록 요청한다고 멤버 VI가 실행되는 방식에 영향을 미치지 않습니다.



노트 LLB는 동일한 이름의 파일을 가질 수 없습니다. 따라서 클래스 계층구조에서 동일한 이름을 갖는 다이나믹 멤버 VI들이 있다면, 같은 LLB에 이 클래스들을 넣을 수 없습니다.



노트 직하위 클래스의 VI가 직상위 클래스의 VI를 덮어쓸 때, 직하위의 VI는 직상위의 VI와 재호출 셋팅, 선호하는 실행 셋팅, 우선 순위 셋팅, 커넥터 팬 터미널, 커넥터 팬 패턴, **접근 영역**이 일치해야 합니다.

블록다이어그램의 다이나믹 디스패치 SubVI를 더블 클릭하고 구현 대상 선택 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 현재 메모리에 있는 다이나믹 디스패치 SubVI의 모든 구현 대상을 본 후 SubVI의 **하나 또는 그 이상의 구현 대상 열기**를 합니다.

새로 만들기 >> 덮어쓰기 위한 VI를 선택하여 다이나믹 디스패치 멤버 VI의 상위 구현을 덮어쓸 VI를 생성하는 경우, 또다른 다이나믹 디스패치 멤버 VI를 생성하게 됩니다. 왜냐하면, 상위 멤버 VI를 덮어쓰는 VI는 상위와 똑같은 이름을 갖고 또한 다이나믹 디스패치 터미널을 포함하기 때문입니다. LabVIEW는 [직상위 클래스 메소드 호출] 노드를 적절한 **다이나믹 디스패치 입력 및 출력 클래스 터미널**과 VI가 상위 VI와 일치하기 위해 필요한

다른 터미널과 함께 블록 다이어그램에 놓습니다. LabVIEW는 덮어쓰기 상위 멤버 VI가 없는 경우 덮어쓰기 위한 VI 옵션을 비활성화합니다.



노트(FPGA Module) FPGA VI를 생성할 때 특정 제한이 있는 다이나믹 디스패치를 사용할 수 있습니다.

다이나믹 디스패치 출력

커넥터 팬의 출력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 다이나믹 디스패치 출력(권장)을 선택하여 LabVIEW 클래스 출력 터미널을 다이나믹으로 설정할 수 있습니다. 다이나믹 디스패치 출력 터미널이 있는 VI를 SubVI로 호출하면, 다이나믹 디스패치 출력은 다이나믹 디스패치 입력 터미널에 연결된 와이어와 같은 데이터 타입으로 변경됩니다. 한 예로, Vehicle 클래스의 와이어를 다이나믹 디스패치 입력 터미널에 연결하면, 멤버 VI의 출력은 입력과 같습니다. 이 경우 출력은 Vehicle 클래스입니다. 사용자는 다이나믹 디스패치 입력 터미널과 다이나믹 디스패치 출력 터미널 사이에 있는 데이터를 수정할 수 있습니다. 그러나, LabVIEW 클래스의 런타임 안정성을 보장하려면, 다이나믹 디스패치 입력 터미널로부터의 데이터는 모든 다이나믹 디스패치 출력 터미널로 연결되어야 합니다. 또한, LabVIEW가 정확히 한 번에 다이나믹 디스패치 입력 터미널로부터 읽기를 하고 다이나믹 디스패치 출력 터미널로 쓰기를 하도록 하기 위해서, 사용자는 다이나믹 디스패치 블록 다이어그램 터미널을 구조 안에 놓을 수 없습니다.



노트 다이나믹 디스패치 입력과 출력이 있는 다이나믹 디스패치 멤버 VI를 디버깅하고 있다면, 다이나믹 디스패치 입력에서 출력으로 연결되는 와이어에 에러가 있는지 확인할 수 있습니다. 다이나믹 입력에서 시작하는 와이어가 런타임 데이터 타입을 변경할 수 있는 어떤 함수도 통과하지 않는 경우, 와이어 배경색은 보통의 흰색이 아닌 회색으로 나타납니다. 와이어가 데이터 타입을 변경할 수 있는 함수를 통과하는 경우, 와이어 배경색은 빨간색으로 바뀝니다. 다이나믹 출력이 바르게 작동하려면, LabVIEW 클래스의 데이터 타입을 바꿔서는 안 됩니다.

LabVIEW 클래스 출력 데이터가 입력과는 다르게 나타날 작업을 멤버 VI의 블록 다이어그램에서 완성하려는 경우, 커넥터 팬의 LabVIEW 클래스 다이나믹 디스패치 터미널이 다이나믹 디스패치 출력(권장)이 아닌 권장으로 설정되었는지 확인합니다. 한 예로, LabVIEW 클래스 입력이 Vehicle이고 LabVIEW 클래스 출력이 Truck이기를 사용자가 원한다면, 커넥터 팬에서 LabVIEW 클래스의 기본 터미널을 변경해야 합니다. 다른 방법으로, 새 VI를 사용하여 **멤버 VI 생성**을 해서 수동으로 커넥터 팬 터미널을 설정할 수 있습니다.



노트 다이나믹 디스패치 입력과 출력이 있는 멤버 VI의 케이스 또는 이벤트 구조를 사용하는 경우, 출력 터널에서 연결되지 않으면 기본값 사용을 선택할 수 없습니다. 출력 터널에서 연결되어 있지 않으면 기본 사용을 사용하면, LabVIEW는 VI를 깨뜨립니다. 구조 안에 있는 모든 케이스를 반드시 연결해야 합니다. 자동으로 입력 및 출력 터널이 연결되도록 **터널을 설정**하는 방법도 있습니다.

다이나믹 디스패치를 위한 메모리 복사본의 최적화

위에서 설명한 것처럼, **다이나믹 디스패치 메소드**를 생성할 때, 각 직하위 클래스는 직상위 클래스에서 정의한 퍼블릭 및 보호된 메소드를 모두 상속합니다. 직상위 클래스는 이러한 멤버 VI를 자체적인 버전으로 덮어쓰거나 확장시킬 수 있습니다. VI가 다이나믹 디스패치 메소드를 호출할 때, LabVIEW는 런타임이 될 때까지 어느 버전의 메소드를 가져

오게 될지를 알지 못합니다. 따라서 LabVIEW는 호출자 VI의 메모리 할당을 최적화하여, 직하위 클래스가 포함하는 멤버 VI는 모두 직상위 클래스에 있는 멤버 VI와 똑같이 설정될 것이라고 가정합니다. 직상위 VI에서 입력이 상수인 경우, 호출자 VI는 직하위 VI에서도 이 입력이 상수라고 가정합니다. 직상위 클래스 멤버 VI에서 입력이 출력을 반환하는 경우, 호출자 VI는 이 입력이 모든 직하위 클래스 멤버 VI에서도 똑같은 동작을 취할 것으로 가정합니다.

이러한 가정 중 어느 하나라도 틀린다면, 최적화 동작은 소용없게 됩니다. 예를 들어, 사용자가 변경되지 않은 여러 입력을 포함하는 멤버 VI를 생성하는 경우, 호출자 VI는 이 입력들이 앞으로 영원히 변경되지 않을 것이라고 가정합니다. 하지만, 어떤 덮어쓰기 동작이든지 모두 일부 또는 전체 입력을 변경할 수 있습니다. 또 다른 예로, 직상위 클래스 멤버 VI에서 어떤 입력도 출력에 와이어로 연결하지 않은 경우, 호출자 VI는 입력과 메모리를 공유하는 출력이 하나도 없다고 가정합니다. 하지만, 이후에 이어지는 멤버 VI에서 입력이 출력에 연결될 수도 있습니다. LabVIEW는 직하위 클래스에서 기대하지 않은 동작이 발생할 때 이를 처리할 코드를 생성해야 하며, 이로 인해 최적화 동작의 효율성이 떨어지게 됩니다.

직하위 클래스 VI에서 발생 가능한 모든 경우를 반영하여, 최대한 일관성 있는 직상위 클래스 VI를 작성함으로써 최적화 전략을 더욱 개선할 수 있습니다. 각 터미널에서 예상하지 못한 동작을 정확하게 나타낼 수 있는 다이나믹 디스패치 메소드를 생성할 수 있도록, In Place 원소 구조를 사용하는 것도 최적화 전략을 개선하는 하나의 방법입니다. In Place 원소 구조를 직상위 클래스 내에 있는 다이나믹 디스패치 VI 안에 놓고, In Place 원소 구조에 여러 쌍의 In Place 원소 노드를 놓습니다. 이러한 노드 쌍을 사용함으로써, 어떤 입력을 어떤 출력과 연결하고, 어떤 입력을 상수로 취급하고, 어떤 입력을 변경된 것으로 취급할 지 등을 LabVIEW에게 전달할 수 있습니다. 이를 통해 LabVIEW는 호출자 VI를 최적화할 수 있게 됩니다.

다이나믹 디스패치 터미널 사용 예제는 다음의 예제를 참조합니다.

- labview\examples\Object-Oriented Programming\Dynamic Dispatching\Dynamic Dispatching.lvproj
- labview\examples\Object-Oriented Programming\Dynamic Terminals\Dynamic Terminals.lvproj

LabVIEW 클래스 개발하기

LabVIEW 객체 지향 프로그래밍이 다른 객체 지향 프로그래밍과 개념상 유사하지만, LabVIEW가 데이터 흐름 프로그래밍 환경이기 때문에 LabVIEW가 LabVIEW 클래스 데이터를 다루고 상호작용하는 법 및 사용자가 클래스 코드를 디버깅하는 법에서 어느 정도 차이가 있습니다.

LabVIEW의 객체는 참조가 아닌 값으로 인식됩니다. LabVIEW는 **클러스터와 배열**에 전통적으로 적용되는 동일한 규칙을 따르는 객체 복사본을 생성합니다.

컨스트럭터와 디스트럭터

LabVIEW 객체 지향 프로그래밍의 컨스트럭터와 디스트럭터는 함축적입니다. LabVIEW 클래스 데이터를 초기화하기 위해 컨스트럭터를 호출할 필요는 없습니다. LabVIEW는 클래스를 초기화해야 할 필요가 있을 때마다 언제든지 기본 컨스트럭터를 호출합니다. 클래스 초기화는 보통 프런트패널 컨트롤이나 블록 다이어그램 상수에서 진행됩니다. LabVIEW는 사용자가 프라이빗 데이터 컨트롤에 설정한 기본값으로 클래스 값을 초기화

합니다. LabVIEW가 더 이상 LabVIEW 클래스 정보를 필요로 하지 않을 때 LabVIEW는 **클러스터 및 배열**의 경우와 같은 방법으로 메모리 할당 해제 작업을 합니다. 클래스 데이터에 다른 값을 설정하고 싶다면, 새 값을 설정하는 멤버 VI를 생성해야 합니다. 한 예로, Vehicle 클래스를 생성할 때 Vehicle 클래스의 기어 개수 기본값을 3으로 설정한 후 Truck 클래스가 상속하는 기어 개수에 다른 값을 지정하고 싶은 경우에, 사용자는 Vehicle 클래스의 값을 변경하는 멤버 VI를 생성해야 합니다. 새 값을 설정하는 한 방법으로, 입력은 없지만 기어 개수 값을 지정하고자 하는 클래스를 출력하는 멤버 VI를 생성하여, 클래스 데이터 타입의 블록 다이어그램 상수를 생성하는 것이 있습니다.

데이터 패턴화 및 패턴화 해제하기

LabVIEW는 데이터를 **패턴화된 데이터**로 저장합니다. [패턴화된 문자열로]와 [문자열로부터 패턴화 해제] 함수는 모든 클래스 데이터 타입을 처리할 수 있습니다. LabVIEW는 자동으로 데이터를 패턴화 및 패턴화 해제 합니다. 모든 **타입 설명자**가 LabVIEW에서 패턴화된 데이터 타입의 데이터 타입을 갖고 있지만, LabVIEW 클래스의 패턴화된 데이터는 자체적으로 이 정보와 클래스 버전 정보를 갖고 있습니다. 패턴화 해제된 LabVIEW 클래스 관련 정보를 LabVIEW 클래스가 보유하고 있기 때문에, 사용자가 클래스를 이동시키거나 삭제해서 LabVIEW 클래스를 찾을 수 없는 경우에는 LabVIEW가 데이터를 패턴화 해제할 수 없으며 에러 메시지가 나타납니다. 찾을 수 없는 SubVI가 있을 때 LabVIEW가 보여주는 동작과 유사하게, 멤버 VI를 열어보면 LabVIEW 클래스 데이터를 찾지 못하는 컨트롤을 LabVIEW가 비활성화시킨 것을 확인할 수 있습니다. 깨진 컨트롤과 대응하는 LabVIEW 클래스를 로드할 경우, 데이터는 패턴화 해제되며 컨트롤은 더 이상 깨져있지 않습니다.



노트

또한 [패턴화된 XML로] 및 [XML로부터 패턴화 해제] 함수를 사용하여 클래스 데이터를 처리할 수 있습니다.

데이터 변경하기

메모리에 LabVIEW 클래스가 가지고 있는 버전보다 더 최근의 LabVIEW 클래스 버전으로 컨트롤 데이터가 쓰여진 경우, LabVIEW 클래스 컨트롤은 경고 리포트를 합니다. LabVIEW는 사용자가 프라이빗 데이터 컨트롤 또는 클래스 상속을 바꿀 때마다 LabVIEW 클래스의 버전 번호를 증가시킵니다. 클래스 프로퍼티 대화 상자에서 LabVIEW 클래스 버전 번호를 확인할 수 있습니다. LabVIEW가 자동으로 클래스 버전 번호를 증가시키는 경우는 다음과 같습니다:

- LabVIEW 클래스 상속이 변경될 경우.
- LabVIEW 클래스의 프라이빗 데이터 컨트롤이 변경될 경우. 컨트롤 추가, 대체, 순서 재설정, 삭제: 컨트롤형 변경: 수정된 타입 정의 업데이트 등이 있습니다.
- 새 LabVIEW 버전에 LabVIEW 클래스가 로드하는 경우.
- 클래스 프로퍼티 대화 상자에서 사용자가 수동으로 버전 번호를 증가시킵니다. 새 클래스 개정을 나타내기 위해 사용자가 이러한 작업을 할 수 있습니다.

클래스의 이름을 바꿀 경우, LabVIEW는 이를 새로운 클래스로 인식하여 클래스의 변경 히스토리를 삭제하고 버전 번호를 1.0.0.0.으로 재설정합니다. LabVIEW가 클래스 이름을 변경하는 경우는 사용자가 다음과 같은 작업을 할 때입니다:

- 폴다운 메뉴에서 파일 >> 다른 이름으로 저장을 선택하여 클래스 이름을 바꾸는 경우.

- 클래스를 소유 라이브러리로 이동시키는 경우.
- 클래스를 소유 라이브러리에서 이동시키는 경우.
- 디스크의 .lvclass 파일 이름을 바꾸는 경우.



노트 다른 변경사항 없이 원래의 클래스 이름으로 되돌리는 경우, 변경 히스토리가 더 이상 존재하지 않기 때문에 이 클래스는 원래 클래스와 동일하지 않습니다. 예를 들어, Vehicle 클래스를 Vehicle #1으로 이름을 바꾼 후 다시 Vehicle로 변경할 경우, Vehicle 클래스는 원래의 클래스와 동일하지 않습니다.

LabVIEW 클래스의 버전 번호를 추적하는 것은 LabVIEW 클래스를 통해 어플리케이션을 개발하는 LabVIEW 클래스 사용자에게 유용합니다. 예를 들어, 클래스의 프라이빗 데이터 컨트롤에서 부호없는 32비트 정수를 사용하는 어플리케이션 상에 사용자가 LabVIEW 클래스를 갖는 경우가 있습니다. LabVIEW 클래스 개발자는 프라이빗 데이터 컨트롤의 숫자형 컨트롤을 배정도 부동수로 변경시키는 LabVIEW 클래스 버전을 사용자에게 보낼 수 있습니다. LabVIEW가 버전 변경사항을 추적하며 모든 LabVIEW 클래스에서 어떻게 데이터를 패턴화 및 패턴화 해제하는지 알고 있기 때문에, 사용자는 이전 버전의 LabVIEW 클래스를 새로운 버전으로 대체하고 어떠한 변경도 없이 프로그램을 실행할 수 있습니다.



노트 사용자가 이후 LabVIEW 클래스 버전으로부터의 데이터 패턴화 해제를 시도하는 경우, LabVIEW는 에러를 반환합니다. 예를 들어, 버전 번호가 1.0.0.3인 데이터를 갖고 있지만 메모리 상의 LabVIEW 클래스 버전은 1.0.0.2일 경우, 이러한 에러가 발생할 수 있습니다.

LabVIEW에서 이전 버전의 LabVIEW 클래스가 발견되는 경우, LabVIEW는 다음 중 하나의 방법으로 데이터 변경을 처리합니다:

- LabVIEW 클래스 계층구조에서 클래스를 제거하면, LabVIEW는 사용자가 제거한 클래스의 패턴화된 데이터를 삭제합니다. 예를 들어, 클래스 C는 클래스 B로부터, 클래스 B는 클래스 A로부터 상속을 하는 경우가 있습니다. 이때 클래스 A로부터 직접 상속하도록 클래스 C의 상속을 변경하면 LabVIEW는 인스턴스에 있는 클래스 B의 패턴화된 데이터를 제거합니다.
- LabVIEW 클래스 계층구조에서 계층을 추가하는 경우, LabVIEW는 사용자가 삽입한 클래스의 클래스 기본 데이터를 추가합니다. 예를 들어, 클래스 C는 클래스 A로부터 상속을 하는 경우가 있습니다. 이때 사용자가 클래스 C의 상속을 변경하여 클래스 A로부터 상속받는 클래스 B로부터 상속하도록 만들면 LabVIEW는 클래스 B의 클래스 기본 데이터를 삽입합니다.
- **탭 순서** 마지막에 있는 클래스의 프라이빗 데이터 클러스터에 원소를 추가하는 경우, LabVIEW 이전 데이터를 패턴화 해제하고 클래스 기본 데이터를 추가합니다.
- 클래스의 프라이빗 데이터 클러스터로부터 원소를 삭제하고 새 원소를 추가하지 않는 경우, LabVIEW는 이전 데이터를 패턴화 해제하고 원소에 관한 정보를 삭제합니다.
- 클래스 프라이빗 데이터 클러스터의 탭 순서를 변경하는 경우, LabVIEW는 이에 따라서 패턴화 해제된 데이터의 순서를 재설정합니다.
- 같은 편집 내에서 클래스의 프라이빗 데이터 클러스터를 추가, 삭제, 대체, 또는 순서 재설정하는 경우, LabVIEW는 상기의 단계를 조합하여 적절히 이전 데이터를 변경하는 과정을 생성 및 실행합니다.

LabVIEW는, 예를 들면 1.0.0.2에서부터 1.0.0.6로 방식으로, 항상 순차적으로 여러 버전에 변경을 적용합니다.



노트 일단 클래스 버전이 증가하면 LabVIEW 클래스를 되돌릴 수 없습니다. LabVIEW는 LabVIEW 클래스 컨트롤과 인디케이터 내의 데이터 변경을 실행 취소할 수 있는 충분한 정보가 없습니다. 현재의 변경사항을 되돌리고 싶으면, 어떤 변경사항도 저장하지 않습니다. 클래스와 클래스를 참조하는 모든 VI를 언로드한 후, 디스크에서 다시 로드합니다.

사용자가 LabVIEW 클래스를 편집할 때 LabVIEW가 데이터를 변경하는 방법에 대한 추가적인 정보는 ni.com의 관련 문서를 참조합니다.

LabVIEW 클래스 캐스팅하기

[보다 일반적인 클래스로] 함수를 사용하여 클래스를 업캐스트할 수 있고, [보다 구체적인 클래스로] 함수를 사용하여 클래스를 다운캐스트할 수 있습니다. [VI 서버 컨트롤 참조 번호](#)와 같이 상속 계층구조가 있는 참조 번호 데이터 타입에 이와 같은 함수를 사용할 수 있으며, 그 기능은 동일합니다. [보다 일반적인 클래스로] 함수를 사용한다고 데이터가 바뀌지는 않지만 데이터를 전달하는 와이어 타입은 바뀔 수 있습니다. 하지만 VI가 실행 상태일 때에는 이 함수가 어떠한 영향도 미치지 않습니다. [보다 일반적인 클래스로] 함수의 주요 용도는 이러한 데이터 변환에 대해 엄격한 코드 형식을 사용하는 프로그래머를 위해 강제 변환점을 제거하는 데에 있습니다.

[보다 구체적인 클래스로] 함수를 사용하면 에러가 발생하지 않는 한 데이터가 변경되지 않습니다. 실행 중인 와이어의 데이터가 보다 구체적인 클래스의 데이터가 아닐 경우, 함수는 에러를 반환하며 출력 데이터는 와이어 타입의 기본값입니다. [보다 구체적인 클래스로] 함수의 일차적인 용도는 직상위 클래스의 값에 대해 타입 테스트하는 데에 있습니다. 종종 프로그래머들은 동일한 직상위 클래스 와이어를 여러 [보다 구체적인 클래스로] 함수 호출에 연결시켜 각각 각기 다른 [보다 구체적인 클래스로] 이동하게 만들고, 이후 어떤 [보다 구체적인 클래스로] 함수가 에러를 반환하지 않는지에 따라 다른 코드를 실행시킵니다. 이는 매우 비효율적입니다. 사용자가 이러한 테스트 메소드를 사용할 생각이 있고 직상위 클래스를 수정할 수 있는 권한이 있다면, 직상위 클래스에 [다이나믹 멤버 VI 쓰기](#)를 하고 각각의 직하위 클래스가 각자에게 적합한 기능으로 덮어쓰도록 할 수 있습니다.

런타임에서 LabVIEW가 사용자가 직상위 클래스 객체를 받는 SubVI에 직하위 클래스 객체를 와이어로 연결하는 것을 감지하는 경우, LabVIEW는 해당 SubVI의 출력을 자동으로 직하위 클래스 객체로 다운캐스트할 수 있습니다. 자동 다운캐스트는 [보다 구체적인 클래스] 함수를 필요로 하지 않습니다. 하지만 SubVI에 연결하는 클래스 객체가 해당 SubVI가 받는 입력과 호환가능한 경우에만 LabVIEW에서 자동 다운캐스팅이 발생합니다. 예를 들어, 변수에 한 클래스 객체를 저장하고 그 변수 데이터를 SubVI에 와이어로 연결하는 경우에 LabVIEW는 그 SubVI가 변수에서 원래 저장한 데이터와 같은 클래스에 속하는 데이터를 포함하고 있는지 보장할 수 없습니다. [런타임 클래스 유지] 함수를 사용하면, LabVIEW에서 SubVI에 연결하는 클래스 객체가 해당 SubVI가 받는 클래스 객체와 호환가능한지 점검할 수 있습니다. 두 객체가 호환되지 않는 경우, 이 함수는 에러를 반환하고 출력 데이터의 클래스를 SubVI가 받는 직상위 클래스로 설정합니다. 또한, 데이터 값 참조와 함께 이 함수를 사용할 수 있습니다. [데이터 값 참조 원소 읽기 / 쓰기] 경계 노드는 런타임 타입을 유지해야 합니다. [런타임 클래스 유지] 함수를 사용하여, [데이터 값 참조 원소 쓰기] 노드에 사용자가 와이어로 연결하는 클래스 객체가 [데이터 값 참조 원소 읽기] 경계 노드에 연결하는 클래스 객체와 호환가능한지 점검할 수 있습니다.



노트 LabVIEW에서는 재귀 SubVI를 자동으로 다운캐스팅할 수 없습니다. 자동 다운캐스팅을 하게 되면, 다이내믹 디스패치 VI의 블록 다이어그램이나 데이터 값 참조에 대한 In Place 원소 구조 내부에서 SubVI를 호출할 경우 호출자 VI가 깨어집니다. 해당 재귀 SubVI의 블록 다이어그램에서 [런타임 클래스 유지] 함수를 사용하여, SubVI에 연결하는 클래스 객체가 SubVI가 받는 클래스 객체와 호환되는지 확인할 수 있습니다. 클래스 객체가 호환되는 경우, 재귀 사이클 외부에 있는 호출자 VI는 깨지지 않습니다.

LabVIEW 클래스 잠그기와 잠금해제하기

LabVIEW 클래스 잠그기를 설정하여 LabVIEW 클래스 사용자가 어플리케이션 내부 구현의 일부분인 멤버 VI를 보는 것을 금지할 수 있습니다. LabVIEW 클래스를 잠그면 프라이빗 멤버 VI에 대한 접근을 막음으로써 어플리케이션에 발생할 수 있는 에러를 방지합니다. **프로젝트 라이브러리**와 관련해서, LabVIEW 클래스에 암호 보호를 추가한다고 멤버 VI에 암호 보호가 추가되는 것은 아닙니다. 멤버 VI는 개별적으로 암호 보호를 해야 합니다.

LabVIEW 클래스 잠그기를 하면, 클래스 와이어 위로 커서를 이동할 때 오직 데이터 타입만이 기본 도움말 윈도우에 나타납니다. 잠금해제 상태의 LabVIEW 클래스의 와이어 위로 커서를 이동하면, 기본 도움말 윈도우는 해당 클래스의 프라이빗 데이터와 모든 잠금 해제 상태인 직상위 클래스의 프라이빗 데이터를 보여줍니다. LabVIEW가 일반 프로브에 디스플레이하는 정보에도 이와 유사한 규칙이 적용됩니다. **LabVIEW 클래스에서 프로브 사용하기**에서 LabVIEW 클래스와 일반 프로브 사용에 관한 정보를 참조합니다.



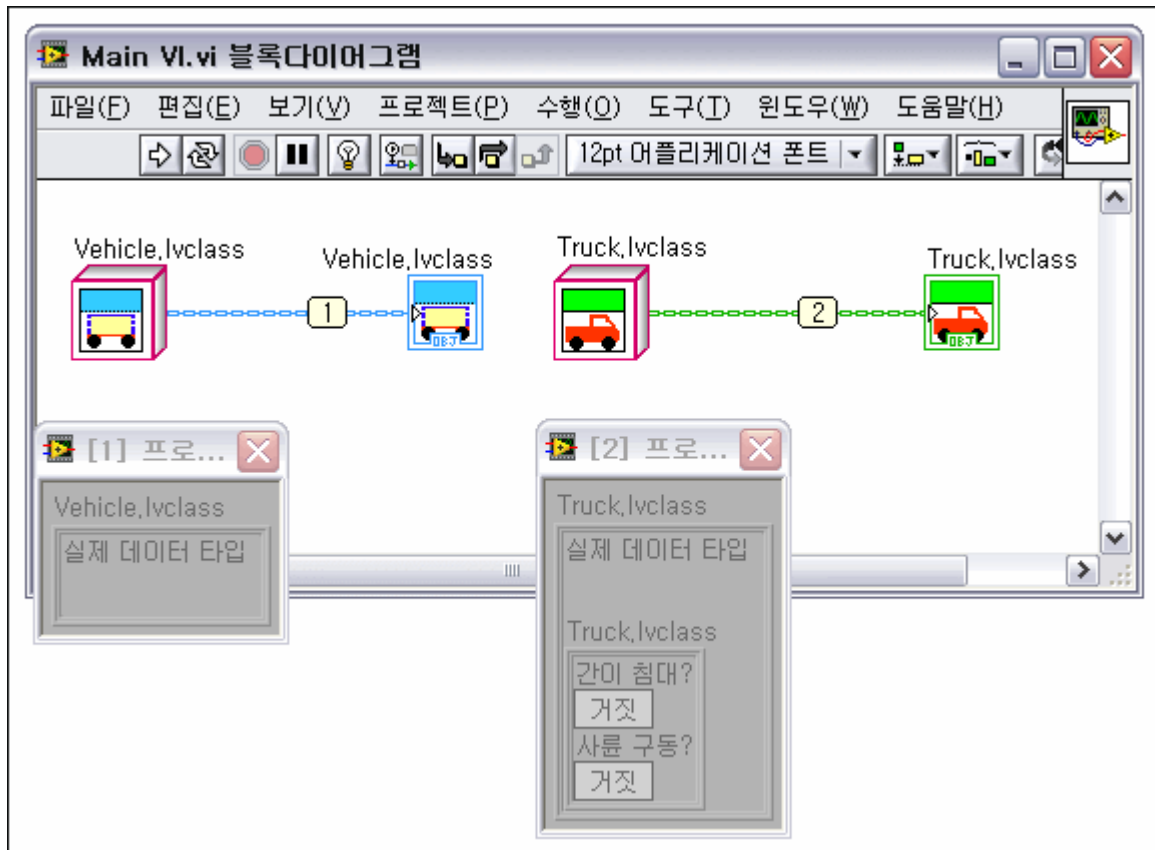
노트 LabVIEW 클래스 와이어와 LabVIEW 클래스 컨트롤에 대해서 기본 도움말 윈도우가 다른 정보를 디스플레이함을 확인하십시오. LabVIEW 클래스 컨트롤은 최종 사용자를 위해 클래스 설명 정보를 디스플레이하는 반면, LabVIEW 클래스 와이어는 LabVIEW 클래스 사용자를 위해 데이터 타입에 대한 정보를 추가적으로 디스플레이합니다.

LabVIEW 클래스 사용자는 오직 허용되는 클래스 정보만을 볼 수 있기 때문에, 개발하고 있는 LabVIEW 클래스를 적절하게 문서화하도록 주의하십시오. 클래스 프로퍼티 대화 상자에서 **클래스 설명 추가**를 할 수 있으며 VI 프로퍼티 대화 상자에서 **멤버 VI 설명 추가**를 할 수 있습니다.

LabVIEW 클래스에서 프로브 사용하기

일반 프로브와 사용자 프로브를 사용하여 LabVIEW 클래스를 디버그할 수 있습니다.

LabVIEW 클래스 와이어에 일반 프로브를 사용할 수 있습니다. LabVIEW 클래스 와이어의 일반 프로브는 실행 중인 클래스 이름, 프라이빗 클래스 데이터, 그리고 모든 직상위 클래스의 프라이빗 클래스 데이터를 보여줍니다. 잠금 상태의 LabVIEW 클래스에서 일반 프로브를 사용하면, 오직 실행 중인 클래스의 이름과 클래스 계층구조 내에서 잠금해제 상태의 모든 프라이빗 데이터를 프로브에서 볼 수 있습니다. 다음의 이미지에서, 왼쪽의 Vehicle 클래스는 잠금 상태이고 LabVIEW는 클래스의 프라이빗 데이터를 디스플레이하지 않습니다: 기어 개수, 차문 개수, 색, 차종, 모델 등이 이에 해당합니다. 트럭 클래스는 잠금 상태가 아니며 LabVIEW는 이 경우에는 프라이빗 데이터를 디스플레이합니다. Vehicle 클래스가 잠금 상태이기 때문에 LabVIEW가 Truck 클래스의 일반 프로브에 Vehicle 클래스에서 상속한 프라이빗 데이터를 디스플레이하지 않는다는 것을 확인할 수 있습니다.



LabVIEW 클래스에서 **사용자 정의 기본 프로브 생성하기**는 유용합니다. 와이어 타입과 동일한 클래스 타입이거나 또는 상위 와이어 타입을 가지는 LabVIEW 클래스 와이어에 대해서만 사용자 프로브를 사용할 수 있습니다. LabVIEW 클래스 개발자는 클래스의 멤버고 클래스의 프라이빗 데이터를 직접 디스플레이할 수 있는 사용자 프로브를 생성할 수 있습니다. LabVIEW 클래스 사용자는 사용자 기본 프로브를 생성하여 클래스의 퍼블릭 메소드를 통해 사용 가능한 모든 정보를 디스플레이할 수 있습니다.

클래스 개발자는 LabVIEW 클래스 사용자를 위해 **사용자 프로브를 기본 프로브로 설정**할 수 있습니다. 잠금 상태인 LabVIEW 클래스의 일반 프로브는 데이터가 프라이빗이기 때문에 LabVIEW 클래스 사용자가 데이터 값에 대한 어떤 정보에도 접근할 수 없도록 합니다. 사용자 프로브를 생성하고 이를 LabVIEW 클래스의 기본 프로브로 설정함으로써, 클래스 개발자는 개발자가 클래스를 잠금 상태로 만든 후에 적절한 데이터를 디스플레이 하는 기본 프로브를 LabVIEW 클래스 사용자에게 제공할 수 있습니다. 사용자 프로브가 LabVIEW 클래스의 멤버여야 LabVIEW 개발자가 이를 기본 프로브로 설정할 수 있습니다.

다른 개발자와 사용자에게 LabVIEW 클래스 배포하기

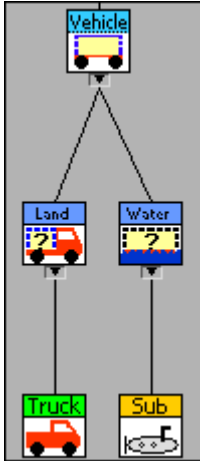
개발한 LabVIEW 클래스를 다른 LabVIEW 클래스 개발자와 LabVIEW 클래스 사용자에게 배포할 수 있습니다. 클래스를 배포하는 방법에는 여러가지가 있으므로 가장 적합한 방법을 선택하도록 합니다. **어플리케이션 빌더**를 사용하여 **압축(Zip) 파일을 생성**함으로써 하나 혹은 그 이상의 클래스를 배포할 수 있습니다. 또한 배포하기 이전에 **LabVIEW 클래스 잠그기**를 하여 프라이빗 데이터와 멤버 VI에 대한 LabVIEW 클래스 사용자의 접근을 제한할 수도 있습니다. 클래스를 잠그면 어플리케이션에 에러가 발생하는 것을 방지할 수 있습니다.



팁 .NET interop 어셈블리를 통해 LabVIEW 클래스들에 접근할 수 있습니다. LabVIEW는 사용자가 지정한 LabVIEW 클래스에 대해 .NET 클래스를 생성할 수 있습니다. 이 .NET 클래스들은 .NET 어셈블리를 통해서 접근하실 수 있습니다.

LabVIEW 클래스 객체 비교하기

비교 함수를 사용하여 LabVIEW 클래스 객체를 비교합니다.



동일한 클래스에 있는 두 객체(예를 들어, Truck 클래스의 두 객체)를 비교하면, LabVIEW는 클래스 계층구조의 모든 레벨에서 데이터를 비교합니다. 이는 LabVIEW가 클러스터의 클러스터를 비교하는 방식과 유사합니다.

서로 다른 클래스에 있는 두 객체(예를 들어, Truck 클래스의 한 객체의 Submarine 클래스의 한 객체)를 비교하면, LabVIEW는 두 객체가 공통적으로 갖는 상위 클래스 모두에 대해 데이터를 비교합니다. 이 경우, LabVIEW는 두 객체의 유일한 공통 상위 클래스인 Vehicle 클래스의 데이터를 비교합니다.

서로 다른 클래스의 두 객체를 비교할 때, 공통의 상위 클래스에 있는 모든 데이터가 같은 경우에 LabVIEW는 다음 기준에 따라 객체 순서를 정합니다.

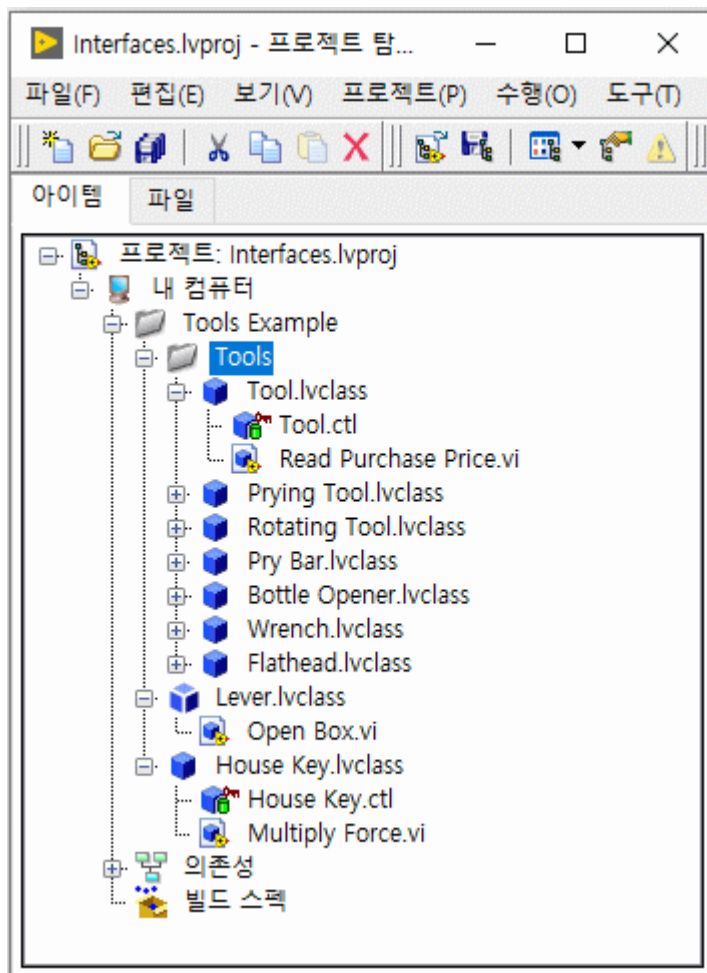
1. LabVIEW는 직상위 클래스가 직하위 클래스보다 우선하도록 순서를 정합니다. 예를 들어, Vehicle 클래스의 객체를 Land Vehicle 클래스의 객체와 비교하는 경우에 LabVIEW는 직상위 클래스인 Vehicle 클래스의 객체가 우선하도록 순서를 정합니다.
2. LabVIEW는 클래스 이름의 글자순에 따라 비교하는 클래스의 순서를 정합니다. 예를 들어, Truck 클래스의 객체와 Submarine 클래스의 객체를 비교하는 경우에 LabVIEW는 공통 직상위 클래스로서 Vehicle 클래스 객체의 데이터를 먼저 비교합니다. Vehicle 클래스 객체의 데이터가 같은 경우, LabVIEW는 Truck 객체의 데이터를 우선 처리합니다. 첫번째 비교하는 클래스(Land Vehicle 및 Water Vehicle)에서 글자순 기준으로 Land Vehicle이 우선하기 때문입니다.

LabVIEW 인터페이스

LabVIEW에서 인터페이스는 프라이빗 데이터 컨트롤이 없는 클래스라고 생각할 수 있지만, 그 작은 차이로 인해 소프트웨어 구조에서 인터페이스를 클래스와는 전혀 다른 용도로 사용할 수 있습니다. 구체적으로, 인터페이스에서는 다중 상속이 가능합니다.

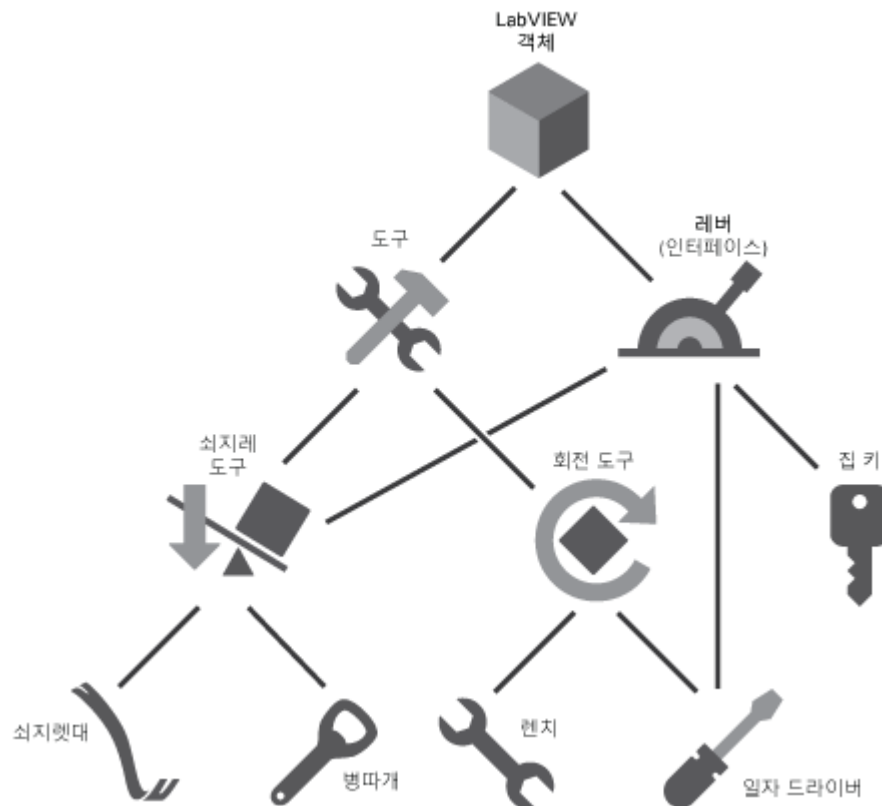
인터페이스는 객체가 그 역할을 수행하는 방법을 정의하지 않고 수행할 수 있는 역할을 선언합니다. 인터페이스에서 상속함으로써 클래스는 객체가 해당 역할을 수행한다고 선언하며 동작 수행 방법을 지정하는 책임을 갖게 됩니다. 클래스가 여러 인터페이스를 상속하는 경우, 해당 객체를 다른 역할을 필요로 하는 여러 소프트웨어 모듈에 전달할 수 있습니다.

다음 프로젝트에는 다양한 도구의 여러 하위 클래스를 가진 도구 클래스가 포함되어 있습니다. 또한 이 프로젝트에는 레버 인터페이스가 포함되어 있습니다. 도구 클래스에는 클래스의 데이터를 정의하는 컨트롤이 있지만 인터페이스에는 프라이빗 데이터가 없기 때문에 레버 인터페이스에는 컨트롤이 없습니다. 클래스는 단색 큐브 로 표시됩니다. 인터페이스는 큐브의 면들 로 표시됩니다. 인터페이스와 클래스는 같은 파일 확장자 .lvclass를 사용합니다.



다음 이미지는 도구 클래스, 집 키 클래스 및 레버 인터페이스 간의 상속 관계를 보여줍니다. 도구 클래스와 레버 인터페이스는 모두 LabVIEW 객체를 상속합니다. 또한 쇠지레 도구 클래스 및 일자 드라이버 클래스도 자신만의 메소드 및 각 상위 클래스에 대해 고유한 메소드를 갖는 것 외에도 레버로도 사용될 수 있기 때문에 레버 인터페이스를 상속합니다.

니다. 이들은 레버 인터페이스의 메소드를 상속합니다. 레버는 클래스가 아닌 인터페이스이므로 이 다중 상속에는 문제가 없습니다. 서로 관련이 없는 3개의 클래스(쇠지레 도구, 일자 드라이버, 집 키)에는 LabVIEW 객체 이외의 공통 상위 항목(레버)이 있습니다.



인터페이스 명명 권장 방법

다음 가이드라인에 따라 생성한 인터페이스에 인터페이스의 목적을 나타내는 이름을 지정합니다.

- 객체가 인터페이스를 상속하는 경우, 객체의 기능을 설명하는 형용사나 부사를 사용합니다. 예를 들어, 인터페이스가 전압을 측정할 수 있는 하드웨어를 나타내는 경우, 인터페이스의 이름을 `Can Measure Voltage.lvclass`로 지정합니다. 해당 인터페이스를 상속하는 모든 클래스나 인터페이스는 전압을 측정할 수 있습니다.
- 인터페이스에서 상속하는 클래스의 항목을 설명하는 명사를 사용하십시오. 예를 들어 인터페이스가 레버 도구를 나타내는 클래스의 항목을 설명하는 경우, 인터페이스 이름을 `Lever.lvclass`로 지정합니다.
- 이름의 시작 부분에 대문자 "I"를 사용하지 마십시오. 대부분의 텍스트 프로그래밍 언어에서는 인터페이스를 클래스와 구별하기 위해 일반적으로 인터페이스 이름 앞에 대문자 "I"를 붙이지만, LabVIEW는 문양을 사용하여 인터페이스와 클래스를 구별합니다. 또한 LabVIEW 개발 환경의 대부분은 의도적으로 클래스와 인터페이스를 동일하게 취급합니다. 메소드 호출자는 일반적으로 기본 타입이 인터페이스인지 클래스인지 상관하지 않습니다. 따라서 "I"를 사용하는 것을 피함으로써 호출자 코드를 리팩토링하지 않고 클래스를 인터페이스로 또는 그 반대로 변환할 수 있습니다.

인터페이스 사용에 대한 예제는 다음 프로젝트를 참조하십시오.

- labview\examples\Object-Oriented Programming\Basic Interfaces\Basic Interfaces.lvproj
- labview\examples\Object-Oriented Programming\Actors and Interfaces\Actors and Interfaces.lvproj

어플리케이션에서 LabVIEW 클래스 및 인터페이스 사용하기

LabVIEW 클래스 및 인터페이스는 사용자 정의된 데이터 타입입니다. LabVIEW 클래스 및 인터페이스 개발자가 이러한 데이터 타입을 생성하고 배포합니다. LabVIEW 클래스 또는 인터페이스의 사용자는 LabVIEW 클래스나 인터페이스를 생성하는 방법을 알아야 할 필요는 없지만, 어플리케이션에서 클래스나 인터페이스가 정의하는 데이터 타입을 사용하는 방법과 LabVIEW 클래스나 인터페이스를 사용하는 코드를 디버깅할 때 이용할 수 있는 정보, 그리고 새로운 버전의 LabVIEW 클래스나 인터페이스가 프로그래머가 작성한 어플리케이션에 미칠 수 있는 영향에 대해 이해해야 합니다.

클래스 또는 인터페이스의 인스턴스 생성하기

LabVIEW 클래스 또는 인터페이스는 프로젝트 내에 있을 필요가 없습니다. **팔레트에 LabVIEW 클래스 또는 인터페이스를 추가**하여 쉽게 클래스나 인터페이스를 찾아서 블록 다이어그램이나 프런트패널 윈도우에 끌어올 수 있습니다. 받거나 **생성**한 LabVIEW 클래스 또는 인터페이스의 인스턴스를 생성하려면, 클래스 라이브러리 파일 .lvclass를 사용하고자 하는 프런트패널 또는 블록다이어그램에 컨트롤이나 상수로 놓습니다. LabVIEW 사용자 정의 데이터 타입을 다른 **LabVIEW 데이터 타입**과 유사하게 사용할 수 있습니다.

클래스 탐색기 윈도우를 사용하여, 블록다이어그램에서 LabVIEW 클래스나 인터페이스의 **멤버 vi를 놓을** 수 있습니다.

LabVIEW 클래스 및 인터페이스 강제 변환하기

LabVIEW는 다음 규칙을 사용하여 클래스와 인터페이스를 가장 가까운 공통 상위 클래스 또는 인터페이스로 강제 변환합니다.

- 입력 클래스와 인터페이스가 서로 관련되어 있지 않은 경우, LabVIEW는 입력을 LabVIEW 객체로 강제 변환합니다.
- 입력 클래스와 인터페이스에 클래스나 인터페이스에 상관없이 하나의 공통 상위 항목만 있는 경우, LabVIEW는 입력을 공통 상위 항목으로 강제 변환합니다.
- 입력 클래스와 인터페이스에 여러 개의 공통 상위 항목이 있고 그 중 하나가 클래스인 경우, LabVIEW는 입력을 공통 상위 클래스로 강제 변환합니다. LabVIEW가 입력을 공통 상위 인터페이스로 강제 변환하게 하려면, [보다 일반적인 클래스로] 함수를 사용하여 입력 중 하나를 원하는 상위 타입으로 캐스트합니다.
- 입력 클래스와 인터페이스에 여러 개의 공통 상위 항목이 있고 그것이 모두 인터페이스인 경우, LabVIEW에서 vi는 깨집니다. LabVIEW가 입력을 공통 상위 인터페이스로 강제 변환하게 하려면, [보다 일반적인 클래스로] 함수를 사용하여 입력 중 하나를 원하는 상위 타입으로 캐스트합니다.

잠금 상태의 LabVIEW 클래스 또는 인터페이스에서 기본 도움말과 프로브 사용

LabVIEW 클래스 또는 인터페이스 개발자는 클래스나 인터페이스를 배포할 때 **LabVIEW 클래스나 인터페이스를 잠금으로 설정**할 수 있습니다. LabVIEW 클래스 또는 인터페이스 개발자가 LabVIEW 클래스나 인터페이스를 잠금으로 설정하면, 기본 도움말 윈도우는 오직 클래스나 인터페이스의 데이터 타입 및 개발자가 제공하는 클래스에 대한 설명만을 디스플레이합니다. LabVIEW는 또한 **잠금 상태의 클래스나 인터페이스의 일반 프로브**에서도 클래스나 인터페이스의 데이터 타입만을 디스플레이합니다. **LabVIEW 클래스나 인터페이스의 멤버가 아닌 사용자 프로브**를 생성할 수 있지만, 멤버가 아닌 사용자 프로브는 오직 클래스나 인터페이스의 퍼블릭 인터페이스만을 디스플레이할 수 있습니다. LabVIEW 클래스 또는 인터페이스 개발자는 또한 클래스나 인터페이스에 대해 **사용자 기본 프로브**를 설정하여 디버깅을 위한 특정 클래스 또는 인터페이스 정보를 제공할 수 있습니다. LabVIEW 클래스 또는 인터페이스 개발자와 같이 작업하여 LabVIEW 클래스나 인터페이스 작업을 하는 데에 필요한 적절한 수준의 문서 또는 프로브 정보를 제공할 수 있도록 합니다.

다이나믹 디스패치 VI

LabVIEW 클래스 또는 인터페이스의 일부 멤버 VI는 다이나믹 디스패치될 수 있습니다. 다이나믹 디스패치 메소드는 다형성 VI와 유사합니다. 다형성 VI는 사용자가 연결하는 데이터 타입에 따라 어떤 VI를 호출할지 결정하는 반면에, 다이나믹 디스패치 메소드는 런타임이 시작될 때까지 대기한 후 입력에 도착하는 데이터에 따라 클래스 계층구조에서 호출할 멤버 VI를 결정합니다. LabVIEW 클래스 또는 인터페이스 와이어가 자신의 데이터 타입 또는 모든 직하위 데이터 타입을 전달할 수 있기 때문에, LabVIEW는 런타임에 클래스 계층구조의 어느 멤버 VI를 실행할지 결정합니다. 블록다이어그램의 다이나믹 디스패치 멤버 VI를 더블 클릭하여 블록다이어그램 상에 있는 특정 VI를 볼 수 있습니다. 클래스 계층구조의 다른 다이나믹 디스패치 멤버 VI를 보려면, 클래스 계층구조 안에 있는 각각의 개별 VI를 봐야 합니다.

데이터 변경하기

LabVIEW 클래스 또는 인터페이스의 버전 번호를 추적하는 것은 LabVIEW 클래스나 인터페이스를 통해 어플리케이션을 개발하는 LabVIEW 클래스 사용자에게 유용합니다. 예를 들어, 클래스의 프라이빗 데이터 컨트롤에서 부호없는 32비트 정수를 사용하는 어플리케이션 상에 사용자가 LabVIEW 클래스를 갖는 경우가 있습니다. LabVIEW 클래스 개발자는 프라이빗 데이터 컨트롤의 숫자형 컨트롤을 배정도 부동수로 변경시키는 LabVIEW 클래스 버전을 사용자에게 보낼 수 있습니다. LabVIEW가 버전 변경사항을 추적하며 모든 LabVIEW 클래스에서 어떻게 데이터를 패턴화 및 패턴화 해제하는지 알고 있기 때문에, 사용자는 이전 버전의 LabVIEW 클래스를 새로운 버전으로 대체하고 어떠한 변경도 없이 프로그램을 실행할 수 있습니다.



노트

사용자가 이후 LabVIEW 클래스 버전으로부터의 데이터 패턴화 해제를 시도하는 경우, LabVIEW는 에러를 반환합니다. 예를 들어, 버전 번호가 1.0.0.3인 데이터를 갖고 있지만 메모리 상의 LabVIEW 클래스 버전은 1.0.0.2일 경우, 이러한 에러가 발생할 수 있습니다.

LabVIEW 프런트패널

프런트패널 컨트롤과 인디케이터

다음 서브 팔레트를 사용하여 LabVIEW 어플리케이션의 프런트패널을 만들 컨트롤과 인디케이터를 선택합니다.



노트

사용할 수 있는 프런트패널 컨트롤과 인디케이터는 선택한 서브팔레트의 스타일에 따라 다릅니다. 사용할 수 있는 프런트패널 컨트롤과 인디케이터는 일반, 실버, 시스템 또는 클래식 서브팔레트를 참조하십시오.

서브팔레트	설명
데이터 컨테이너	배열과 클러스터 컨트롤 및 인디케이터를 사용하여 정렬되지 않은 순서로 데이터 원소를 그룹화합니다. 세트 또는 맵 컨트롤 및 인디케이터를 사용하여 정렬된 순서로 데이터 원소를 그룹화합니다. 행렬 컨트롤과 인디케이터를 사용하여 행렬 데이터를 입력하고 보여줍니다.
불리언	불리언 컨트롤과 인디케이터를 사용하여 참/거짓 값을 입력하고, 버튼, 스위치, 빛 등의 형태로 나타냅니다.
컨테이너	컨테이너 컨트롤과 인디케이터를 사용하여 다른 컨트롤과 인디케이터를 그룹화하거나, 현재 VI의 프런트패널에 다른 VI의 프런트패널을 나타냅니다.
장식	장식을 사용하여 VI의 기능에 영향을 미치지 않고 프런트패널 객체를 그룹으로 모으거나, 분리합니다. 장식에는 상자, 라인, 화살표 같은 객체가 있습니다.
그래프	그래프와 차트 인디케이터를 사용하여 숫자형 데이터를 플롯하고 보여줍니다.
I/O	I/O 이름 컨트롤과 인디케이터를 사용해서 I/O 하드웨어 설정과 관련된 이름을 I/O VI에 전달하여 인스트루먼트 또는 디바이스와 통신합니다.
리스트, 테이블 & 트리	리스트와 트리 컨트롤 및 인디케이터를 사용하여 아이템 리스트를 제공합니다. 테이블을 사용하여 데이터 문자열을 행과 열로 나타냅니다.
숫자형	숫자 컨트롤과 인디케이터를 사용하여 숫자 데이터를 입력하고 보여줍니다.
참조 번호	참조 번호 컨트롤과 인디케이터를 사용하여 파일, 디렉토리, 디바이스 및 네트워크 연결을 참조합니다.
링 & 열거형	링과 열거형 컨트롤 및 인디케이터를 사용하여 사용자가 선택할 수 있는 아이템 리스트를 생성합니다. 링 컨트롤과 인디케이터는 숫자형 데이터 타입만 지원합니다. 열거형 컨트롤과 인디케이터는 상응하는 문자열이 포함된 숫자형 데이터 타입을 지원합니다.

서브팔레트	설명
문자열 & 경로	문자열 컨트롤과 인디케이터를 사용하여 텍스트를 입력하거나 보여줍니다. 경로 컨트롤과 인디케이터를 사용하여 파일과 디렉토리 경로를 입력하거나 반환합니다.
배리언트 & 클래스	배리언트와 클래스 컨트롤 및 인디케이터를 사용하여 배리언트와 객체 지향 데이터와 상호 작용합니다.



관련 정보

[컨테이너 컨트롤과 인디케이터](#) (50 페이지 참조)

[볼리언 컨트롤과 인디케이터](#) (47 페이지 참조)

볼리언 서브팔레트에 있는 볼리언 컨트롤과 인디케이터를 사용하여, 버튼, 스위치, LED와 같은 볼리언(참/거짓) 값을 입력하고 표시합니다.

[컨테이너 컨트롤과 인디케이터](#) (50 페이지 참조)

[그래프 서브팔레트](#) (78 페이지 참조)

[I/O 이름 컨트롤과 인디케이터](#) (81 페이지 참조)

[리스트, 테이블, 트리 컨트롤과 인디케이터](#) (84 페이지 참조)

[숫자형 컨트롤과 인디케이터](#) (86 페이지 참조)

[참조 번호 컨트롤과 인디케이터](#) (87 페이지 참조)

[링 및 열거형 타입 컨트롤과 인디케이터](#) (88 페이지 참조)

[문자열 및 경로 컨트롤과 인디케이터](#) (89 페이지 참조)

[배리언트 및 클래스 컨트롤과 인디케이터](#) (92 페이지 참조)

[팔레트 항목 보이기와 숨기기](#) (803 페이지 참조)

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

프론트패널 만들기

프론트패널은 VI의 사용자 인터페이스입니다. 일반적으로, 프론트패널을 먼저 만든 후 프론트패널에 생성한 입력과 출력에 작업을 수행하기 위해서 블록다이어그램을 디자인합니다.

각각 VI의 대화식 입력과 출력 터미널인 컨트롤과 인디케이터를 사용하여 프론트패널을 만듭니다. 컨트롤은 노브, 누름 버튼, 다이얼 및 기타 입력 메커니즘입니다. 인디케이터는 그래프, LED, 그리고 기타 출력 디스플레이입니다. 컨트롤은 인스트루먼트의 입력 메커니즘을 시뮬레이션하고 VI의 블록다이어그램에 데이터를 제공합니다. 인디케이터는 인스트루먼트의 출력 메커니즘을 시뮬레이션하고 블록다이어그램에서 수집하거나 생성하는 데이터를 디스플레이합니다.

보기>> 컨트롤 팔레트를 선택하여 컨트롤 팔레트를 디스플레이한 후, 컨트롤 팔레트에서 컨트롤과 인디케이터를 선택하여 프런트패널에 놓습니다.



관련 정보

[블록다이어그램 만들기](#) (52 페이지 참조)

[프런트패널 컨트롤과 인디케이터](#) (544 페이지 참조)

[프런트패널 컨트롤과 인디케이터](#) (544 페이지 참조)

[프런트패널 객체 설정하기](#) (546 페이지 참조)

[프런트패널 설정하기](#) (548 페이지 참조)

[컨트롤 서브팔레트](#) (79 페이지 참조)

[사용자 인터페이스 디자인하기](#) (553 페이지 참조)

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

[라벨 붙이기](#) (1315 페이지 참조)

[x컨트롤 생성하기](#) (1030 페이지 참조)

[LabVIEW에서 끌어서 놓기](#) (51 페이지 참조)

프런트패널 객체 설정하기

프로퍼티 대화 상자나 바로 가기 메뉴를 사용하여 상수, 컨트롤, 인디케이터, 숫자형 함수가 나타나거나 작동하는 방식을 설정합니다. 기본 도움말을 포함하며 객체에 여러 프로퍼티를 동시에 설정할 수 있는 대화 상자를 사용하여 상수, 컨트롤, 인디케이터 또는 숫자형 함수를 설정하려고 할 때 프로퍼티 대화 상자를 사용합니다. 손쉽게 일반 상수, 컨트롤, 인디케이터, 함수 프로퍼티를 설정하기 위해서는 **바로 가기 메뉴**를 사용합니다. 프로퍼티 대화 상자와 바로 가기 메뉴에서 사용할 수 있는 옵션은 객체에 따라 다릅니다. 바로 가기 메뉴를 사용하여 설정한 대부분의 옵션은 프로퍼티 대화 상자에 반영되며, 프로퍼티 대화 상자를 사용하여 설정한 대부분의 옵션은 바로 가기 메뉴에 반영됩니다.

상수, 컨트롤, 인디케이터 또는 숫자형 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체에 대한 프로퍼티 대화 상자를 디스플레이합니다. VI가 실행되는 동안에는 프로퍼티 대화 상자를 디스플레이할 수 없습니다.



팁

여러 개의 객체에서 한 개의 일반적인 작업을 동시에 수행하려면, 객체를 선택한 후 바로 가기 메뉴를 사용하거나 프로퍼티 대화 상자를 사용합니다.

또한 **사용자 컨트롤 또는 인디케이터를 생성**하여 사용 가능한 프런트패널 객체 세트를 확장할 수 있습니다. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **급>> 사용자 정의를 선택하여 컨트롤 또는 인디케이터를 사용자 정의**합니다. 생성한 사용자 컨트롤 또는 인디케이터를 디렉토리 또는 LLB에 저장한 후 다른 프런트패널에서 사용자 컨트롤 또는 인디케이터를 사용할 수 있습니다.

옵션 원소 보이기와 숨기기

프런트패널 컨트롤과 인디케이터는 라벨, 캡션, 디지털 디스플레이 등의 보이거나 숨길 수 있는 옵션 원소를 가집니다. 프런트패널 객체에 대한 **프로퍼티** 대화 상자의 모양 페이지에서 컨트롤 또는 인디케이터의 보이는 원소를 설정합니다. 또한, 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 아이টে임을 선택한 후 사용 가능한 옵션에서 선택하여 보이는 원소를 설정할 수 있습니다.

컨트롤을 인디케이터로 또는 인디케이터를 컨트롤로 바꾸기

LabVIEW는 객체의 초기 설정을 컨트롤 팔레트에 있는 컨트롤과 인디케이터의 일반적인 용도에 따라서 지정하고 있습니다. 예를 들어, 토글 스위치는 일반적으로 입력 메커니즘으로 사용되기 때문에 프런트패널에 토글 스위치를 놓으면 컨트롤로 나타납니다. 프런트패널에 LED를 놓으면, LED는 일반적으로 출력 디바이스로 사용되기 때문에 인디케이터로 나타납니다.

일부 팔레트는 같은 타입이나 객체의 클래스에 대해서 컨트롤과 인디케이터를 포함합니다. 예를 들어, 숫자형 팔레트는 숫자형 컨트롤과 숫자형 인디케이터를 포함합니다. 숫자 입력 또는 숫자 출력을 가질 수 있기 때문입니다.

객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 인디케이터로 변경을 선택하여 컨트롤을 인디케이터로 변경하고 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 컨트롤로 변경을 선택하여 인디케이터를 컨트롤로 변경할 수 있습니다.

프런트패널 객체 대체하기

프런트패널 객체를 다른 컨트롤 또는 인디케이터로 대체할 수 있습니다. 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택할 때, 임시 컨트롤 팔레트가 나타납니다. 임시 컨트롤 팔레트에서 컨트롤 또는 인디케이터를 선택하여 프런트패널의 현재 객체를 대체합니다.

바로 가기 메뉴에서 대체를 선택할 경우 원래 객체의 이름, 설명, 기본값, 데이터 흐름 방향(컨트롤 또는 인디케이터), 색, 크기 등을 최대한 유지합니다. 숫자형 터미널을 다른 숫자형 터미널로 대체하면 LabVIEW는 원래의 형을 보존하려고 합니다. 그러나 컨트롤이 새로운 데이터 타입을 지원하는 않는 경우, 새로운 객체는 객체 고유의 데이터 타입을 유지합니다. 객체의 터미널 또는 로컬 변수의 와이어는 블록 다이어그램에 남아 있지만, 깨져 있을 수 있습니다. 예를 들어, 숫자 터미널을 문자열 터미널로 대체한 경우, 원래 와이어는 블록 다이어그램에 남아 있지만 깨져 있을 것입니다.

새 객체가 대체하는 객체와 비슷할수록, 원래 특성을 더 많이 유지합니다. 예를 들어, 슬라이드를 다른 스타일의 슬라이드로 대체하는 경우, 새 슬라이드는 같은 크기, 스케일, 값, 이름, 설명 등을 가집니다. 대신에 슬라이드를 문자열 컨트롤로 대체하는 경우, LabVIEW는 오직 이름, 설명, 데이터 흐름 방향만을 유지합니다. 왜냐하면 슬라이드와 문자열은 공통점이 많지 않기 때문입니다.

또한 편집>>복사와 붙여넣기를 통해서 클립보드에 객체를 복사하고 클립보드에서 객체를 붙여넣기해서 기존 프런트패널 컨트롤과 인디케이터를 대체할 수 있습니다. 이 방법은 이전 객체의 특성을 보존하지는 않으나, 와이어는 객체에 연결되어 있습니다.

프런트패널 설정하기

프런트패널 객체의 탭 순서를 설정하고, 반입된 그래픽을 사용하고, 윈도우 크기가 바뀔 때 프런트패널 객체를 자동으로 크기 조정하도록 설정하는 등, 프런트패널을 사용자 정의할 수 있습니다.

컨트롤의 키보드 바로 가기 키 설정하기

사용자가 마우스 없이 프런트패널을 **조작할 수 있도록** 컨트롤에 키보드 바로 가기 키를 지정할 수 있습니다. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>키 조작을 선택하여 **프로퍼티** 대화 상자의 키 조작 페이지를 디스플레이합니다.



노트

LabVIEW는 숨겨진 컨트롤의 키보드 바로 가기 키에는 반응하지 않습니다.

VI가 실행 중일 때 사용자가 키보드 바로 가기 키를 누르면, 해당 컨트롤이 포커스됩니다. 컨트롤이 텍스트 또는 숫자형 컨트롤인 경우, LabVIEW는 텍스트를 하이라이트하고 편집할 수 있습니다. 컨트롤이 불리언인 경우, 스페이스바, <Enter> 또는 숫자 키패드의 <Enter> 키를 눌러서 값을 바꿉니다.

인디케이터에는 데이터를 입력할 수 없기 때문에 인디케이터의 고급>>키 조작 바로 가기 메뉴 아이템은 비활성화됩니다.



노트 키 다운 이벤트를 사용하여 사용자가 키보드에서 특정 키를 누를 때 이벤트를 생성하도록 할 수 있습니다.

키 조작으로 버튼 동작 컨트롤하기

기능 키를 프런트패널을 컨트롤하는 다양한 버튼에 연결할 수 있습니다.



노트 (macOS) 기능 키를 키보드 바로 가기 키로 사용하는 것은 기능 키가 Exposé에서 바로 가기 키로 할당되지 않은 경우에만 가능합니다. LabVIEW는 macOS에서 <VolumeUp>, <VolumeDown> 및 <F15>부터 <F24>까지의 기능 키를 바로 가기 키로 지원하지 않습니다.

VI의 버튼을 대화 상자 버튼과 동일하게 동작하도록 하여 <Enter> 키나 숫자 키패드의 <Enter> 키를 누르는 것이 버튼을 클릭하는 것과 같도록 설정할 수 있습니다. <Enter> 키를 대화 상자 버튼에 연결할 경우, LabVIEW는 자동으로 버튼 주위에 두꺼운 경계를 그립니다.

불리언 컨트롤을 선택한 후 <Enter> 키를 누를 경우, 다른 컨트롤이 키보드 바로 가기 키로 <Enter> 키를 사용하더라도 불리언 컨트롤은 바뀝니다. 지정된 <Enter> 키보드 바로 가기 키는 불리언 컨트롤이 선택되지 않은 때에만 적용됩니다.

프런트패널 객체의 탭 순서 설정하기

프런트패널의 컨트롤과 인디케이터는 탭 순서라 불리는 순서를 가집니다. 이 순서는 프런트패널에서의 위치와는 관련이 없습니다. LabVIEW는 사용자가 프런트패널에 컨트롤을 놓은 순서에 따라 탭 순서를 결정합니다. 프런트패널에 처음으로 생성하는 컨트롤 또는 인디케이터는 0번 원소이며, 두번째는 1번, 이와 같이 이어집니다. 컨트롤 또는 인디케이터를 삭제할 경우, 탭 순서는 자동으로 조절됩니다.

탭 순서는 VI가 실행되는 동안 사용자가 <Tab> 키를 누를 때 LabVIEW가 선택하는 컨트롤과 인디케이터의 순서를 결정합니다. 또한 탭 순서는 **프런트패널 데이터를 로그**할 때 생성한 데이터로그 파일의 레코드에 나타나는 컨트롤과 인디케이터의 순서를 결정합니다.

편집» 탭 순서 설정을 선택하여 프런트패널 객체의 **탭 순서를 설정**할 수 있습니다.

VI가 실행되는 동안 사용자가 <Tab> 키를 눌러서 컨트롤에 접근하는 것을 막으려면, **프로퍼티** 대화 상자의 키 조작 페이지에서 탭으로 이동 시 이 컨트롤 건너뛰기 확인란에 확인 표시를 합니다.



노트 프런트패널에 분리자 막대를 추가하고 **구획을 생성**하면 프런트패널에 있는 객체의 탭 순서가 바뀌지 않습니다. 어떤 구획에 있는 컨트롤이든 탭 순서에서의 다음 컨트롤이 될 수 있습니다.

객체 색칠하기

사용자가 LabVIEW에 있는 많은 객체들의 색을 변경할 수 있습니다. 대부분의 프런트패널 객체, 프런트패널 상의 구획, 블록다이어그램 작업 공간의 색을 변경할 수 있습니다. 시스템 컨트롤과 인디케이터의 색은 변경할 수 없습니다. 이들 객체는 사용자가 설정한 시스템의 색으로 나타나기 때문입니다.

색칠 도구를 사용하여 객체 또는 작업공간에서 마우스 오른쪽 버튼을 클릭한 후 프런트패널 객체, 프런트패널 상의 구획, 블록다이어그램 작업 공간의 색을 바꿉니다. 또한, 도구» 옵션을 선택한 후 항목 리스트에서 환경을 선택하여 몇몇 객체에 대한 기본색을 바꿀 수 있습니다.

프런트패널에서 색을 사용할 경우 다음 가이드라인을 참조하십시오.

- 색은 사용자가 중요한 정보를 취할 때 방해가 될 수 있습니다. 예를 들어 배경이 노란색이나 녹색, 또는 밝은 주황색인 경우 빨간 위험등을 식별하기 어렵습니다.
- 일부 플랫폼에서는 색이 제한적으로 제공됩니다. 다른 플랫폼에서 일관성을 유지하려는 경우 일반적으로 검정색, 흰색 및 회색이 최적의 옵션입니다.
- 디바이스 상태를 나타내는 유일한 인디케이터로 색을 사용해서는 안 됩니다. 플롯에 라인 스타일을 사용하여 구별 요소를 추가합니다. 색맹 증상이 있는 사용자의 경우 색만으로 상태의 변화를 감지하기 어렵기 때문입니다.
- 배경색으로는 연한 회색, 흰색 또는 파스텔 색을 사용하십시오. 프런트패널 배경과 일반적인 컨트롤에는 색 선택기에서 첫 번째 행의 색을 사용합니다. 에러 알림과 같이 중요한 컨트롤을 하이라이트하는 경우에는 색 선택기에서 두 번째 행의 색을 사용합니다.

반입된 그래픽 사용하기

다른 어플리케이션에서 그래픽을 반입하여 구획 배경, 링 컨트롤의 아이템, 다른 컨트롤과 인디케이터의 일부로 사용할 수 있습니다.

LabVIEW는 BMP, JPEG, 애니메이션 GIF, MNG, 애니메이션 MNG, PNG를 포함하여 대부분의 표준 그래픽 포맷을 지원합니다. LabVIEW는 또한 투명성을 지원합니다.

하지만, 반입된 그래픽은 스크린 업데이트를 지연시킬 수 있습니다. 그래픽 객체 위에 인디케이터와 컨트롤을 놓지 않도록 하여, 인디케이터가 업데이트될 때마다 LabVIEW가 객체를 다시 그리지 않도록 하십시오.



팁 큰 배경 그림 위에 컨트롤을 놓아야 하는 경우, 그림을 여러 개의 작은 객체로 나누어서 개별적으로 반입하십시오. 큰 그래픽일수록 그리는 시간이 길어지기 때문입니다.

다음 중 하나의 방법을 사용하여 그래픽을 반입합니다.

Windows

macOS

Linux



팁 **다른 플랫폼**에 VI를 로드할 때 반입되는 그래픽의 모양을 확인하십시오. 예를 들면 불규칙한 모양의 macOS PICT 파일을 Windows나 Linux에서 열게 되면 흰색 배경의 직사각형 비트맵으로 나타나는 경우도 있습니다.

그래픽을 반입한 후, 크기 조정이나 순서 재설정이 필요한 경우도 있습니다. 반입된 GIF는 크기 조정할 수 없습니다.

그래픽을 프런트패널에 반입하기

다음의 방법 중 하나를 사용하여 프런트패널 배경으로 그래픽을 반입할 수 있습니다.

- 구획의 스크롤 막대에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여, 구획 프로퍼티 대화 상자를 디스플레이합니다. 배경 탭에서 배경, 탐색, 위치 옵션을 사용하여 **구획의 배경 이미지를 설정**합니다.
- 그래픽을 클립보드에 복사하고 프런트패널에 붙여넣기를 합니다. 또한 편집»클립보드에 그림 반입을 선택하고 그래픽을 프런트패널에 붙여넣기할 수도 있습니다.



노트 (Windows와 macOS) 이미지를 복사한 후 붙여넣기하여 이미지를 반입하는 경우, 해당 이미지는 투명성을 잃게 됩니다.

객체 정렬 및 간격 조절하기

편집»패널 눈금 정렬 활성화를 선택하여 프런트패널의 눈금 정렬을 활성화하고 객체를 놓을 때 **객체를 정렬**합니다. 또는, 편집»패널 눈금 정렬 비활성화를 선택하여 눈금 정렬을 비활성화하고 보이는 눈금을 사용하여 수동으로 객체를 정렬합니다. <Ctrl-#> 키를 눌러서 눈금 정렬을 활성화하거나 비활성화할 수 있습니다. 프랑스어 키보드에서는, <Ctrl-~> 키를 누릅니다.

(macOS) <Command-#> 키를 누릅니다. (Linux) <Alt-#> 키를 누릅니다.

블록다이어그램에서도 정렬 눈금을 사용할 수 있습니다. 객체가 고르게 정렬되고 간격이 조절되면 직선 와이어를 사용하여 객체를 서로 연결할 수 있습니다. 이로 인해 블록다이어그램을 알아보기가 용이해집니다.

도구»옵션을 선택한 후 항목 리스트에서 프런트패널을 선택하여, 눈금을 숨기거나 사용자 정의합니다.

객체를 놓은 후에 객체를 정렬하려면 객체를 선택한 후 도구 모음의 객체 정렬 풀다운 메뉴를 선택하거나 편집»아이템 정렬을 선택합니다. 객체의 간격을 균일하게 하려면, 객체를 선택한 후 도구 모음의 객체 간격 조절 풀다운 메뉴를 선택하거나 편집»아이템 간격 조절을 선택합니다.

객체 그룹화하기 및 잠그기

위치 도구를 사용하여 그룹화 및 잠금 설정을 하려는 프런트패널 객체를 선택합니다. 도구 모음의 순서 재설정 버튼을 클릭한 후 풀다운 메뉴에서 그룹 또는 고정을 선택합니다. 그룹화된 객체는 위치 도구를 이용하여 위치를 옮기거나 크기를 바꿀 때, 상호 간의 상대적인 배열과 크기를 유지합니다. 잠겨진 객체는 프런트패널에서 움직이지 않습니다. 그룹화되거나 잠근 객체는 그룹화나 고정을 해제하기 전까지는 프런트패널이나 블록다이어그램에서 삭제할 수 없습니다. 객체에 대한 그룹화와 잠금 설정을 동시에 할 수 있습니다. 동시에 객체를 그룹화하고 잠금으로 설정하는 경우, 프런트패널이나 블록다이어그램에서 지우기 전에 그룹화와 잠금 설정을 해제해야 합니다. 위치 도구 이외의 다른 도구들도 그룹화되고 잠금 설정된 객체에서 사용할 수 있습니다.

객체 크기 조정하기

대부분의 프런트패널 객체는 크기를 바꿀 수 있습니다. 위치 도구를 크기 조정이 가능한 객체 위로 움직일 때, 객체의 크기를 조정할 수 있는 포인트에서 크기 조정 핸들이 나타납니다. 객체를 크기 조정할 때, 폰트 크기는 변하지 않습니다. 객체의 그룹을 크기 조정하면 그룹 내의 모든 객체들의 크기가 조정됩니다.

디지털 숫자형 컨트롤과 인디케이터 같은 일부 객체는 크기 조정할 때 수평 또는 수직으로만 크기가 조정됩니다. 노브와 같은 다른 객체는 크기 조정할 때 같은 비율을 유지합니다. 위치 커서는 똑같이 나타나지만, 객체를 둘러싸고 있는 경계 점선은 한 방향으로만 움직입니다.

객체를 크기 조정할 때 수동으로 증가 방향을 제한할 수 있습니다. 확대를 수직 또는 수평으로 제한하거나 객체의 현재 비례를 유지하려면, <Shift> 키를 누르면서 크기 조정 핸들을 클릭하고 끕니다. 중심점 주위로 객체의 크기를 조정하려면, <Ctrl> 키를 누르면서 크기 조정 핸들을 클릭하여 끕니다. 확대를 수직 또는 수평으로 제한하거나 객체의 현재

비율을 유지하면서 중심점 주변에서 크기를 조정하려면, <Ctrl-Shift> 키를 누르면서 크기 조정 핸들을 클릭하여 끕니다.

(macOS) <Ctrl> 키 대신 <Option> 키를 누릅니다. (Linux) <Ctrl> 키 대신 <Alt> 키를 누릅니다.

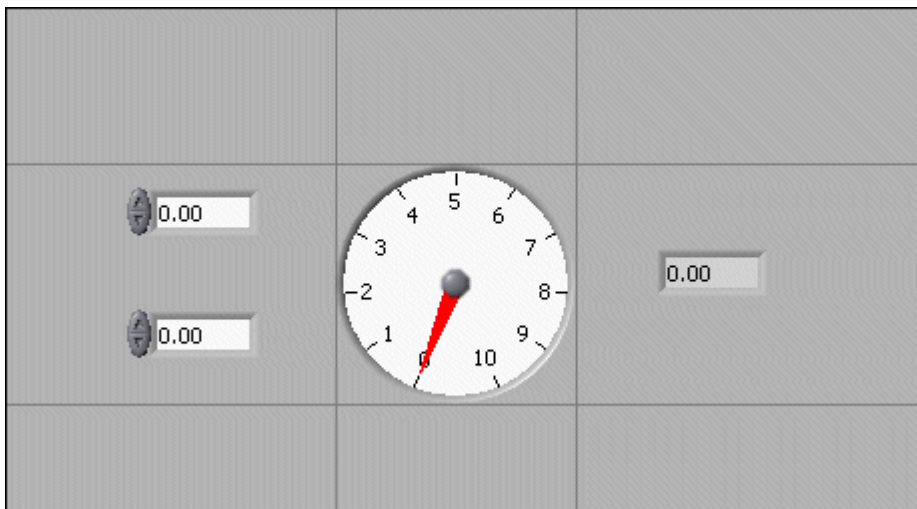
여러 객체를 같은 크기로 조정하려면, 객체를 선택한 후 도구 모음의 객체 크기 조정 풀다운 메뉴를 선택합니다. 모든 선택된 객체를 최대 또는 최소 객체의 폭과 높이로 크기 조정할 수 있으며, 모든 선택된 객체를 특정한 픽셀 크기로 조정할 수 있습니다. 프런트패널에 분리자 막대를 추가하고 구획을 생성하면, <Shift> 키를 누르고 크기를 조정하고 싶은 객체를 선택하여 여러 구획에 있는 다른 객체들을 선택할 수 있습니다.

프런트패널 객체 스케일하기

구획의 크기를 조정할 때, 구획 크기에 따라서 스케일되거나 자동으로 크기 조정되도록 프런트패널 객체를 설정할 수 있습니다. 프런트패널의 한 객체를 스케일하거나, 소유하는 구역의 모든 객체를 스케일하도록 설정할 수 있습니다. 하지만, 모든 객체를 스케일하도록 설정하거나, 객체를 먼저 그룹화하거나, 또는 객체들이 각각 다른 구획에 있지 않는 한, 프런트패널에서 여러 객체를 스케일하도록 설정할 수 없습니다. 또한, 하나의 객체가 구획을 채우도록 설정할 수 있습니다.

구획의 크기가 변할 때 단일 객체가 자동으로 스케일되도록 설정하려면, 해당 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 구획에 따라 객체 스케일 조정을 선택합니다. 하나의 프런트패널 객체가 스케일되도록 설정한 경우, 객체는 구획 크기의 변화에 따라서 자동으로 크기 조정됩니다. 프런트패널의 다른 객체는 이전 배치와 일관성을 유지할 수 있도록 위치를 옮기지만, 구획의 새로운 크기에 맞도록 스케일되지는 않습니다.

다음 프런트패널에서 보이는 것처럼, 프런트패널의 한 객체를 자동으로 스케일하도록 설정하면 곧바로 회색 라인이 프런트패널을 여러 영역으로 나눕니다. 그 영역은 스케일하려는 객체를 기준으로 하여 다른 프런트패널 객체의 위치를 정의합니다. 구획의 크기를 조정할 때, 자동적으로 스케일하도록 설정한 객체는 크기를 재조정하고 그 원래의 위치를 중심으로 위치를 재조정합니다. VI를 실행할 때 회색 라인이 사라집니다.



구획의 크기가 변할 때 소유하는 구획의 모든 객체가 자동으로 스케일되도록 설정하려면, 분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 구획 크기 조정>>크기 조정에 따라 객체 스케일 조정을 선택합니다. 구획에서 스케일하는 아이템은

사용자가 구획 크기 조정을 완료했을 때가 아니라 구획의 크기를 조정하는 동안에 스케일됩니다. 구획의 크기가 변할 때 소유하는 구역의 모든 객체가 자동으로 스케일되도록 설정하려면, 분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후 해당 구획의 바로 가기 메뉴에서 **구획 크기 조정** > **구획에 따라 모든 객체 스케일 조정**을 선택합니다. 구획의 모든 객체가 스케일되도록 설정한 경우, 모든 객체는 구획 크기의 변화에 따라서 자동으로 크기와 위치가 조정됩니다. 구획의 모든 객체가 스케일되도록 설정하고 구획의 하나의 객체가 이미 스케일되도록 설정된 경우, 대화 상자가 나타나 구획에서 하나의 객체 또는 모든 객체에 대한 자동 스케일링 설정 사이에서 선택하도록 입력 청합니다. 또한, 스케일링 모드 설정 메소드를 사용하여 모든 구획의 객체가 프로그램적으로 스케일되도록 설정할 수 있습니다.

LabVIEW가 자동으로 객체를 스케일할 때, 객체를 수동으로 크기 조정할 때와 같은 규정을 따릅니다. 예를 들어, 일부 객체는 수평 또는 수직으로만 크기 조정이 되며 객체의 크기를 조정할 때 폰트 크기는 유지됩니다.

LabVIEW가 자동으로 객체를 스케일한 후에, 구획을 원래 크기로 되돌려 놓아도 객체는 정확히 원래의 크기로 스케일되지 않는 것입니다. VI를 저장하기 전에, 편집 > 실행 취소를 선택하여 원래 프론트패널 윈도우와 객체 크기를 복원합니다.

배열을 스케일하거나 또는 배열 속의 객체를 스케일하도록 설정할 수 있습니다. 배열을 스케일하도록 설정한 경우, 배열에서 볼 수 있는 행과 열의 개수를 조정합니다. 배열 속의 객체를 스케일하도록 설정한 경우, 배열 속의 행과 열의 개수는 변하지 않고 크기만 변합니다.

또한 클러스터를 스케일하거나 또는 클러스터 속의 객체를 스케일하도록 설정할 수 있습니다. 클러스터가 스케일되도록 설정한 경우, 클러스터 내의 객체는 스케일되지 않습니다. 클러스터 내의 객체가 스케일되도록 설정한 경우, 사용자가 클러스터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **자동 크기 조정** > **크기에 맞게 조정**을 선택하지 않는 이상 클러스터는 스케일되지 않습니다.

하나의 객체가 구획을 채우도록 설정하려면, 해당 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **컨트롤을 구획에 맞춤**을 선택합니다. 이 객체는 구획에 있는 유일한 객체여야 하는데, 그렇지 않다면 구획에 맞추도록 선택한 객체가 다른 객체들과 겹쳐집니다. 객체는 구획을 채우도록 크기를 조정하며, 구획 크기의 변화에 따라 자동으로 크기를 조정합니다.

윈도우 크기를 조정하지 않고 프론트패널에 공간을 추가하거나 공간을 절약하기

윈도우 크기를 조정하지 않고 프론트패널에 **공간을 추가하거나 절약**할 수 있습니다.

사용자 인터페이스 디자인하기

VI가 사용자 인터페이스 또는 대화 상자로 사용될 때, 프론트 패널의 모습 및 레이아웃은 매우 중요합니다. **프론트패널을 디자인**할 때 인스트루먼트 또는 다른 디바이스와 비슷하게 만들어서, 사용자가 수행할 동작을 이해하기 쉽도록 합니다. 프론트패널 컨트롤과 인디케이터, 분리자 막대와 구획, 윈도우 설정 등을 사용하여 프론트패널의 활용도를 높입니다. 또한, **이벤트를 사용**하여 사용자 인터페이스의 기능을 향상시킬 수 있습니다.

프론트패널 디자인하기

프론트패널의 주요 구성요소는 컨트롤과 인디케이터입니다. 다음 가이드라인을 사용하여 사용자 인터페이스로 사용할 프론트패널을 디자인합니다:

- 사용자가 어떻게 VI와 논리적으로 상호작용하는지를 고려합니다. 이에 따라 컨트롤과 인디케이터를 그룹화합니다.
- 서로 관련된 컨트롤을 클러스터 안에 놓거나 관련 컨트롤 주변에 장식용 경계선을 두릅니다.
- **장식** 팔레트에 위치한 장식을 사용하여 상자, 라인 그리고 화살표로 객체를 그룹으로 모으거나 구분합니다.
- 프론트패널을 알아보기 쉽도록 객체 사이에 빈 공간을 두십시오. 빈 공간은 또한 사용자가 실수로 다른 컨트롤 또는 버튼을 클릭하는 것을 방지합니다.
- 와이어 위에 다른 객체를 놓지 않습니다. 컨트롤이나 인디케이터의 일부에만 라벨이나 다른 객체가 겹쳐 있는 경우 스크린 업데이트가 느려지고 컨트롤 또는 인디케이터가 깜박거릴 수 있습니다.
- 버튼에 시작, 중지, 저장과 같은 특정한 이름을 지정합니다. 일반적인 용어를 사용합니다.
- 객체 정렬 및 간격 조절 풀다운 메뉴를 사용하여 레이아웃을 깔끔하고 보기 쉽게 만듭니다.

폰트 및 색 설정하기

또한 폰트와 색을 설정하여 사용자 인터페이스의 활용도를 높일 수 있습니다. LabVIEW는 **내장 폰트**를 기본 시스템 폰트로 정의합니다. VI를 플랫폼 사이에서 이동하는 경우, LabVIEW는 자동으로 내장 폰트를 업데이트하여 현재 플랫폼의 기본 시스템 폰트와 일치하도록 합니다. 사용할 수 없는 폰트를 사용하는 VI를 여는 경우, LabVIEW는 가장 비슷한 폰트로 이를 대체합니다. 또한, LabVIEW는 폰트 대체와 유사한 방법으로 색을 대체합니다. VI의 색 중 하나를 사용할 수 없는 경우, LabVIEW는 이를 가장 가까운 색으로 대체합니다. 시스템 색은 컴퓨터에 따라 다릅니다. 따라서 시스템 색을 사용하는 모든 VI는 현재 사용 중인 컴퓨터 고유의 시스템 색을 반영합니다. 시스템 색은 패널 & 객체, 윈도우, 하이라이트의 모양과 시스템 색 위에 나타나는 텍스트의 색상도 결정합니다. 옵션 대화 상자에서 **사용자 정의 색을 설정하고 시스템 색을 확인**할 수 있습니다.

분리자 막대와 구획 사용하기

도구 모음이나 **상태 막대**와 같은 분리자 막대를 사용하여 프론트패널 윈도우에 전문적인 사용자 인터페이스를 만들 수 있습니다. **분리자 막대를 생성**해서 구획이라고 불리는 여러 영역으로 프론트패널을 분리할 수 있습니다. 각 구획은 자체적인 구획 좌표, 컨트롤, 인디케이터를 갖고 있어서, 개별 프론트패널처럼 동작합니다. 사용자는 각각의 구획을 개별적으로 스크롤할 수 있습니다. 분리자 막대는 특정 구획에 있는 컨트롤을 다른 구획에 있는 컨트롤과 분리하지만 모든 컨트롤의 터미널은 동일 블록 다이어그램에 있습니다.

새 VI를 생성할 때, 프론트패널은 윈도우를 채우는 하나의 단일 구획을 갖습니다. 프론트패널은 이 구획을 소유하며 해당 구획의 직상위입니다. 구획에 분리자 막대를 놓으면, 분리자 막대는 프론트패널 객체 계층구조 상에서 구획을 대체하게 되고 두 개의 새 직하위 구획을 생성합니다. 프론트패널은 분리자 막대를 소유하고 분리자 막대는 두 직하위 구획을 소유합니다. 직하위 구획 중 하나에 분리자 막대를 새로 놓게 되면, 새 분리자 막대

는 해당 구획을 대체하며 새로 만들어진 두 직하위 구획의 직상위 구획이 됩니다. 이 계층구조는 2진 트리를 형성하고 프런트패널은 최상위 구획을 소유합니다.

분리자 막대와 구획을 사용하는 예제를 보려면 labview\examples\Controls and Indicators\Containers에 있는 Resizable Panel with Splitter Bars VI를 참조하십시오.

분리자 막대 설정하기

분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후, 다음 바로 가기 메뉴 옵션 중 하나를 선택하여 분리자 막대를 설정합니다:

- 보이는 아이템—분리자 막대 라벨을 프런트패널에 추가합니다. LabVIEW는 기본적으로 분리자 막대 라벨을 숨깁니다.
- 생성—사용자가 프로퍼티 노드, VI 서버 참조 또는 인보크 노드를 생성하여 프로그램적으로 분리자 막대를 설정할 수 있습니다.
- 전체 분리자 계층구조 제거—선택한 분리자 막대와 계층구조 상에서 직하위인 모든 분리자 막대를 제거합니다. 객체는 프런트패널에서 동일한 스크린 위치를 유지합니다. 다른 분리자의 직상위가 아닌 분리자 막대를 선택하면, LabVIEW는 이 옵션을 비활성화합니다.
- 인접하는 분리자 제거—선택한 분리자 막대와 해당 분리자 막대와 끝이 닿는 모든 분리자 막대를 제거합니다.
- 분리자 크기 조정—윈도우 크기를 조정할 때 구획에 미치는 영향을 컨트롤할 수 있도록 허용합니다. 분리자 크기 조정 바로 가기 메뉴에서 옵션을 선택하면, 윈도우 크기를 조정함으로써 다른 구획이 커지거나 작아지는 동안 해당 구획이 동일한 크기로 유지되거나 고정되도록 설정할 수 있습니다. 또한 비례하여 분리자 크기 조정을 선택하여 윈도우 크기를 조정할 때 두 구획이 모두 커지거나 작아지도록 허용할 수 있습니다.
- 고정—현재 위치에 분리자 막대를 고정시킵니다.
- 분리자 스타일—분리자 막대의 스타일을 사용자 정의할 수 있도록 합니다.
- 위쪽 구획/아래쪽 구획/왼쪽 구획/오른쪽 구획—선택한 구획을 설정할 수 있도록 합니다. 구획을 선택할 때, 선택한 구획을 설정하는 데에 쓰이는 옵션 포함하는 새 바로 가기 메뉴가 열립니다. 구획 설정하기에 대한 추가적인 정보는 이 토픽의 **구획 설정하기** 섹션을 참조하십시오.

또한 분리자 프로퍼티를 사용하여 프로그램적으로 분리자 막대를 설정할 수 있습니다.

구획 설정하기

구획의 스크롤 바에서 마우스 오른쪽 버튼을 클릭하거나 분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후, 설정하려는 구획을 선택하고 다음 바로 가기 메뉴 옵션 중 하나를 선택하여 구획을 설정합니다:

- 보이는 아이템—구획 라벨을 프런트패널에 추가합니다. LabVIEW는 기본적으로 구획 라벨을 숨깁니다.
- 생성—사용자가 프로퍼티 노드, VI 서버 참조 또는 인보크 노드를 생성하여 프로그램적으로 구획을 설정할 수 있습니다.

- 수평 스크롤 막대—수평 스크롤 막대가 항상 켜기, 항상 끄기 또는 실행 중 끄기인지 선택할 수 있습니다.
- 수직 스크롤 막대—수직 스크롤 막대가 항상 켜기, 항상 끄기 또는 실행 중 끄기인지 선택할 수 있습니다.



노트

프런트패널 윈도우의 스크롤 막대에서 마우스 오른쪽 버튼을 클릭하여, 수평이나 수직 스크롤 막대를 끄면, 윈도우 모양 사용자 정의 대화 상자에서 수평 스크롤 막대 보이기 옵션이나, 수직 스크롤 막대 보이기 옵션을 활성화하여 스크롤 막대를 다시 디스플레이할 수 있습니다.

- 구획 크기 조정—윈도우 크기를 조정할 때 구획에 미치는 영향을 컨트롤할 수 있습니다. 크기가 조정된 구획의 객체가 구획의 아래쪽, 위쪽, 왼쪽, 또는 오른쪽에 남아있을지를 선택할 수 있습니다. 또한 윈도우 크기를 조정할 때 구획에 있는 객체를 스케일할지 여부도 선택할 수 있습니다.

프런트패널에 분리자 막대가 없는 경우, VI 프로퍼티 대화 상자에서 단일 구획의 일부 프로퍼티를 설정할 수 있습니다. 그러나, 프런트패널에 분리자 막대를 추가하는 경우, 옵션 설정이 모든 구획에 적용될 수 없기 때문에 LabVIEW는 VI 프로퍼티 대화 상자에 있는 메뉴 옵션을 제한합니다. 사용자는 각각의 구획을 개별적으로 설정해야 합니다. 예를 들어, 프런트패널에 분리자 막대를 추가하면 VI 프로퍼티 대화 상자의 윈도우 크기 페이지에 있는 윈도우가 크기 조정될 때 프런트패널의 모든 객체 스케일 확인란이 비활성화됩니다.

또한 구획 프로퍼티를 사용하여 구획을 프로그램적으로 설정할 수 있습니다.

대화 상자 디자인하기

파일»VI 프로퍼티를 선택한 후 항목 풀다운 메뉴에서 윈도우 모양을 선택하고 대화 상자 옵션을 선택하여 메뉴 모음과 스크롤 막대를 숨기고 각 플랫폼에서 표준 대화 상자처럼 보이고 동작하는 VI를 생성합니다.

VI가 같은 스크린 위치에 연속적으로 나타나는 **대화 상자**를 포함하는 경우, 첫번째 대화 상자의 버튼이 두번째 대화 상자의 버튼과 같은 곳에 있지 않도록 구성합니다. 사용자가 첫번째 대화 상자의 버튼을 더블 클릭할 때 무심코 다음 대화 상자의 버튼을 클릭할 수 있기 때문입니다.

VI가 실행 중일 때 사용자가 취소 동작을 수행하기 위해 키보드 바로 가기 키 <Ctrl-Z>를 누를 수도 있습니다. 이로 인해 닫힌 대화 상자가 다시 열리는 등, 대화 상자가 예기치 않은 동작을 일으킵니다. 이 동작을 방지하려면, 편집»실행 취소 옵션을 포함하지 않는 **사용자 메뉴를 생성**하거나, VI에 이벤트 구조를 추가하고 메뉴 선택(어플리케이션) 이벤트를 **설정**하여 명령을 처리합니다.

생성한 대화 상자의 시스템 팔레트에 위치한 시스템 컨트롤을 사용합니다.

스크린 크기 선택하기

VI를 디자인할 때, 다른 스크린 해상도를 가진 컴퓨터에서 **프런트패널을 디스플레이할 수 있는**지를 고려합니다.

컨트롤을 지나치게 뽁뽁하게 채우거나 선명한 레이아웃을 해치지 않는 범위 내에서 최대한 프런트패널을 작게 만듭니다. 일부 디스플레이의 경우 해상도가 제한되는데, 특히 LCD 디스플레이와 터치스크린의 경우 그렇습니다.

또한 프런트패널은 작은 스크린을 사용하는 사용자의 편의를 위해 스크린 왼쪽 상단 코너에서 열리도록 설정합니다. 자주 사용하는 VI들을 함께 배치하여 사용자가 최소한 각 VI의 작은 부분이라도 볼 수 있도록 합니다. 자동으로 열리는 프런트패널은 스크린 중앙에 두어, 사용자가 다양한 크기의 모니터에서 VI를 보다 쉽게 볼 수 있도록 합니다.

파일»VI 프로퍼티에서, 항목 풀다운 메뉴의 윈도우 크기를 선택한 후 다른 모니터 해상도에서 윈도우의 비율 유지 확인란에 확인 표시를 하여 스크린 해상도에 대해 상대적인 프런트패널 윈도우 비율을 유지하도록 합니다.

프런트패널 객체 컨트롤하기

참조 번호와 클래식 참조 번호 팔레트에 위치한 컨트롤 참조 번호 컨트롤을 사용하여 프런트패널 객체 참조를 다른 VI에 전달합니다. 또한, 프런트패널 객체 또는 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성»참조를 선택하여 VI 서버 참조를 생성합니다. 덧붙여서, VI 서버 참조 상수로부터 프런트패널 객체를 블록다이어그램에 놓을 수 있습니다. 블록다이어그램에 있는 참조에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 컨트롤 찾기를 선택합니다. VI 서버 참조를 subVI에 전달한 후, 프로퍼티 노드와 인보크 노드를 사용하여 프로퍼티를 읽고 쓰며 참조된 프런트패널 객체의 메소드를 수행합니다.



노트

컨트롤 참조는 VI를 자동적으로 메모리에 남겨두지 않으며, VI가 메모리에 남아 있어야 할지의 여부를 제어하지 않습니다. VI가 메모리에 남아있지 않으면, 컨트롤 참조는 무효하게 됩니다. 컨트롤 참조를 사용할 때 VI가 메모리에 남아있게 하려면, [VI 참조 열기] 함수를 사용해서 VI를 가져오고, [참조 닫기] 함수를 사용해서 이 VI를 적절한 방법으로 닫습니다. 또한 VI의 프런트패널을 열어놓아도 VI가 계속 메모리에 남아있습니다.

또한, **이벤트를 사용**하여 프런트패널 객체를 통하여 프로그램적으로 블록다이어그램의 동작을 컨트롤할 수 있습니다.

컨트롤 참조를 사용하여 데이터를 전달할 수 있으나, 성능을 개선하려면 컨트롤 자체에서 데이터를 읽고 쓰거나 **로컬 또는 글로벌 변수**를 대신 사용하십시오.

컨트롤 참조는 텍스트 기반 프로그래밍 언어의 사용자 인터페이스 객체 참조에 대응합니다. 컨트롤 참조는 텍스트 기반 프로그래밍 언어의 포인터와는 대응하지 않습니다.

컨트롤 참조를 사용하는 예제는 labview\examples\Application Control\VI Server\Control References 디렉토리의 Control References VI를 참조하십시오. 관련 예제 찾기

엄격한 타입과 엄격하지 않은 타입의 컨트롤 참조 번호

엄격한 타입의 컨트롤 참조 번호는 정확히 동일한 데이터 타입의 컨트롤 참조 번호를 수용합니다. 예를 들어, 엄격한 타입의 컨트롤 참조 번호의 타입이 32비트 정수 슬라이드인

경우, 32비트 정수 슬라이드만 컨트롤 참조 번호 터미널에 연결할 수 있습니다. 컨트롤 참조 번호 터미널에 8비트 정수 슬라이드, 배정도 스칼라 슬라이드, 또는 32비트 정수 슬라이드의 클러스터를 연결할 수 없습니다.

컨트롤에서 생성한 컨트롤 참조는 기본으로 엄격한 타입입니다. 프런트패널 컨트롤 참조의 왼쪽 아래 코너에 있는 붉은색 별표는 이 컨트롤 참조가 엄격한 타입임을 나타냅니다. 블록 다이어그램에서 컨트롤 참조 터미널에 연결된 프로퍼티 노드 또는 인보크 노드에 (엄격함)이 표시되어 해당 컨트롤 참조가 엄격한 타입임을 나타냅니다.



노트 기계적인 래치 동작은 엄격한 타입의 컨트롤 참조와 호환되지 않기 때문에 기계적인 래치 동작을 가진 볼리언 컨트롤은 엄격하지 않은 타입의 컨트롤 참조를 생성합니다.

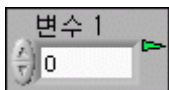
엄격하지 않은 타입의 컨트롤 참조 번호는 더 다양한 데이터 타입을 받습니다. 예를 들어, 엄격하지 않은 타입의 컨트롤 참조가 슬라이드인 경우, 32비트 정수 슬라이드, 단정도 슬라이드, 또는 32비트 정수 클러스터의 슬라이드를 컨트롤 참조 터미널에 연결할 수 있습니다. 엄격하지 않은 타입의 컨트롤 참조 타입이 컨트롤인 경우, 모든 타입 컨트롤의 컨트롤 참조를 컨트롤 참조 터미널에 연결할 수 있습니다.



노트 프로퍼티 노드를 엄격하지 않은 타입의 컨트롤 참조 터미널에 연결하면 값 프로퍼티는 **배리언트 데이터**를 생성합니다. 이 데이터는 사용하기 전에 변환해야 할 경우도 있습니다. 차트의 히스토리 데이터 프로퍼티는 차트 참조가 엄격한 타입일 경우에만 사용 가능합니다.

프런트패널 데이터 연결의 문제 해결

프런트패널 컨트롤이나 인디케이터를 편집 시 공유 변수에 연결하면, 작은 투명 인디케이터가 프런트패널 객체 옆에 나타납니다. VI를 실행할 때, 공유 변수 엔진으로의 연결이 제대로 이루어진 경우, 다음 그림에서처럼 인디케이터가 녹색이 됩니다.



연결이 성공적이지 못한 경우, 인디케이터는 빨간색이 됩니다. 또한, 커서를 인디케이터 위로 이동하는 경우 상태 메시지가 나타납니다.



노트

다음의 방법을 사용하여 이 인디케이터를 숨길 수 있습니다:

- 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> PSP LED를 선택합니다.
- 데이터 연결: LED 보이기 프로퍼티를 사용하여 프로그램적으로 인디케이터를 숨깁니다.

프런트패널에서 데이터를 읽거나 쓸 때 에러가 발생하는 경우, 다음 가능한 원인을 조사하십시오:

- 공유 변수를 반드시 **배포**해야 합니다.
- 공유 변수가 상주하는 타겟의 디바이스가 작동을 하는지와 프런트패널 데이터 연결을 지원하는지 확인합니다. 프런트패널 데이터 연결을 지원하려면 LabVIEW 모듈 및 툴킷에 대한 문서를 참조하십시오.

LabVIEW의 변수

로컬 변수, 글로벌 변수, 피드백 노드

LabVIEW에서는 블록 다이어그램의 터미널을 사용하여 프런트패널 객체의 데이터를 읽거나 씁니다. 그러나 프런트패널의 객체는 블록 다이어그램 터미널이 하나만 있고, 어플리케이션이 여러 위치에서 이 터미널의 데이터에 접근해야 하는 경우가 있을 수 있습니다.

로컬 변수와 글로벌 변수는 어플리케이션에서 와이어로 연결할 수 없는 위치에서 정보를 전달합니다. 하나의 VI 내에서의 여러 위치에서 프런트패널 객체에 접근해야 하는 경우, 로컬 변수를 사용합니다. 글로벌 변수를 사용하여 여러 VI에 접근하고 데이터를 전달합니다.

피드백 노드를 사용하면, 이전 VI나 루프 실행에서 넘어온 데이터를 저장할 수 있습니다.

피드백 노드에 대한 예제는 Structures\Feedback Node 디렉토리에 있는 Feedback Node - Building an Array를 참조하십시오.



관련 정보

[로컬 변수](#) (560 페이지 참조)

[피드백 노드: 루프 반복 사이에 데이터 저장하기](#) (371 페이지 참조)

[로컬 변수](#) (560 페이지 참조)

[글로벌 변수](#) (560 페이지 참조)

[피드백 노드: 루프 반복 사이에 데이터 저장하기](#) (371 페이지 참조)

[변수 읽기와 쓰기](#) (561 페이지 참조)

[로컬과 글로벌 변수를 사용시 유의 사항](#) (562 페이지 참조)

[로컬 변수 생성하기](#) (1087 페이지 참조)

[로컬 변수의 데이터 방향 변경하기](#) (1086 페이지 참조)

[글로벌 변수 생성하기](#) (1088 페이지 참조)

[글로벌 변수의 데이터 방향 변경하기](#) (1088 페이지 참조)

로컬 변수

프런트패널 객체에 대한 접근 권한이 없거나 블록다이어그램 노드 사이에 데이터를 전달해야 할 때, 단일 VI에 있는 프런트패널 객체에 대한 로컬 변수를 생성합니다. 로컬 변수를 생성하면, 블록다이어그램에는 나타나지만 프런트패널에는 나타나지 않습니다.

로컬 변수를 이용하여 프런트패널의 컨트롤 값을 읽거나 인디케이터에 값을 쓸 수 있습니다. 로컬 변수에 값을 쓰는 것은 다른 일반적인 터미널에 데이터를 전달하는 것과 비슷합니다. 그러나 로컬 변수를 이용하면 컨트롤인 것처럼 데이터를 쓰거나 인디케이터인 것처럼 데이터를 읽을 수 있습니다. 결국 로컬 변수를 이용하여 입력과 출력 모두로 프런트패널의 객체에 접근할 수 있습니다.

예를 들어 사용자 인터페이스에 사용자가 로그인해야 하는 경우, 새 사용자가 로그인할 때마다 새 로그인과 암호를 기입할 수 있도록 빈칸으로 초기화할 수 있습니다. 사용자가 로그인할 때 로그인과 암호 문자열 컨트롤에서 값을 읽고, 사용자가 로그아웃할 때 빈 문자열을 이 컨트롤에 쓰려면 로컬 변수를 사용해야 합니다.

로컬 변수 생성하기

기존의 프런트패널의 객체나 블록다이어그램의 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>로컬 변수를 선택하여 **로컬 변수를 생성**합니다. 이 객체의 로컬 변수 아이콘이 블록다이어그램에 나타납니다.

또는 함수 팔레트에서 로컬 변수를 선택하여 블록다이어그램에 놓을 수 있습니다. 다음에서 보는 것처럼 로컬 변수 노드는 아직 컨트롤이나 인디케이터에 연결되어 있지 않습니다.



로컬 변수를 컨트롤이나 인디케이터에 연결하려면, 로컬 변수 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이템 선택을 선택합니다. 확장된 바로 가기 메뉴에는 고유 라벨이 있는 프런트패널 객체가 모두 나열됩니다.

LabVIEW는 로컬 변수를 프런트패널의 객체에 연결할 때 고유 **라벨**을 사용하기 때문에, 프런트패널의 컨트롤과 인디케이터에 설명적인 고유 라벨을 할당해야 합니다.

글로벌 변수

글로벌 변수를 사용하여 동시에 실행되는 여러 VI에 접근하고 데이터를 전달합니다. 글로벌 변수는 LabVIEW에 포함되어 있는 LabVIEW 객체입니다. 글로벌 변수를 생성하면, LabVIEW는 자동적으로 블록다이어그램은 없고 프런트패널만 있는 특수한 글로벌 VI를 생성합니다. 글로벌 VI의 프런트패널에 컨트롤과 인디케이터를 추가하여 포함된 글로벌 변수의 데이터 타입을 정의합니다. 이 프런트패널은 몇몇 VI가 데이터에 접근할 수 있는 컨테이너입니다.

예를 들어, 동시에 실행되는 두 개의 VI가 있다고 가정합니다. 각 VI는 While 루프를 포함하며 웨이브폼 차트에 데이터 포인트를 씁니다. 첫번째 VI에 불리언 컨트롤이 있어서 두 VI 모두를 종료시킵니다. 한 개의 불리언 컨트롤로 양쪽 루프를 모두 종료하기 위해서는

글로벌 변수를 반드시 사용해야 합니다. 동일한 VI 내의 블록다이어그램에 두 개의 루프가 있다면, 로컬 변수를 이용하여 루프를 종료할 수 있습니다.

글로벌 변수 생성하기

다음과 같이 함수 팔레트에서 **글로벌 변수를 선택**하여 블록다이어그램에 놓습니다.



글로벌 VI의 프런트패널을 디스플레이하려면 이 글로벌 변수 노드를 더블 클릭하십시오. 일반 프런트패널에서 했던 방법과 동일하게 컨트롤과 인디케이터를 프런트패널에 놓습니다.

LabVIEW는 고유 **라벨**을 이용하여 글로벌 변수를 구분하기 때문에, 설명적인 고유 라벨로 프런트패널의 컨트롤과 인디케이터에 라벨을 붙입니다.

각각이 하나의 프런트패널 객체를 가진 여러 개의 단일 글로벌 VI를 만들거나, 비슷한 변수를 함께 그룹으로 만들기 위해 여러 개의 프런트패널 객체를 가진 하나의 글로벌 VI를 만들 수 있습니다.

글로벌 VI의 프런트패널에 객체를 놓은 다음, 이것을 저장하고 원래 VI의 블록다이어그램으로 돌아옵니다. 그 후 접근하려고 하는 글로벌 VI의 객체를 선택해야 합니다. 글로벌 변수 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프런트패널의 객체를 선택합니다. 이 바로 가기 메뉴는 고유 라벨을 가진 글로벌 VI에 있는 모든 프런트패널 객체를 나열합니다. 글로벌 변수 노드에서 마우스 오른쪽 버튼을 클릭하고 아이템 선택 바로 가기 메뉴에서 프런트패널의 객체를 선택할 수도 있습니다.

글로벌 변수 노드의 복사본을 생성하면, LabVIEW는 새로운 글로벌 변수 노드를 복사한 변수 노드의 글로벌 VI와 연관시킵니다.

변수 읽기와 쓰기

로컬 또는 글로벌 변수를 생성한 후, 변수에서 데이터를 읽거나 변수에 데이터를 쓸 수 있습니다. 기본으로 새 변수는 데이터를 받을 수 있도록 설정되어 있습니다. 이러한 종류의 변수는 인디케이터로 작동하며, 쓰기 로컬 변수 또는 쓰기 글로벌 변수입니다. 새 데이터를 로컬 변수 또는 글로벌 변수에 쓰면, 연결된 프런트패널 컨트롤이나 인디케이터는 새 데이터에 따라 업데이트됩니다.

또한, 변수를 데이터 소스, 또는 읽기 로컬이나 읽기 글로벌처럼 동작하도록 설정할 수 있습니다. 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 읽기로 변경을 선택하여, 이 변수가 컨트롤처럼 동작하도록 설정합니다. 이 노드가 실행되면, VI는 연결된 프런트패널의 컨트롤이나 인디케이터에서 데이터를 읽습니다.

변수가 데이터를 제공하는 것이 아니라 블록다이어그램에서 데이터를 받도록 변경하려면, 변수에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 쓰기로 변경을 선택하십시오.

블록다이어그램에서 컨트롤과 인디케이터를 구별하는 것과 같은 방법으로 읽기 로컬 또는 글로벌을 쓰기 로컬 또는 글로벌과 구별할 수 있습니다. 읽기 로컬 또는 글로벌은 컨트롤과 유사한 굵은 경계선이 있습니다. 쓰기 로컬 또는 글로벌에는 인디케이터처럼 얇은 경계선이 있습니다.

로컬과 글로벌 변수를 사용시 유의 사항

로컬과 글로벌 변수는 고급 LabVIEW 개념에 속합니다. 이 변수는 원래 LabVIEW 데이터 흐름 실행 모델의 일부가 아닙니다. 로컬과 글로벌 변수를 사용할 때 블록다이어그램이 읽기 어려워질 수 있으므로 주의해서 변수를 사용해야 합니다. 커넥터 팬 대신에 로컬 또는 글로벌 변수를 사용하거나 시퀀스 구조의 각 프레임마다 값에 접근하기 위해 로컬과 글로벌 변수를 사용하는 것과 같이 이들을 잘못 사용하면, VI에서 예상치 못한 결과를 초래할 수도 있습니다. 블록다이어그램 전체를 지나는 긴 와이어를 피하기 위해서나, 또는 **데이터 흐름** 대신에 변수를 사용한다는지와 같이 로컬 변수 및 글로벌 변수를 과도하게 사용하면 성능이 느려질 수 있습니다.

로컬과 글로벌 변수 초기화하기

로컬 변수나 글로벌 변수를 초기화하려면 VI를 실행하기 전, 알려진 값을 변수에 씁니다. 확인하지 않는 경우, VI가 올바르게 작동할 수 있도록 하는 데이터가 변수에 포함되어 있을 수 있습니다. 변수의 초기값이 연산 결과에 의존할 경우, LabVIEW가 반드시 다른 동작을 위해 변수에 접근하기 전에 변수에 해당 값을 쓰도록 합니다. 쓰기 동작이 VI의 나머지 부분과 병렬로 작동하도록 연결하면 **경합 조건**이 발생할 수 있습니다.

VI의 다른 작업들이 실행되기 전에 변수가 처음으로 시작되도록 하려면 변수에 초기값을 쓰는 코드를 **시퀀스 구조**의 첫 번째 프레임이나 SubVI에 따로 두고, 해당 SubVI를 연결하여 블록다이어그램의 데이터 흐름에서 첫 번째로 실행되도록 합니다.

VI가 변수를 처음으로 읽기 전에 이 변수를 초기화하지 않으면 이 변수는 연결된 프런트 패널 객체의 기본값을 갖습니다.

글로벌 또는 로컬 변수 값이 다이어그램의 다른 항목에 연결되면, 변수가 변경될 때 와이어의 값이 변경되지 않습니다. 와이어 값을 업데이트하려면, 변경 후에 변수의 값을 읽어야 합니다.

경합 조건

두 개 혹은 그 이상의 코드가 병렬 실행되고, 공유 메모리에 접근할 수 있을 때 경합 조건이 발생할 수 있습니다. 각 코드가 독립적이라면, LabVIEW가 어떤 순서로 공유하는 리소스에 접근할지 구분할 수 없습니다.

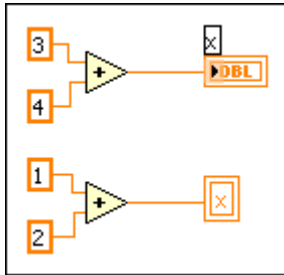
경합 조건은 예상치 못한 결과를 초래할 수도 있습니다. 예를 들어, 두 개의 독립적인 코드가 큐에 접근할 때, 사용자가 LabVIEW가 큐에 어떤 순서로 접근할지 제어하지 않으면, 경합 조건이 발생할 수도 있습니다.

경합 조건은 프로그램의 타이밍이 바뀔 때 나타나거나, 사라질 수 있기 때문에 위험할 수도 있습니다. 경합 조건은 OS를 바꿨을 때, 또는 LabVIEW 버전이나 다른 소프트웨어를 바꿨을 때 발생할 수 있습니다.

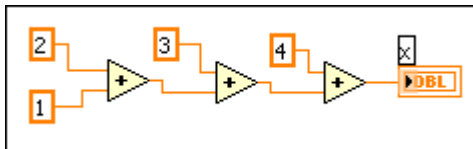
OS나 LabVIEW 버전의 업그레이드 등으로 VI 타이밍에 변화가 생기면, 같은 데이터에 접근하는 병렬 코드를 찾아 타이밍 조건을 이용하여 어떤 작업이 먼저 실행될지를 제어합니다.

로컬 변수와 글로벌 변수 이용 시 경합 조건

경합 조건은 하나 이상의 작업이 함께 공유하는 데이터 값을 업데이트할 때마다 발생할 수 있지만, 로컬과 글로벌 변수를 사용할 때, 또는 외부 파일을 사용할 때 종종 발생하기도 합니다. 다음 블록다이어그램은 **로컬 변수**에서 경합 조건의 예제를 보여줍니다.



이 VI의 출력(로컬 변수 x의 값)은 어떤 작업이 먼저 실행하느냐에 따라 달라집니다. 각 작업이 x에 다른 값을 쓰면, 결과가 7인지 3인지 알 수 없습니다. 일부 프로그래밍 언어에서는, 위에서 아래로의 데이터 흐름 패러다임이 실행 순서가 됩니다. LabVIEW에서는, 와이어링을 사용하여 경합 조건이 생기지 않게 하면서 변수에서 여러 작업을 수행할 수 있습니다. 다음 블록다이어그램은 로컬 변수 대신 와이어링을 사용해 추가적인 작업을 수행합니다.



팁

로컬이나 글로벌 변수에서 하나 이상의 작업을 수행해야 하는 경우, 작업의 실행 순서를 결정하십시오.

두 작업이 병렬적으로 글로벌 변수를 업데이트하려고 할 때에도 경합 조건이 발생합니다. 글로벌 변수를 업데이트하기 위해 값을 읽고, 수정하고, 원래의 위치에 다시 쓰는 작업이 수행됩니다. 첫 작업이 읽기-수정하기-쓰기 동작을 수행하고 뒤이어 두번째 작업이 따르면, 그 결과는 예상할 수 있고 바르게 나옵니다. 첫번째 작업이 읽은 후에 두번째 작업이 읽으면, 두 작업 모두가 값을 수정하고 쓰게 됩니다. 이 동작은 읽기-수정하기-쓰기 경합 조건을 야기하고, 유효하지 않거나 찾을 수 없는 값을 생성합니다.

글로벌 변수와 관련된 경합 조건을 방지하려면 **기능적인 글로벌 변수**를 사용하여 변수에서 해당 작업에 접근하는 코드의 중요 섹션을 보호할 수 있습니다. 여러 로컬 또는 글로벌 변수 대신 하나의 기능적인 글로벌 변수를 사용하면 한 번에 하나의 작업만 실행되므로 작업이 서로 상충되거나 저장된 데이터에 상충되는 값을 할당하는 것을 피할 수 있습니다.

루프 내 변수 동작

변수가 여러 병렬 루프를 컨트롤하는 경우, 루프는 변수가 작성된 후 실행되어야 하며 각 루프의 각 반복에서 변수를 읽어야 합니다.

시스템이 다시 시작되면, 변수를 읽는 루프가 실행되기 전에 변수를 적절한 시작 조건으로 설정해야 합니다. 그렇지 않으면 루프가 조기에 정지됩니다.

로컬 변수 사용 시 메모리 관리

로컬 변수는 데이터 버퍼의 복사본을 생성합니다. 로컬 변수에서 데이터를 읽어올 때, 이와 연결된 컨트롤의 데이터에 대한 새로운 버퍼를 생성하게 됩니다.

로컬 변수를 사용하여 대량의 데이터를 블록 다이어그램의 한 위치에서 다른 위치로 이동하는 경우, 일반적으로 와이어를 사용한 데이터 전송보다 더 많은 메모리를 사용하게 되어 결국 실행 속도가 느려지게 됩니다. 실행시 데이터를 저장해야 하는 경우, 시프트 레지스터 사용을 고려하십시오.

글로벌 변수 사용 시 메모리 관리

글로벌 변수에서 데이터를 읽어올 때, LabVIEW는 그 글로벌 변수에 저장된 데이터의 복사본을 생성합니다.

대용량의 배열이나 문자열을 다루는 경우, 글로벌 변수를 관리하려면 상당한 시간과 메모리가 요구됩니다. 배열의 글로벌 변수를 다루는 경우, LabVIEW는 한 개의 배열 원소를 수정하더라도 배열 원소 전체를 저장하고 편집하므로 비효율적입니다. 어플리케이션의 여러 장소에서 글로벌 변수를 읽으면, 여러개의 메모리 버퍼가 만들어지므로 비효율적이며 시스템의 성능을 저하시킵니다.

LabVIEW에서 고려해야 할 성능 및 메모리 관리에 대한 추가적인 정보는 [성능과 메모리 관리하기](#)를 참조하십시오.

공유 변수 기술 이해하기

네트워크 공유 변수는 공유 변수 엔진(SVE)이라고 하는 소프트웨어 구성요소를 통해 네트워크로 데이터를 공개합니다. 공유 변수 엔진은 LabVIEW를 설치할 때 사용자 컴퓨터에 서비스로 설치되며, NI Publish-Subscribe Protocol (NI-PSP)이라고 하는 독점 기술을 사용하여 공유 변수 업데이트를 관리합니다. publish-subscribe라는 용어는 발신자(writer 또는 publisher)가 특정 수신자(reader 또는 subscriber)에게 데이터를 전송하지 않는 통신 모델을 나타냅니다. 오히려 발신자(publisher)는 서버에 업데이트를 전송하고, 이 경우 공유 변수 엔진, 즉 수신자(subscriber)는 해당 업데이트를 서버에서 받습니다.



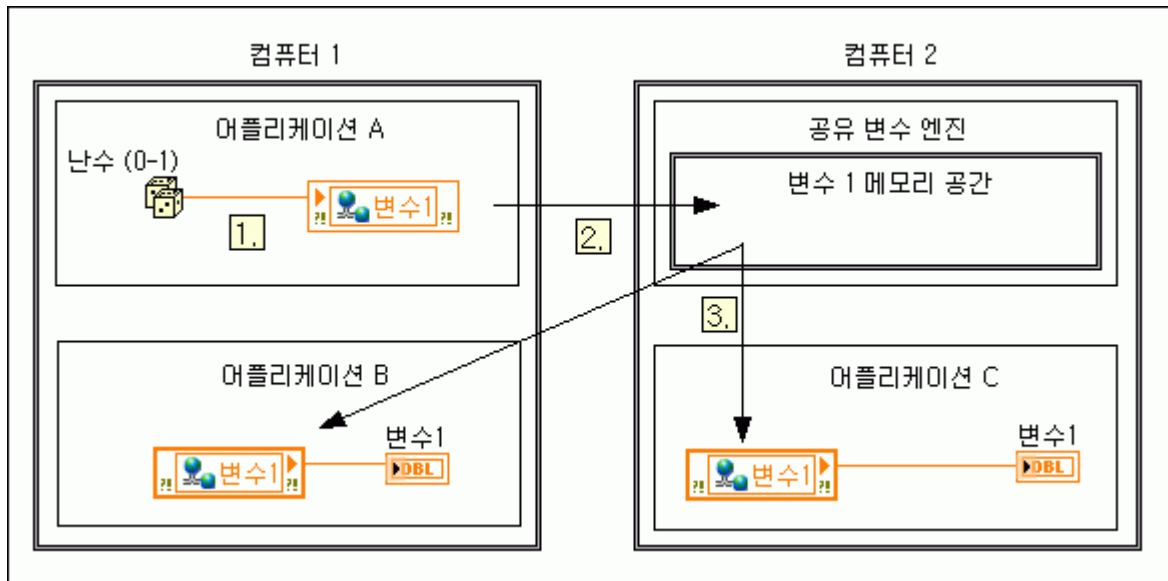
노트

네트워크 공유 변수를 방화벽이나 라우터를 통해 전송하려면, [방화벽과 네트워크 주소 전송\(NAT\) 라우터를 설정](#)해야 합니다.

다음 그림은 공유 변수 노드를 사용하여 공유 변수 엔진이 어떻게 NI-PSP로 공유 변수 업데이트를 관리하는지 보여줍니다.



노트 **공유 변수 노드**는 **공유 변수의 읽고 쓰기**에 사용할 수 있는 여러 프로그래밍 도구 중 하나입니다.



다음과 같은 이벤트가 위의 그림에서 발생합니다:

1. 어플리케이션 A에서, [난수 (0-1)] 함수는 변수 1에 대응하는 공유 변수 노드에 난수를 씁니다.

노트 네트워크를 통해 공유 변수를 읽고 쓰려면, 우선 **네트워크에서 해당 공유 변수의 이용이 가능**하도록 해야 합니다.

2. 어플리케이션 A의 공유 변수 노드는 공유 변수 엔진에 요청을 보내 변수 1의 값을 업데이트합니다.
3. 공유 변수 엔진은 어플리케이션 B와 C의 변수 1에 대응하는 공유 변수 노드에 새로운 값을 승인하여 보냅니다.

이전 그림에서 컴퓨터 1이 어플리케이션 A에 있는 변수 1의 발신자와 어플리케이션 B에 있는 변수 1의 수신자를 호스트하지만, 어플리케이션 A는 어플리케이션 B에 직접 새로운 값을 쓸 수 없습니다. 그 대신, 어플리케이션 A가 컴퓨터 2의 공유 변수 엔진(SVE)에 요청을 보내야만 변수 1을 읽는 모든 어플리케이션을 업데이트할 수 있습니다. 따라서, 이러한 업데이트 관련 지연 때문에 최근값만을 공개할 경우 공유 변수가 이상적입니다. 연속적으로 데이터를 스트리밍하려면 **네트워크 스트림**을 사용합니다.

네트워크에서 공유 변수의 이용 가능하게 하기

공유 변수 엔진(SVE)에 공유 변수를 배포해야만 네트워크를 통해 해당 공유 변수를 읽고 쓸 수 있습니다. 공유 변수를 배포할 때 LabVIEW는 프로젝트 라이브러리에 있는 모든 다른 공유 변수를 포함하여, 해당 공유 변수를 포함하는 **프로젝트 라이브러리**를 공유 변수 엔진(SVE)에 배포합니다. 그리고 나서 공유 변수 엔진은 각 공유 변수에 대한 메모리 공간을 포함하는, 해당 프로젝트 라이브러리에 대한 메모리 공간을 생성합니다.

NI-PSP 데이터 아이템

NI-PSP를 사용하여 네트워크를 통해 LabVIEW가 공개하는 모든 데이터는 NI-PSP 데이터 아이템입니다. 일반적인 NI-PSP 데이터 아이템은 다음을 포함합니다:

- 네트워크 공유 변수
- 네트워크 공개가 활성화된 I/O 변수
- OPC 서버의 데이터 아이템
- FieldPoint 모듈의 데이터 아이템

LabVIEW는 네트워크 경로로 NI-PSP 데이터 아이템을 식별하는데, 네트워크 경로는 데이터 아이템이 위치한 컴퓨터의 이름, 데이터 아이템이 있는 프로세스의 이름, 데이터 아이템의 이름으로 구성됩니다. 예를 들어, 네트워크 경로 `\\computer1\mylib\myvariable`는 `computer1`로 명명된 컴퓨터의 `mylib` 프로젝트 라이브러리에서 `myvariable`라는 이름의 공유 변수를 식별합니다. 공유 변수가 `mysublib`라는 이름의 서브프로젝트 라이브러리에 존재하는 경우, `\\computer1\mylib\mysublib\myvariable`와 같이 공유 변수 경로는 서브프로젝트 라이브러리를 포함합니다.

기존 데이터 소스 또는 **컨트롤과 인디케이터**로 공유 변수를 읽고 쓸 때 NI-PSP 데이터 아이템의 네트워크 경로를 지정하거나 NI-PSP 데이터 아이템을 탐색합니다.

공유 변수 엔진 에러

공유 변수 엔진은 서버 상태 업데이트 및 유효하지 않은 프로세스 설정과 같은 심각한 에러를 리포트합니다. Windows 이벤트 뷰어에서 이러한 에러를 볼 수 있습니다. Windows 제어판에서 관리 도구 디렉토리의 이벤트 뷰어를 탐색합니다. 관리 도구 디렉토리의 정확한 위치는 사용하는 Windows 버전에 따라 다릅니다.

데이터 통신 방법

공유 변수로 최근값 공개하기

공유 변수는 데이터를 읽고 쓸 수 있는 메모리 공간입니다. 단일 프로세스 공유 변수를 보유한 단일 컴퓨터나 네트워크 공유 변수를 보유한 여러 컴퓨터에서 공유 변수를 읽고 쓸 수 있으며, 이로써 NI Publish-Subscribe Protocol (NI-PSP)을 사용하여 네트워크로 데이터를 공개할 수 있습니다. 공유 변수를 사용하여 데이터 세트의 최근값만 하나 이상의 컴퓨터로 공개합니다.



노트

데이터 포인트를 한 컴퓨터에서 다른 컴퓨터로 전송하는데, National Instruments는 네트워크 스트림으로 계속 명령을 보내거나 데이터를 스트리밍하도록 권장합니다. 사용자 어플리케이션의 필요에 가장 적합한 네트워킹 기능을 선택하는 방법에 대한 더 자세한 정보는 네트워크를 통해 데이터 전송하기를 참조하십시오.

일반적으로, 최근값을 공유 변수로 공개하려면 다음 태스크를 완료해야 합니다:

1. 공유 변수를 생성합니다.
2. 공유 변수를 설정합니다.
3. 사용자 어플리케이션이 공유 변수를 읽고 쓸 수 있도록 프로그래밍합니다.
4. 네트워크에서 공유 변수의 이용이 가능하도록 합니다.

Real-Time Module 및 DSC Module로 공유 변수의 기능 확장하기

LabVIEW Real-Time Module 및 DSC Module은 공유 변수에 대한 추가 설정 및 프로그래밍 옵션을 제공합니다. 예를 들면, Real-Time Module로 공유 변수의 리얼타임 FIFO를 활성화하고 DSC Module로 공유 변수의 스케일링 및 알람 프로퍼티를 설정할 수 있습니다. 공유 변수의 기능 확장에 대한 더 자세한 정보는 Real-Time Module 및 DSC Module 도움말을 참조하십시오.

기타 변수 타입 사용하기

이 모음에서는 네트워크 공유 변수를 사용하여 네트워크를 통해 최근값을 공개하는 과정을 주로 설명하고 있습니다. 하지만, 다른 변수 타입으로 다음 태스크를 수행할 수도 있습니다:

- NI 스캔 엔진을 통한 I/O 데이터로의 단일 포인트 접근을 위해 I/O 변수를 사용합니다.
- 글로벌 변수를 사용하여 단일 컴퓨터에서 데이터를 읽고 씁니다.
- 로컬 변수를 사용하여 단일 VI에서 데이터를 읽고 씁니다.



관련 정보

[공유 변수와 기타 NI-PSP 데이터 아이템을 컨트롤 및 인디케이터에 연결하기](#) (586 페이지 참조)

[네트워크 스트림을 사용해서 어플리케이션 디자인하기](#) (615 페이지 참조)

[공유 변수 읽고 쓰는 방법 선택하기](#) (573 페이지 참조)

[공유 변수 생성하기](#) (573 페이지 참조)

[공유 변수 설정하기](#) (568 페이지 참조)

[공유 변수 노드를 사용하여 공유 변수를 읽기와 쓰기](#) (582 페이지 참조)

[네트워크에서 공유 변수의 이용 가능하게 하기](#) (601 페이지 참조)

[I/O 변수 사용하기 \(Real-Time, Windows\)](#) (1643 페이지 참조)

[Point-by-Point Analysis and Array-Based Analysis in](#) (1636 페이지 참조)

[스캔 I/O 데이터 접근하기 \(Real-Time, Windows\)](#) (1653 페이지 참조)

NI 스캔 엔진을 사용하여 스캔 I/O 데이터에 대한 단일 포인트에 접근합니다.

[글로벌 변수](#) (560 페이지 참조)

[로컬 변수](#) (560 페이지 참조)

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

공유 변수 설정하기

공유 변수 프로퍼티 대화 상자를 사용하여 **프로젝트 탐색기** 윈도우의 공유 변수를 설정합니다. 이 대화 상자를 디스플레이하려면, **새로운 공유 변수를 생성**하거나 프로젝트 탐색기 윈도우의 기존 공유 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택합니다.

다음 리스트는 일부 일반적인 공유 변수 설정 옵션의 기능 및 용도를 설명합니다:

- 이름 텍스트 박스에 공유 변수의 이름을 입력합니다.
- **데이터 타입** 풀다운 메뉴에서 공유 변수의 데이터 타입을 선택합니다. 사용하려는 데이터 타입이 없는 경우, 사용자 컨트롤로부터를 선택하여 **사용자 컨트롤**의 데이터 타입을 사용합니다.
- 변수를 읽고 쓸 컴퓨터의 네트워크 위치에 따라 생성하려는 공유 변수의 타입을 선택합니다. 변수 타입 풀다운 메뉴의 다음 옵션에서 선택할 수 있습니다:
 - 네트워크를 선택하여 같은 네트워크 상의 원격 컴퓨터와 타겟에 읽고 쓰려는 공유 변수를 생성합니다.
 - 단일 프로세스를 선택하여 하나의 컴퓨터에 읽고 쓰려는 공유 변수를 생성합니다.
- 변수 타입 리스트에서 네트워크 공개를 선택하는 경우, 네트워크 페이지의 버퍼링 사용 확인란에 확인 표시를 하여 공유 변수 데이터의 클라이언트 측 **버퍼링**을 활성화할 수 있습니다. 데이터 타입 풀다운 메뉴에서 선택한 데이터 타입에 따라 버퍼링 사용 섹션에 추가적인 값을 입력해야 할 수도 있습니다.



노트

이 옵션을 선택하면, 공유 변수가 데이터를 FIFO 버퍼에 저장합니다. 그러나 버퍼가 용량에 달하는 경우 여전히 새 값이 이전 값을 덮어쓸 수 있기 때문에 이 옵션이 손실 없는 데이터 통신을 보장하지는 않습니다. 사용자 어플리케이션에 손실 없는 통신이 필요한 경우, 네트워크 스트림을 사용하여 **어플리케이션 사이에서 지속적으로 데이터를 스트리밍**합니다. 공유 변수 버퍼링 동작에 대한 더 상세한 설명은 National Instruments 웹 사이트를 참조하십시오.

- 변수 타입 리스트에서 네트워크를 선택하는 경우, 네트워크 페이지의 단일 발신자 확인란에 확인 표시를 하면 한 번에 하나의 어플리케이션이 쓰도록 할 수 있습니다. 이 옵션을 선택하면, 공유 변수에 연결된 첫번째 어플리케이션이 해당 공유 변수에 값을 쓸 수 있고, 이에 반해 이후 어플리케이션은 값을 쓸 수 없습니다. 첫번째 어플리케이션이 연결을 해제할 때, 큐에 있는 다음 어플리케이션이 공유 변수에 값을 쓸 수 있습니다.



데이터 통신 방법

공유 변수에 스케일링 설정하기

스케일링은 값의 범위를 측정 단위에서 계산된 범위로 변환하는데 유용합니다. 각 데이터 타입의 **공유 변수**의 스케일링을 설정할 수 있습니다. 공유 변수 프로퍼티 대화 상자의 스케일링 페이지에서 스케일링을 설정할 수 있습니다.

(DSC Module) 또한 공유 변수 IO 프로퍼티를 사용하여 스케일링을 설정할 수 있습니다.

숫자형 공유 변수 스케일링하기

숫자형 공유 변수의 **선형 및 제곱근 스케일링**을 사용할 수 있습니다.

많은 경우 어플리케이션에서는 디바이스 서버에 사용되는 기본 데이터를 어플리케이션의 사용자에게 적합한 공학 단위의 형태로 만들기 위해 조작해야 할 필요가 있습니다. 공유 변수 프로퍼티 대화 상자의 스케일링 페이지에서 공유 변수에 대한 기본 범위와 공학 범위의 두 범위 사이의 간단한 변환을 수행할 수 있습니다. 기본 범위란 디바이스 서버가 사용하는 값을 의미합니다. 공학 범위란 **공유 변수 엔진**과 HMI(Human Machine Interface) 어플리케이션에서 사용되는 값을 말합니다.

선형 및 제곱근 스케일링

선형 스케일링은 다음 방정식으로 정의됩니다:

$$m * \text{기본} + b$$

여기서 b는 공학 최소 스케일, m은 공학 최대 스케일 - 공학 최소 스케일 / 기본 최대 스케일 - 기본 최소 스케일입니다.

제곱근 스케일링은 다음 방정식으로 정의됩니다:

$$b + m * \sqrt{\text{기본} - o}$$

여기서 b는 공학 최소 스케일, m은 공학 최대 스케일 - 공학 최소 스케일 / sqrt(기본 최대 스케일 - 기본 최소 스케일)이며, o는 기본 최소 스케일입니다.

다음 테이블은 0에서 100의 범위의 공유 변수를 0에서 10의 범위의 공학 단위로 스케일 하여 보여줍니다.

기본 단위	선형 스케일	제곱근 스케일
0	0	0
4	.4	2
9	.9	3
10	1	3.16
16	1.6	4
20	2	4.47
25	2.5	5
30	3	5.48

기본 단위	선형 스케일	제공된 스케일
36	3.6	6
40	4	6.32
49	4.9	7
50	5	7.07
60	6	7.75
64	6.4	8
70	7	8.37
80	8	8.94
81	8.1	9
90	9	9.49
100	10	10

다음 테이블은 0에서 100의 범위의 공유 변수를 15에서 30의 범위의 공학 단위로 스케일 하여 보여줍니다. 오프셋 스케일링의 결과는 보다 복잡합니다.

기본 단위	선형 스케일	제공된 스케일
0	15	15
4	15.60	18
10	16.50	19.74
16	17.40	21
20	18	21.71
30	19.50	23.22
36	20.40	24
40	21	24.49
50	22.50	25.61
60	24	26.62
64	24.60	27
70	25.50	27.55
80	27	28.42

기본 단위	선형 스케일	제곱근 스케일
90	28.5	29.23
100	30	30

선형 스케일링 예제

이 예제에서 디바이스 서버는 0~5V의 전압을 반환합니다. 전압은 위치 센서와 연결되어 있으며, 실제 위치는 0 V가 50 cm, 5 V가 100 cm로 맵핑되어 센티미터로 측정됩니다. 다음 테이블은 스케일링 후 공유 변수의 값을 보여줍니다.

기본 단위 (V)	선형 스케일 (cm)
0	50
1	60
2	70
3	80
4	90
5	100

제곱근 스케일링 예제

이 예에서는 유량 미터가 차동 압력을 사용하여 사용하여 액체의 유량을 측정합니다. 디바이스 서버는 4 ~ 20 mA 값을 제공합니다. 실제 유량은 분당 갤런 (GPM) 단위로 측정되며, 여기서 4 mA는 0 GPM, 20 mA는 100 GPM에 해당합니다. 다음 테이블은 스케일링 후 공유 변수의 값을 보여줍니다.

기본 단위 (mA)	제곱근 스케일 (GPM)
4	0
5	25
6	35.36
7	43.3
8	50
10	61.24
13	75
15	82.92

기본 단위 (mA)	제공된 스케일 (GPM)
20	100

(DSC Module) Scaling.UnitsSharedVariableIO 프로퍼티를 사용하여 공유 변수에 단위를 지정합니다. 위의 예에서, 분당 갤론을 Scaling.Units 프로퍼티의 상수로 지정할 수 있습니다.

불리언 공유 변수 스케일링하기

불리언 공유 변수의 **반전 스케일링**을 사용할 수 있습니다. 반전 스케일링을 활성화하면, LabVIEW는 서버와 통신할 때 불리언 값을 반전시킵니다.

U32 비트 필드 공유 변수 스케일링하기

U32 비트 필드 공유 변수에 대한 **마스크 반전 및 마스크 선택 스케일링**을 사용할 수 있습니다.

마스크 스케일링은 U32 비트 필드에만 적용됩니다. LabVIEW는 필요한 경우 비트 필드를 숫자로 또는 숫자를 32비트 필드로 변환하여 저장합니다. 비트 필드는 32개의 값의 집합을 유지하며, 각 값은 1 또는 0로 제한되어 있습니다. 마스크 스케일링을 사용하면, 주의를 기울여야 할 배열 멤버를 결정할 수 있습니다. 마스크로 필터된 모든 비트는 0으로 보입니다. 필터되지 않은 비트는 실제 값을 반영하여 0 또는 1로 보입니다. 다음의 마스크에서,

0000000000000000000011111111111111 0x0000FFFF

여기서 0은 필터링해야 할 위치를 표시하고 1을 리포트해야 할 위치를 표시하고, 비트 배열을

10110011100011001100111000110010

다음과 같이 읽습니다:

000000000000000000001100111000110010

 **노트**
반전 마스크는 마스크된 비트를 반전시키지 않습니다.

다음 테이블은 스케일링을 U32 비트 필드 스케일링으로 설정된 공유 변수의 예를 보여줍니다.

공유 변수 이름	기본값	마스크 반전 스케일	마스크 선택 스케일	스케일된 값
공유 변수 1	0x0000000F	0x00000000	0x000000FF	0x0000000F
공유 변수 2	0x0000000F	0x00000033	0x000000FF	0x0000003C

공유 변수 이름	기본값	마스크 반전 스케일	마스크 선택 스케일	스케일된 값
공유 변수 3	0x0000000F	0x00000033	0x0000000F	0x0000000C
공유 변수 4	0x0000000F	0x00000000	0x00000033	0x00000003
공유 변수 5	0x0000000F	0x00000033	0x00000033	0x00000030
공유 변수 6	0x000000FF0	0x0000000F	0x000000FF	0x000000FF

공유 변수 생성하기

다음 단계를 따라 **공유 변수**를 생성합니다.



노트

프로젝트를 열어야만 공유 변수를 생성할 수 있습니다.

1. **프로젝트 탐색기** 윈도우의 타겟, 프로젝트 라이브러리, 또는 프로젝트 라이브러리 내의 폴더에서 마우스 오른쪽 버튼을 클릭합니다.

노트 타겟에서 또는 프로젝트 라이브러리 내에 위치하지 않은 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기»변수를 선택할 경우, LabVIEW는 공유 변수를 새 프로젝트 라이브러리 내에 놓습니다.

2. 바로 가기 메뉴에서 새로 만들기»변수를 선택합니다. 공유 변수 프로퍼티 대화 상자가 열립니다.
3. 이 대화 상자를 사용하여 **공유 변수를 설정**합니다.
4. 확인 버튼을 클릭하여 공유 변수를 생성합니다.

공유 변수를 생성한 후, **사용자 어플리케이션 필요에 가장 적합하게 공유 변수를 읽고 쓰는 방법을 선택**합니다.



데이터 통신 방법

공유 변수 읽고 쓰는 방법 선택하기

공유 변수를 읽고 쓰는 방법을 선택할 때, 다음을 포함한 몇가지 요소를 고려해야 합니다:

- 업데이트 속도
- 프로그래밍 복잡도
- 확장성

다음 테이블에서는 네트워크를 통한 공유 변수의 읽기 및 쓰기 방법의 용도, 장점, 단점을 설명합니다.



노트
어플리케이션이 다음 중 하나 이상의 기능으로 공유 변수를 읽고 쓰도록 프로그래밍한 후, 네트워크에서 공유 변수의 이용이 가능하도록 하려면 **공유 변수를 배포**해야 합니다.

공유 변수 읽고 쓰는 방법	권장 사용	장점	단점
프로그램적 으로 공유 변수 사용	다수의 공유 변수 읽고 쓰기. 예를 들어, 특정 프로젝트 라이브러리 에서 배정도 데이터 타입의 공유 변수 50개를 읽어야 할 경우, 해당 공유 변수를 프로그램적으로 읽는 것이 가장 효율적인 방법입니다.	- 하나의 공유 변수 함수가 여러 공유 변수에 접근할 수 있습니다. 다른 방법은 한 번에 하나의 공유 변수만 읽고 쓸 수 있습니다. - 확장 가능한 한 프로젝트에 50개의 공유 변수를 추가할 경우, 블록 다이어그램을 업데이트 할 필요 없이 각 변수를 읽고 쓸 수 있습니다. I/O 변수에 접근 가능.	어플리케이션이 특정 기준을 충족하는 공유 변수를 읽고 쓰도록 프로그래밍해야 하기 때문에 다른 방법보다 더 많은 프로그래밍이 필요.

공유 변수 읽고 쓰는 방법	권장 사용	장점	단점
공유 변수 노드 사용	- 소수의 공유 변수 읽고 쓰기. 예를 들어, 10개 공유 변수의 값을 업데이트할 필요가 있을 경우, 공유 변수 노드를 사용할 수 있습니다. - HMI(Human-Machine Interface) 어플리케이션에서 빠른 속도로 데이터 읽고 쓰기. 예를 들어, 초당 한 번 온도 센서의 최근값을 디스플레이할 필요가 있을 경우, 공유 변수 노드를 사용합니다. - I/O 변수에 접근 가능.	- 프로그래밍의 편의성. 공유 변수 노드를 어플리케이션에 추가하려면, 프로젝트 탐색기 윈도우에서 사용자 어플리케이션의 블록다이어그램으로 공유 변수를 끌어옵니다. - LabVIEW는 대응하는 공유 변수 이름으로 각 공유 변수 노드에 라벨을 붙입니다.	- 각 공유 변수 노드는 하나의 공유 변수에만 접근할 수 있습니다. - 확장 불가능. 추가적인 공유 변수를 읽고 쓰려 할 때마다 공유 변수 노드를 블록다이어그램에 추가해야 합니다.

공유 변수 읽고 쓰는 방법	권장 사용	장점	단점
컨트롤과 인디케이터 사용	HMI 어플리케이션에서 느린 속도로 데이터 읽고 쓰기. 예를 들어, 몇 초마다 한 번 온도 센서의 최근값을 디스플레이할 필요가 있을 경우, 인디케이터로 공유 변수를 읽습니다.	프로그래밍의 편의성. 컨트롤이나 인디케이터로 공유 변수를 읽고 쓰려면, 프로젝트 탐색기 윈도우에서 사용자 어플리케이션의 프런트패널로 공유 변수를 끌어옵니다.	• 데이터를 강제하 지 않음. 그러므 로 읽고 쓰는 데이터 로 수행할 수 있는 작업이 제한됩 니다. • 어떤 컨트롤 및 인디케 이터가 어떤 공유 변수에 접근하 는지 나타나 지 않음. • 공유 변수 노드를 사용하 는 것보다 더 느린 처리 속도. • I/O 변수에 대한 접근 불가.

공유 변수 읽고 쓰는 방법	권장 사용	장점	단점
기존 데이터 소스 사용	공유 변수 기능을 지원하지 않는 데이터 소스에 대해 공유 변수 기능 이용하기. 예를 들어, 리얼타임 타겟은 DSC Module 로깅 및 알람 프로퍼티를 지원하지 않습니다. 그러므로 리얼타임 타겟의 공유 변수에 대해 DSC Module 로깅 및 알람 프로퍼티를 설정할 필요가 있을 경우, 사용자 개발 컴퓨터에서 새로운 공유 변수를 생성하고, CompactRIO 타겟의 공유 변수로 해당 공유 변수를 쓰고, 로깅 및 알람이 되도록 개발 컴퓨터의 공유 변수를 설정할 수 있습니다.	- I/O 변수에 접근 가능. - 공유 변수에 설정 옵션 추가.	데이터 소스에 대해 공유 변수 기능을 이용할 필요가 없거나 데이터 소스가 이미 필요한 기능을 지원할 경우, 데이터 소스를 직접 읽고 쓸 수 있기 때문에 이 방법은 비효율적입니다.
DataSocket 함수 사용	Mac 및 Linux 컴퓨터에서 공유 변수 읽고 쓰기.	DataSocket 함수는 Mac 및 Linux 컴퓨터에서 공유 변수를 읽고 쓰는데 사용할 수 있는 유일한 프로그래밍 도구입니다.	- 공유 변수를 읽고 쓰는 다른 방법보다 더 느린 처리 속도. - I/O 변수에 대한 접근 불가.
변수 웹 서비스를 사용하여	HTTP를 지원하는 웹 활성화된 플랫폼에서 공유 변수 읽고 쓰기	- 변수 웹 서비스는 HTTP를 지원하는 여러 웹 활성화된 플랫폼에서 공유 변수를 읽고 쓰는데 사용할 수 있는 유일한 프로그래밍 도구입니다. - I/O 변수에 접근 가능.	HTTP를 지원하는 웹 활성화된 플랫폼에서 공유 변수 기능을 이용할 필요가 없는 경우, 이 메소드는 비효율적이며 다른 방법에 비해 더 많은 프로그래밍을 요구합니다.

데이터 통신 방법 홈

공유 변수를 프로그램적으로 읽기와 쓰기

다수의 공유 변수에 접근하는 어플리케이션이 있는 경우, 공유 변수 함수를 사용하여 프로그램적으로 해당 공유 변수를 읽고 쓸 수 있습니다. 각 공유 변수 노드가 하나의 공유 변수에만 접근할 수 있는 반면, 하나의 공유 변수 함수로 여러 공유 변수에 접근할 수 있기 때문에, **공유 변수 노드를 사용**하는 것보다 이러한 프로그래밍 기술이 종종 더 깔끔하고 확장 가능합니다.

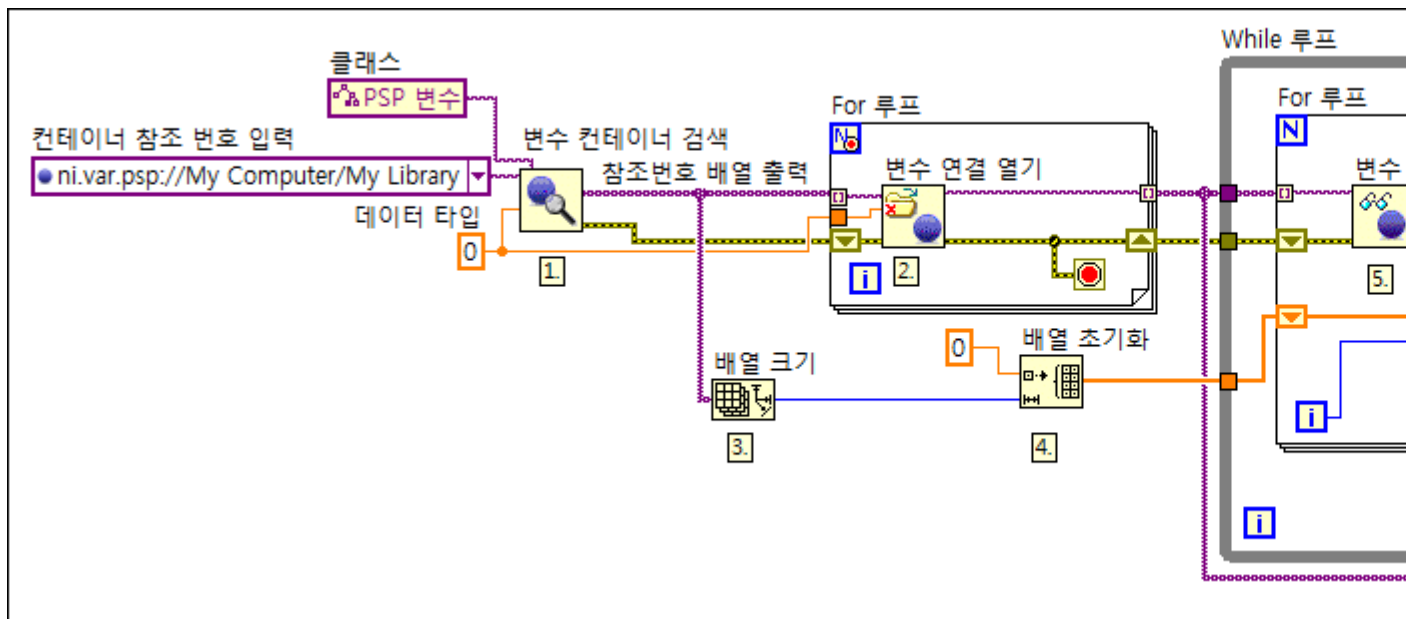


노트

다음 타입의 공유 변수에 프로그램적으로 접근할 수 있습니다:

- PSP 변수 — **NI Publish-Subscribe Protocol (NI-PSP)**을 사용하여 네트워크에 공개된 데이터 아이템. 네트워크 공유 변수, 네트워크 공개가 활성화된 I/O 변수, NI-DAQmx I/O 서버를 통해 공개된 DAQ 채널이 PSP 변수의 일반적인 예에 포함됩니다.
- I/O 변수—I/O 데이터에 대한 단일 포인트 접근을 위해 **NI 스캔 엔진**을 사용하는 공유 변수.

다음 그림은 공유 변수를 프로그램적으로 읽고, 해당 값을 미리 할당된 배열로 놓고, 해당 값을 처리하는 예를 보여줍니다.



다음과 같은 이벤트가 위의 그림에서 발생합니다:

1. [변수 컨테이너 검색] 함수는 My Library 아래에서 모든 배정도 데이터 타입 공유 변수를 찾고, 참조 번호 배열 출력은 해당 공유 변수에 대한 참조의 배열을 반환합니다.
2. [변수 연결 열기] 함수는 변수 식별자 URL의 배열을 사용하여 각 공유 변수에 연결합니다. 에러가 발생하면 For 루프는 멈춥니다.
3. [배열 크기] 함수는 공유 변수 참조의 배열을 사용하여 [변수 컨테이너 검색] 함수가 찾은 공유 변수의 개수를 계산합니다.

4. [배열 초기화] 함수는 공유 변수의 개수를 사용하여 공유 변수 데이터에 대한 배열을 생성합니다. While 루프 밖에서 공유 변수 데이터에 대한 배열을 초기화함으로써, 이 어플리케이션은 CPU 사용을 최적화합니다.
5. [변수 읽기] 함수는 [변수 연결 열기] 함수가 열었던 각각의 공유 변수를 읽습니다.
6. [배열 부분 대체] 함수는 [배열 초기화] 함수가 생성한 배열에 공유 변수 데이터를 저장합니다.
7. Process Data subVI는 공유 변수 데이터의 배열을 처리합니다.
8. 에러가 발생할 때까지 또는 사용자가 정지 버튼을 클릭할 때까지 읽기 및 처리 작업은 While 루프에서 지속적으로 실행됩니다.
9. [변수 연결 닫기] 함수는 공유 변수 연결을 끊습니다.

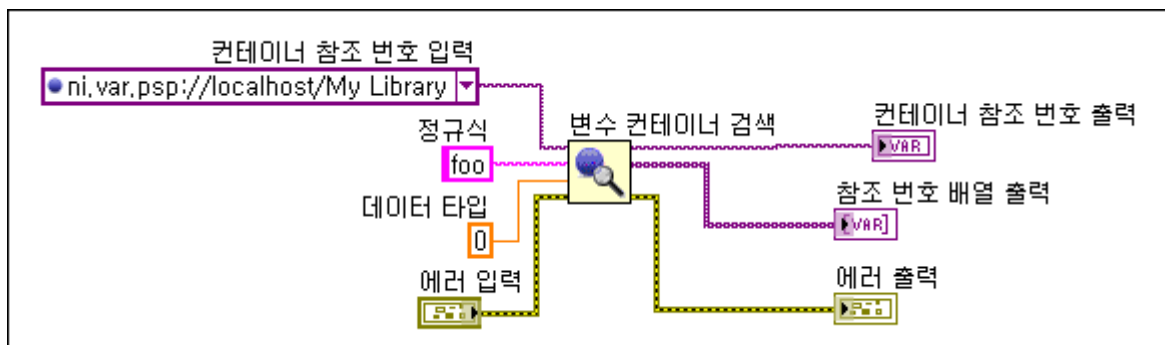
이 어플리케이션은 공유 변수 노드를 사용하는 것과 동일한 기능을 달성합니다. 그러나 이 어플리케이션은 프로그램적으로 공유 변수에 접근하기 때문에, 블록다이어그램을 업데이트하지 않고도 수백 또는 수천 개의 변수로 확장할 수 있습니다. 예를 들어, My Library에 변수1부터 변수10까지, 10개의 변수가 포함된다고 가정합니다. 일괄 변수 생성 대화 상자를 사용하여 My Library에 변수11부터 변수500까지 추가할 수 있습니다. My Library를 재배포한 후, 다음에 VI를 실행하면, 변수11부터 변수500까지 블록다이어그램에 추가할 필요 없이 변수1부터 변수500까지 읽습니다.

다음 섹션에서는 위 예의 프로그래밍 기술에 대한 정보가 제공됩니다.

공유 변수 찾기

[변수 컨테이너 검색] 함수를 사용하여 특정한 기준을 충족하는 공유 변수와 변수 컨테이너를 검색합니다. 예를 들어, 다음 블록다이어그램에 보이는 VI는 이 함수를 사용하여 주어진 컨테이너에서 다음의 기준을 만족시키는 공유 변수를 찾습니다.

- 로컬 컴퓨터의 My Library에 존재
- 배정도 데이터 타입
- **변수 식별자 URL**은 문자열 foo 포함



노트 변수 객체 탐색 대화 상자를 사용하여 편집 시에 공유 네트워크나 현재 프로젝트에서 변수와 변수 컨테이너를 탐색할 수 있습니다. 위 그림의 컨테이너 참조 번호 입력 상수와 같은 변수 컨트롤이나 상수에서 드롭다운 화살표를 클릭한 후 탐색을 선택하여 변수 객체 탐색 대화 상자를 디스플레이합니다.

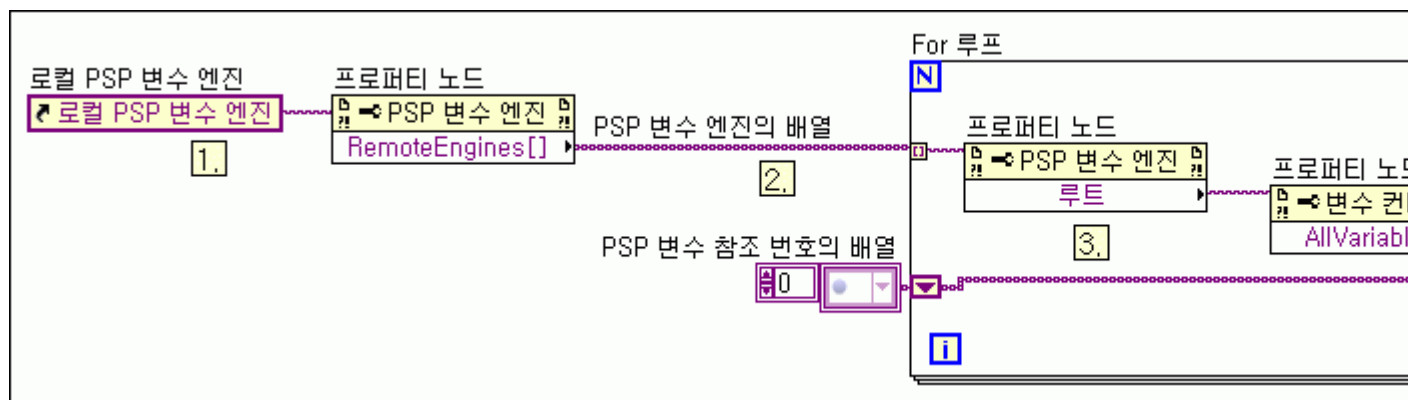


주의 프로그램식으로 찾은 공유 변수를 읽고 쓸 때 잘못된 데이터 타입의 데이터를 읽거나 쓸 수 있습니다. 변수 컨테이너 검색 함수의 데이터 타입 입력을 사용해서 잘못된 데이터 타입의 공유 변수를 읽거나 쓰는 것을 방지합니다.

또한 다음 변수 컨테이너 프로퍼티와 메소드를 사용하여 프로그램적으로 공유 변수를 찾을 수 있습니다.

- 모든 변수 프로퍼티
- 직하위 프로퍼티
- 하위 찾기 메소드

다음 그림의 VI는 모든 변수 프로퍼티를 사용하여 로컬 네트워크에 호스트된 모든 PSP 변수를 찾습니다.



다음과 같은 이벤트가 위의 그림에서 발생합니다:

1. 로컬 변수 객체 참조 상수는 로컬 PSP 변수 엔진에 대한 참조를 원격 엔진 프로퍼티로 전달합니다.
2. 원격 엔진 프로퍼티는 네트워크의 다른 컴퓨터에 호스트된 PSP 변수 엔진의 배열을 반환합니다. 그런다음 이 VI가 For 루프에서 각 PSP 변수 엔진을 반복합니다.
3. 루트 프로퍼티는 각 원격 컴퓨터에서 루트 PSP 변수 컨테이너에 대한 참조를 얻습니다.
4. 모든 변수 프로퍼티는 각 원격 컴퓨터에 호스트된 모든 변수에 대한 참조의 배열을 얻습니다.
5. VI는 네트워크에서 발견된 모든 변수를 모든 PSP 변수 배열에 추가합니다.



팁 앞의 블록다이어그램을 만드는 출발점으로, 변수 객체 클래스에서 미리 설정한 변수 프로퍼티 노드를 사용할 수 있습니다.

공유 변수 연결 열기 및 닫기

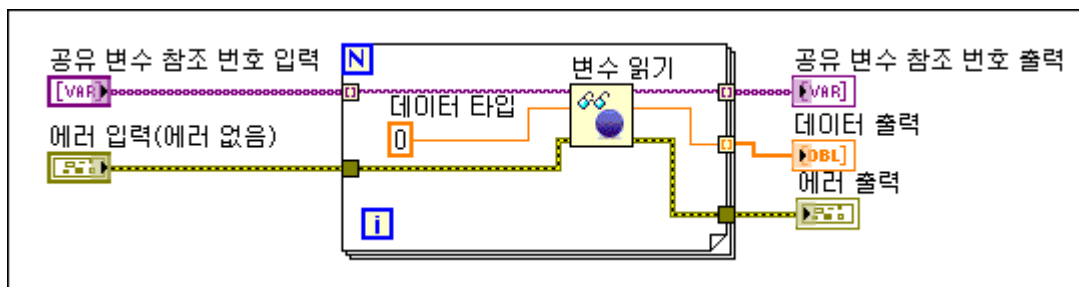
다음 블록다이어그램에서 보이는 것처럼, [변수 연결 열기] 함수를 사용하여 프로그램적으로 변수에 대한 연결을 열 수 있습니다.



어플리케이션이 공유 변수에 대한 접근을 마치면, For 루프에서 [변수 연결 닫기] 함수를 사용하여 모든 변수 참조를 닫습니다. 이 함수는 각 변수 연결에 할당되었던 메모리 리소스를 해제합니다.

공유 변수 읽기와 쓰기

[변수 읽기] 함수를 사용하여 프로그램적으로 실시간 변수 값을 데이터를 읽습니다. 편집 시에 특정한 공유 변수로 연결해야 하는 공유 변수 노드와 달리, [변수 읽기] 함수는 공유 변수 참조 번호 입력 입력을 포함하며, 이 입력을 사용하여 실행 시에 읽을 공유 변수를 지정할 수 있습니다. 예를 들어, 다음 블록 다이어그램에서처럼 For 루프안에 변수 읽기 함수를 사용하여 참조 번호 배열이 지정한 여러 공유 변수를 읽을 수 있습니다.



[변수 쓰기] 함수를 사용하여 프로그램적으로 실시간 변수 값을 쓸 수 있습니다. 예를 들어, For 루프내에 [변수 쓰기] 함수를 사용하여 참조 번호 배열이 지정한 여러 공유 변수에 쓸 수 있습니다.



팁 배열로 I/O 변수 컨테이너를 읽고 쓰면 성능을 최적화하고, 블록 다이어그램을 간소화할 수 있습니다.



노트 공유 변수에 8 킬로바이트 미만의 데이터를 쓰는 경우, LabVIEW가 이 데이터를 네트워크를 통해 전송하는데 소요되는 시간은 최대 10밀리초입니다. 이러한 지연을 없애려면 [공유 변수 데이터 비우기] VI를 사용하여 공유 변수 데이터를 즉시 전송합니다.

데이터 통신 방법 홈

공유 변수 노드를 사용하여 공유 변수를 읽기와 쓰기

(Real-Time, Windows) 공유 변수 노드는 대응하는 **공유 변수**를 가리키는 블록다이어그램 객체입니다. 공유 변수 노드를 사용하여 공유 변수의 값을 읽고 씁니다.



노트

- 어플리케이션이 다수의 공유 변수에 접근하는 경우, **프로그램적으로 공유 변수에 접근**하여 보다 깔끔하고 확장 가능한 블록다이어그램을 생성합니다.
- (Real-Time, Windows) 프로젝트에 공유 변수 노드가 있고 프로젝트 탐색기 윈도우에 대응하는 공유 변수가 없는 VI를 열면 공유 변수 노드가 깨집니다. 찾을 수 없는 공유 변수와 관련된 모든 프런트패널 컨트롤도 깨집니다. 이런 동작은 Windows에서만 나타나며 프로젝트의 VI를 열 때에만 발생합니다. VI를 주요 어플리케이션 인스턴스에서 여는 경우, LabVIEW는 찾을 수 없는 공유 변수에 대해 알려주지 않습니다.

블록다이어그램에 공유 변수 노드 추가하기

다음과 같은 방법으로 공유 변수 노드를 블록다이어그램에 추가할 수 있습니다:

- 프로젝트 탐색기 윈도우에서 같은 프로젝트에 있는 VI의 블록다이어그램으로 공유 변수를 끌어옵니다.
- 공유 변수 팔레트에서 공유 변수 노드를 선택, 블록다이어그램에 추가합니다. 그리고 나서 다음 단계를 따라 해당 공유 변수 노드가 프로젝트 탐색기 윈도우에서 대응하는 공유 변수를 가리키도록 해야 합니다.
 - 공유 변수 노드에서 마우스 오른쪽 버튼을 클릭하여 변수 탐색 대화 상자를 디스플레이합니다.
 - 프로젝트/네트워크 트리 리스트에서 공유 변수를 탐색합니다.
 - 확인 버튼을 클릭합니다.
- (Real-Time, FPGA, DSC Module) 컨트롤, 인디케이터 또는 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 공유 변수 노드로 변경을 선택합니다. 또한, 출력 터미널에서 마우스 오른쪽 버튼을 클릭하면 바로 가기 메뉴에서 생성 >> 공유 변수 노드를 선택할 수 있습니다.

공유 변수 노드가 데이터를 읽을지 또는 쓸지 지정하기

다음 단계를 따라 공유 변수 노드가 프로젝트 탐색기 윈도우의 대응하는 공유 변수로 데이터를 읽을지 또는 쓸지 지정합니다.

- 공유 변수 노드에서 마우스 오른쪽 버튼을 클릭합니다.
- 바로 가기 메뉴의 다음 옵션 중 하나를 선택합니다:
 - 접근 모드 >> 읽기를 선택하여 공유 변수 노드로 공유 변수의 데이터를 읽습니다. 이 옵션을 선택하면, 다음 그림에서 보이는 것처럼, 출력이 공유 변수 노드의 오른쪽에 나타납니다.



- 접근 모드 >> 쓰기를 선택하여 공유 변수 노드로 공유 변수에 데이터를 씁니다. 이 옵션을 선택하면, 다음 그림에서 보이는 것처럼, 입력이 공유 변수 노드의 왼쪽에 나타납니다.



노트 공유 변수에 8 킬로바이트 미만의 데이터를 쓰는 경우, LabVIEW가 이 데이터를 네트워크를 통해 전송하는데 소요되는 시간은 최대 10밀리초입니다. 이러한 지연을 없애려면 [공유 변수 데이터 비우기] VI를 사용하여 **공유 변수 데이터를 즉시 전송**합니다.

마지막 읽기 이후 공유 변수 값이 업데이트되었는지 확인하기

다음 단계를 따라 공유 변수 노드의 타임스탬프 인디케이터를 사용하여 공유 변수 값이 마지막 읽기 이후 업데이트되었는지 확인합니다.

	노트 - LabVIEW는 기본적으로 타임스탬프가 되도록 네트워크 공유 변수를 설정합니다. 단일 프로세스 공유 변수의 타임스탬프 값을 기록하려면, 우선 공유 변수 프로퍼티 대화 상자의 변수 페이지에 있는 타임스탬프 활성화 확인란에 확인 표시를 합니다. - 타임스탬프를 활성화하면 추가의 메모리와 CPU 오버헤드가 요구되어 어플리케이션의 정확성에 영향을 줄 수 있기 때문에 LabVIEW의 I/O 변수 노드에서 타임스탬프는 기본 설정으로 비활성화되어 있습니다. 타임스탬프를 사용하려는 경우에만 I/O 변수 노드에 대한 타임스탬프를 활성화합니다.

- 공유 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타임스탬프 >> 보이기를 선택합니다. 타임스탬프 출력이 다음 그림에서 보이는 것처럼 빨간색 모래시계로 공유 변수 노드의 오른쪽에 나타납니다.



- 타임스탬프 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 인디케이터를 선택하여 타임스탬프 데이터를 디스플레이하는 프런트패널 인디케이터를 생성합니다.

공유 변수 노드가 타임아웃되었는지 확인하기

다음 단계를 따라 공유 변수 노드의 타임아웃 기간을 지정하고 공유 변수 노드가 타임아웃되었는지 확인합니다.

	노트 데이터를 읽도록 설정된 공유 변수 노드에 대해서만 타임아웃 컨트롤을 활성화할 수 있습니다.

1. 블록다이어그램의 공유 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타임아웃 >> 보이기를 선택합니다.
2. 다음 그림의 왼쪽에 파란색 시계로 보이는 ms 타임아웃 컨트롤을 사용하여 타임아웃을 밀리초 단위로 지정합니다. 다음 그림의 오른쪽에 빨간색 시계로 보이는 타임아웃? 인디케이터를 사용하여 공유 변수 노드가 지정된 기간에 새 데이터를 읽는지 아니면 타임아웃되었는지 확인합니다.



공유 변수 노드를 다른 타겟으로 옮길 때 공유 변수 노드가 깨질지 지정하기

(Real-Time, Windows) 공유 변수 노드를 공유 변수에 연결하는 방법을 지정하려면, 노드를 절대적 또는 특정 타겟으로 설정합니다.

절대 공유 변수 노드는 항상 공유 변수를 생성한 타겟의 공유 변수에 연결됩니다. 절대 공유 변수 노드가 기본입니다. VI에 절대 공유 변수 노드가 포함되어 있고 VI를 새 타겟에서 여는 경우, 노드는 공유 변수를 생성한 타겟의 공유 변수로 연결을 시도하기 때문에 VI가 깨집니다. VI와 변수를 다른 타겟으로 옮기지 않으려면, 절대 공유 변수 노드를 사용합니다.

특정 타겟 공유 변수 노드는 항상 공유 변수 노드가 포함된 VI가 실행되는 타겟의 공유 변수에 연결됩니다. 특정 타겟 공유 변수 노드가 포함된 VI를 새로운 타겟으로 옮기는 경우, 특정 타겟 공유 변수 노드는 로컬 호스트의 공유 변수에만 연결되기 때문에 공유 변수도 새로운 타겟으로 옮겨야 합니다. 로컬 호스트의 공유 변수에 접근 중이고 VI와 변수를 다른 타겟으로 옮기려면, 특정 타겟 공유 변수 노드를 사용합니다.

절대 공유 변수 노드를 특정 타겟으로 변경하려면, 블록다이어그램의 공유 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 참조 모드 >> 특정 타겟을 선택합니다. 특정 타겟 공유 변수 노드는 노드에 \ 문양이 나타납니다. 다음 이미지는 특정 타겟 공유 변수 노드를 나타냅니다.



특정 타겟 공유 변수 노드를 절대적으로 변경하려면, 공유 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 참조 모드 >> 절대를 선택합니다. 다음 이미지는 절대 공유 변수 노드를 나타냅니다.



데이터 통신 방법 홈

변수 웹 서비스를 사용하여 공유 변수 읽기와 쓰기

변수 웹 서비스를 사용하여 HTTP를 통해 네트워크에 공개된 공유 변수에 접근하는 어플리케이션을 생성할 수 있습니다. 변수 웹 서비스를 통해 [NI Publish-Subscribe](#)

Protocol(NI-PSP)을 사용하는 모든 공유 변수에 접근할 수 있습니다. 변수 웹 서비스는 Open Data Protocol(OData)을 사용하여 XML 및 JSON 포맷으로 응답합니다.



노트

변수 웹 서비스는 다음 OData 파라미터를 지원하지 않습니다: \$filter, \$select, \$top, \$skip, \$links, \$orderby, \$inlinecount, \$batch.

Real-Time 타겟에 변수 웹 서비스 설치하기

RT 타겟에서 변수 웹 서비스를 사용하려면, NI Measurement & Automation Explorer (MAX)에서 LabVIEW Real-Time 소프트웨어 마법사를 사용하여 RT 타겟에 변수 웹 서비스 12.0.0을 설치하십시오.

사용자에게 새 권한 부여하기

변수 웹 서비스를 사용하려면, NI 웹 기반 설정 및 모니터링에서 적절한 사용자에게 적절한 권한을 부여해야 합니다. 변수 웹 서비스에는 다음 같은 권한이 있습니다:

- VariableBrowse
- VariableRead
- VariableWrite
- VariableCreateDeleteGroup
- VariableReadGroup

이 권한은 관리자에게만 부여되도록 기본 설정되어 있습니다. 사용자에게 권한을 부여하는 것에 대한 정보는 NI 웹 기반 설정 및 모니터링을 **시작**하고 웹 서버 설정 페이지에서 도움말을 클릭하십시오.

사용자를 인증하고 이러한 권한을 사용하려면 NI 인증에 로그인해야 합니다. 또한, 관리자는 everyone 그룹에 5개의 권한을 모두 부여하여 인증을 비활성화할 수도 있습니다.



데이터 통신 방법

컨트롤 및 인디케이터로 공유 변수 읽고 쓰기

해당 공유 변수를 컨트롤 및 인디케이터에 연결하여 HMI(Human-Machine Interface) 어플리케이션에서 **공유 변수** 및 기타 NI-PSP 데이터 아이템을 읽고 씁니다. **공유 변수를 프론트패널 컨트롤이나 인디케이터에 연결**하면 해당 컨트롤이나 인디케이터는 직접 공유 변수에 값을 쓰거나 공유 변수의 값을 읽습니다.



노트

프론트패널 객체는 네트워크 공유 변수에만 연결할 수 있습니다.

예를 들어, 보일러 밸브의 온도를 모니터할 필요가 있을 경우, 컨트롤 및 인디케이터에 공유 변수를 연결하여 최근 온도값을 제한된 범위로 다음과 같이 디스플레이하고 처리합니다:

- 공유 변수의 값을 봅니다. 예를 들어, 프런트패널 인디케이터를 하나의 온도 변수에 연결할 경우, 인디케이터는 해당 변수의 최근값을 디스플레이합니다.
- 한 컴퓨터의 VI로부터 오는 데이터를 다른 컴퓨터의 VI에 있는 컨트롤이나 인디케이터로 디스플레이하거나 사용합니다. 예를 들어, 다른 컴퓨터의 VI가 평균 밸브 온도를 계산하여 해당 데이터를 공유 변수에 쓰는 경우, 프런트패널 인디케이터를 해당하는 동일한 해당 공유 변수에 연결하여 사용자 컴퓨터의 VI에서 동일한 데이터를 디스플레이할 수 있습니다. 또는 사용자 VI의 컨트롤을 공유 변수에 연결하고 해당 컨트롤을 subVI나 함수에 연결하여 데이터를 사용할 수 있습니다.
- 여러 컴퓨터의 컨트롤을 조작하고 읽습니다. 예를 들어, 여러 컴퓨터의 VI 정지 버튼을 동일한 공유 변수에 연결할하는 경우, 각각의 해당 컨트롤이 온도 센서를 끌 수 있습니다.



노트 Windows에서는 opc, ftp, http **프로토콜**을 사용하고, 모든 LabVIEW 지원 플랫폼에서는 dstp, file **프로토콜**을 사용하여 **프런트패널 객체를 DataSocket 서버 아이템에 연결**할 수 있습니다. 그러나 프런트패널 데이터 연결을 사용하여 **공유 변수 엔진(SVE)**을 통해 원격으로 프런트패널 객체를 컨트롤하면, 공유 변수 및 **NI-PSP** 데이터 아이템으로의 연결뿐만 아니라 더 빠르고 신뢰성 높은 데이터 전송이 이루어집니다.



데이터 통신 방법 홈

공유 변수와 기타 NI-PSP 데이터 아이템을 컨트롤 및 인디케이터에 연결하기

컨트롤 및 인디케이터로 데이터를 읽고 쓰려면 프로젝트 탐색기 윈도우에서 VI의 프런트패널로 공유 변수를 끌어 옮겨서 공유 변수에 연결된 컨트롤을 생성합니다. 또한, 다음 단계를 완료하여 더 많은 설정 옵션을 지정하거나 별도의 프로젝트에서 공유 변수를 읽고 쓸 수 있습니다.

- 프런트패널의 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티 대화 상자를 디스플레이합니다.
- 데이터 연결 페이지를 탐색합니다.
- 데이터 연결 선택 리스트에서 **공유 변수 엔진(NI-PSP)**을 선택합니다.
- 프런트패널 객체를 연결하려는 공유 변수나 **NI-PSP** 데이터 아이템을 선택합니다.
 - 탐색 버튼을 클릭하여 소스 아이템 선택 대화 상자를 디스플레이합니다.
 - 네트워크 소스 풀다운 메뉴의 다음 옵션에서 선택합니다:
 - 프로젝트 아이템을 선택하여 프런트패널 객체를 현재 프로젝트의 공유 변수에 연결합니다.

- 네트워크 아이템을 선택하여 프런트패널 객체를 다른 프로젝트의 공유 변수나 기타 NI-PSP 데이터 아이템에 연결합니다.



노트

공유 변수 또는 데이터 아이템을 이동시키거나 네트워크에서 이용할 수 없게 되면, 사용자 VI는 깨집니다.

3. 공유 변수나 NI-PSP 데이터 아이템을 선택한 후 확인 버튼을 클릭합니다.
5. 데이터 연결 페이지의 접근 타입 풀다운 메뉴 아래에서, 다음 옵션을 선택하여 프런트패널 객체가 공유 변수나 데이터 아이템과 상호작용하는 방법을 지정합니다:
 - 읽기 전용을 선택하여 공유 변수나 데이터 아이템에서 읽습니다.
 - 쓰기 전용을 선택하여 공유 변수나 데이터 아이템에 씁니다.
 - 읽기/쓰기를 선택하여 공유 변수나 데이터 아이템에서 읽고 씁니다.
6. 확실한 연결 방법에 대한 정보는 [프런트패널 연결의 문제 해결](#)을 참조하십시오.

변수 식별자 URL

LabVIEW는 변수 식별자 URL로 각 공유 변수를 식별합니다. [프로그램적으로 공유 변수를 읽고 쓰기](#) 위해, 공유 변수 함수의 공유 변수 참조 번호 입력 입력으로 해당 공유 변수의 변수 식별자 URL을 지정할 수 있습니다.



노트

또한, [프로그램적으로 공유 변수를 검색하여](#) 비교적 적은 LabVIEW 코드로 여러 공유 변수를 읽고 쓸 수 있습니다.

공유 변수 구조

다음 리스트는 공유 변수 및 변수 식별자 URL과 관련된 LabVIEW 구조 세트를 요약합니다.

- 공유 변수—데이터를 읽고 쓸 수 있는 메모리 공간.
- PSP 변수—[NI Publish-Subscribe Protocol\(NI-PSP\)](#)을 사용하는 네트워크에 공개된 데이터 아이템. 네트워크 공유 변수, 네트워크 공개가 활성화된 I/O 변수, NI-DAQmx I/O 서버를 통해 공개된 DAQ 채널이 PSP 변수의 일반적인 예에 포함됩니다.
- I/O 변수—I/O 데이터에 대한 단일 포인트 접근을 위해 [NI 스캔 엔진](#)을 사용하는 공유 변수.
- 변수 컨테이너—변수의 세트. 가장 일반적인 변수 컨테이너 타입은 [프로젝트 라이브러리](#)입니다. 그러나, I/O 모듈, 무선 센서 네트워크 (WSN) 노드, EtherCAT 디바이스와 같은 다른 아이템도 변수 컨테이너로 동작할 수 있습니다. 변수 객체 탐색 대화 상자를 사용하여 현재 네트워크에 배포된 모든 변수 컨테이너를 찾을 수 있습니다.
- 변수 엔진—변수 컨테이너와 변수의 계층구조를 관리하는 런타임 구성 요소.
- 프로젝트 라이브러리—다른 프로젝트 라이브러리를 포함하여, VI, 타입 정의, 공유 변수, 팔레트 파일, 기타 파일의 집합. 공유 변수를 포함하는 프로젝트 라이브러리는 변수 컨테이너입니다. 그러나 모든 변수 컨테이너가 프로젝트 라이브러리는 아닙니다.

- 타겟 이름—LabVIEW 프로젝트에서 내 컴퓨터 또는 RT 타겟과 같은 타겟 아이템에 할당된 이름.

노트 (Real-Time Module) RT 타겟 프로퍼티 대화 상자의 일반 프로퍼티 페이지에서 이름 필드를 사용하여 RT 타겟 이름을 지정할 수 있습니다.

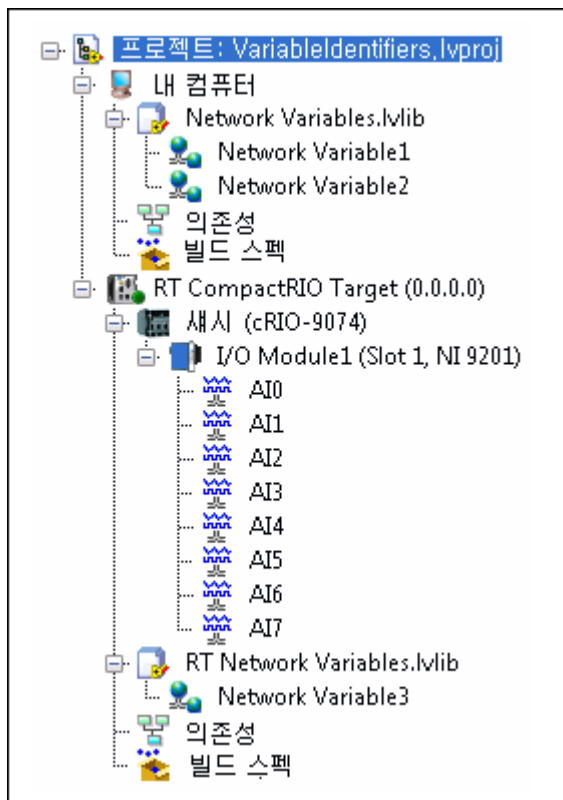
변수 식별자 URL 구문

변수 식별자의 URL 구문은 다음과 같습니다:

[변수 엔진]://[호스트 이름]/[컨테이너 이름]/[변수 이름]

모든 변수 식별자 URL의 컨테이너 이름 및 변수 이름 구성요소를 지정해야 합니다. 그러나 클래스가 포함하는 경우, 변수 엔진을 생략할 수 있습니다. 또한 VI가 실행되는 컴퓨터와 같은 컴퓨터에서 변수가 호스트되는 경우에도 호스트 이름을 생략할 수 있습니다.

예를 들어, 다음의 LabVIEW 프로젝트는 네트워크 공유 변수(Network Variables.lvlib 컨테이너 아래에 리스팅됨) 및 I/O 변수(I/O Module1 컨테이너 아래에 리스팅됨)를 포함합니다.



다음 테이블에서는 이 프로젝트의 여러 변수에 대한 전체 변수 식별자 URL을 보여줍니다.

변수 이름	변수 타입	변수 식별자 URL
Network Variable1	네트워크 공유 변수	<code>ni.var.psp://My Computer/Network Variables/Network Variable1</code>
AI0	I/O 변수	<code>ni.var.io://RT CompactRIO Target/I%2F0 Module1/AI0</code>
Network Variable3	네트워크 공유 변수	<code>ni.var.psp://RT CompactRIO Target/RT Network Variables/Network Variable3</code>

변수 식별자 URL은 대소문자를 구분하지 않습니다. 그러나 분석 에러가 발생하지 않게 하려면 예약된 문자를 해당하는 **이스케이프 코드**로 대체해야 합니다. 예를 들어, / 문자를 이스케이프 코드 %2F로 대체해야 합니다.

URL의 변수 엔진 구성 요소는 읽거나 쓰려는 변수 타입에 따라 달라집니다. 다음 테이블에서는 공유 변수 함수가 지원하는 각 공유 변수 타입에 대한 변수 엔진 URL을 보여줍니다.

타입	변수 엔진 URL
네트워크 공유 변수, I/O 변수 또는 I/O 가명 (원격 접근)	<code>ni.var.psp</code>
I/O 변수 또는 I/O 가명 (로컬 접근)	<code>ni.var.io</code>

PSP 변수 엔진이나 I/O 변수 엔진을 사용하여 네트워크에 공개된 I/O 변수 또는 I/O 가명에 접근할 수 있습니다. 네트워크에 공개된 I/O 변수 또는 I/O 가명에 로컬로 읽거나 쓰려면 I/O 변수 엔진(`ni.var.io`)을 사용합니다. 네트워크에 공개된 I/O 변수 또는 I/O 가명에 원격으로 읽거나 쓰려면 PSP 변수 엔진(`ni.var.psp`)을 사용합니다.

URL의 변수 엔진 구성 요소는 포함되어 있으므로, 다음 클래스 중 하나를 사용할 때 옵션이 됩니다:

- PSP 변수 엔진
- I/O 변수 엔진
- PSP 변수 컨테이너
- I/O 변수 컨테이너
- PSP 변수
- I/O 변수

이 경우, URL의 변수 엔진 구성 요소를 생략하고 다음 URL 구문을 사용할 수 있습니다:

`//[호스트 이름]/[컨테이너 이름]/[변수 이름]`

변수 엔진을 포함하지 않는 일반적인 클래스를 사용하고 변수 엔진을 지정하지 않은 경우, LabVIEW는 다음과 같은 우선 순위로 변수 엔진을 검색하여 변수 식별자 URL 결정을 시도합니다.

1. I/O 변수 엔진
2. PSP 변수 엔진

이같은 경우, 일치하는 URL이 여러 엔진에 있으면, LabVIEW는 가장 처음 발견한 일치하는 URL을 사용합니다.

변수 식별 URL의 [호스트 이름] 구성 요소는 옵션입니다. 호스트 이름을 DNS 이름, 타겟 이름 또는 IP 주소로 식별할 수 있습니다. 그러나 변수 식별자 URL에서 호스트 이름을 지정하지 않으면, LabVIEW는 호스트가 VI를 실행하는 컴퓨터와 같은 컴퓨터에 있다고 가정합니다. 또한 localhost 문자열을 사용하여 VI를 실행하는 컴퓨터를 지정할 수 있습니다. 이름이 충돌하지 않게 하려면, 가능한 유일한 이름을 사용하십시오.

호스트 이름을 지정할 필요가 없으면, 다음의 URL 구문을 사용할 수 있습니다:

/[컨테이너 이름]/[변수 이름]

지정한 호스트 이름이 한가지 이상((예를 들어, 프로젝트 이름과 DNS 이름)으로 해석되는 경우, LabVIEW는 다음과 같은 우선 순위로 지정된 호스트 이름의 유효한 해석을 검색합니다.

1. 타겟 이름 (LabVIEW 프로젝트에서 지정된 이름)
2. DNS 이름
3. IP 주소

컨테이너에 있는 각 공유 변수의 이름은 고유해야 합니다. 그러나 변수 ID에 컨테이너 이름이 포함되어 있으므로, 다른 컨테이너의 변수는 같은 이름을 공유하면서, 유일한 변수 ID를 유지할 수 있습니다.

변수 식별자 URL 이스케이프 코드

이스케이프 코드를 사용하여 **변수 식별자 URL**을 지정할 때 예약된 문자를 대체합니다. 다음 테이블은 각 예약된 문자에서의 이스케이프 코드를 포함합니다.



노트

변수 식별자 URL의 예약된 문자 세트는 Uniform Resource Identifier (URI) Generic Syntax 스펙 (RFC 3986)이 지정하는 예약된 문자 세트와는 조금 다릅니다.

문자	이스케이프 코드
#	%23
%	%25
[	%5B
]	%5D

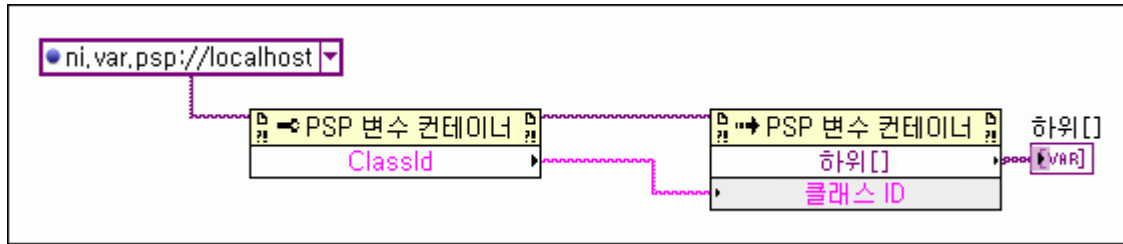
문자	이스케이프 코드
;	%3B
/	%2F
?	%3F
:	%3A
@	%40
&	%26
\$	%24
!	%21
*	%2A
+	%2B
=	%3D

변수 메소드 사용하기

하위 찾기 및 Is Kind Of 메소드와 같은 일부 변수 메소드를 사용하려면, 클래스 ID 입력으로 변수 클래스의 클래스 ID를 지정해야 합니다. 다음 테이블은 LabVIEW에 포함된 변수 클래스의 클래스 ID를 보여줍니다.

클래스 이름	클래스 식별자
변수 객체	VariableObject
변수	변수
PSP 변수	PSP변수
I/O 변수	IOVariable
변수	변수
변수 컨테이너	VariableContainer
PSP 변수 컨테이너	PSPVariableContainer
I/O 변수 컨테이너	IOVariableContainer
변수 엔진	VariableEngine
PSP 변수 엔진	PSPVariableEngine
I/O 변수 엔진	IOVariableEngine

변수 객체에서 상속받는 클래스의 클래스 ID 프로퍼티를 사용하여 프로그램적으로 클래스 ID를 얻을 수도 있습니다. 예를 들면, 다음 그림에서, 클래스 ID 프로퍼티는 PSP 변수 컨테이너의 클래스 ID를 하위 찾기 메소드로 전달합니다. 그리고 나서 하위 찾기 메소드는 로컬 호스트의 PSP 변수 컨테이너에 대한 참조의 배열을 반환합니다.



기존 데이터 소스를 사용하여 공유 변수 쓰기

공유 변수를 기존 데이터 소스에 연결할 때, 공유 변수는 해당 데이터 소스와 동일한 업데이트를 받습니다. 공유 변수에 대해 가능한 데이터 소스는 DAQ 채널, 다른 공유 변수와 같은 **NI Publish-Subscribe Protocol(NI-PSP)** 데이터 아이템, 별도의 프로젝트에 정의된 I/O 서버의 데이터를 포함합니다.



노트

- 네트워크 공유 변수만 연결할 수 있습니다.
- 현재 **배포**된 데이터 아이템에만 공유 변수를 연결할 수 있습니다.
- 공유 변수에 8 킬로바이트 미만의 데이터를 쓰는 경우, LabVIEW가 이 데이터를 네트워크를 통해 전송하는데 소요되는 시간은 최대 10밀리초입니다. 이러한 지연을 없애려면 [공유 변수 데이터 비우기] VI를 사용하여 **공유 변수 데이터를 즉시 전송**합니다.

공유 변수 기능을 지원하지 않는 데이터 소스에 대해 공유 변수 기능을 이용할 필요가 있을 경우, 공유 변수를 기존 데이터 소스에 연결합니다. 예를 들어, CompactRIO 타겟의 공유 변수에 대해 로깅 및 알람 프로퍼티를 설정할 필요가 있다고 가정합니다. 리얼타임 타겟이 로깅 및 알람을 지원하지 않기 때문에, 사용자 개발 컴퓨터에서 새로운 공유 변수를 생성하고, 해당 공유 변수를 CompactRIO 타겟의 공유 변수에 연결하고, 로깅 및 알람이 되도록 개발 컴퓨터의 공유 변수를 설정해야 합니다. CompactRIO 타겟의 공유 변수가 공유 변수 엔진(SVE)에 새 데이터를 전송하면, 공유 변수 엔진은 사용자 개발 컴퓨터에서 공유 변수의 값을 업데이트하고, 해당 공유 변수는 로깅 및 알람 태스크를 수행합니다.



노트 공유 변수를 네트워크 상의 기존 데이터 소스에 연결하면, LabVIEW는 데이터 소스의 설정 변경을 추적하지 않습니다. 따라서, 위의 예에서, CompactRIO 타겟의 변수 이름을 변경할 경우, 사용자 개발 컴퓨터의 공유 변수 설정 또한 업데이트해야 하며, 그렇지 않을 경우 해당 공유 변수는 깨집니다.

다음 단계를 따라 공유 변수를 기존 데이터 소스에 연결합니다.

1. **공유 변수를 생성**하거나 **프로젝트 탐색기** 윈도우의 공유 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 공유 변수 프로퍼티 대화 상자를 디스플레이합니다.
2. 가명 활성화 확인란에 확인 표시를 합니다.

3. 연결 대상 풀다운 메뉴의 다음 옵션에서 선택합니다:

- 프로젝트 변수를 선택하여 공유 변수를 현재 프로젝트의 NI-PSP 데이터 아 이템에 연결합니다.
- PSP URL을 선택하여 공유 변수를 네트워크의 NI-PSP 데이터 아 이템에 연 결합니다.

노트 로컬 서브넷 상에 있지 않은 컴퓨터에 데이터 아 이템이 있는 경우, 도구»공유 변수»컴퓨터 등록을 선택해야만 해당 컴퓨터를 등록할 수 있습니다. 컴퓨터 이름 또는 컴퓨터의 IP 주소를 지정한 후에, 공유 변수를 해당 컴퓨터의 데이터 소스에 연결할 수 있습니다.

- I/O 변수를 선택하여 OPC 서버와 같은 I/O 서버의 데이터 아 이템에 공유 변 수를 연결합니다.

4. 다음 중 하나의 방법을 사용하여 공유 변수를 연결하려는 데이터 소스를 선택합 니다:

- 탐색 버튼을 클릭하여 변수 탐색 대화 상자로 데이터 소스를 탐색합니다.
- 경로 텍스트 박스에 데이터 소스의 **네트워크 경로**를 입력합니다.

5. 확인 버튼을 클릭합니다.

데이터 통신 방법 홈

실시간 데이터를 프로그램적으로 공유하기

DataSocket VI와 함수를 사용하여 프로그램적으로 실시간 데이터를 설정하고, 읽고, 작 성할 수 있습니다.

(Real-Time, Windows) National Instrument는 **공유 변수 노드를 VI 및 함수에 연결**하여 프 로그램식으로 실시간 데이터를 공유할 것을 권장합니다.



노트

오직 Windows 또는 RT 타겟에서만 LabVIEW Real Time Module을 사용하여 공유 변수를 생성, 설정, 호스트할 수 있습니다. DataSocket VI 및 함수를 사용하여 다 른 플랫폼의 공유 변수를 읽거나 씁니다.

공유 변수 사용하기

공유 변수를 사용하여, VI 사이에서 또는 와이어로 연결할 수 없는 블록다이어그램 상의 위치 사이에서 데이터를 공유합니다. 공유 변수를 사용하여 약간의 블록다이어그램 프 로그래밍 또는 프로그래밍 없이 데이터를 공유할 수 있습니다.

DataSocket VI와 함수에 대해, 웹 브라우저에서 URL을 지정하는 것과 같은 방법으로 URL 을 지정하여 데이터를 쓰거나 읽습니다.

URL 지정하기

데이터를 전송할 때 URL은 Windows에서는 opc, ftp, http과 같은 통신 프로토콜을 사용하고, LabVIEW가 지원하는 모든 플랫폼에서는 psp, dstp, file과 같은 통신 프로토콜을 사용합니다. URL에 사용한 프로토콜은 어떠한 데이터를 쓰려고 하는지 또는 어떻게 네트워크를 설정하는지에 영향을 받습니다.

DataSocket VI와 함수를 사용하여 데이터를 읽거나 쓸 때, 또는 **컨트롤 연결을 활성화**할 때 다음의 프로토콜을 사용할 수 있습니다:

- **NI Publish-Subscribe Protocol (psp)**—네트워크와 사용자 로컬 컴퓨터 사이에서 데이터를 전송하는 National Instruments의 기술입니다. 이 프로토콜을 사용할 때 VI는 **공유 변수 엔진 (SVE)**과 통신합니다. psp 프로토콜을 사용하여 **공유 변수** 및 서버 또는 디바이스 데이터 아이템에 연결합니다. 데이터 아이템에 URL에 추가되는 이름을 제공해야 합니다. 데이터 연결은 이름을 공유 변수 엔진의 특정한 데이터 아이템의 주소를 나타내는데 사용합니다. 또한 **프런트패널 데이터 연결을 사용할** 때 이 프로토콜을 사용할 수 있습니다. 또한 fieldpoint 프로토콜을 NI Publish-Subscribe Protocol (NI-PSP)의 가명으로 사용할 수 있습니다.
- **DataSocket Transport Protocol (dstp)**—이 프로토콜을 사용할 때 VI는 DataSocket 서버와 통신합니다. 데이터에 URL에 추가되는 이름있는 태그를 제공해야 합니다. 데이터 연결은 이름있는 태그를 DataSocket 서버의 특정한 데이터 아이템의 주소를 나타내는데 사용합니다. 이 프로토콜을 사용하려면, DataSocket 서버를 실행해야 합니다.
- (Windows) **OLE for Process Control (opc)**—산업 자동화 작업으로 생성된 데이터와 같은 공유 리얼타임 생산 데이터를 위해 특별히 디자인되었습니다. 이 프로토콜을 사용하려면, OPC 서버를 실행해야 합니다.
- (Windows) **파일 전송 프로토콜 (ftp)**—이 프로토콜을 사용하여 FTP 서버에서 데이터를 읽는 파일을 지정할 수 있습니다.

노트 DataSocket 함수를 사용하여 FTP 사이트에서 텍스트 파일을 읽으려면 URL의 끝에 [text]를 추가하십시오.

- **file**—이 프로토콜을 사용하여 데이터를 포함하는 로컬 또는 네트워크 파일로의 링크를 제공할 수 있습니다.
- (Windows) **http**—이 프로토콜을 사용하여 데이터를 포함하는 웹 페이지의 링크를 제공할 수 있습니다.

노트 DataSocket 함수를 사용한 http 프로토콜을 통해 텍스트 파일을 읽으려면 URL의 끝에 [text]를 추가하십시오.

다음 테이블은 각 프로토콜 URL의 예를 보여줍니다.

URL	예제
	공유 변수의 경우: psp://computer/library/shared_variable
psp	서버와 디바이스 데이터 아이템과 같은 NI-PSP 데이터 아이템의 경우: psp://computer/process/data_item fieldpoint://host/FP/module/channel
dstp	dstp://servername.com/numeric에서 numeric은 데이터의 이름있는 태그입니다.
	opc:\National Instruments.OPCTest\item1
opc	opc:\\computer\National Instruments.OPCModbus\Modbus Demo Box.4:0 opc:\\computer\National Instruments.OPCModbus\Modbus Demo Box.4:0?updaterate=100&deadband=0.7
ftp	ftp://ftp.ni.com/datasocket/ping.wav ftp://ftp.ni.com/support/00README.txt[text]
	file:ping.wav
file	file:c:\mydata\ping.wav file:\\computer\mydata\ping.wav
http	http://ni.com

psp, dstp, opc, URL을 사용하여 실시간 데이터를 공유합니다. 이런 프로토콜은 원격과 로컬 컨트롤 및 인디케이터를 업데이트할 수 있기 때문입니다. ftp와 file URL을 사용하여 파일로부터 데이터를 읽습니다. 이 프로토콜은 원격과 로컬 컨트롤 및 인디케이터를 업데이트할 수 없기 때문입니다.

DataSocket VI와 함수가 지원하는 데이터 포맷

DataSocket VI와 함수를 사용하여 어떤 LabVIEW 데이터 타입의 데이터도 전달할 수 있습니다. 또한, DataSocket VI와 함수를 사용하여 다음 데이터를 읽고 쓸 수 있습니다:

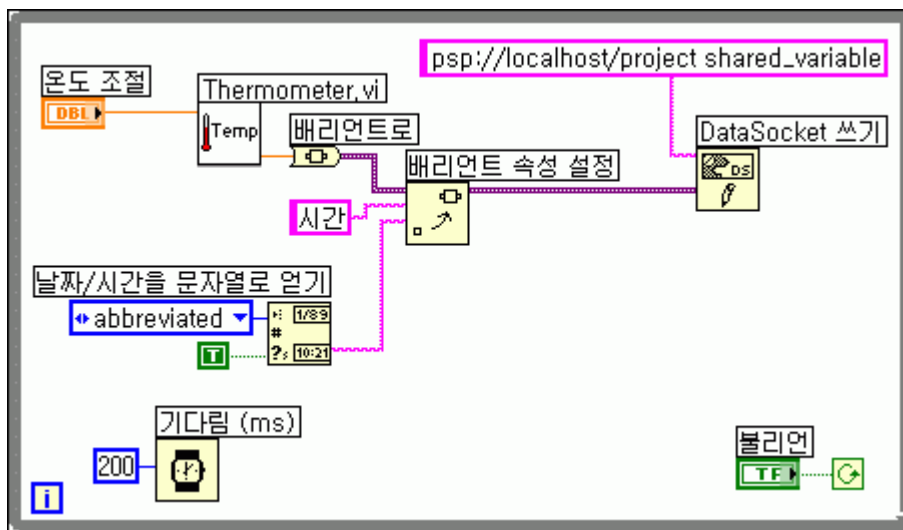
- 원시 텍스트—문자열을 문자열 인디케이터에 전달하려면 원시 텍스트를 사용합니다.

- 탭을 사용한 텍스트—배열에 데이터를 쓸 때 스프레드시트에서처럼 탭을 사용한 텍스트를 사용합니다. LabVIEW는 탭을 사용한 텍스트를 데이터의 배열로 해석합니다.
- .wav 데이터—사운드를 VI 또는 함수에 쓰려면 .wav 데이터를 사용합니다.
- 배리언트 데이터—National Instruments Measurement Studio ActiveX control과 같은 다른 어플리케이션으로부터 데이터를 읽는 경우 배리언트 데이터를 사용합니다.

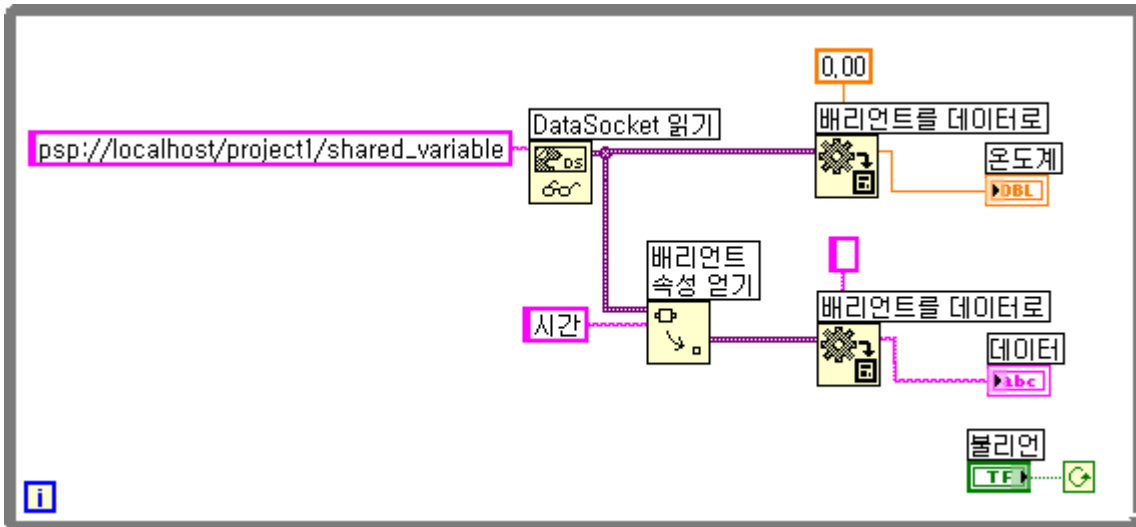
DataSocket 함수와 배리언트 데이터

데이터를 다른 어플리케이션에서 읽을 때처럼, 프로그램적으로 데이터를 읽는 VI나 기타 어플리케이션이 데이터를 다시 원래 데이터 타입으로 변환할 수 없을 때 배리언트 데이터를 사용합니다. 또한, 데이터 타입이 추가적인 속성을 허용하지 않을 때 배리언트 데이터를 사용하여 타임스탬프나 경고와 같은 속성을 읽거나 쓰는 데이터에 추가할 수 있습니다.

[배리언트로] 함수를 사용하여 프로그램적으로 데이터 연결에 쓴 데이터를 배리언트 데이터로 변환합니다. 다음 블록다이어그램은 연속적으로 온도를 읽어 이 데이터를 배리언트 데이터로 변환하고 속성으로서 타임스탬프를 데이터에 추가합니다.



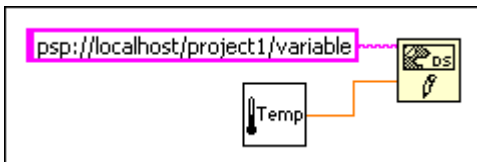
다른 VI에서 이 실시간 데이터를 읽는 경우, 이 VI는 배리언트 데이터를 자신이 조작할 수 있는 데이터 타입으로 변환해야 합니다. 다음 블록다이어그램은 온도 데이터를 연속적으로 읽고, 배리언트 데이터를 온도값으로 변환하고, 읽혀진 값에 해당되는 타임스탬프 속성을 가져오며, 프런트패널에 온도값과 타임스탬프를 디스플레이합니다.



DataSocket 함수를 사용하여 실시간 데이터 읽고 쓰기

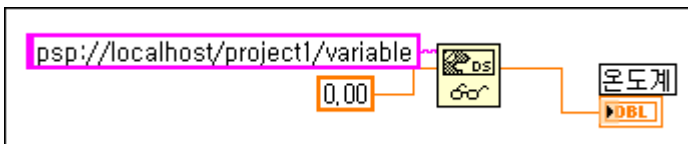
블록다이어그램에서, DataSocket 함수 또는 (Real-Time, Windows) 공유 변수 노드를 사용하여 데이터를 프로그램식으로 읽거나 쓸 수 있습니다. 이는 데이터 연결을 사용자 정의하거나 다이나믹하게 변경하는데 유용합니다. DataSocket 함수를 공유 변수와 함께 사용하여 프로그램적으로 공유 변수 읽기 또는 쓰기를 시작하거나 끝낼 수 있으며, 버퍼 크기를 지정하거나 어떤 공유 변수를 읽을지 다이나믹하게 결정할 수 있습니다. 공유 변수 컨트롤을 DataSocket 함수에 연결하여 입력 공유 변수를 선택할 수 있습니다.

[DataSocket 쓰기] 함수를 사용하여 프로그램적으로 실시간 데이터를 쓸 수 있습니다. 다음 블록다이어그램은 숫자값을 쓰는 방법을 보여줍니다.



[DataSocket 쓰기] 함수는 다형성이므로, 연결에 쓰는 데이터는 임의의 포맷 또는 LabVIEW 데이터 타입이 될 수 있습니다.

[DataSocket 읽기] 함수를 사용하여 프로그램적으로 실시간 데이터를 읽을 수 있습니다. 다음의 블록다이어그램은 데이터를 읽는 방법과 배정도 부동소수로 변환하는 방법을 보여줍니다.



[DataSocket 읽기] 함수의 타입 입력에 컨트롤 또는 상수를 연결하여 실시간 데이터를 특정 데이터 타입으로 변환합니다. 타입을 지정하지 않을 경우, [DataSocket 읽기] 함수의 데이터 출력은 배리언트 데이터를 반환하며, 이러한 경우 [배리언트를 데이터로] 함수로 조작할 수 있습니다.

프로그램적으로 데이터 연결 열기와 닫기

[DataSocket 열기]와 [DataSocket 닫기] 함수를 사용하여 데이터 연결을 열 때와 닫을 때를 컨트롤합니다. [DataSocket 열기] 함수를 사용하여 데이터 연결을 시작하면, 다음 조건 중의 하나가 만족되기 전까지 계속 연결됩니다: [DataSocket 닫기] 함수를 사용하여 연결을 닫거나, VI를 닫거나, VI의 실행을 멈춥니다. VI가 실행을 중지하면, SubVI의 데이터 연결도 모두 닫힙니다. [DataSocket 열기] 함수의 URL 입력은 유효한 URL만 허용합니다. [DataSocket 열기] 함수는 [DataSocket 읽기] 함수 및 [DataSocket 쓰기] 함수의 URL 입력으로서 사용할 수 있는 데이터 연결 참조 번호를 반환합니다.

데이터 연결에 DataSocket 함수를 사용한 예제는 labview\examples\Data Communication\DataSocket\Simple DataSocket\Simple DataSocket.lvproj의 VI를 참조하십시오.



관련 정보

[공유 변수 노드를 사용하여 공유 변수를 읽기와 쓰기](#) (582 페이지 참조)

[공유 변수로 최근값 공개하기](#) (566 페이지 참조)

[I/O 컨트롤과 인디케이터](#) (1001 페이지 참조)

[실시간 데이터를 프로그램적으로 쓰기](#) (1478 페이지 참조)

[블록다이어그램 객체](#) (52 페이지 참조)

[실시간 데이터를 프로그램적으로 공유하기](#) (593 페이지 참조)

[프로그램적으로 실시간 데이터 읽기](#) (1481 페이지 참조)

[프로그램적으로 데이터 연결 열기와 닫기](#) (1478 페이지 참조)

[실시간 데이터를 프로그램적으로 공유하기](#) (593 페이지 참조)

데이터 버퍼링하기

네트워크 **공유 변수** 또는 **psp**나 **dstpURL**을 사용하여 프로그램적으로 **데이터**를 공유하는 경우, LabVIEW는 기본적으로 가장 최신의 값만을 모든 수신자(reader)에게 씁니다. 한 클라이언트가 서버에 값을 쓰는 속도가 다른 클라이언트가 그 값을 읽는 속도보다 더 빠를 경우, 서버의 새 값은 클라이언트가 그 값을 읽기 전에 처리되지 않은 이전 값을 덮어 씁니다. 수신자가 다음 값을 받기 전에 값을 받지 못한 경우 데이터는 손실됩니다. 이 처리되지 않은 데이터의 손실이 서버나 클라이언트에서 발생할 수 있습니다. 데이터를 읽고 가장 최근에 서버에 쓰여진 값만을 받으려고 하는 경우, 이와 같은 데이터 유실은 문제가 되지 않습니다. 그러나 서버에 쓰여지는 모든 값을 받으려고 하는 경우나 데이터 유실이 허용되지 않는 전송의 경우에는 클라이언트의 데이터를 버퍼링해야 합니다. 공유 변수 프로퍼티 대화 상자의 네트워크 페이지에서 버퍼링 사용 확인란에 확인 표시를 하여 공유 변수에 대한 버퍼링을 활성화합니다.

클라이언트측 버퍼링

버퍼링은 수신자와 발신자 사이의 잠재적인 시간차를 줄여주나 데이터 전송을 보장하지는 않습니다. 서버나 클라이언트의 버퍼에 있는 데이터가 버퍼의 크기보다 커지는 경우, 버퍼는 예전 값을 버리고 새 값을 받게 됩니다. 공유 변수 프로퍼티 대화 상자의 네트

워크 페이지에서 공유 변수의 버퍼 크기를 설정하여 공유 변수가 이전 데이터를 덮어쓰기 전에 얼마나 많은 데이터 패킷을 보유하는지 설정할 수 있습니다.

또한 프로그램적으로 각 이이템의 버퍼가 보유할 수 있는 최대 바이트 수와 최대 값 수를 패킷으로 지정할 수 있습니다. National Instruments는 데이터를 버퍼할 때 최대 버퍼 바이트와 최대 버퍼 패킷 프로퍼티 모두를 지정하도록 권장합니다. 들어오는 데이터가 최대 바이트 수 또는 최대값 수를 초과하는 경우, LabVIEW는 오래된 데이터를 버퍼에서 삭제하여 새 데이터를 받습니다.

프로그램적으로 버퍼링을 설정할 때 [배리언트 속성 설정] 함수에 쓰여진 데이터를 연결하여 발신자의 각 값을 고유하게 식별하고 수신자에서 버려진 시퀀스 ID를 확인함으로써 데이터 스트림에서 버려진 값을 감지합니다.

8 킬로바이트 미만의 데이터를 가진 공유 변수를 연결하면, LabVIEW가 이 데이터를 네트워크에 전송하는데 최대 10 밀리초까지 소요됩니다. 공유 변수 데이터 비우기 VI를 사용하여 즉시 버퍼를 비우고 이러한 지연을 제거할 수 있습니다.

버퍼가 활성화된 클라이언트가 모든 데이터를 받도록 하려는 경우, [DataSocket 쓰기] 함수를 사용하여 프로그램적으로 **실시간 데이터를 씁니다**. 프런트패널에서 데이터를 쓰면 데이터를 잃을 수 있습니다.



노트

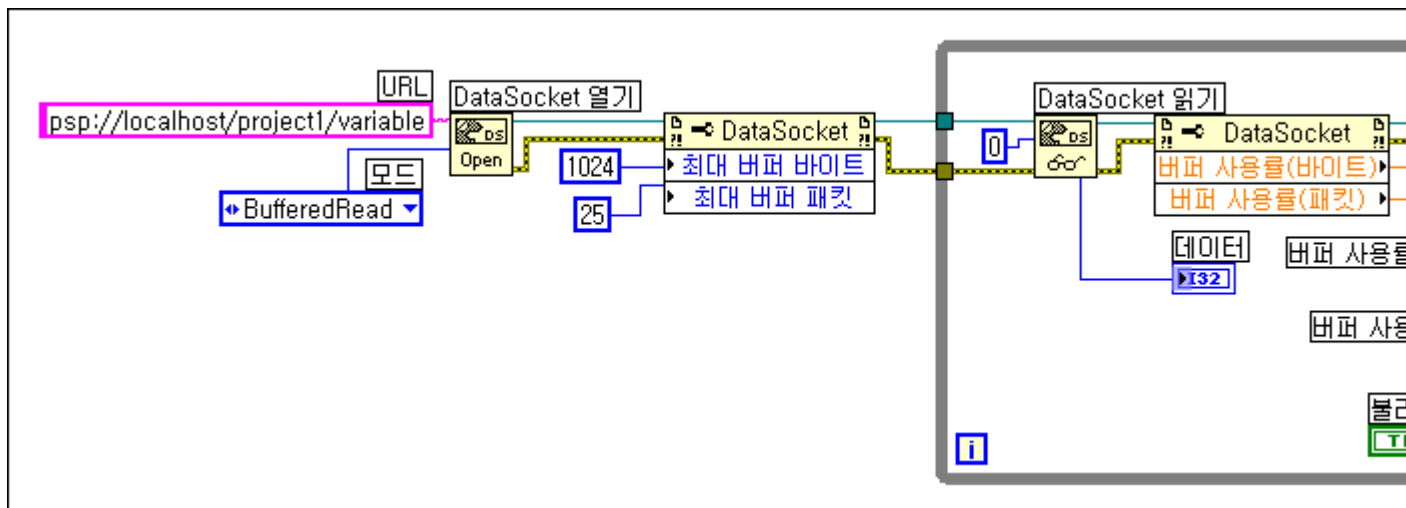
또한, 클라이언트 버퍼링은 opc와 file과 같은 다른 프로토콜에도 적용됩니다. 그러나 일부 경우에 이러한 서버는 **데이터를 잃을 수** 있습니다.

[데이터 소켓 열기] 함수의 모드 입력을 BufferedRead 또는 BufferedReadWrite로 설정하고, 프로퍼티 노드를 사용하여 FIFO(First-In-First-Out) 버퍼의 **사이즈에 대해 DataSocket 프로퍼티를 설정**합니다. 이와 같은 방법은 LabVIEW가 값이 변경될 때마다 덮어쓰는 대신 클라이언트가 받는 값을 버퍼에 저장하도록 합니다.



노트 DataSocket 프로퍼티를 사용하여 FIFO 버퍼의 크기를 설정하는 경우, [DataSocket 열기] 함수의 모드 입력을 BufferedRead 또는 BufferedReadWrite로 설정해야 합니다.. 그렇지 않으면, 연결이 되어도 서버의 아이템은 버퍼에 저장되지 않습니다.

다음 블록다이어그램은 프로그램적으로 버퍼링을 사용합니다.



또한, 공유 데이터가 DataSocket 함수를 사용할 때 문자열?

maxbytes=1000&maxpackets=10을 URL의 끝에 추가하여 버퍼링을 활성화할 수 있습니다. 이때 1000은 버퍼가 포함하는 바이트의 최대 개수이고 10은 버퍼가 포함하는 패킷의 최대 개수입니다.



노트 버퍼링은 서버가 쓰는 데이터를 **DataSocket 읽기** 함수를 사용하여 읽을 때 적용됩니다. **프런트패널 DataSocket 데이터 연결**을 사용하여 데이터를 읽을 때에는 버퍼링을 사용할 수 없습니다. 공유 변수 프로퍼티 대화 상자의 **네트워크** 페이지에 있는 버퍼링을 활성화하고 컨트롤을 공유 변수에 연결한 경우, **공유 변수 엔진**을 통해 버퍼링을 사용하여 **프런트패널 데이터 연결**을 할 수 있습니다.

DataSocket 프로퍼티를 사용하여 진단 정보 보고하기

지정한 버퍼에 대한 **진단 정보를 요청**하기 위해서 버퍼 사용률(바이트) 프로퍼티 또는 버퍼 사용률(패킷) 프로퍼티를 사용합니다. 현재 버퍼의 크기가 적절한지 파악하기 위해 클라이언트에서 사용 중인 버퍼의 퍼센트를 확인하기 위하여 이들 프로퍼티를 사용합니다. 이들 프로퍼티의 값이 버퍼의 최대값에 근접하는 경우, 서버가 쓰는 모든 값을 확실히 받기 위해서 **버퍼의 크기를 증가**시킵니다. 또한 DataSocket 함수를 포함하는 While 루프가 실행되는 속도를 증가시켜 읽는 **속도를 증가**시킬 수 있습니다.

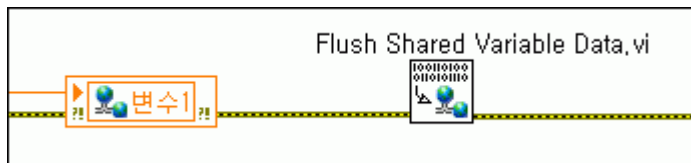
서버측 버퍼링

클라이언트측 버퍼링을 설정할 때, 서버가 네트워크가 받을 수 있는 속도보다 빨리 데이터를 전송하는 경우 데이터의 유실이 발생할 수 있습니다. **NI Publish-Subscribe Protocol(NI-PSP)**의 서버측 버퍼링이나 dstp 버퍼링을 사용할 수 있습니다. NI-PSP 버퍼링을 사용하려면 버퍼링을 활성화한 기존 공유 변수에 연결해야 합니다. dstp 버퍼링을 사용하려면, 또한 DataSocket 서버 관리자를 사용하여 서버측 버퍼링을 설정해야 합니다. DataSocket 서버의 서버측 버퍼링에 대한 추가적인 내용은 DataSocket 서버 도움말을 참조하십시오. 다른 프로토콜의 경우, 해당 서버의 서버측 버퍼링에 대한 추가적인 내용은 서버 문서를 참조하십시오.

즉시 네트워크를 통해 공유 변수 데이터 전송하기

LabVIEW는 공유 변수에 쓰는 데이터를 8 킬로바이트 버퍼에 임시로 저장합니다. LabVIEW는 버퍼가 꽉 차거나 10 밀리초가 지나면 네트워크를 통해 해당 데이터를 전송합니다. 네트워크를 통해 데이터 패킷을 보낼 때마다 오버헤드가 초래되므로, 이 디자인은 전송하는 데이터 패킷의 수를 줄임으로써 처리량을 증가시킵니다.

그러나 공유 변수에 8 킬로바이트 이하의 데이터를 쓰는 경우 이 디자인은 보다 긴 지연 시간을 초래합니다. 10 밀리초의 지연을 없애고 지연 시간을 최소화하려면, 다음 그림에서 보여지는 것처럼, 공유 변수에 쓴 후, [공유 변수 데이터 비우기] VI를 사용하여 네트워크를 통해 즉시 모든 공유 변수 데이터를 전송합니다.



앞의 그림의 경우 공유 변수 노드가 쓴 데이터를 비우지만, 다른 방법으로 쓴 공유 변수 데이터를 비울 수도 있습니다. 예를 들어, **프로그램식으로 공유 변수 쓰기**를 하는 경우, [변수 쓰기] 함수 다음에 [공유 변수 데이터 비우기] VI를 사용합니다.



노트

[공유 변수 데이터 비우기] VI를 사용하면, LabVIEW는 네트워크를 통해 아직 전송되지 않은 모든 공유 변수 데이터를 전송하는데, 여기에는 다른 어플리케이션 및 동일한 어플리케이션의 다른 부분에서 모든 공유 변수와 함께 쓴 데이터까지 포함됩니다. 그러므로, 단일 컴퓨터의 모든 공유 변수 발신자를 위해 지연 시간을 최소화할 필요가 있을 때 [공유 변수 데이터 비우기] VI가 유용합니다. 그러나 그 중 일부 공유 변수에 높은 처리률이 필요할 경우, [공유 변수 데이터 비우기] VI는 해당 공유 변수에 부정적인 영향을 미칠 수 있습니다.



데이터 통신 방법 홈

네트워크에서 공유 변수의 이용 가능하게 하기

공유 변수를 **공유 변수 엔진(SVE)**에 배포해야만 해당 공유 변수를 네트워크를 통해 읽고 쓸 수 있습니다. 공유 변수를 배포하면 공유 변수 엔진에서 공유 변수에 대한 메모리 공간이 생성됩니다. 공유 변수의 배포를 해제하면 해당 공유 변수에 할당되었던 메모리 리소스가 해제됩니다.

공유 변수 노드로 공유 변수를 읽고 쓰는 VI를 실행할 때 LabVIEW는 해당 공유 변수를 포함하는 **프로젝트 라이브러리**를 자동으로 배포합니다. 그러나 어플리케이션 종료 시에 LabVIEW는 프로젝트 라이브러리의 배포를 자동으로 해제하지 않습니다. 그러므로 **공유 변수 노드로 공유 변수를 읽고 쓰기** 전에 공유 변수를 배포할 필요는 없지만 해당 공유 변수에 할당된 메모리 리소스를 해제하려면 해당 공유 변수의 배포를 해제해야 합니다. 또한, **다른 방법으로 공유 변수 읽고 쓰기**가 진행 중이면, 공유 변수를 명백히 배포해야 합니다. 다음 리스트에서는 공유 변수를 배포하고 배포 해제하는데 사용할 수 있는 방법을 설명합니다.

- **편집 시 방법**—소수의 공유 변수에 접근하는 어플리케이션을 개발할 때에는 편집 시에 공유 변수를 배포하고 배포 해제합니다.
- **실행 시 방법**—어플리케이션을 독립 어플리케이션으로 배포하려면 공유 변수를 실행 시에 배포하거나 배포 해제합니다. 실행 시에 공유 변수를 배포하고 배포 해제하는 것은 어플리케이션이 다수의 공유 변수에 접근하는 경우에도 유용합니다.



노트

공유 변수를 사용하는 독립 어플리케이션을 배포하려는 경우, LLB 또는 실행 파일에 .lvlib 파일을 포함시키지 마십시오. 어플리케이션 프로퍼티 대화 상자의 소스 파일 선택 페이지를 사용하여 .lvlib 파일의 대상을 실행 파일 또는 LLB 밖의 대상으로 변경합니다.

편집 시에 공유 변수 배포 및 배포 해제하기

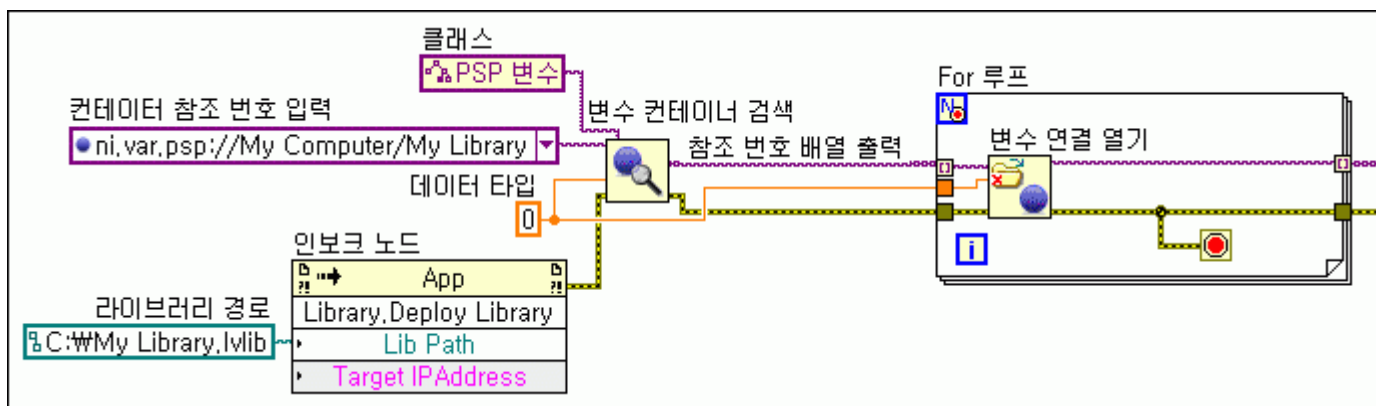
공유 변수를 배포하려면 공유 변수를 포함하는 **프로젝트 탐색기** 윈도우의 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 배포를 선택합니다. 이 작동으로 공유 변수 엔진의 프로젝트 라이브러리에 있는 모든 공유 변수가 배포됩니다. 공유 변수 노드를 사용하는 경우 이외의 방법으로 공유 변수를 읽고 쓰는 어플리케이션을 실행하기 전에 이러한 작동을 수행합니다.

공유 변수의 배포를 해제하려면, 공유 변수를 포함하는 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 배포 해제를 선택합니다.

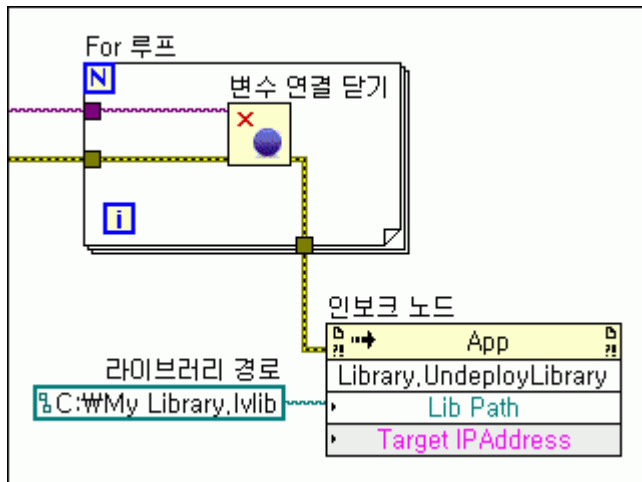
실행 시에 공유 변수 배포 및 배포 해제하기

(Windows) 실행 시에 공유 변수를 배포하려면 어플리케이션 프로퍼티 대화 상자의 공유 변수 배포 페이지에서 어플리케이션 실행 시 공유 변수 라이브러리 배포 확인란에 확인 표시를 합니다.

또한, 공유 변수를 포함하는 라이브러리를 배포하고 배포 해제하려면, 어느 플랫폼에서나 라이브러리 배포 및 라이브러리 배포 해제 메소드를 사용할 수 있습니다. 예를 들어, 다음 그림에서, 라이브러리 배포 메소드는 [변수 컨테이너 검색] 함수가 해당 프로젝트 라이브러리의 공유 변수를 검색하기 전에 My Library.lvlib를 배포합니다. 그리고 나서 [변수 연결 열기] 함수는 해당 공유 변수로 연결합니다.



다음 그림에서, 라이브러리 배포 해제 메소드는 [변수 연결 닫기] 함수가 해당 프로젝트 라이브러리의 공유 변수와 연결을 끊은 후에 My Library.lvlib의 배포를 해제합니다.



LabVIEW에서 데이터 전송

데이터 전송하기

LabVIEW에서 데이터 전송 방법 선택 시 도움이 필요하면 LabVIEW의 데이터 통신 방법을 참조하십시오. 또한 특정 데이터 전송의 사례에 대한 정보는 LabVIEW 도움말의 다음 토픽 모음을 참조하십시오.

- 네트워크를 통해 데이터 전송하기
- 채널 와이어를 사용하여 병렬 코드 섹션 사이에서 데이터 통신하기
- 시간 결정성이 있는 어플리케이션에서 데이터 공유하기(Real-Time Module)
- 데이터 저장하기 및 전송하기(FPGA Module)



관련 정보

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

[공유 변수 읽고 쓰는 방법 선택하기](#) (573 페이지 참조)

[채널 와이어를 사용하여 병렬 코드 섹션 사이에서 데이터 통신하기](#) (697 페이지 참조)

LabVIEW에서의 데이터 통신 방법

LabVIEW에는 용도별로 사용하기에 적합한 다양한 데이터 통신 방법이 포함되어 있습니다. 사용자의 어플리케이션에 적합한 데이터 통신 방법을 구현하려면, 다음 토픽에 제시된 개요를 참조하십시오. 특정 데이터 통신 방법에 대한 자세한 정보는 해당 데이터 통신 방법의 제목을 클릭하십시오.

데이터 통신 타입:	데이터 흐름의 원소	버퍼 인터페이스	변수 인터페이스
용도:	대부분의 블록 다이어그램 객체 사이에서 데이터를 전송하기.	값을 덮어쓰거나 손실을 방지하는 것이 중요한 경우, 각 위치 사이에서 데이터를 전송하기.	최신 값이 가장 중요한 경우, 메모리에 즉시 접근할 수 있는 값을 저장하기. 경합 조건이 발생하지 않도록 주의해야 합니다.
예:	한 VI의 출력에서 다른 VI의 입력으로 데이터 전달하기.	웨이브폼 얻기; 메시지나 명령 전송하기; 이미지 스트리밍하기.	현재 온도 모니터하기; 시스템 상태 모니터하기.
항목:	— 와이어 — - 시프트 레지스터 — 피드백 노드 — - 채널 와이어	내부 타겟 또는 PC — 큐 — RT FIFO — 사용자 이벤트 — AMC(Asynchronous Message Communication) 참조 라이브러리 네트워크 — 네트워크 스트림 — TCP — STM(Simple TCP Messaging) 참조 라이브러리 FPGA 내부 — Target-Scoped FIFO — VI-Defined FIFO — DRAM FPGA에서 호스트로/호스트에서 FPGA로 — DMA FIFO FPGA에서 FPGA로 — Peer-to-Peer(P2P) FIFO	내부 타겟 또는 PC — 로컬 변수 — 글로벌 변수 — 기능적인 글로벌 변수(FGV) — 단일 프로세스 공유 변수 — 데이터 값 참조 — 현재 값 테이블(CVT) 네트워크 — 네트워크 공유 변수 — 프로그램적 공유 변수 API — 웹 서비스 — UDP FPGA 내부 — 글로벌 변수 — 메모리 아이템 — 레지스터 아이템 FPGA에서 호스트로/호스트에서 FPGA로 — 읽기/쓰기 컨트롤 — NI 스캔 엔진 및 변수 — 사용자 정의 I/O 변수

데이터 흐름의 원소

LabVIEW에서 **데이터 흐름 프로그래밍 환경**의 장점을 활용하려면 와이어, 시프트 레지스터, 피드백 노드, 채널 와이어를 어플리케이션의 주요 데이터 통신 방법으로 사용해야 합니다.

메소드	속성
와이어	블록 다이어그램 객체 사이에서 주요 데이터 전송
시프트 레지스터	루프 반복 사이에서 주요 데이터 전송
피드백 노드	코드 반복 사이에서 주요 데이터 전송(루프가 필요하지 않음)
채널 와이어	병렬 코드 섹션 간 비동기식 통신을 위한 주요 데이터 전송

버퍼 인터페이스

한 위치에서 다른 위치로 데이터를 전송할 때 값을 덮어쓰거나 손실하고 싶지 않은 경우에는 버퍼 인터페이스를 사용합니다. 버퍼 인터페이스는 자주 FIFO(선입 선출) 단일 원소 또는 여러 원소의 버퍼로 표현되며, 하나 이상의 발신자가 버퍼의 한쪽에 데이터를 추가하고 다른 한쪽에서 수신자가 데이터를 받습니다.

내부 타겟 또는 PC

방법	속성
큐	- 크기 조정 및 접근 옵션이 유연함 - 타이밍 결정성이 높지 않음
RT FIFO (Real-Time Module)	- 시간 결정성이 있음 - 폴링 모드(기본)에서는 읽기/쓰기 대기 시간을 시간에 결정적으로 처리하지만 CPU 오버헤드가 높아짐 - 방해 모드에서는 읽기/쓰기 대기 시간에서 CPU 사용이 최적화되지만, 작업을 시간에 결정적으로 처리하지 않음 - 크기가 고정되어 있고 접근 옵션이 제한되어 있어서 큐보다 유연성이 적음
사용자 이벤트	이벤트 드리븐 프로그래밍 (406 페이지 참조) 어플리케이션의 여러 다른 부분 사이의 이벤트 드리븐 커뮤니케이션 - 객체에 프로그램적인 변경이 가해질 때 반응함

방법	속성
AMC(Asynchronous Message Communication) 라이브러리 (ni.com에서 이용가능)	- 단일 프로세스 및 프로세스 사이에서 메시지 통신 - 메시징 프레임워크 설계 시, 큐에 비해 사용이 훨씬 용이함 - 큐 메시지 핸들러(QMH) 템플릿을 포함함 노트: 이것은 ni.com/samplecodelicense에 있는 Sample Code License Terms에 의거한 커뮤니티 콘텐츠입니다.

네트워크

방법	속성
네트워크 스트림	1:1 통신 - TCP보다 구현이 쉬움 - 자동으로 연결 해제 및 다시 연결을 처리함 - 단방향 - Windows와 Real-Time 시스템에서만 사용 가능
TCP	- 네트워크 스트림 지원이 없어 네트워크 통신을 필요로 하는 어플리케이션에 대한 기능의 폭이 넓음 - 하위 레벨 프로토콜을 사용하면 구현이 복잡해짐
STM (Simple TCP Messaging) 참조 라이브러리 (ni.com에서 이용 가능)	- 하위 레벨 TCP 함수의 메시지 래퍼 1:N 단일 서버에서 여러 클라이언트간의 통신을 구현하는데 사용됨 노트: 이것은 ni.com/samplecodelicense에 있는 Sample Code License Terms에 의거한 커뮤니티 콘텐츠입니다.

FPGA 내부

방법	속성
Target-Scoped FIFO (FPGA Module)	구현 옵션은 FIFO가 사용하는 저장 타입(플립 플롭, 록업 테이블 또는 블록 메모리)을 지정함

방법	속성
VI-defined FIFO (FPGA Module)	- 구현 옵션은 FIFO가 사용하는 저장 타입(폴립 폴름, 록업 테이블 또는 블록 메모리)을 지정함 - 각 인스턴스의 리소스를 복제하여 재호출 SubVI에서의 리소스 충돌을 방지함 - 여러 개의 프로젝트에서 VI에 코드를 재사용하는 것을 용이하게 함 - 프로젝트 탐색기 윈도우에서 확인 또는 사용자 정의할 수 없음
DRAM (FPGA Module)	- 메모리 아이템 또는 CLIP 인터페이스를 사용하여 DRAM(Dynamic Random Access Memory)과 상호작용함 - Target-Scoped 및 VI-Defined FIFO에 비해 대용량의 버퍼된 메모리 공간을 제공함 - 시간 결정성이 없는 접근 지연 - 일부 타겟에서만 사용 가능

FPGA에서 호스트로/호스트에서 FPGA로

방법	속성
DMA FIFO (FPGA Module)	DMA를 사용하여 호스트로(부터) 대용량의 데이터를 전송

FPGA에서 FPGA로

방법	속성
Peer-to-Peer (P2P) FIFO (FPGA Module)	타겟 간의 1:1 통신

변수 인터페이스

변수 인터페이스를 사용하면 수신자와 발신자가 최소한의 흐름 제어 또는 흐름 제어 없이 접근이 가능한 메모리에 값을 저장할 수 있습니다. 최신값이 가장 중요하여 이전 값을 덮어써도 되는 경우에는 반드시 변수를 사용해야 합니다. **경합 조건이 발생하지 않도록 주의해야 합니다.**

내부 타겟 또는 PC

방법	속성
로컬 변수	단일 VI에서 접근할 수 있는 데이터를 저장함

방법	속성
글로벌 변수	동일한 LabVIEW 어플리케이션 인스턴스에 여러 VI가 접근할 수 있는 데이터를 저장함
기능적인 글로벌 변수(FGV)	- 초기화하지 않은 시프트 레지스터를 사용하여 글로벌 데이터를 저장하는 VI - 스택이나 큐 버퍼와 같은 복잡한 데이터 구조뿐만 아니라, 간단한 글로벌 변수 구현 가능 - 호출 코드가 변수에 수행할 특정 작업을 정의하여 경쟁 조건을 방지함
단일 프로세스 공유 변수	- 시간 결정성이 있는 옵션이 있음 - 네트워크 용도로 스케일 가능함
데이터 값 참조	- 큰 데이터 세트를 저장함 - 메모리 관리를 돕고, 데이터 복제를 방지함
현재 값 테이블(CVT) (ni.com에서 이용 가능)	- 별도의 구성요소를 갖는 어플리케이션의 변수에 대한 공통 데이터 저장소 - I/O 작업에 중점을 두어, 계산시 공유 변수보다 적은 리소스를 사용함 노트: 이것은 ni.com/samplecode/license에 있는 Sample Code License Terms에 의거한 커뮤니티 콘텐츠입니다.

네트워크

방법	속성
네트워크 공유 변수	- 시간 결정성이 있는 옵션이 있음 - 값을 모니터하는데 이상적임 - 네트워크 데이터 서버(공유 변수 엔진)에 값을 저장함 1:N 및 N:1 네트워크 설정을 위한 기본 연결 관리자
프로그래밍 공유 변수 API	- 노드 기반의 정적 연결 대신 변수에 다이나믹한 접근을 제공하여 설계의 확장성을 향상시킴 - 단일 변수 API 함수를 사용하여 여러 변수에 접근함 - 네트워크 I/O 변수에 접근함
웹 서비스	- 표준 HTTP 프로토콜을 사용한 네트워크 데이터 통신 - 텍스트, 이미지 및 동영상과 같은 표준 MIME 데이터 타입을 스트리밍함 - LabVIEW VI를 사용하여 웹서비스를 빌드하고 배포함

방법	속성
UDP	- 디바이스에서 프로세스 사이의 하위 레벨 통신을 위한 단일캐스트(1:1) 및 멀티캐스트(1:N)를 지원함 - 손실없는 데이터 전송이 요구되지 않는 어플리케이션에서 TCP보다 성능이 더 높음 - 하위 레벨 프로토콜을 사용하면 구현이 복잡해짐 - 클라이언트의 데이터 수신을 보장하지 않음

FPGA 내부

방법	속성
글로벌 변수	동일한 LabVIEW 어플리케이션 인스턴스에 여러 VI가 접근할 수 있는 데이터를 저장함
메모리 아이템 (FPGA Module)	- 값을 여러 데이터 포인트로 저장할 수 있음 - 구현 옵션은 메모리 아이템이 사용하는 저장 타입(블록 메모리, 특업 테이블 또는 DRAM)을 지정함 - Target-Scoped(프로젝트에서 설정) 및 VI-Defined(VI에서 재호출용으로 설정) 옵션
레지스터 아이템 (FPGA Module)	- 클럭 사이클마다 상태를 게시할 수 있음 - Target-Scoped(프로젝트에서 설정) 및 VI-Defined(VI에서 재호출용으로 설정) 옵션 - FIFO보다 FPGA 리소스 소비가 적음

FPGA에서 호스트로/호스트에서 FPGA로

방법	속성
읽기/쓰기 컨트롤 (FPGA Module)	FPGA 타겟의 최상위 FPGA VI의 컨트롤이나 인디케이터를 읽거나 씸
NI 스캔 엔진 및 변수 (FPGA Module + 사용자의 타겟에 있는 NI 스캔 엔진 지원)	- FPGA I/O 데이터와 RT 호스트 사이의 주기적인 단순한 통신 - FPGA 타겟과 RT 호스트 사이에서 I/O 데이터를 전송하는데 필요한 코드를 줄임
사용자 정의 I/O 변수 (FPGA Module)	- 하이브리드 모드에서, 스캔 엔진을 사용하여 사용자 정의된 데이터를 전송함 - FPGA VI와 LabVIEW RT 호스트 VI 사이의 데이터 전송 - 사용자 정의한 FPGA 처리된 I/O 데이터 전송

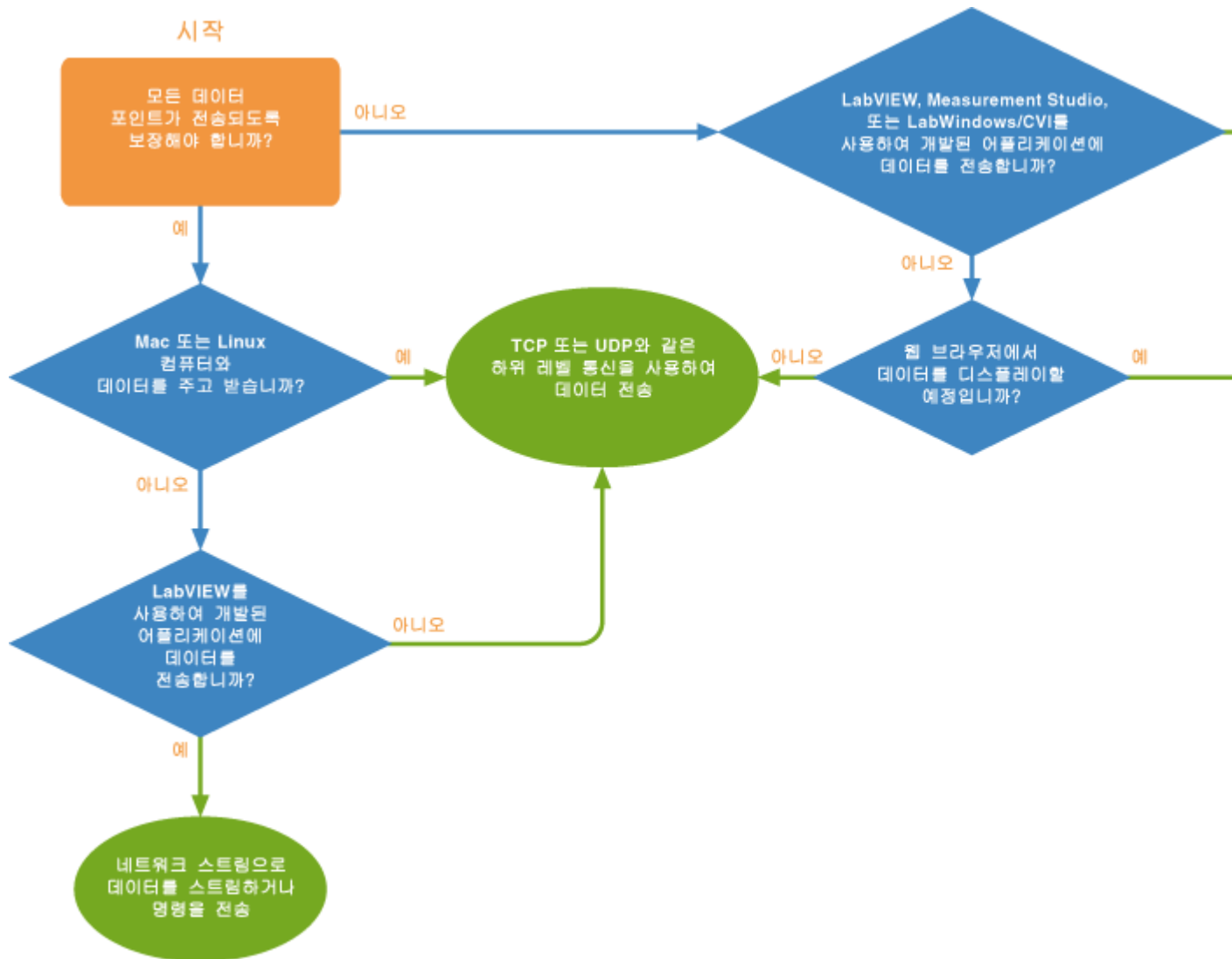
네트워크를 통해 데이터 전송하기

어플리케이션의 네트워킹 기능을 선택할 때, 가장 중요한 요소는 어플리케이션이 사용하는 통신 모델입니다. 각기 다른 통신 모델에는 서로 다른 데이터 전송 요건이 요구되며 이는 각기 다른 네트워킹 기능으로 충족되도록 고안됩니다. 다음 테이블에서는 가장 흔히 사용되는 통신 모델과 각 통신 모델에 대해 권장되는 네트워킹 기능에 대해 설명합니다.

통신모델	설명	예	권장 네트워킹 기능
데이터 처리	최근 값만 전송, 일반적으로 하나 이상의 타겟에서 호스트 컴퓨터로.	임베디드 컨트롤러가 온도 데이터를 수집하고 주기적으로 데이터 포인트를 호스트 컴퓨터로 전송하여 사용자가 시스템의 상태를 모니터링할 수 있는 어플리케이션.	공유 변수
스트리밍	높은 처리량을 보이는 모든 데이터 포인트 전송, 일반적으로 한 컴퓨터에서 데이터를 쓰면 다른 컴퓨터에서 해당 데이터를 읽어서 디스플레이하거나 처리함.	임베디드 컨트롤러가 모터에서 진동 데이터를 수집하여 해당 데이터의 모든 포인트를 분석과 처리를 위해 호스트 컴퓨터로 스트리밍하는 어플리케이션.	네트워크 스트림*
명령 기반	한 컴퓨터로부터(명령자라고 함) 대기시간이 낮은 데이터 전송이 이루어지고 이로 인해 다른 컴퓨터에서(작업자라고 함) 특정 이벤트가 트리거됨. 명령 기반의 통신은 일반적으로 드물고 데이터 포인트를 놓쳐서는 안됩니다.	사용자가 HMI(Human-Machine Interface) 어플리케이션에서 정지 버튼을 클릭하면, 해당 어플리케이션은 컨베이어 벨트를 컨트롤하는 CompactRIO 타겟으로 정지 명령을 보내 컨베이어 벨트를 멈춥니다.	네트워크 스트림*

* Windows와 Real-Time 시스템에서만 사용 가능.

사용자 컴퓨터에서 실행되는 OS와 데이터를 전송하는 어플리케이션에 따라, 위의 테이블에서 설명된 것 이외에 다른 네트워킹 기능을 사용해야 할 수도 있습니다. 다음 흐름도로 사용자 어플리케이션의 요구조건에 가장 적합한 네트워킹 기능을 결정합니다.



다음 네트워킹 기능을 LabVIEW에서 사용할 수도 있습니다:

- Bluetooth VI와 함수를 사용하여 Bluetooth 디바이스에 무선 연결을 설정합니다.
- IrDA 함수를 사용하여 IrDA(Infrared Data Association) 디바이스에 무선 연결을 설정합니다.
- **LabVIEW 웹 서버의 리모트 프런트패널**을 사용하여 VI 프런트패널을 원격으로 보거나 컨트롤합니다.
- **OPC 시스템에 연결하기**를 통해 산업용 네트워크의 실시간 데이터를 공유합니다.



노트

리얼타임 어플리케이션의 일반적인 경우처럼 어플리케이션이 시스템 시작 시 자동으로 로드되는 경우, 네트워크 설정이 완료되기 전에 어플리케이션이 시작될 수 있습니다. 시스템이 네트워크 인터페이스에 연결 요청을 보낼 때까지 네트워크 연결이 실패할 수 있습니다. [설정된 네트워크 기다림] VI를 사용하여, 시스템이 요청을 보낼 수 있을 때까지 자동 네트워크 연결 요청을 지연하는 것이 좋습니다.

네트워크 클라이언트 및 서버로서 LabVIEW

LabVIEW를 클라이언트로 사용하여 데이터를 읽거나 기능을 다른 어플리케이션에서 사용할 수 있고 또는 LabVIEW를 서버로 사용하여 LabVIEW의 기능을 다른 어플리케이션에 제공할 수 있습니다. 프로퍼티 노드를 통해 프로퍼티에 접근하고 인보크 노드를 통해 메소드를 실행함으로써 **VI 서버**를 사용하여 로컬과 원격 컴퓨터의 VI를 컨트롤할 수 있습니다.

컴퓨터 사이에서 통신하기 전에, 통신할 때 사용하는 네트워크를 정해야 합니다. 사용할 수 있는 프로토콜은 HTTP와 TCP/IP입니다. 어플리케이션에 따라 필요한 프로토콜이 다릅니다. 예를 들어, HTTP 프로토콜은 웹 문서를 작성하기에 이상적이지만, 이를 이용하여 다른 VI에서 작성한 데이터를 듣는 VI를 만들 수 없습니다. 그러기 위해서는, TCP 프로토콜을 사용합니다. LabVIEW는 컴퓨터간에 통신하기 위해 사용하는 여러가지 **하위 레벨 프로토콜**을 지원합니다.

(Windows) ActiveX 서버나 클라이언트로서 LabVIEW에서 **ActiveX 기술**을 사용할 수 있습니다.

어플리케이션 사이에서 데이터 스트리밍하기 및 명령 전송하기

데이터 스트리밍하기와 **명령 전송하기**는 두 개의 LabVIEW 어플리케이션 간에 데이터를 전송하는 **통신 모델**입니다. 네트워크 스트림을 사용하여 이러한 태스크를 수행할 수 있습니다. 네트워크 스트림은 데이터 손실이 없고 단일 방향 일대일 통신 채널로, 발신자와 수신자 **끝포인트**로 구성됩니다. 네트워크 스트림 함수 및 네트워크 스트림 끝포인트 프로퍼티를 사용하여 네트워크 스트림으로 어플리케이션을 디자인합니다.



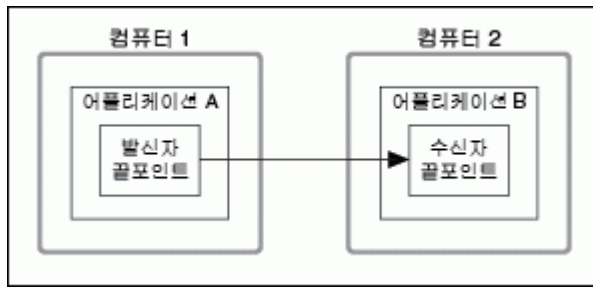
노트

네트워크 스트림은 Windows와 Real-Time 시스템에서만 사용할 수 있습니다.

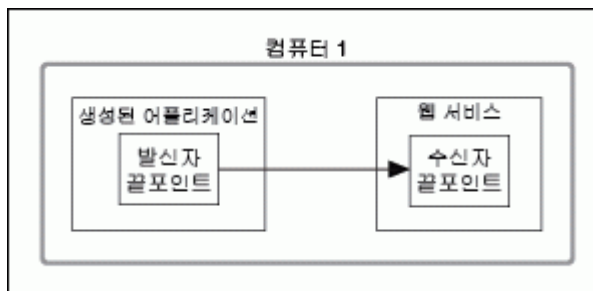
네트워크 스트림을 사용하여 LabVIEW 클래스 및 대부분의 참조 번호를 제외한 모든 LabVIEW 데이터 타입을 전송할 수 있습니다. 네트워크 스트림을 사용하여 전송할 수 있는 유일한 참조 번호는 비전 이미지 데이터 타입입니다. 가장 빠른 속도로 전송되는 데이터 타입은 다음과 같습니다:

- 숫자 스칼라
- 불리언
- 숫자 스칼라의 1D 배열
- 불리언의 1D 배열

다음 그림은 네트워크 스트림을 설정하여 데이터를 전송하는 몇 가지 방법을 보여줍니다.



위의 그림에서는 각각 다른 컴퓨터에 있는 두 개의 어플리케이션 사이에서 네트워크 스트림으로 데이터가 전송됩니다.

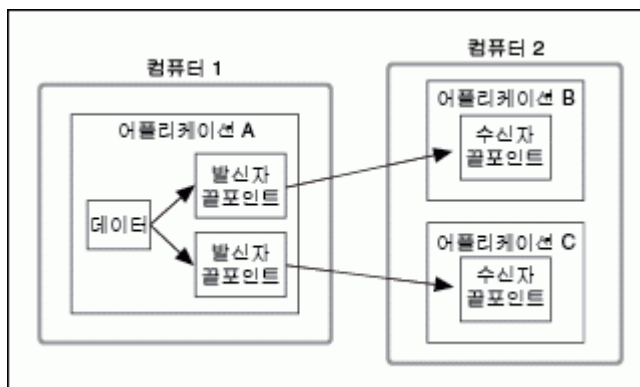


위의 그림에서는 빌드된 어플리케이션과 웹 서비스 사이에서 네트워크 스트림으로 데이터가 전송됩니다.

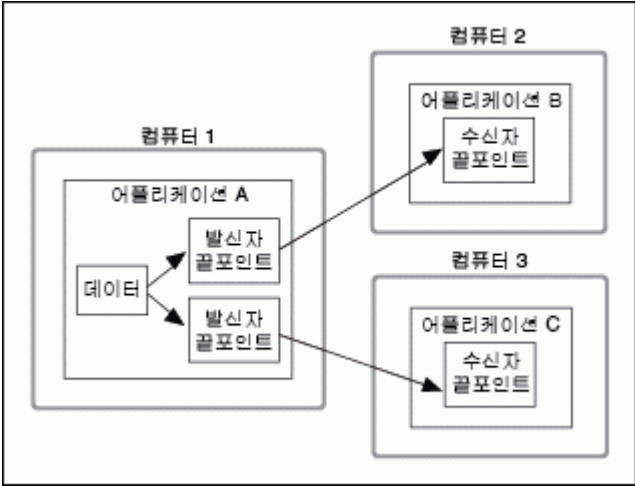


노트 네트워크 통신을 시작하려면 네트워크 스트림은 네트워크 상에서가 아닌 일반 데이터 전송 방법보다 더 많은 오버헤드를 필요로 합니다. 단일 어플리케이션 내의 루프 사이에서 데이터를 전송하는데, National Instruments는 다음 방법의 사용을 권장합니다:

- 큐 동작 함수
- (Real-Time Module) **RT FIFO** 함수
- (Real-Time Module) **RT FIFO가 활성화된 공유 변수**



위의 그림에서는 한 컴퓨터에 있는 하나의 어플리케이션에서 다른 컴퓨터에 있는 두 개의 어플리케이션으로 네트워크 스트림을 통해 데이터가 전송됩니다.

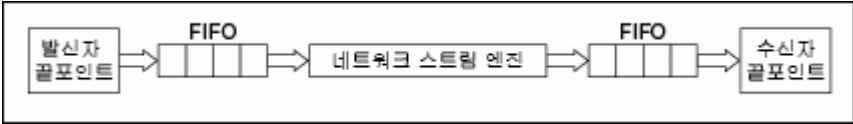


위의 그림에서는 한 컴퓨터에 있는 하나의 어플리케이션에서 각기 다른 컴퓨터에 있는 두 개의 어플리케이션으로 네트워크 스트림을 통해 데이터가 전송됩니다.

네트워크 스트림 엔진

각 끝 포인트는 FIFO 버퍼를 사용해서 데이터를 전송합니다. 네트워크 스트림 엔진(NSE)은 LogosXT를 사용해서 발신자 끝 포인트의 FIFO 버퍼로부터 수신자 끝 포인트의 FIFO 버퍼로 데이터를 전송합니다.

다음 그림은 네트워크 스트림에서 데이터가 흐르는 모습을 보여줍니다.



위의 그림에서 데이터는 다음의 순서로 전달됩니다.

- 1. 발신자 끝 포인트가 FIFO 버퍼에 데이터를 씁니다.
- 2. NSE가 네트워크를 통해 수신자 끝 포인트의 다른 FIFO 버퍼로 데이터를 전송합니다.
- 3. 수신자 끝 포인트가 위에 언급한 FIFO 버퍼로부터 데이터를 읽습니다.

공유 변수 대신 네트워크 스트림을 사용해야 하는 경우를 결정하기

데이터 세트의 최신값을 여러 컴퓨터에 공개하려면 공유 변수를 사용하십시오. 이와 반대로 한 대의 컴퓨터에서 모든 데이터 포인트를 로그하려면 네트워크 스트림을 사용하십시오.

예를 들어 압력계를 사용해서 파이프라인에서 천연 가스의 압력을 유지하도록 하는 펌프의 진동을 감지한다고 가정합니다. 펌프가 오작동하지 않도록 CompactRIO 타겟에서 진동 데이터를 처리하여 베어링의 결함을 모니터링합니다. 하지만, CompactRIO 타겟에는 데이터를 분석할만큼 충분한 메모리가 없습니다. 그러므로 이 데이터를 저장하고, 분석

하고, 디스플레이할만큼의 충분한 메모리를 가진 데스크탑 컴퓨터에 이 데이터를 보내야 합니다.

공유 변수는 데이터의 가장 최신값만을 공개하도록 최적화되어 있기 때문에, 매우 중요한 데이터 포인트를 놓칠 수도 있습니다. 그러나 네트워크 스트림은 모든 데이터 포인트를 데스크탑 컴퓨터로 스트림해서 사용자가 엔진의 상태를 모니터할 수 있도록 합니다.



노트 네트워크 스트림은 리얼타임(RT)이나 시간에 결정적인 루프에서는 지터를 야기할 수도 있습니다. 그러므로, 네트워크 스트림을 사용해서 시간에 결정적인 루프로부터 데이터를 스트림하려면, 먼저 우선 순위가 낮은 루프와 데이터를 공유하는 것을 권장합니다. 그 후, 네트워크 스트림을 사용해서 다른 어플리케이션으로 데이터를 스트리밍합니다.

네트워크 스트림을 사용해서 어플리케이션 디자인하기

네트워크 스트림을 사용하여 Windows와 Real-Time 시스템 지원 어플리케이션을 디자인할 수 있습니다:

- 데이터 스트리밍
- 명령 전송
- 끝포인트에 대한 정보 디스플레이

네트워크 스트림을 사용하여 어플리케이션을 설계하는 예제는 `labview\examples\Data Communication\Network Streams\Simple Network Streams` 디렉토리의 `Simple Network Streams.lvproj`를 참조하십시오.



관련 정보

[어플리케이션 사이에서 데이터 스트리밍하기](#) (615 페이지 참조)

[어플리케이션 사이에서 명령 전송하기](#) (617 페이지 참조)

[끝포인트에 대한 정보 디스플레이하기](#) (621 페이지 참조)

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

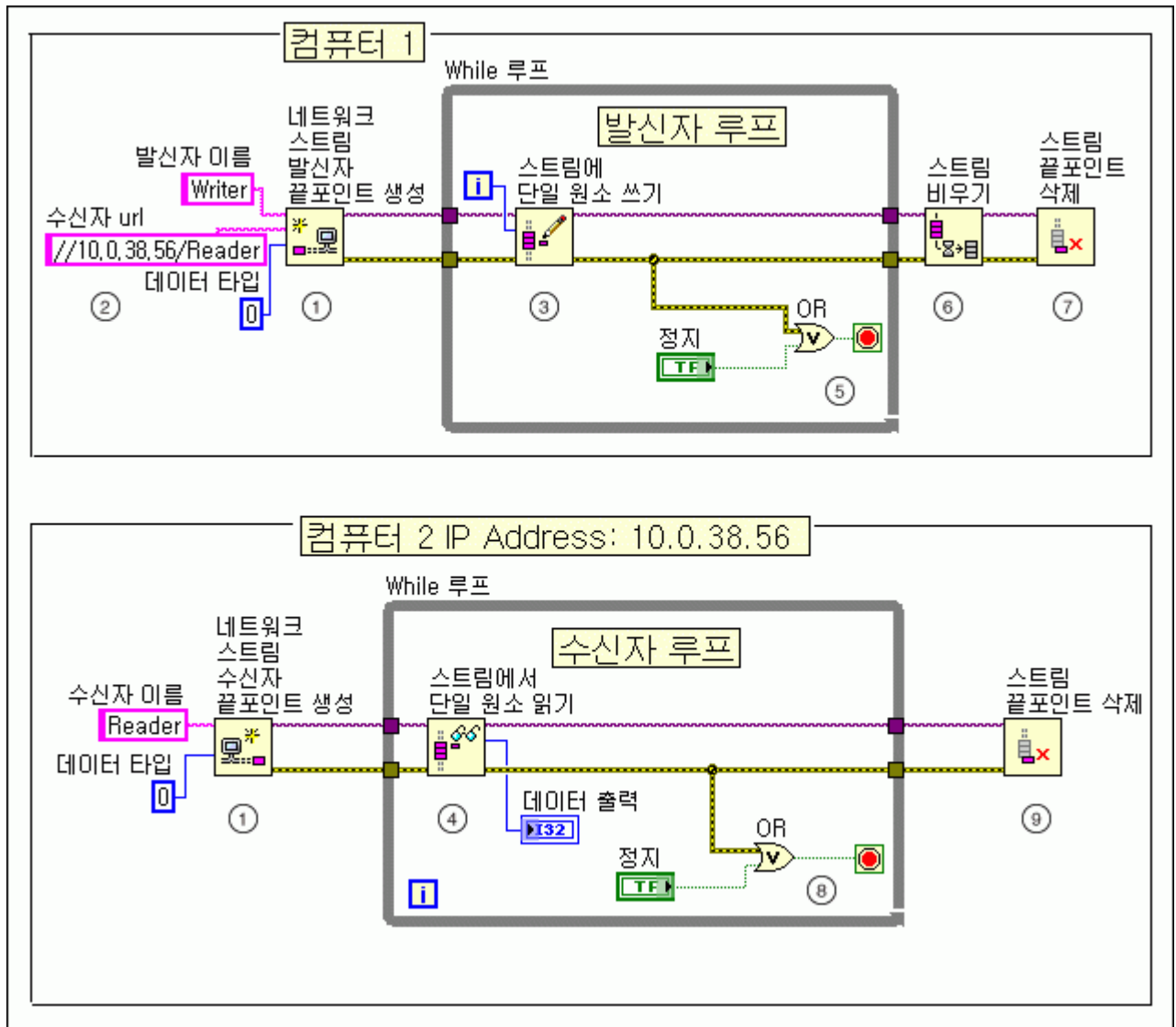
어플리케이션 사이에서 데이터 스트리밍하기

다음 그림은 네트워크 스트림 함수를 사용하여 각기 다른 컴퓨터에 있는 두 개의 어플리케이션 사이에서 데이터를 스트리밍하는 사례를 보여줍니다.



노트

네트워크 스트림은 Windows와 Real-Time 시스템에서만 사용할 수 있습니다.



다음과 같은 이벤트가 위의 그림에서 발생합니다:

	[네트워크 스트림 발신자 끝포인트 생성] 함수는 컴퓨터 1에 발신자 끝포인트를 생성하고, [네트워크 스트림 수신자 끝포인트 생성] 함수는 컴퓨터 2에 수신자 끝포인트를 생성합니다.
① 노트	[네트워크 스트림 발신자 끝포인트 생성]이나 [네트워크 스트림 수신자 끝포인트 생성] 함수는 아무 순서대로 실행해도 됩니다. 이러한 함수는 이 중 한 함수에 있는 타임아웃(ms) 터미널이 다른 함수가 실행되기 전에 만료되지 않는 한, 네트워크 스트림을 생성합니다.
②	발신자 끝포인트는 수신자 끝포인트의 끝포인트 URL을 사용해서 수신자 끝포인트와 연결을 형성합니다.

	쓰기 루프 내에서, [스트림에 단일 원소 쓰기] 함수는 While 루프의 반복 i 터미널 값을 지속적으로 스트림에 씁니다.
③	 노트 또한 [스트림에 멀티플 원소 쓰기] 함수를 사용해서 한 개 이상의 데이터 포인트를 동시에 스트리밍할 수 있습니다. 예를 들어, 이 함수를 사용하면 매 초마다 웨이브폼 데이터의 샘플 두 개를 한꺼번에 스트리밍할 수 있습니다.
④	 노트 또한 [스트림에서 멀티플 원소 읽기] 함수를 사용해서 스트림을 읽을 수도 있습니다. 다른 어플리케이션에서 [스트림에 단일 원소 쓰기] 함수를 사용했다 하더라도 [스트림에서 멀티플 원소 읽기] 함수를 사용할 수 있으며, 그 반대도 마찬가지입니다.
⑤	사용자가 컴퓨터1에서 정지 버튼을 클릭할 때까지 데이터는 지속적으로 스트리밍됩니다.
⑥	[스트림 비우기] 함수는 모든 남아있는 데이터를 수신자 끝포인트로 전송합니다.
⑦	[스트림 끝포인트 삭제] 함수는 발신자 끝포인트를 삭제합니다.
⑧	수신자 끝포인트는 연결이 끊겼기 때문에 에러를 받으며, 데이터 흐름은 While 루프를 종료합니다.
⑨	[스트림 끝포인트 삭제] 함수는 수신자 끝포인트를 삭제하고, 이로 인해 스트림이 삭제됩니다.

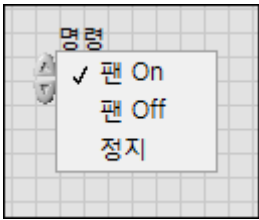
어플리케이션 사이에서 명령 전송하기

다음 단계를 따라 네트워크 스트림 함수를 사용하여 한 컴퓨터에서 다른 컴퓨터로 **명령**을 전송합니다.

 **노트**
네트워크 스트림은 Windows와 Real-Time 시스템에서만 사용할 수 있습니다.

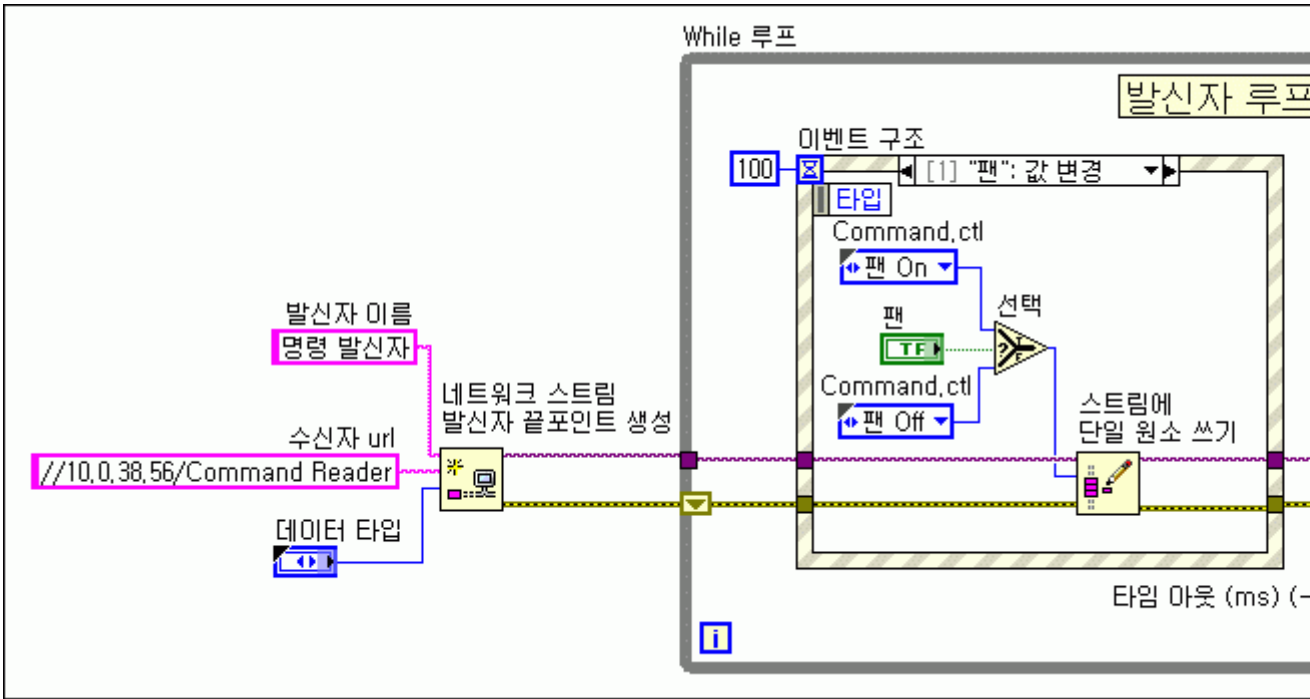
1. 각 명령에 대한 엔트리로 **열거형 타입 컨트롤**의 **타입 정의를 생성**합니다. 예를 들어, 다음 그림은 팬을 켜고, 팬을 끄고, 팬을 컨트롤하는 어플리케이션을 정지시키는 것과 같은 명령을 포함하는 열거형 타입 컨트롤의 타입 정의의 예를 보여줍니다.

노트 타입 정의를 생성하면 컨트롤의 한 인스턴스를 업데이트할 경우 다른 모든 인스턴스도 함께 업데이트됩니다.

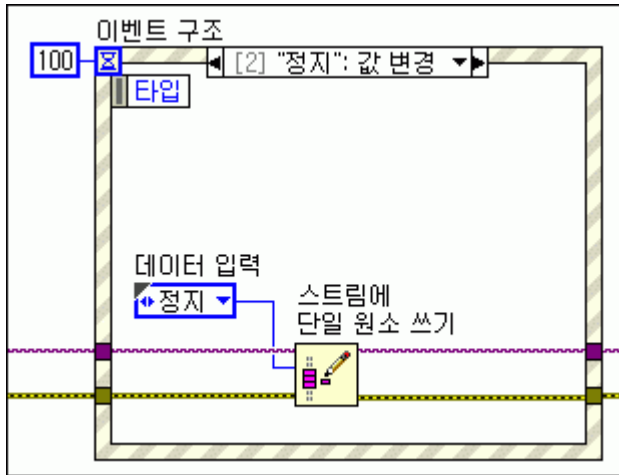


- 2. 데이터 스트리밍하기 과정과 같이 쓰기 루프와 읽기 루프를 생성합니다.
- 3. 쓰기 루프 내에서, 이벤트 구조를 설정하여 사용자 입력을 기반으로 하는 명령을 전송합니다. 예를 들어, 다음 그림에서, 사용자가 팬 버튼을 클릭할 때, [스트림에 단일 원소 쓰기] 함수는 팬 On 또는 팬 Off 명령을 전송합니다.

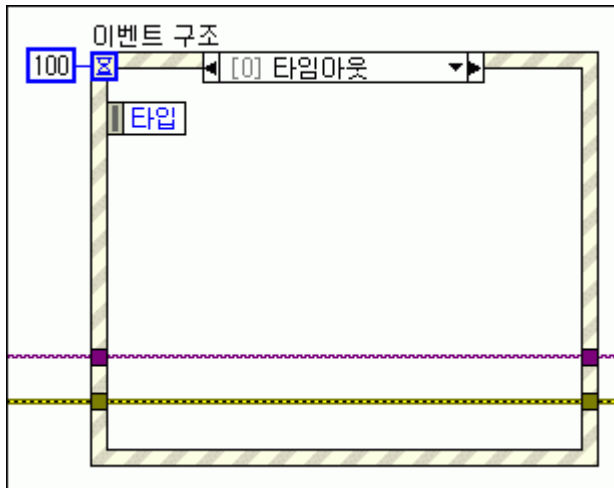
노트 쓰기 루프 내의 [스트림 비우기] 함수를 사용하여 다음 그림에서 보이는 것처럼 지연 시간을 최소화하고 네트워크를 통해 명령이 즉시 전달되도록 합니다.



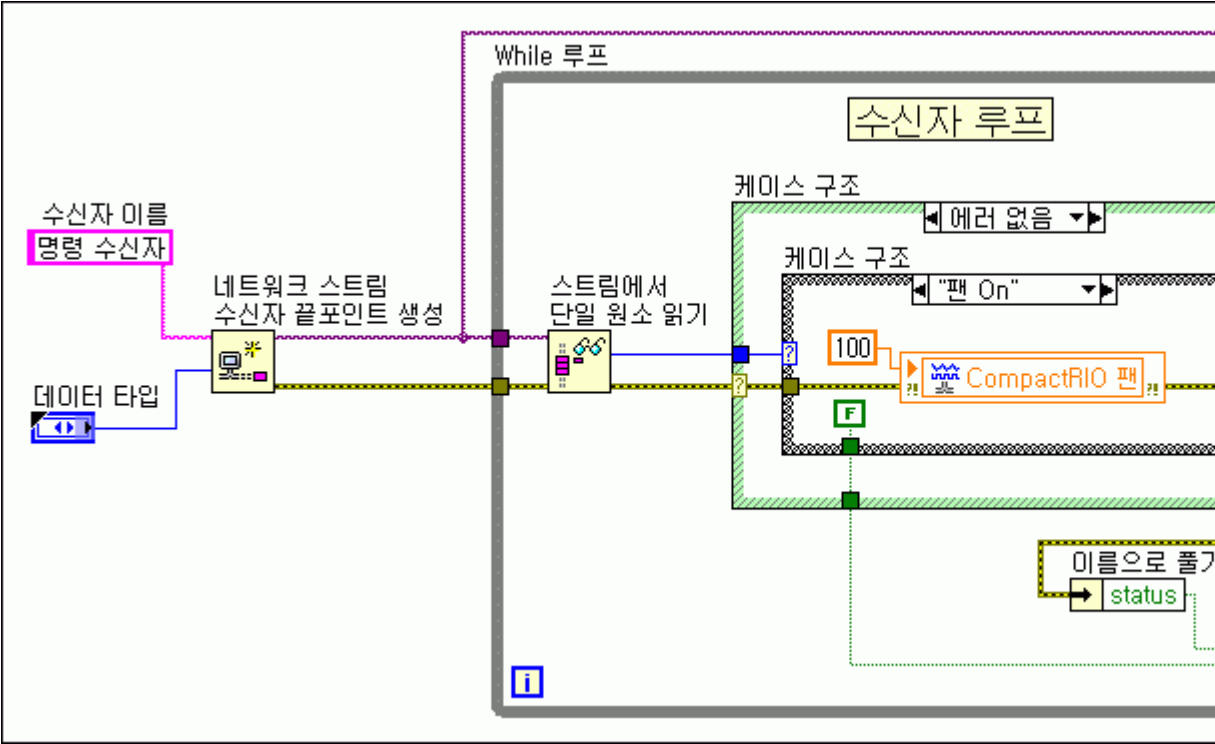
유사하게, 다음 그림에서 사용자가 정지 버튼을 클릭할 때, [스트림에 단일 원소 쓰기] 함수는 정지 명령을 전송합니다.



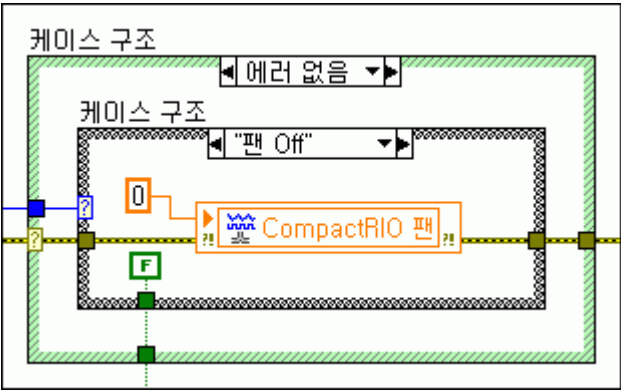
4. 다음 그림에서 보이는 것처럼, 타임아웃되는 경우 동작이 멈추도록 이벤트 구조를 설정합니다.



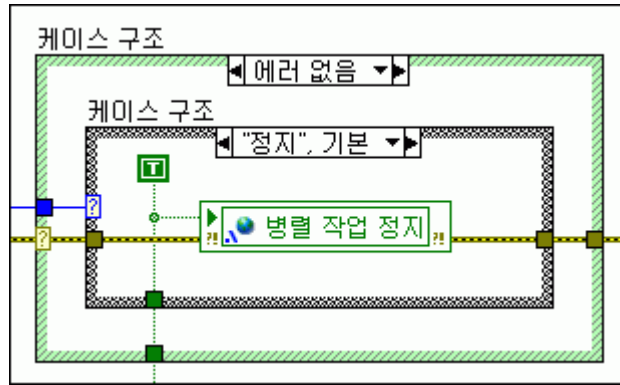
5. 읽기 루프 내에서, 명령을 분석하고 에러 조건을 처리합니다.
1. 케이스 구조를 읽기 루프 안에 놓고 [스트림에서 단일 원소 읽기] 함수의 에러 출력 터미널을 케이스 선택자에 연결합니다.
 2. 케이스 구조를 에러 없음 케이스 안에 놓고 [스트림에서 단일 원소 읽기] 함수의 데이터 출력 터미널을 케이스 선택자에 연결합니다. 이 케이스 구조를 사용하여 쓰기 루프에서 보낸 명령을 분석합니다. 예를 들어, 다음 그림에서, [스트림에서 단일 원소 읽기] 함수가 팬 On 명령을 받으면, 케이스 구조는 팬 On 케이스로 바뀌고 팬을 컨트롤하는 **I/O 변수**가 100의 강도로 설정되는데, 이로써 팬이 작동하게 됩니다.



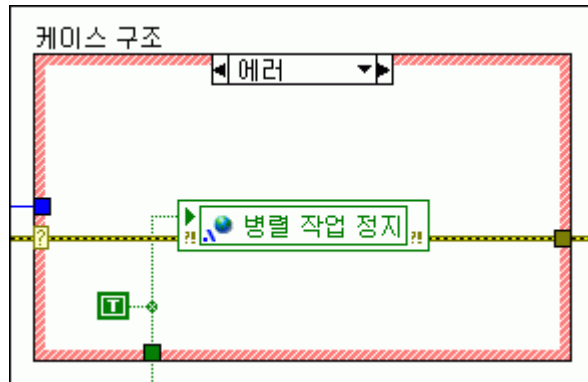
이와 마찬가지로, 다음 그림에서 [스트림에서 단일 원소 읽기] 함수가 팬 off 명령을 받으면 케이스 구조는 팬 off 케이스로 바뀌고 팬을 컨트롤하는 I/O 변수가 0의 강도로 설정되는데, 이로써 팬이 작동을 멈추게 됩니다.



마지막으로, 다음 그림에서 [스트림에서 단일 원소 읽기] 함수가 정지 명령을 받으면 케이스 구조는 정지 케이스로 바뀌고, 참 상수로 읽기 루프를 정지시키고, **단일 프로세스 공유 변수**를 사용하여 VI 내의 모든 병렬 작업을 정지시킵니다.



3. 에러 케이스 내에서, 다음 그림에서 보이는 것처럼, 코드를 작성하여 에러 발생 시 읽기 루프와 모든 병렬 작업을 정지시킵니다.

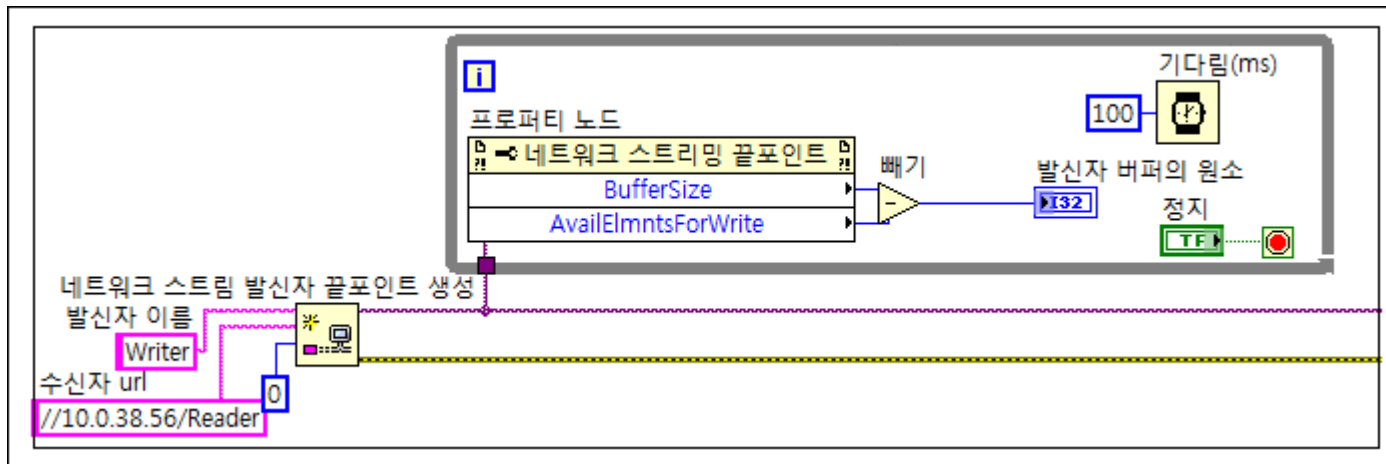


끝포인트에 대한 정보 디스플레이하기

네트워크 스트림 끝포인트 프로퍼티를 사용하여 **끝포인트**에 대한 정보를 디스플레이할 수 있습니다. 다음 단계를 따라 네트워크 스트림 끝포인트 프로퍼티를 사용합니다.

1. 블록 다이어그램에 네트워크 스트림 프로퍼티 노드를 추가합니다.
2. 끝포인트 참조 번호 와이어를 프로퍼티 노드의 참조 입력에 연결합니다. [네트워크 스트림 수신자 끝포인트 생성] 함수의 수신자 끝포인트 참조나, [네트워크 스트림 발신자 생성] 함수의 발신자 끝포인트 터미널, 또는 다른 아무 네트워크 스트림 함수의 끝포인트 출력 터미널에서 이 참조를 얻습니다.
3. 네트워크 스트림 끝포인트 프로퍼티 노드의 프로퍼티 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티 선택을 선택하여 원하는 프로퍼티를 선택합니다.
4. (옵션) 위치 도구를 사용해서 네트워크 스트림 끝포인트 프로퍼티 노드의 **크기를 조정**하면, 한 개 이상의 프로퍼티를 동시에 디스플레이할 수 있습니다.

다음 그림은 네트워크 스트림 끝포인트 프로퍼티의 사용 예를 보여줍니다.



이 예에서 버퍼 크기 프로퍼티에서 쓰기 가능한 원소를 빼면 발신자 끝포인트의 버퍼 안의 원소 개수가 나타납니다.

네트워크 스트림 끝포인트

끝포인트는 데이터를 교환하는 어플리케이션의 일부입니다. 모든 네트워크 스트림에는 발신자 끝포인트와 수신자 끝포인트가 반드시 있어야 합니다. 발신자 끝포인트는 스트림에 데이터를 씁니다. 수신자 끝포인트는 스트림으로부터 데이터를 읽습니다. LabVIEW는 각 끝포인트를 끝포인트 URL로 나타내는데, 이를 통해 끝포인트를 서로 연결할 수 있습니다. 두 개의 끝포인트가 연결될 때, 네트워크 스트림이 생성됩니다. 네트워크 스트림 함수를 사용하여 끝포인트를 생성하고 끝포인트 간에 데이터를 스트림합니다.



관련 정보

[네트워크 스트림 끝포인트 URL 지정하기](#) (622 페이지 참조)

[네트워크 스트림 끝포인트끼리 연결하기](#) (627 페이지 참조)

[네트워크 스트림을 사용해서 어플리케이션 디자인하기](#) (615 페이지 참조)

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

네트워크 스트림 끝포인트 URL 지정하기

LabVIEW는 모든 네트워크 스트림 끝포인트를 URL로 식별합니다. [네트워크 발신자 끝포인트 생성] 함수의 발신자 이름 터미널이나, [네트워크 수신자 끝포인트 생성] 함수의 수신자 이름 터미널을 사용해서 끝포인트의 이름을 지정하면, LabVIEW는 그 이름을 사용해서 다음의 구문으로 URL을 생성합니다.

```
ni.dex:// host_name : context_name / endpoint_name
```

다음 테이블에서는 이 URL의 각 구성요소를 설명합니다.



노트

각 컴퓨터에서는 단 한 개의 어플리케이션만이 기본 내용을 지정할 수 있습니다. 그러므로, **네트워크 스트림을 사용하는 한 대의 컴퓨터 안에 여러 개의 어플리케이션이 있을 경우**, 어플리케이션의 각 끝포인트에 이름을 지정하는 대신 URL을 지정해야 합니다.

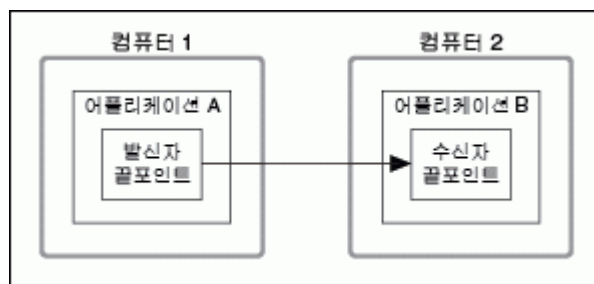
유효한 네트워크 스트림을 생성하려면, 끝포인트 URL을 사용해서 발신자와 수신자 끝포인트가 서로 연결되도록 요청합니다. 원격 끝포인트의 URL을 [네트워크 스트림 발신자 끝포인트 생성] 함수의 수신자 url 입력이나, [네트워크 스트림 수신자 끝포인트 생성] 함수의 발신자 url에 연결하여 이 작업을 수행합니다.

이 터미널에 지정해야 하는 URL은 원격 끝포인트의 네트워크 위치에 따라 다양합니다. 다음은 끝포인트가 서로 연결되도록 요청하는 여러 가지 예를 보여줍니다.



노트 끝포인트 URL은 대소문자를 구분하지 않습니다. 그러나, 끝포인트 URL을 지정할 때, 분석 에러가 발생하지 않게 하려면, 예약된 문자를 해당하는 **이스케이프 코드**로 대체해야 합니다.

네트워크 스트림을 사용하는 한 개의 어플리케이션을 실행하는 컴퓨터로 스트리밍하기

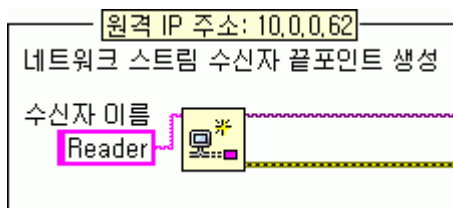


위의 그림에서처럼, 네트워크 스트림을 사용하는 한 개의 어플리케이션을 실행하는 원격 컴퓨터의 끝포인트에 연결할 때 끝포인트 간의 연결을 요청하려면, 다음의 구문을 가진 URL을 사용해야 합니다.

```
// host_name / endpoint_name
```

다음 단계를 따라 이 구문을 가진 URL을 사용하여 두 끝포인트 간의 연결을 요청합니다.

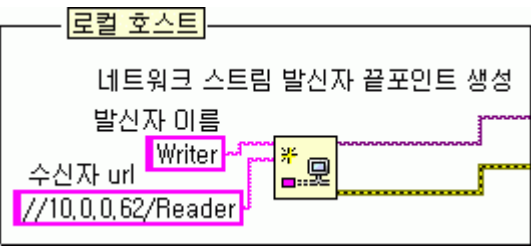
1. 다음 그림에 보이는 대로 [네트워크 스트림 수신자 끝포인트 생성] 함수를 사용해서 원격 컴퓨터에 수신자 끝포인트를 생성합니다.



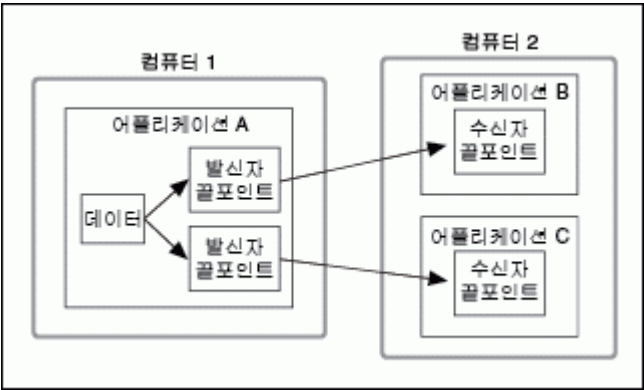
위의 그림에서 10.0.0.62는 host_name이고 Reader는 endpoint_name입니다. 따라서 이 끝포인트의 URL은 //10.0.0.62/Reader.

노트 LabVIEW는 다음의 순서로 호스트 이름의 유효한 해석을 검색합니다.

- 1. 타겟 이름 (LabVIEW 프로젝트에서 지정된 이름)
 - 2. DNS 이름
 - 3. IP 주소
2. 다음 그림에서 보이는 것처럼, [네트워크 스트림 발신자 끝포인트 생성] 함수를 사용해서 로컬 호스트에 발신자 끝포인트를 생성하고, 수신자 url 터미널을 사용해서 수신자 끝포인트의 URL을 지정합니다.



네트워크 스트림을 사용하는 여러 개의 어플리케이션을 실행하는 컴퓨터로 스트리밍하기

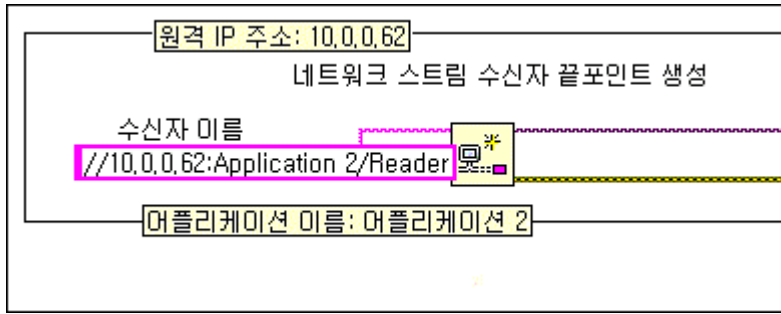


위의 그림에서처럼, 네트워크 스트림을 사용하는 여러 개의 어플리케이션을 실행하는 원격 컴퓨터의 끝포인트에 연결할 때 끝포인트 간의 연결을 요청하려면 다음의 구문을 가진 URL을 사용해야 합니다.

```
// host_name : context_name / endpoint_name
```

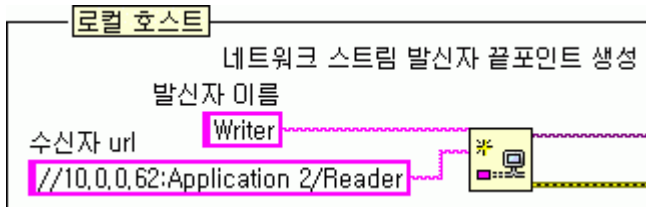
다음 단계를 따라 이 구문을 가진 URL을 사용하여 두 끝포인트 간의 연결을 요청합니다.

- 1. 다음 그림에 보이는 대로 [네트워크 스트림 수신자 끝포인트 생성] 함수를 사용해서 원격 컴퓨터에 수신자 끝포인트를 생성합니다.

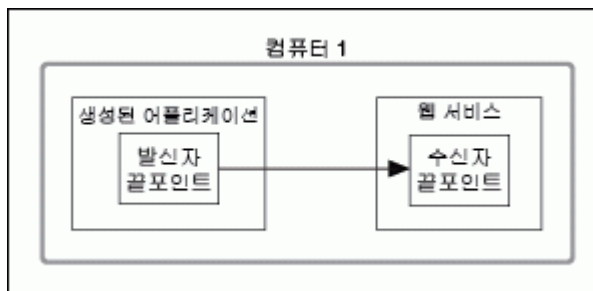


네트워크 스트림을 사용하는 여러 개의 어플리케이션을 실행하는 컴퓨터에 끝포인트를 생성할 때는 위의 그림에서 보는 것처럼 URL에 반드시 host_name, context_name 및 endpoint_name이 포함되어야 합니다. 이름 대신에 URL을 지정하면, 한 대의 컴퓨터에 있는 여러 개의 어플리케이션에서 같은 내용을 사용하지 않게되며, 이로써 같은 내용을 사용하기 때문에 두 끝포인트가 연결되지 못하는 상황을 방지할 수 있습니다.

2. 다음 그림에서 보이는 것처럼, [네트워크 스트림 발신자 끝포인트 생성] 함수를 사용해서 로컬 호스트에 발신자 끝포인트를 생성하고, 수신자 url 터미널을 사용해서 수신자 끝포인트의 URL을 지정합니다.



로컬 호스트의 어플리케이션으로 스트리밍하기

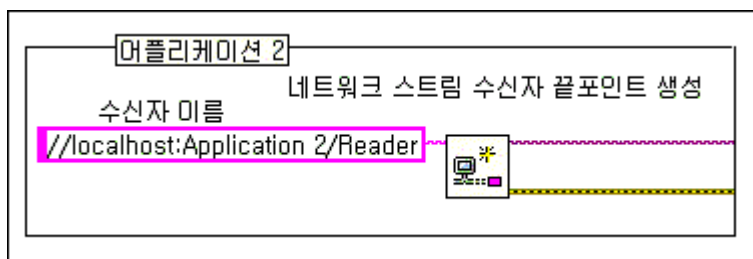


위의 그림에서처럼 로컬 호스트의 다른 어플리케이션 안에 있는 끝포인트에 연결할 때, 끝포인트 간의 연결을 요청하려면 다음의 구문을 가진 URL을 사용해야 합니다.

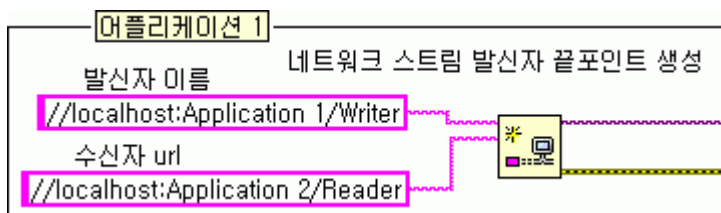
```
// localhost : context_name / endpoint_name
```

다음 단계를 따라 이 구문을 가진 URL을 사용하여 두 끝포인트 간의 연결을 요청합니다.

1. 다음 그림에 보이는 대로 [네트워크 스트림 수신자 끝포인트 생성] 함수를 사용해서 한 개의 어플리케이션 안에서 수신자 끝포인트를 생성합니다.



2. 다음 그림에서 보이는 것처럼, [네트워크 스트림 끝포인트 생성] 함수를 사용해서 다른 어플리케이션 안에 발신자 끝포인트를 생성하고, 수신자 url 터미널을 사용해서 수신자 끝포인트의 URL을 지정합니다.



노트 로컬 호스트가 네트워크 스트림을 사용하는 두 개의 어플리케이션을 실행하기 때문에 끝포인트에 이름 대신 URL을 지정해야 합니다.

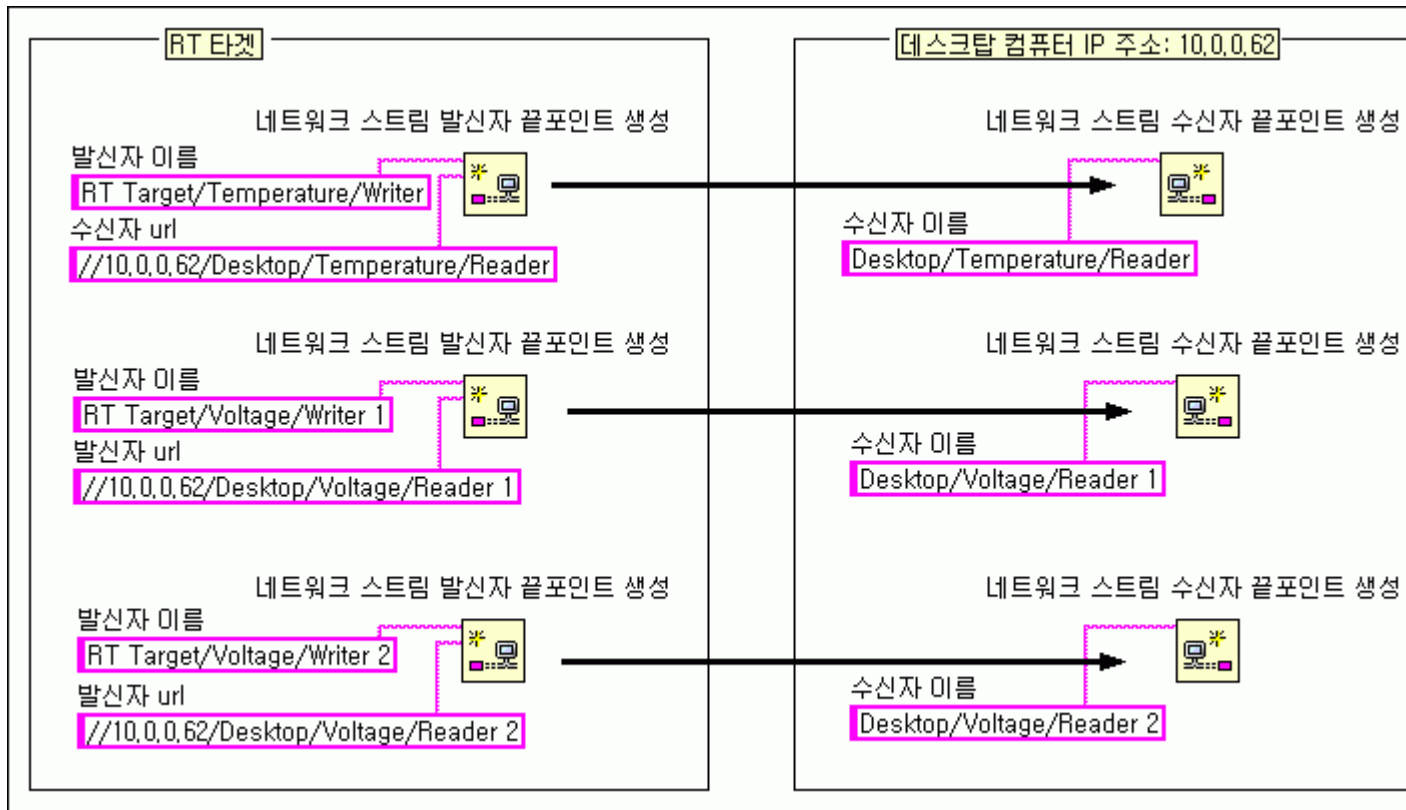
네트워크 스트림 끝포인트 이름 정리하기

끝포인트가 스트리밍하는 데이터, 끝포인트가 자리한 컴퓨터 또는 다른 특성 등을 묘사하는 부분을 끝포인트의 이름에 첨부해서 끝포인트를 정리할 수 있습니다. 여러 대의 컴퓨터에 여러 개의 끝포인트가 있을 때 이런 방법을 사용해서 끝포인트를 정리하면, 각 발신자 끝포인트가 대응하는 수신자 끝포인트에 연결되도록 할 수 있습니다.

다음 단계를 따라 끝포인트를 이름 별로 정리합니다.

1. [네트워크 스트림 발신자 끝포인트 생성] 함수의 발신자 이름 터미널 또는 [네트워크 스트림 수신자 끝포인트 생성] 함수의 수신자 이름 터미널을 연결합니다.
2. 이 터미널에 지정한 이름의 각 부분 사이에 슬래시를 놓습니다.

예를 들어, Real-Time 타겟에 있는 한 개의 어플리케이션 안에 세 개의 발신자 끝포인트가 있다고 가정해 봅시다. 하나는 온도를 측정하고, 다른 두 개는 압력 신호를 측정합니다. 이 끝포인트는 각각 데스크탑 컴퓨터 안에 있는 어플리케이션에 있는 세 개의 대응하는 수신자 끝포인트와 연결됩니다. 다음 그림은 각 끝포인트에 지정할 수 있는 이름의 예를 보여줍니다. 화살표는 수신자 끝포인트가 어느 발신자 끝포인트로 연결되어야 할 것인지 나타냅니다.



위의 그림에서 각 끝포인트의 이름은 끝포인트가 있는 타겟, 끝포인트가 스트리밍하는 데이터 타입과 더불어 끝포인트가 데이터를 읽는지, 쓰는지 나타냅니다. 이렇게 이름을 지정하면 어떤 끝포인트가 어떤 끝포인트와 대응하는지 나타낼 수 있습니다.



노트

끝포인트의 이름에는 같은 어플리케이션 안에 있는 다른 끝포인트의 이름 일부를 사용할 수 없습니다. 예를 들어, 위의 예에서 발신자 끝포인트를 RT Target/Voltage라고 이름지은 후, 다른 발신자 끝포인트를 RT Target/Voltage/Writer 1이라고 이름지을 경우, 네트워크 스트림을 생성하는 데 어려움이 발생할 수 있습니다.



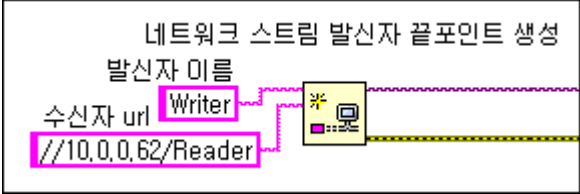
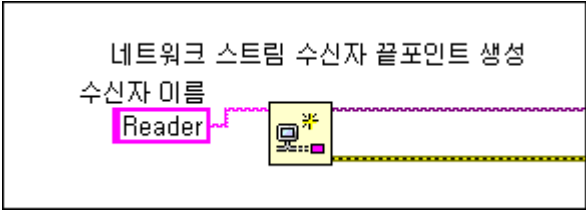
데이터 통신 방법

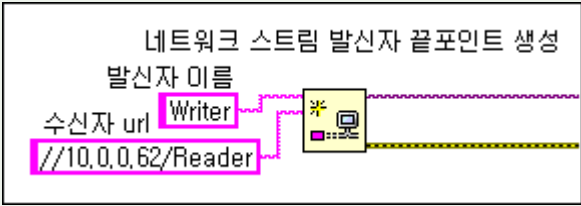
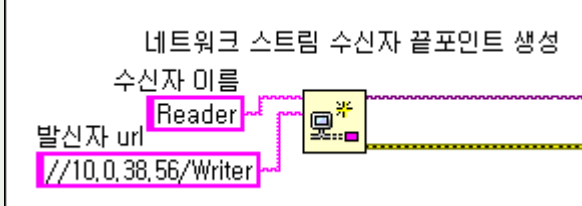
네트워크 스트림 끝포인트끼리 연결하기

[네트워크 스트림 발신자 끝포인트 생성] 함수는 실행 시 발신자 끝포인트를 생성하고 수신자 url 입력에서 지정한 URL을 사용하여 [네트워크 스트림 수신자 끝포인트 생성] 함수로 생성한 수신자 끝포인트에 연결합니다. 끝포인트끼리 연결되면 유효한 네트워크 스트림이 생성됩니다.

이 입력을 연결하는 것은 유효한 네트워크 스트림을 생성하는 여러 방법 중의 하나입니다. 또한 수신자 url 입력을 연결하지 않은 상태로 놔두어도 괜찮습니다. 그러나 이번 경우에는 반드시 발신자 끝포인트의 URL을 [네트워크 스트림 수신자 끝포인트 생성] 함수의 발신자 url 입력에 연결해야 합니다. 이 두 방법은 동등하며, 수신자와 발신자 끝포인트를 연결함으로써 유효한 네트워크 스트림을 생성합니다.

다음 테이블은 이 두 방법의 사용 예제를 보여줍니다.

발신자 끝포인트	수신자 끝포인트	노트
네트워크 스트림 발신자 끝포인트 생성 발신자 이름 수신자 url Writer //10.0.0.62/Reader 	네트워크 스트림 수신자 끝포인트 생성 수신자 이름 Reader 	끝포인트를 이관하면, 보낼 데이터가 수신자 끝포인트와 연결됩니다. 그러므로, 수신자 끝포인트에 연결된 발신자 끝포인트

발신자 끝포인트	수신자 끝포인트	노트
네트워크 스트림 발신자 끝포인트 생성 발신자 이름 수신자 url Writer //10.0.0.62/Reader 	네트워크 스트림 수신자 끝포인트 생성 수신자 이름 Reader 발신자 url //10.0.0.38.56/Writer 	끝포인트를 이관하면, 수신자 끝포인트가 발신자 끝포인트와 연결됩니다. 그러므로, 발신자 끝포인트에 연결되는 수신자 끝포인트

데이터 통신 방법

네트워크 스트림 끝포인트 버퍼에 메모리 할당

네트워크 스트림 **끝포인트**는 FIFO 버퍼를 사용해서 데이터를 전송합니다. [네트워크 스트림 발신자 끝포인트 생성] 및 [네트워크 스트림 수신자 끝포인트 생성] 함수의 데이터 타입 및 버퍼 크기 입력이 버퍼가 메모리를 얼마나 필요로 하는지 결정합니다. 이러한 함수의 원소 할당 모드 입력은 초기 버퍼 메모리를 스칼라 타입이 아닌 데이터 타입에 할당할 시점을 결정합니다.

할당 모드와 관계없이, 사용자 어플리케이션이 사용하는 메모리 양에 영향을 미치는 것은 버퍼 크기입니다. 어플리케이션을 테스트할 때는 항상 **메모리 사용량을 모니터**하십시오.

네트워크 스트림 끝포인트 버퍼의 할당 모드 지정하기

[네트워크 스트림 발신자 끝포인트 생성] 및 [네트워크 스트림 수신자 끝포인트 생성] 함수의 원소 할당 모드를 사용하여 할당 모드를 지정합니다.



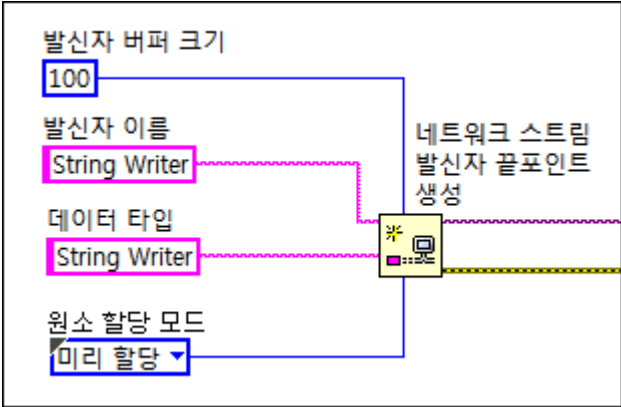
노트

원소 할당 모드 입력은 스칼라 데이터 타입의 전송에 영향을 미치지 않습니다. 스칼라 데이터 전송은 미리 할당 모드에서 수행됩니다.

어플리케이션이 스칼라 타입이 아닌 데이터를 전송하고 실행 전에 각 데이터 원소의 최대 크기를 알 수 없는 경우, 필요할 때 할당 모드를 사용하십시오. 필요할 때 할당 모드에서 LabVIEW는 네트워크 스트림이 생성될 때 각 데이터 원소의 포인터 또는 핸들을 생성하는데 필요한 양의 메모리만 할당합니다. LabVIEW는 구 후 각 원소가 버퍼에 들어갈 때 추가 메모리를ダイナ믹하게 할당합니다.

어플리케이션이 스칼라 타입이 아닌 데이터를 전송하고 실행 전에 각 데이터 원소의 최대 크기를 알 수 있는 경우, 미리 할당 모드를 사용하십시오. 미리 할당 모드에서 [네트워크 스트림 발신자 끝포인트 생성] 및 [네트워크 스트림 수신자 끝포인트 생성] 함수는 데이터 타입 입력 및 발신자 버퍼 크기 또는 수신자 버퍼 크기 입력에 근거하여 처음 버퍼 메모리를 할당합니다.

끝포인트 버퍼를 미리 할당하려면 데이터 타입 입력에 예상 크기가 최대인 원소를 연결해야 합니다. 다음 그림은 끝포인트가 생성될 때 100개의 문자열 원소에 각각 예제 문자열과 같은 최대 크기의 메모리를 할당하는 [네트워크 스트림 발신자 끝포인트 생성] 함수를 보여줍니다.



노트 실행 시 미리 할당된 메모리 양보다 원소가 더 많은 메모리를 필요로 하는 경우, 추가적인 메모리가ダイナ믹하게 할당됩니다.

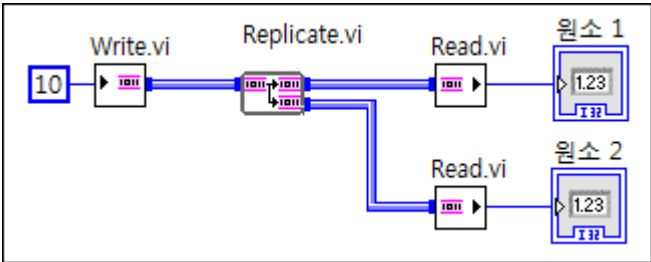


데이터 통신 방법

채널 복제하기

채널을 복제하면 채널에 데이터 복사본이 생성되고, 이 채널이 두 개의 다른 수신자 끝포인트로 연결되는 두 개의 발신 채널로 분리됩니다. LabVIEW는 이 수신자 끝포인트 복사본을 같은 채널 템플릿의 복제 끝포인트로 나타냅니다. 복제 끝포인트는 한 개의 채널 입력과 두 개의 채널 출력을 가집니다. 두 개 채널 출력은 각각 채널 입력으로부터 개별 데이터 복사본을 받고, 각각 개별적으로 받은 데이터 복사본을 읽습니다.

다음 그림은 스트림 템플릿의 복제 채널을 보여줍니다.



노트
채널을 복제하려면, 채널 와이어에서 마우스 오른쪽 버튼을 클릭한 후 삽입>복제를 선택합니다. 삽입>복제 옵션은 일부 템플릿 채널에 대해서만 사용할 수 있습니다.

타입 전달

타입 전달은 LabVIEW 컴파일 과정 중, 여러 데이터 타입에 따라 데이터 타입을 변경하는 터미널에 적합한 데이터 타입을 전달하고 구문 에러를 감지하는 분석 단계입니다. 이 단계는 VI를 편집할 때마다 실행됩니다.

예를 들어, 정수를 [감소] 함수의 입력에 연결하는 경우 이 출력의 데이터 타입은 정수입니다. 입력에 부동소수를 전달하는 경우, 출력 역시 부동소수입니다. 배열과 클러스터 역시 같은 식입니다. 블록다이어그램에서 입력 데이터 타입에 기반하여 출력 데이터 타입을 결정하는 것을 타입 전달이라고 합니다. [감소] 함수의 입력에 열거형을 연결하는 경우, 이 함수에서 열거형을 감소하는 것은 가능한 연산이 아니기 때문에 블록다이어그램에서 이 타입을 전달할 수 없고, VI는 구문 에러를 보고합니다.

타입 설명자

LabVIEW는 블록다이어그램의 각 와이어와 터미널을 데이터 타입과 연관시킵니다. LabVIEW는 타입 설명자라고 불리는 메모리의 구조를 사용하여 이 타입을 추적합니다. 타입 설명자는 LabVIEW에서 모든 데이터 타입을 나타낼 수 있는 32비트 정수 시퀀스입니다. 따로 언급되지 않은 이상, 숫자값은 16진법 포맷으로 표기됩니다.

타입은 집합을 이룰 수 있습니다. 이 집합은 여러가지 다른 타입으로 구성될 수 있습니다. 예를 들어 불리언 배열에서, 배열이 하나의 타입이고 배열의 불리언 원소는 또 다른 타입입니다. LabVIEW에서 이러한 복합 타입의 모든 타입 설명자는 리스트에 저장됩니다.

읽을 수 있는 포맷

[패턴화된 문자열로] 함수 또는 [배리언트를 패턴화된 문자열로] 함수를 사용하여 타입 설명자를 여러 다른 포맷으로 읽을 수 있습니다.



노트

LabVIEW가 타입 설명자를 저장하는 포맷이 LabVIEW 8.0에서 변경되었습니다. LabVIEW 7.x와 그 이전 버전은 16비트의 패턴 형태로 타입 설명자를 저장합니다. LabVIEW 8.0과 그 이후 버전은 32비트의 패턴 형태로 타입 설명자를 저장합니다. [패턴화된 문자열로] 함수의 7.x 데이터 변환 모드를 사용하는 경우, LabVIEW 7.x 및 이전 버전의 정보가 적용됩니다. 다음 정보는 LabVIEW 8.0 또는 이후 버전에 적용됩니다. NI는 **배리언트** 데이터 타입을 사용하여 이러한 포맷 변경이 야기할 수 있는 충돌을 방지할 것을 권장합니다.

[패턴화된 문자열로] 함수를 사용하여 배리언트 데이터를 패턴화하는 경우, LabVIEW는 배리언트와 그 속성을 비롯하여 모든 내용을 패턴화합니다. 이 함수를 사용할 때 타입 설명자 저장 버퍼의 포맷은 다음과 같습니다.

32비트	nTDs TDs
var size	nTypesUsed
var size	TypesUsed

어디서 nTDs	는 버퍼에 포함된 원소 타입 설명자의 개수입니다.
TDs	는 nTDs 타입 설명자의 리스트입니다.

var size	는 변수의 크기로 나타내는 값에 따라 16 또는 32비트의 변수로 저장됩니다. 값이 32768보다 작은 경우 두 바이트로 저장됩니다. 값이 32768이거나 그 이상인 경우 네 바이트로 저장되며 높은 비트는 1로 설정됩니다.
nTypesUsed	는 버퍼를 참조하는 객체가 사용한 타입 설명자의 실제 개수입니다.
TypesUsed	는 사용되는 모든 TDs 타입의 인덱스 리스트입니다.

[배리언트를 패턴화된 문자열로] 함수를 사용하는 경우, LabVIEW는 배리언트만 패턴화하고 모든 속성을 삭제합니다. 이 함수를 사용할 때 타입 설명자의 포맷은 다음과 같습니다.

[크기] [타입 코드] [타입 특정 정보] [이름 (해당되는 경우)]

어디서 [크기]	는 바이트로 나타낸 첫 번째이며, 크기 워드를 포함합니다. [크기]의 최소값은 4입니다. 모든 타입 설명자에 이름 (파스칼 문자열)을 추가할 수 있습니다. 이렇게하면 [크기]의 값은 이름의 길이를 2의 배수로 반올림한만큼 증가됩니다. 크기는 16비트입니다.
[타입 코드]	는 두 번째 단어입니다. LabVIEW는 내부적으로 사용하도록 타입 코드의 높은 자리 바이트를 예약합니다. 두 개의 타입 설명자가 동등한지 비교할 때는 이 바이트를 무시해야 합니다. 타입 코드의 높은 자리 바이트가 같지 않은 경우에도 두 개의 타입 설명자는 같습니다. 타입 코드는 16비트입니다.
[타입 특정 정보]	는 타입 설명자에 대한 추가적인 정보입니다. 배열과 클러스터는 구조 또는 여러 데이터 타입의 집합인데, 그 이유는 다른 여러 타입의 참조를 포함하기 때문입니다. 예를 들면, 클러스터 타입은 그 원소 각각의 타입에 대한 추가적인 정보를 가지고 있습니다.
[이름]	은 파스칼 문자열입니다. 이름은 있을 수도 있고 없을 수도 있습니다.

타입 코드는 다음의 테이블에 리스트되어 있는 것처럼 단정도 또는 확장형 부동소수와 같은 실제 타입 정보를 인코딩합니다. 이 타입 코드 값은 LabVIEW의 향후 버전에서 변경될 수 있습니다.

데이터 타입

다음 테이블은 숫자 및 비숫자 데이터 타입, 타입 코드, 타입 설명자의 리스트입니다. 타입 코드 앞에 [크기]의 최소값이 나타납니다. 일부 데이터 타입의 경우, [타입 특정 정보]가 가질 수 있는 값이 타입 코드 뒤에 나타나기도 합니다.

	노트 모든 타입 설명자가 이름을 추가로 가질 수 있습니다. 이름은 파스칼 문자열입니다.

데이터 타입	타입 설명자(16진법의 숫자)
8비트 정수	0004 xx 01

데이터 타입	타입 설명자(16진법의 숫자)
16비트 정수	0004 xx 02
32비트 정수	0004 xx 03
64비트 정수	0004 xx 04
부호없는 8비트 정수	0004 xx 05
부호없는 16비트 정수	0004 xx 06
부호없는 32비트 정수	0004 xx 07
부호없는 64비트 정수	0004 xx 08
단정도 부동소수	0004 xx 09
배정도 부동소수	0004 xx 0A
확장형 부동소수	0004 xx 0B
복소수 단정도 부동소수	0004 xx 0C
복소수 배정도 부동소수	0004 xx 0D
복소수 확장형 부동소수	0004 xx 0E
열거형 8비트 정수	<nn> xx 15 <k><k pstrs>
열거형 16비트 정수	<nn> xx 16 <k><k pstrs>
열거형 32비트 정수	<nn> xx 17 <k><k pstrs>
단정도 물리량	<nn> xx 19 <k><k base-exp>
배정도 물리량	<nn> xx 1A <k><k base-exp>
확장형 물리량	<nn> xx 1B <k><k base-exp>
복소수 단정도 물리량	<nn> xx 1C <k><k base-exp>
복소수 배정도 물리량	<nn> xx 1D <k><k base-exp>
복소수 확장형 물리량	<nn> xx 1E <k><k base-exp>
불리언	0004 xx 21
문자열	0008 xx 30 <dim>
경로	0008 xx 32 <dim>
그림	0008 xx 33 <dim>
배열	<Nn> xx 40 <k dims><원소 타입 설명자>

데이터 타입	타입 설명자(16진법의 숫자)
클러스터	<Nn> xx 50 <k elems><원소 타입 설명자>
웨이브폼	<Nn> xx 54 <웨이브폼 타입><원소 타입 설명자>
참조 번호	<nn> <참조 번호 타입 코드>
배리언트	<Nn> xx 53
nn = 길이, xx = LabVIEW에 의해 예약됨, k = 숫자, k pstrs = 파스칼 문자열의 개수, k base-exp = 밑지수 쌍의 개수, k dims = 차원의 개수, k elems = 원소의 개수, dim은 32 비트 정수입니다.	

앞의 테이블에서 볼 수 있는 것처럼 타입 설명자의 크기 필드에서 최소값은 4입니다. 그러나, 모든 타입 설명자가 이름(파스칼 문자열)을 추가로 가질 수 있습니다. 이 경우에, 크기 필드는 이름의 길이(파스칼 문자열)를 2의 배수로 반올림한 것만큼 커집니다.

열거형 8비트 정수

am, fm, fm_stereo 아이템에 대한 이름이 없는 열거형 8비트 정수의 다음 예제에서, 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

0016 0015 0003 0261 6D02 666D 0966 6D20 7374 6572 656F

타입 설명자는 다음과 같은 워드를 가집니다.

- 0016—전체 22바이트를 나타냅니다.
- 0015—열거된 8비트 정수의 타입 코드를 나타냅니다.
- 0003—3개의 아이템이 있다는 것을 나타냅니다.
- 0261 6D02 666D 0966 6D20 7374 6572 656F—타입 코드별 정보를 나타냅니다. 예를 들어, 0261 6D는 문자열의 크기를 16진수로 나타내는 02 접두사가 있는 am의 ASCII 인코딩입니다.

동일한 열거형 8비트 정수가 radio라는 이름으로 지정된 경우, 해당 타입 설명자는 다음과 같이 해석될 수 있습니다.

001C 4015 0003 0261 6D02 666D 0966 6D20 7374 6572 656F 0572 6164 696F

다음은 두 설명자를 비교한 내용입니다.

①	길이가 다름
②	타입 코드가 동일함
③	열거형 값의 형이 동일함
④	이름, 파스칼 문자열 radio

물리량

m/s의 단위를 가진 배정도 물리량의 다음 예제에서, 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

000E 001A 0002 0002 FFFF 0003 0001

타입 설명자는 다음과 같은 워드를 가집니다.

- 000E—전체 14 바이트를 나타냅니다
- 001A—배정도 물리량의 타입 코드를 나타냅니다.
- 0002—2개의 밀-지수 쌍을 나타냅니다.
- 0002—2번째 밀 인덱스를 나타냅니다.
- FFFF (-1)—초의 지수입니다.
- 0003—미터의 밀 인덱스를 나타냅니다.
- 0001—미터의 지수입니다.

LabVIEW는 물리량을 디스플레이하기 위해 사용되는 단위에는 상관없이 모든 물리량을 기본 단위로 내부적으로 저장합니다.

다음 테이블은 LabVIEW가 사용하는 9개의 밀과 밀의 형을 인덱스 0에서 8까지로 보여줍니다.

물리량 이름	단위	약자	밀 값
평면 각도	라디안	rad	0
입체 각도	입체 호도법	sr	1
시간	초	s	2
길이	미터	m	3
질량	킬로그램	kg	4
전류	암페어	A	5
열역학적인 온도	켈빈	K	6
물질의 양	몰	mol	7
광도	칸델라	cd	8

문자열, 경로, 그림 데이터 타입

문자열, 경로, 그림 데이터 타입은 배열 차원 크기와 비슷한 32비트 길이를 가지고 있습니다. 그러나 현재 인코드되어 있는 유일한 값은 16진수 FFFFFFFF이며, 이는 크기가 변할 수 있다는 것을 나타냅니다. 현재 모든 문자열, 경로, 그림은 크기가 다양합니다. 실제 길이는 데이터와 함께 저장됩니다.

배열과 클러스터 데이터 타입

배열과 클러스터 데이터 타입 각각이 그 자신의 타입 코드를 가지고 있습니다. 또한 배열과 클러스터 데이터 타입은 그 원소의 데이터 타입과 배열의 차원이나 클러스터의 원소의 수에 대한 추가적인 정보를 가지고 있습니다.

배열

배열을 위한 타입 코드는 40입니다. 배열의 타입 설명자는 다음과 같은 포맷을 가집니다.

<Nn> xx 40 <k dims><원소 타입 설명자>

배정도 1D, 부동 소수에 대한 타입 설명자의 다음 예제를 고려합니다. 여기서 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

000E 0040 0001 FFFF FFFF 0004 000A [타입 특정 정보]

타입 설명자는 다음과 같은 워드를 가집니다.

- 000E—원소 타입 인덱스를 포함하여 전체 타입 설명자의 길이이며, 총 12바이트입니다.
- 0040—배열의 타입 코드를 나타냅니다.

- 0001—배열의 차원 수인 1차원을 나타냅니다.
- FFFF FFFF—차원의 크기를 나타내며, 이 크기는 변경이 가능합니다. LabVIEW는 실제 차원 크기를 저장하는데, 이것은 데이터와 함께 항상 제로보다 크거나 제로와 같습니다.
- 0004 000A—원소의 타입 설명자를 나타냅니다. 원소 타입은 배열을 제외한 모든 타입이 될 수 있습니다. 이 예에서 원소는 배정도 부동소수이며 다음과 같은 워드를 가집니다.
 - 0004—원소의 크기인 4바이트를 나타냅니다.
 - 000A—배정도 부동소수의 타입 코드를 나타냅니다.

이 예에 대한 타입 설명자 저장 버퍼의 포맷은 다음과 같습니다.

타입 설명자의 개수	0000 0002	고유한 타입 설명자의 개수는 2입니다.
타입 설명자 0	0004 000A	첫 번째 타입 설명자는 배정도 부동소수를 위한 것입니다.
타입 설명자 1	000E 0040 0001 FFFF FFFF	두 번째 타입 설명자는 배열을 위한 것입니다. 000E는 원소 타입 인덱스를 포함하여 전체 타입 설명자의 길이입니다. 배열의 크기는 변할 수 있으므로 차원은 FFFFFFFF입니다.
사용된 타입의 개수	0002	사용된 타입의 전체 개수는 2입니다.
사용된 타입	0000 0001	사용된 타입은 0과 1입니다.

다음과 같이 불리언 값의 2D 배열을 위한 타입 설명자의 예를 살펴봅니다.

0012 0040 0002 FFFF FFFF FFFF FFFF 0004 0021 [타입 특정 정보]

타입 설명자는 다음과 같은 워드를 가집니다.

- 0012—원소 타입 인덱스를 포함하여 전체 타입 설명자의 길이이며, 총 18바이트를 나타냅니다.
- 0040—배열의 타입 코드를 나타냅니다.
- 0002—배열의 차원 수인 2차원을 나타냅니다.
- FFFF FFFF (-1)—첫 번째 차원의 크기를 나타내며, 이 크기는 변경이 가능합니다.
- FFFF FFFF (-1)—두 번째 차원의 크기를 나타내며, 이 크기는 변경이 가능합니다.
- 0004 0021—원소의 타입 설명자를 나타냅니다. 이 예에서 원소는 불리언이며 다음과 같은 워드를 가집니다.
 - 0004—원소의 크기인 4바이트를 나타냅니다.
 - 0021—불리언의 타입 코드를 나타냅니다.

이 예에 대한 타입 설명자 저장 버퍼의 포맷은 다음과 같습니다.

타입 설명자의 개수	0000 0002	고유한 타입 설명자의 개수는 2입니다.
타입 설명자 0	0004 0021	첫 번째 타입 설명자는 불리언을 위한 것입니다.
타입 설명자 1	0012 0040 0002 FFFF FFFF FFFF FFFF	두 번째 타입 설명자는 배열을 위한 것입니다. 0012는 원소 타입 인덱스를 포함하여 전체 타입 설명자의 길이입니다. 배열의 크기는 변할 수 있으므로 차원은 FFFFFFFF입니다.
사용된 타입의 개수	0002	사용된 타입의 전체 개수는 2입니다.
사용된 타입	0000 0001	사용된 타입은 0과 1입니다.

클러스터

클러스터를 위한 타입 코드는 50입니다. 클러스터의 타입 설명자는 다음과 같은 포맷을 가집니다.

<Nn> xx 50 <k elems><원소 타입 설명자>

두 정수, 하나의 부호있는 16비트 정수 및 하나의 부호없는 32비트 정수를 포함하는 클러스터에 대한 타입 설명자의 다음 예제를 고려합니다. 여기서 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

000E 0050 0002 0004 0002 0004 0007

타입 설명자는 다음과 같은 워드를 가집니다.

- 000E—전체 14 바이트를 나타냅니다
- 0050—클러스터의 타입 코드를 나타냅니다.
- 0002—클러스터의 원소 개수인 2개의 원소를 나타냅니다.
- 0004 0002—첫 번째 원소의 타입 설명자인 부호있는 16비트 정수를 나타냅니다.
 - 0004—4 바이트를 나타냅니다.
 - 0002—부호있는 16비트 정수의 타입 코드를 나타냅니다.
- 0004 0007—두 번째 원소의 타입 설명자인 부호없는 32비트 정수를 나타냅니다.
 - 0004—4 바이트를 나타냅니다.
 - 0007—부호없는 32비트 정수의 타입 코드를 나타냅니다.

타입 설명자 저장 버퍼 포맷의 포맷은 다음과 같습니다.

타입 설명자의 개수	0000 0003	고유한 타입 설명자의 개수는 3입니다.
타입 설명자 0	0004 0002	첫 번째 타입 설명자는 부호있는 16비트 정수를 위한 것입니다.
타입 설명자 1	0004 0007	두 번째 타입 설명자는 부호없는 32비트 정수를 위한 것입니다.
타입 설명자 2	000E 0050 0002	세 번째 타입 설명자는 클러스터를 위한 것입니다. 000E는 클러스터 타입 인덱스를 포함하는 전체 타입 설명자의 길이입니다.
사용된 타입의 개수	0003	사용된 타입의 전체 개수는 3입니다.
사용된 타입	0000 0001 0002	사용된 타입은 0, 1, 2입니다.

배열과 클러스터 타입 설명자가 다른 타입 설명자를 가지고 있기 때문에, 그 설명자는 깊
이 포개어질 수 있습니다.

중복 타입은 타입 리스트에 포함되어있지 않습니다. 다음 예는 배정도 부동소수의 1D 배
열과 하나의 배정도 부동소수를 가진 클러스터의 타입 설명자 리스트입니다.

타입 설명자의 개수	0000 0003	고유한 타입 설명자의 개수는 3입니다.
타입 설명자 0	0005 000A 00	첫 번째 타입 설명자는 원소 타입을 나타내며 같은 타입의 스칼라와 똑같이 나타납니다.
타입 설명자 1	000C 0040 0001 FFFF FFFF 0000	두 번째 타입 설명자는 배열을 위한 것입니다. 000C는 원소 타입 인덱스를 포함한 전체 타입 설명자의 길이입니다. 배열의 크기는 변할 수 있으므로 차원은 FFFFFFFF입니다.
타입 설명자 2	0008 0050 0001 0002	세 번째 타입 설명자는 클러스터를 위한 것입니다. 0008는 클러스터 타입 인덱스를 포함한 전체 타입 설명자의 길이입니다. 클러스터는 인덱스 2에서 타입으로 된 원소 하나를 포함하고 있습니다.
사용된 타입의 개수	0003	사용된 타입의 전체 개수는 4입니다.

사용된 타입	0000 0001 0000 0002	사용된 타입은 0, 1, 0, 2입니다.

인덱스 2에서 클러스터에 참조된 타입은 배열에 사용된 타입에서 찾아볼 수 있어야 합니다. 사용된 타입의 두 번째 원소는 0이며 이는 타입 설명자 0에 대응합니다.

웨이브폼

웨이브폼을 위한 타입 코드는 54입니다. 웨이브폼의 타입을 가지고 있는 워드는 타입 코드 바로 뒤에 옵니다. 웨이브폼 원소의 클러스터의 타입 설명자는 이 워드 뒤에 옵니다. 웨이브폼에는 5가지 항목이 있습니다:

- 아날로그
- 디지털
- 디지털 데이터
- 타임스탬프
- 다이내믹

아날로그 웨이브폼

다음 테이블은 아날로그 웨이브폼의 타입을 나열합니다.

타입	서브타입 코드
8비트 부호있음	14
16비트 부호있음	2
32비트 부호있음	15
64비트 부호있음	19
배정도	3
단정도	5
확장형	10
8비트 부호없음	11
16비트 부호없음	12
32비트 부호없음	13
64비트 부호없음	20
복소수 단정도	16

타입	서브타입 코드
복소수 배정도	17
복소수 확장형	18

예를 들어, 배정도 웨이브폼을 고려합니다. 여기서 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

0006 0054 0003 //웨이브폼 크기, 웨이브폼 타입 코드, 배정도 서브타입 코드 3

0080 0050 0005.//클러스터의 크기, 클러스터 코드, 원소의 개수

<타임스탬프의 타입 설명자>

<dt의 타입 설명자>

<원소 타입 배열의 타입 설명자>

<사용되지 않은 에러 클러스터의 타입 설명자>

<속성의 타입 설명자>

다른 타입의 아날로그 웨이브폼의 타입 설명자는 배정도 웨이브폼과 비슷합니다. 차이점은 배열 원소에 대해 다른 타입이고 다른 서브타입을 갖는다는 것입니다.

디지털 웨이브폼과 디지털 데이터

디지털 웨이브폼은 서브타입 코드 8을 가지는 유일한 웨이브폼 타입입니다.

예를 들어, 다음을 고려합니다. 여기서 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

003E 0054 0008 //디지털 웨이브폼 크기, 웨이브폼 타입 코드, 서브타입 코드 8

<타임스탬프의 타입 설명자>

<dt의 타입 설명자>

<디지털 데이터의 타입 설명자>

<사용되지 않은 에러 클러스터의 타입 설명자>

<속성의 타입 설명자>

디지털 데이터

디지털 데이터는 웨이브폼 타입 코드와 7의 서브타입 코드를 가진 유일한 데이터 타입입니다.

예를 들어, 다음을 고려합니다. 여기서 문자의 각 그룹은 16진수 표기로 표현된 16비트 워드를 나타냅니다.

003E 0054 0007 // 디지털 데이터 크기, 웨이브폼 타입 코드, 서브타입 코드 7

<변이의 타입 설명자>

<데이터의 타입 설명자>

타임스탬프

타임스탬프는 6의 서브타입 코드를 가집니다. 타임스탬프는 64비트 기수를 가진 128비트 고정 소수점 수입니다. LabVIEW는 4개의 정수 클러스터로 타임스탬프를 저장하며, 이때 처음 두 개의 부호 있는 정수는 세계시 기준 [01-01-1904 00:00:00], 1904년 1월 1일 금요일 오전 12시 이후 경과된 초의 개수를 나타냅니다. 다음 2개의 부호 없는 정수(64비트)는 초의 소수 단위를 나타냅니다.

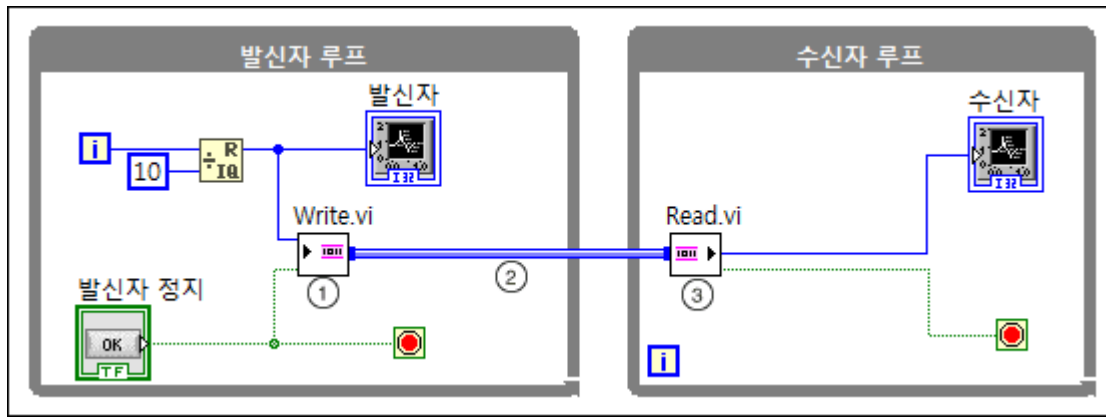
003E 0054 0006 // 타임스탬프 크기, 웨이브폼 타입 코드, 타임스탬프 서브타입 코드 6

<아날로그 웨이브폼의 타입 설명자>

병렬 실행 중인 두 개 루프 사이에서 데이터 전송하기

채널 와이어를 사용하는 것은 한 루프에서 참조 번호나 변수를 쓰고 다른 루프에서 이를 읽도록 하는 것과 비슷합니다. 참조 번호나 변수를 사용하는 대신, 와이어를 사용하여 데이터 전송을 그래픽적으로 표현합니다. 채널 와이어는 실행 순서를 강제 적용하지 않고 두 개의 병렬 코드 섹션을 연결하는 비동기적 와이어입니다. 그러므로 두 코드 섹션 사이에서 데이터 의존성이 발생되지 않습니다.

다음 그림은 채널 와이어를 사용하여 병렬로 실행되는 두 개의 루프 사이에서 데이터를 전송하는 방법을 보여줍니다. 발신자 루프에서, 쓰기 끝포인트는 반복 터미널의 값을 채널에 계속 씁니다. 수신자 루프에서, 읽기 끝포인트는 채널로부터 데이터를 계속 읽습니다. 수신자 루프가 실행되기 전에, 발신자 루프가 완료될 때까지 수신자 루프가 대기해야 하는 것은 아닙니다. 대신, 두 루프가 병렬로 실행됩니다. 사용자가 발신자 루프에서 발신자 정지 버튼을 클릭하면, 쓰기 끝포인트는 채널에 데이터를 쓰던 동작을 즉시 멈추고, 읽기 끝포인트는 채널에 남아있는 데이터를 모두 읽은 후 읽기 동작을 멈춥니다.



다음은 위의 다이어그램에서 중요한 사항을 상세히 설명한 것입니다.



노트

채널에서 데이터 타입이 서로 맞지 않아 강제 변환점이 생기거나 와이어가 깨지면, 끝포인트나 프런트패널 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 원소 타입에 맞추기 또는 채널 타입에 맞추기를 선택합니다.

유의점 및 권장사항

다음 리스트는 채널 와이어를 디버깅할 때 고려해야 할 몇가지 유의점 및 권장사항에 대해 설명합니다.

- 채널 와이어를 프로브할 때, 프로브 관찰 윈도우는 프로브할 채널을 표시하는 플러그의 역할만 합니다. VI가 실행되기 시작하면 채널 프로브를 위한 별도의 윈도우가 나타납니다. 여러 다른 끝포인트가 데이터를 읽거나 쓰는 동안 채널 프로브 윈도우에서 채널의 진행 상태를 확인할 수 있습니다.
- 프로브 관찰 윈도우에서 와이어 프로브가 새 채널에 대해 활성화되면 채널을 일시 중지 옵션을 선택하는 경우, 채널 프로브 윈도우가 열리면 바로 채널 프로브가 일시 정지됩니다. 전체 프로그램의 실행 초기에 이 옵션을 선택하면, 모든 끝포인트의 실행 전에 채널 자체를 일시 정지시킬 수 있습니다. 채널 자체가 일시 정지되면 채널에서 어떤 일이 발생하는지 자세히 살펴볼 수 있습니다.
- 채널 프로브 윈도우는 일시 정지, 조건적 일시 정지, 단계별 실행 등의 디버깅 기능을 제공합니다. 디버깅 옵션은 각 채널 템플릿에 맞게 제공되며, 이로써 철저한 채널 디버깅이 가능합니다.
- 고속 스트림 채널에 대해서는 디버깅 기능을 사용할 수 없습니다. 호출자 VI에서 디버깅이 활성화된 경우에도 고속 스트림 채널은 프로브할 수 없습니다. 이 때문에 고속 스트림 채널을 사용하면 빠르고 매우 효율적인 방식으로 데이터를 전송할 수 있는 것입니다. 하지만 디버깅이 채널과 대다수 호출자 VI의 성능에 미치는 영향은 매우 근소하기 때문에 이를 이유로 다른 템플릿 채널의 사용을 꺼릴 필요는 없습니다.

LabVIEW의 수식과 방정식

수식과 방정식

LabVIEW에서 복잡한 식을 구현하려면, 블록다이어그램에서 복잡하게 많은 연산 함수들을 연결할 필요는 없습니다. 친숙한 수학적 환경에서 식을 만든 후 어플리케이션에 통합할 수 있습니다.



관련 정보

[수식 노드](#) (646 페이지 참조)

[식 노드](#) (648 페이지 참조)

[수식 노드와 식 노드 함수](#) (1070 페이지 참조)

[수식 노드와 식 노드에서 연산자의 우선 순위](#) (1072 페이지 참조)

[스크립트 노드](#) (1082 페이지 참조)

수식 노드

[수식 노드]는 블록다이어그램에서 수학 연산을 수행할 때 사용할 수 있는 편리한 텍스트 기반의 노드입니다. 외부 코드나 어플리케이션에 접근할 필요가 없으며, 방정식을 만들기 위해 하위 레벨의 산술 함수를 연결하지 않아도 됩니다. [수식 노드]에는 텍스트 기반 방정식 표현을 사용할 수 있고, C 프로그래머들에게 친숙한 If 문, While 루프, For 루프, 그리고 Do 루프 등을 사용할 수 있습니다. 이들 프로그래밍 요소들은 C 프로그래밍의 것들과 유사하지만 완전히 똑같지는 않습니다.



노트

C 프로그래밍 언어에 익숙한 경우, 수식 노드를 사용하는 것이 가장 효율적입니다. LabVIEW 도움말은 사용자가 C에 익숙하다고 가정하며, C 프로그래밍과 관련된 토픽은 다루지 않습니다. 더 자세한 정보는 Brian W. Kernighan과 Dennis M. Ritchie의 The C Programming Language를 참고하시거나 Herbert Schildt의 Teach Yourself C를 참조하십시오.

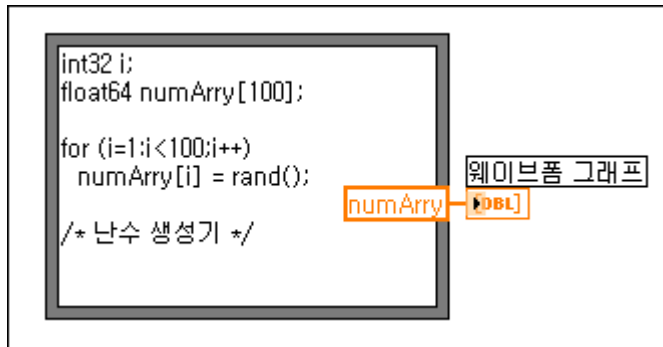
[수식 노드]는 변수가 많거나 복잡한 방정식과 기존 텍스트 기반 코드를 재사용할 때 유용합니다. 기존 텍스트 기반 코드를 블록다이어그램에서 그래픽으로 다시 만들지 않고 [수식 노드]로 복사하여 붙여넣을 수 있습니다.

[수식 노드]는 배열의 인덱스들이 숫자 데이터인지 그리고 비트 연산의 피연산자가 정수 데이터인지 확인하기 위하여 타입 확인을 합니다. 또한 [수식 노드]는 배열의 인덱스 범위가 적합한지 확인합니다. 배열에서, 값이 범위를 벗어날 경우에 값의 기본 설정 값은 제로이며, 값이 범위를 벗어날 경우 할당의 기본 설정은 nop으로 아무런 작업이 일어나지 않음을 나타냅니다.

또한 [수식 노드]는 자동 타입 변환을 수행합니다.

[수식 노드] 사용하기

[수식 노드]는 다음과 같은 [For 루프], [While 루프], [케이스 구조], [다중 시퀀스 구조], [플랫 시퀀스 구조]와 유사하게 크기 변경이 가능한 상자입니다. 그러나, 서브다이얼로그 대신, [수식 노드]는 다음 예제와 같이 세미콜론으로 구분된 하나 또는 그 이상의 C 언어와 유사한 문장을 갖습니다. C와 같이 슬래시/별표의 조합 /* 주석 */ 안에 주석을 넣거나 두 개의 슬래시 // 주석 뒤에 주석을 넣어 주석을 추가할 수 있습니다.



[수식 노드]의 변수

변수를 사용할 때에는, 다음 사항을 기억합니다:

- [수식 노드]의 변수와 방정식에는 개수의 제한이 없습니다.
- 두 개의 입력은 동일한 이름을 가질 수 없고 두 개의 출력도 동일한 이름을 가질 수 없습니다. 그러나 입력 변수명과 같은 출력 변수명은 사용할 수 있습니다.
- 입력 변수를 선언하기 위하여 [수식 노드]의 경계를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 입력 추가를 선택합니다. 입력 변수는 [수식 노드] 내부에서 선언할 수 없습니다.
- 출력 변수를 선언하기 위하여 [수식 노드]의 경계를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 출력 추가를 선택합니다. 출력 변수명은 입력 변수명과 같거나 [수식 노드] 내부에서 선언해준 변수명과 같아야 됩니다.
- 변수를 입력 또는 출력으로 변경하려면 입력이나 출력을 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 입력으로 변경 또는 출력으로 변경을 선택합니다.
- 입력과 출력에 연관되지 않는 변수는 [수식 노드] 내부에서 선언하고 사용할 수 있습니다.
- 모든 입력 터미널은 반드시 연결되어야 합니다.
- 변수는 컴퓨터의 설정에 따라 정확도가 정해지는 부동소수 숫자값일 수 있습니다. 또한 정수나 숫자값의 배열을 변수로 사용할 수 있습니다.



노트 수식 노드는 복소수 데이터 타입이나 행렬 데이터 타입을 지원하지 않습니다. 이러한 지원되지 않는 데이터 타입을 사용하려는 경우, 스크립트 노드를 사용하십시오.

- 변수는 단위를 가질 수 없습니다.

식 노드

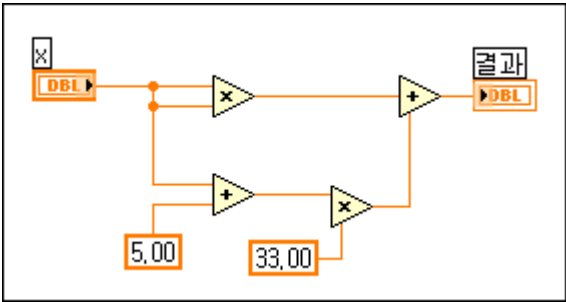
[식 노드]를 사용하여 단일 변수를 갖는 수식이나 방정식을 계산합니다. [식 노드]는 변수가 한 개인 경우 유용하지만 변수가 여러 개인 경우는 복잡해집니다.

[식 노드]는 입력 터미널에 전달한 값을 변수의 값으로 사용합니다. 출력 터미널은 계산한 값을 반환합니다.

예를 들어, 다음과 같은 간단한 정규식을 생각해봅시다:

$$x \times x + 33 \times x + 5)$$

다음 블록다이어그램은 숫자형 함수를 사용하여 이 정규식을 나타낸 것입니다.



다음 그림과 같이 [식 노드]를 사용하면 훨씬 간단한 블록다이어그램을 만들 수 있습니다.



식 노드의 다형성

[식 노드]의 입력 터미널은 연결한 컨트롤이나 상수의 데이터 타입을 가집니다. 출력 터미널은 입력 터미널과 같은 데이터 타입을 가집니다. 입력의 데이터 타입은 모든 복소수가 아닌 스칼라 숫자, 스칼라 숫자의 배열, 또는 스칼라 숫자의 클러스터가 될 수 있습니다. 배열과 클러스터의 경우, [식 노드]는 입력 배열이나 클러스터의 모든 원소에 정규식을 적용합니다.

피드백 노드

피드백 노드를 사용하여 이전 블록다이어그램 실행 또는 루프 반복에서 온 데이터를 저장합니다. 피드백 노드는 이전 실행으로부터의 상태 정보를 저장하고 불러오는데 유용합니다.

피드백 노드는 받은 데이터에 대해 아무 동작도 수행하지 않습니다. 대신 초기화 터미널에서 값을 받아 해당 값을 다음 입력 터미널에 전달합니다. 피드백 노드가 새 값을 받으

면, 노드가 다른 입력 터미널에 그 값을 전달할 때까지 값이 노드에 저장됩니다. 피드백 노드는 피드백 컨트롤 이론과 디지털 신호 처리에서 z^{-1} 블록과 유사합니다.

피드백 노드를 생성하기

SubVI, 함수, 또는 SubVI와 함수 그룹의 출력을 같은 VI, 함수, 또는 그룹의 입력에 연결하고 옵션 대화 상자의 블록다이어그램 페이지에서 사이클 내에 피드백 노드 자동 삽입을 활성화하면 다음에 보이는 것처럼 피드백 노드가 자동으로 나타납니다. LabVIEW는 기본 설정으로 사이클 내에 피드백 노드 자동 삽입을 활성화합니다.



또한 함수 팔레트에서 수동으로 피드백 노드를 선택할 수도 있습니다.

피드백 노드에 대한 예제는 Structures\Feedback Node 디렉토리에 있는 Feedback Node - Building an Array VI를 참조하십시오.

초기화 터미널

초기화 터미널은 피드백 노드의 초기값을 설정합니다. 다음에 보이는 것처럼, 사용자가 피드백 노드를 어떻게 설정하는지에 따라 초기화 터미널의 모양이 변합니다.



피드백 노드를 블록다이어그램에 놓으면, 초기화 터미널은 노드에 연결된 상태로 남아 있으며 노드는 VI가 컴파일 또는 로드할 때마다 매번 글로벌하게 초기화됩니다. 첫 번째 실행에서 피드백 노드의 초기 입력은 **해당 데이터 타입에 대한 기본값**입니다. 하지만, 사용자가 피드백 노드의 **초기 값을 설정**할 수 있습니다. 사용자가 초기 값을 설정하는 경우, 피드백 노드는 VI가 처음으로 호출하여 실행될 때 해당 값으로 초기화됩니다. VI의 두 번째 실행부터는 실행시 초기값이 이전 실행의 마지막 값입니다. **루프 안에서 피드백 노드를 사용**하는 경우, 초기화 터미널을 루프의 왼쪽 에지로 옮겨 루프가 실행될 때마다 노드가 초기화하도록 설정합니다.



노트

(FPGA Module) Single-Cycle Timed 루프 안의 피드백 노드를 초기화할 때, 출력값은 활성화 터미널이 참일 때까지는 초기값과 같습니다. 활성화 터미널이 한 사이클 이상에서 거짓으로 남아있고 초기값이 사이클 마다 변경하는 경우, 출력값 역시 변경됩니다.

활성화 터미널

다음에 보이는 것처럼 활성화 터미널을 사용하여, 피드백 노드를 활성화 또는 비활성화시킬 수 있습니다.



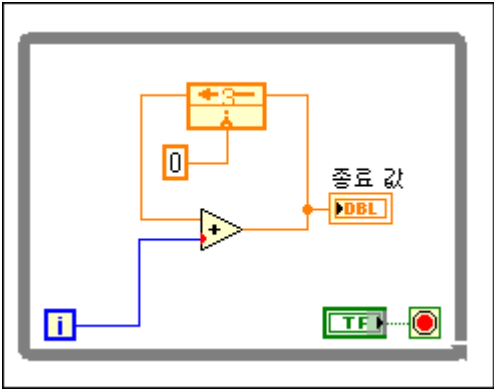
피드백 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 활성화 터미널을 선택하여, 활성화 터미널을 디스플레이할 수 있습니다. 활성화 터미널이 참으로 설정되어 있으면, 피드백 노드는 사용자가 프로퍼티 대화 상자 또는 노드의 바로 가기 메뉴에서 설정한 실행 조건에 맞게 실행됩니다. 활성화 터미널이 거짓으로 설정되어 있으면, 터미널의 값이 다시 참으로 설정될 때까지 피드백 노드는 이전 실행 또는 반복으로 부터의 최종 값을 계속 출력하게 됩니다.

데이터의 샘플

기본으로, 피드백 노드는 이전 실행 또는 반복의 데이터만 저장합니다. 하지만, 실행 및 반복 횟수를 늘려서 피드백 노드의 출력을 지연시킴으로써, 피드백 노드가 연속적인 여러 실행 또는 반복에서의 데이터 샘플을 저장하도록 피드백 노드를 프로퍼티 대화상자의 설정페이지에서 설정할 수 있습니다. 지연을 하나의 실행 또는 반복이 넘도록 증가시키면, 피드백 노드는 사용자가 지정한 지연이 완료될 때까지 초기화 값만 출력합니다. 그 후에 피드백 노드는 저장한 값을 한 번에 하나씩 순서대로 출력하기 시작합니다.

노트 (FPGA Module) 프로퍼티 대화 상자의 FPGA 구현 페이지에서 FPGA 리셋 방법 무시 확인란에 확인 표시를 하는 경우, LabVIEW FPGA Module은 기본적인 레지스터 인스턴스에서 리셋을 제거합니다. 이렇게하면 컴파일러가 플립 플롭 대신 시프트 레지스터 검색 테이블을 사용하여 지연을 구현할 수 있게 됩니다. SRL은 여러 지연을 단일 룩업 테이블(LUT)로 병합하므로 플립 플롭에 비해 FPGA 리소스 사용량을 상당히 감소시킬 수 있습니다.

예를 들어, 지연을 3으로 지정하는 경우에 피드백 노드는 3 번째 루프 반복까지 초기화 값을 출력합니다. 그 이후의 반복에서 피드백 노드는 첫번째 반복, 즉 반복 0으로 부터의 값을 출력하고, 그 다음에는 2번째 반복의 값을 출력하는 방식으로 이어져서 동작합니다. 다음의 블록 다이어그램은 초기화 값이 0이고 지연이 3인 피드백 노드의 동작을 보여줍니다.



루프 반복	피드백 노드가 출력하는 값	마지막 값	피드백 노드가 저장하는 데이터
실행 또는 반복 0 이전	해당 없음	해당 없음	0, 0, 0
0	0	0	0, 0, 0

루프 반복	피드백 노드가 출력하는 값	마지막 값	피드백 노드가 저장하는 데이터
1	0	1	0, 0, 1
2	0	2	0, 1, 2
3	0	3	1, 2, 3
4	1	5	2, 3, 5
5	2	7	3, 5, 7
6	3	9	5, 7, 9
7	5	12	7, 9, 12

위의 블록 다이어그램에서 피드백 노드는 초기화 값 0을 3번 출력한 후에 피드백 노드가 받은 첫번째 값인 0을 출력합니다.



노트 두 개 혹은 그 이상의 실행이나 반복에 지연을 설정할 때 배열을 초기화 터미널에 연결하여 각각의 지연 원소를 임의의 값으로 초기화합니다.

화살표 방향

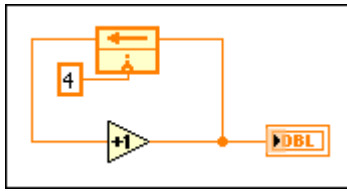
노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 방향 변경을 선택하여 피드백 노드의 화살표 방향을 변경함으로써 와이어를 따라 흐르는 데이터의 흐름을 나타낼 수 있습니다. 화살표의 방향을 바꾸더라도 피드백 노드와 그에 연결된 와이어의 외형적 모양만 변경되며 데이터의 흐름 방향은 변하지 않습니다.

Z 변환 보기

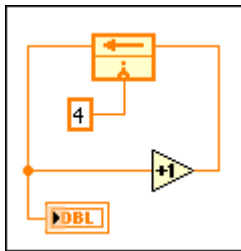
피드백 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 Z 변환 보기를 선택하여, 피드백 노드가 Z 변환 보기를 디스플레이하도록 변경할 수 있습니다. 이 보기를 사용하면 피드백 노드의 모양이 피드백 컨트롤 이론 및 신호 처리에서의 z^{-1} 블록과 유사하게 변경됩니다. 피드백 노드를 Z 변환 보기로 변경하면, 피드백 노드의 외형적 모양만 변하며 노드의 기능은 변하지 않습니다.

프런트패널 인디케이터와 피드백 노드

피드백 노드와 같은 와이어 사이클에 연결된 프런트패널 인디케이터는 와이어 상에서 피드백 노드에 상대적인 위치에 따라 서로 다른 값을 디스플레이합니다. 다음 블록 다이어그램에서 피드백 노드는 초기값을 다음 입력 노드에 전달합니다. 입력 노드는 초기값으로 작동한 후 최종값을 인디케이터에 전달하고 다시 이 값이 피드백 노드에 전달됩니다.



이전 블록다이어그램에서 [증가] 함수는 초기값 4에 작업을 한 후 이를 인디케이터에 전달합니다. 그렇게 되면 프런트패널 인디케이터가 5를 디스플레이합니다. 그러나 다음 블록다이어그램에서 피드백 노드는 같은 데이터 값을 [증가] 함수에 전달하나 인디케이터는 다른 값을 디스플레이합니다.



이전 블록다이어그램에서 피드백 노드 초기값인 4는 [증가] 함수에 도달하기 전에 인디케이터에 도달합니다. [증가] 함수에서 피드백 노드에 전달된 최종값은 5이지만 프런트패널 인디케이터는 초기값인 4를 디스플레이합니다.

LabVIEW 어플리케이션과의 통신

웹 클라이언트에서 LabVIEW 어플리케이션과 통신하기 (Real-Time, Windows)

LabVIEW 독립 어플리케이션이 원격으로 실행되는 경우, 웹 브라우저 또는 HTTP 요청을 사용하는 다른 어플리케이션과 같은 웹 클라이언트를 사용하여 어플리케이션과 통신할 수 있습니다. LabVIEW 웹 서비스를 사용하면 어플리케이션과 클라이언트 사이에서 통신을 할 수 있습니다. 웹 서비스는 어플리케이션과 동일한 타겟에서 실행되며, 어플리케이션에서 데이터를 보내거나 받도록 설계된 HTTP 메소드 VI라는 VI를 실행하여 클라이언트로부터의 HTTP 요청에 응답합니다.

또한 웹 서비스는 LabVIEW 어플리케이션에 독립적으로 실행될 수 있습니다.

다음 도움말 토픽을 사용하여 LabVIEW 웹 서비스를 시작합니다.

시작하기 - 개요: LabVIEW 어플리케이션과의 웹 기반 통신 - 길라잡이: LabVIEW 웹 서비스 어플리케이션 생성 및 공개하기 웹 서비스 개발하기 - 웹 서비스의 구성요소 - HTTP 메소드 VI 개발하기 - 시작 VI 개발하기 - 웹 서비스에 정적 콘텐츠 통합하기	웹 서비스 실행하기 - 웹 서비스 호스트하기 - 웹 서비스 공개하기 웹 서비스 통신하기 - 웹 서비스와 클라이언트 사이에서 데이터 전송하기 웹 서비스용 사용자 인터페이스 생성하기 - WebVI를 사용하여 웹 서비스용 사용자 인터페이스 생성하기

시작하기

- 개요: LabVIEW 어플리케이션과의 웹 기반 통신
- 길라잡이: LabVIEW 웹 서비스 어플리케이션 생성 및 공개하기

웹 서비스 개발하기

- 웹 서비스의 구성요소
- HTTP 메소드 VI 개발하기
- 시작 VI 개발하기
- 웹 서비스에 정적 콘텐츠 통합하기
-

웹 서비스 실행하기

- 웹 서비스 호스트하기
- 웹 서비스 공개하기

웹 서비스 통신하기

- 웹 서비스와 클라이언트 사이에서 데이터 전송하기

웹 서비스용 사용자 인터페이스 생성하기

- WebVI를 사용하여 웹 서비스용 사용자 인터페이스 생성하기



관련 정보

개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows) (655 페이지 참조)

길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 (Windows) (1500 페이지 참조)

웹 서비스의 구성요소 (Real-Time, Windows) (658 페이지 참조)

HTTP 메소드 VI 개발하기 (Real-Time, Windows) (661 페이지 참조)

HTTP 메소드 VI를 초기화하고 통신하는 시작 VI 개발하기 (Real-Time, Windows) (667 페이지 참조)

웹 서비스에 정적 콘텐츠 통합하기(Real-Time, Windows) (1486 페이지 참조)

웹 서비스 호스트하기 (Real-Time, Windows) (670 페이지 참조)

웹 서비스 공개하기 (Real-Time, Windows) (673 페이지 참조)

변수 웹 서비스를 사용하여 공유 변수 읽기와 쓰기 (584 페이지 참조)

LabVIEW에서의 데이터 통신 방법 (603 페이지 참조)

개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows) (655 페이지 참조)

길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 (Windows) (1500 페이지 참조)

웹 서비스의 구성요소 (Real-Time, Windows) (658 페이지 참조)

HTTP 메소드 VI 개발하기 (Real-Time, Windows) (661 페이지 참조)

HTTP 메소드 VI를 초기화하고 통신하는 시작 VI 개발하기 (Real-Time, Windows) (667 페이지 참조)

웹 서비스에 정적 콘텐츠 통합하기(Real-Time, Windows) (1486 페이지 참조)

웹 서비스 호스트하기 (Real-Time, Windows) (670 페이지 참조)

웹 서비스 공개하기 (Real-Time, Windows) (673 페이지 참조)

변수 웹 서비스를 사용하여 공유 변수 읽기와 쓰기 (584 페이지 참조)

개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows)

LabVIEW 웹 서비스를 통해 웹 클라이언트는 네트워크상에서 원격 LabVIEW 독립 어플리케이션과 데이터를 교환할 수 있습니다. 웹 서비스는 클라이언트의 HTTP 요청에 응답하고 서버에서 실행되는 VI 및 그 외 다른 파일들로 구성됩니다.



노트

LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음의 경우, 웹 서비스가 유용합니다:

- 표준 웹 브라우저 및 HTTP를 사용하는 모든 웹 클라이언트를 사용하여 웹 서비스 VI를 인보크하고, URL 및 POST와 같은 표준 HTTP 메소드를 사용하여 데이터를 교환할 수 있습니다. 예를 들면:
 - 어플리케이션에 새 파라미터를 업로드할 때
 - 현재 상태를 불러올 때
- 사용자 정의된 씬(thin) 클라이언트 어플리케이션을 통해 원격으로 임베디드 어플리케이션을 모니터링하고 컨트롤할 수 있습니다. 예를 들면:
 - 어플리케이션 상태를 업데이트할 때
 - 프로세스를 시작 또는 중지할 때
- National Instruments 및 타사의 여러 HTTP 구현 디바이스와 소프트웨어 간에는 어플리케이션 대 어플리케이션 데이터 교환이 가능합니다. 예를 들면:
 - LabVIEW가 아닌 클라이언트에 연결할 때
 - 보안 및 인증을 구현할 때

다음 상황에서는 웹 서비스를 사용하지 마십시오:

- 연속 데이터 스트리밍
- 빠른 폴링
- Real-time 프로세싱

다음 그림은 LabVIEW 웹 서비스 어플리케이션에 관련된 일반적인 프로세스를 보여줍니다.



(1) 웹 서비스 개발하기

웹 서비스는 LabVIEW 어플리케이션과 웹 클라이언트 간의 통신을 가능하게 합니다. 다음 작업을 수행하여 웹 서비스를 개발합니다.

- LabVIEW가 설치되어 있는 호스트 컴퓨터에서 LabVIEW 프로젝트에 웹 서비스를 생성하여 개발합니다.
- HTTP 메소드 VI와 시작 VI와 같은 웹 서비스의 기본 컴포넌트를 개발합니다.
- 정적 콘텐츠 파일을 웹 서비스에 통합합니다.
- 웹 서버에서 웹 서비스를 테스트하고 디버그합니다.

관련 정보

웹 서비스 개발하기

(2) 웹 서비스 호스팅하기

LabVIEW는 NI 웹 서버 또는 어플리케이션 웹 서버를 사용하여 네트워크에서 상에서 웹 서비스를 호스팅합니다. 웹 서비스에 대한 웹 서버 지원은 웹 서비스를 게시하거나 실행하는 방법에 따라 달라집니다.



노트 NI 웹 서버는 Windows OS만을 지원합니다. 어플리케이션 웹 서버는 Windows OS 및 RT 타겟을 모두 지원합니다.

웹 서비스를 사용하여 웹 서비스를 호스팅하기 전에 웹 서버를 활성화하고 설정해야 합니다. 또한 웹 클라이언트와 LabVIEW 웹 서비스 간의 간에 안전한 통신을 수립할 수도 있습니다.

관련 정보

웹 서비스 호스팅하기

(3) 웹 서비스 공개하기

프로젝트에서 다음 작업을 수행하여 웹 서비스가 LabVIEW Development System에 의존하지 않고 실행될 수 있도록 웹 서비스를 공개할 수 있습니다.

- 웹 서비스를 독립 웹 서비스로 호스트 컴퓨터 또는 접속된 RT 타겟의 어플리케이션 웹 서버에 공개합니다.
- (Windows) 웹 서비스를 NI 패키지를 통해 NI 웹 서버에 공개합니다.
- 웹 서비스를 독립 어플리케이션을 통해 Windows 컴퓨터 또는 RT 타겟의 NI 웹 서버 또는 어플리케이션 웹 서버에 공개합니다.
- 웹 서비스를 Windows 설치 프로그램을 통해 Windows 컴퓨터의 어플리케이션 웹 서버에 공개합니다.

웹 서비스는 호스트 서버에 공개되는 방식에 따라 서로 다른 조건으로 실행을 시작하고 정지합니다.

관련 정보

웹 서비스 공개하기

(4) 웹 서비스와 클라이언트 사이에서 데이터 전송하기

웹 클라이언트는 브라우저의 경우와 같이 특정 URL로 HTTP 요청을 전송하여 웹 서비스와 데이터를 교환합니다. LabVIEW는 URL을 각 HTTP 메소드 VI에 맵핑하므로, 어떤 HTTP 메소드 VI가 HTTP 요청을 받을지는 클라이언트가 사용하는 특정 URL에 의해 결정됩니다. 요청이 HTTP 메소드 VI의 특정 파라미터에 할당될 값을 포함할 수도 있습니다. 각 요청 후 HTTP 메소드 VI는 해당 값을 처리하고 응답을 반환합니다. 웹 서비스는 사용자가 설정한 JSON과 같은 특정 포맷으로 웹 클라이언트로 데이터를 반환합니다.

지원되는 웹 클라이언트에는 표준 웹 브라우저, HTML 형식, 타사 소프트웨어와 HTTP 클라이언트 팔레트를 사용하는 VI 등, HTTP가 활성화된 모든 플랫폼이 포함됩니다.

표준 프로세스 사이의 통신 방법을 사용하여 모든 Windows 또는 임베디드 LabVIEW 어플리케이션과 웹 서비스 간의 통신을 구현할 수 있습니다. 예를 들어 [네트워크 스트림] 함수를 사용하여 로컬에서 LabVIEW 어플리케이션에서 웹 서비스로 데이터를 전송할 수 있습니다.

간단한 예로, 웹 클라이언트는 두 개의 숫자를 포함하는 HTTP 요청을 제출하여 HTTP 메소드 VI가 그 두 숫자를 계산하여 총합을 반환하도록 할 수 있습니다.

관련 정보

웹 서비스와 클라이언트 사이에서 데이터 전송하기

(5) 웹 서비스용 사용자 인터페이스 생성하기

웹 브라우저를 통해 웹 서비스의 데이터를 시각화하고 상호작용하는 사용자 인터페이스를 작성하려면 WebVI를 Web 서비스에 통합할 수 있습니다. WebVI는 LabVIEW에서 웹 어플리케이션을 구축하면 HTML, JavaScript, CSS 파일을 생성하는 특수 VI입니다. WebVI는 HTTP 메소드를 통해 웹 서비스와 데이터를 통신하고 데이터를 디스플레이할 수 있습니다.

웹 서비스 개발하기 (Real-Time, Windows)

상위 토픽: 개요: LabVIEW 어플리케이션과의 웹 기반 통신

웹 서비스는 LabVIEW 어플리케이션과 웹 클라이언트 간의 통신을 가능하게 합니다. 웹 서비스는 LabVIEW 프로젝트에서 개발할 수 있습니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 웹 서비스 개요 및 길라잡이를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

웹 서비스는 HTTP 메소드 VI, HTTP 메소드 VI 의존성, 시작 VI 및 정적 콘텐츠 파일로 구성됩니다.

- HTTP 메소드 VI — 웹 서비스의 최상위 VI. 웹 서비스의 핵심 기능이 포함되어 있으며 네트워크를 통해 데이터 웹 클라이언트를 교환합니다.
- HTTP 메소드 VI 의존성 — SubVI, 공유 변수, DAQ 채널 및 HTTP 메소드 VI 안에 나타나는 그 외 컴포넌트들.
- 시작 VI — 네트워크 리소스를 사용하지 않고 HTTP 메소드 VI와 데이터를 교환하고 웹 클라이언트와 데이터를 교환하지 않는 VI.
- 정적 콘텐츠 파일 — HTML 파일, JavaScript 파일, 스타일 시트, 이미지 및 비디오. 예를 들어, WebVI가 웹 페이지에 대해 생성하는 파일.

웹 서비스를 타겟에 공개하기 전, 클라이언트가 HTTP 메소드 VI와 정적 콘텐츠를 예상대로 실행하고 통신하는지 테스트하십시오. 예상치 않은 동작이나 에러가 발견된 경우, 수정 후 웹 서비스를 다시 테스트합니다.



관련 정보

[개요: LabVIEW 어플리케이션과의 웹 기반 통신 \(Real-Time, Windows\)](#) (655 페이지 참조)

[개요: LabVIEW 어플리케이션과의 웹 기반 통신 \(Real-Time, Windows\)](#) (655 페이지 참조)

[길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 \(Windows\)](#) (1500 페이지 참조)

[웹 서비스의 구성요소 \(Real-Time, Windows\)](#) (658 페이지 참조)

[HTTP 메소드 VI 개발하기 \(Real-Time, Windows\)](#) (661 페이지 참조)

[HTTP 메소드 VI를 초기화하고 통신하는 시작 VI 개발하기 \(Real-Time, Windows\)](#) (667 페이지 참조)

[웹 서비스에 정적 콘텐츠 통합하기 \(Real-Time, Windows\)](#) (1486 페이지 참조)

[NI 웹 서버에서 웹 서비스 테스트 및 디버깅하기 \(Windows\)](#) (1488 페이지 참조)

[웹 서비스 구조 및 파일 위치 이해하기 \(Real-Time, Windows\)](#) (669 페이지 참조)

웹 서비스의 구성요소 (Real-Time, Windows)

상위 토픽: [웹 서비스 개발하기](#)

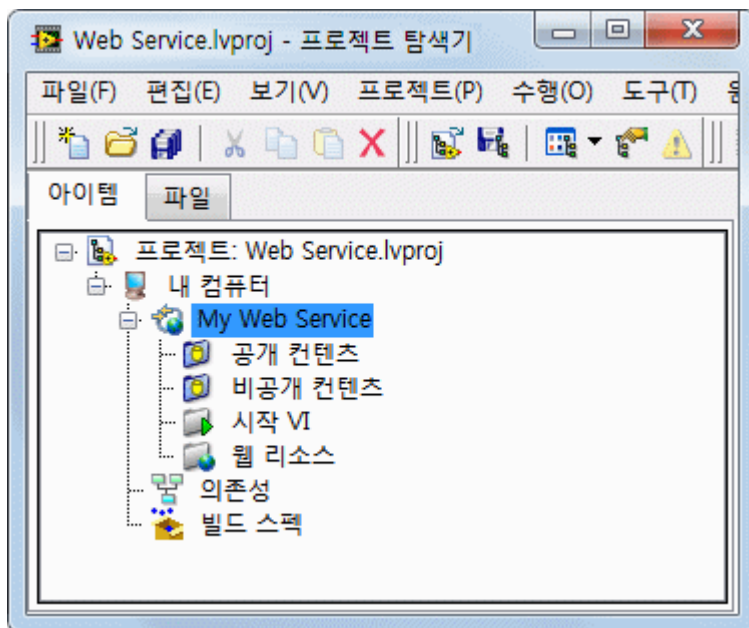
웹 서비스에는 HTTP 메소드 VI, HTTP 메소드 VI 의존성, 시작 VI, 정적 콘텐츠 파일이 포함될 수 있습니다. 웹 서비스 프로젝트 아이템에 웹 서비스 파일을 구성합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 LabVIEW 프로젝트는 웹 서비스 파일의 구성 및 설정에 사용하는 웹 서비스 프로젝트 아이템 및 서브아이템을 포함합니다. 최상위 레벨 웹 서비스 프로젝트 아이템을 추가하려면, 내 컴퓨터와 같은 타겟에서 마우스 오른쪽 버튼을 클릭한 후 새로 만들기>>웹 서비스를 선택합니다.



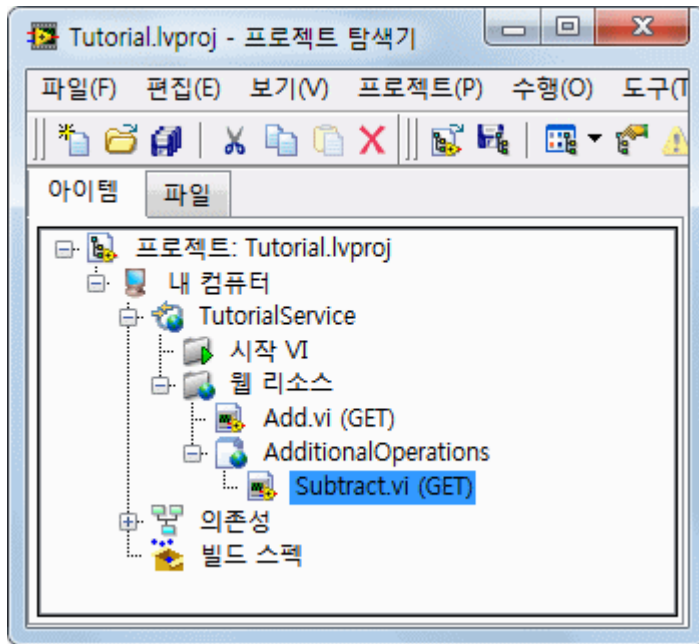
HTTP 메소드 VI

HTTP 메소드 VI는 웹 서비스의 최상위 VI이며, 웹 서비스의 주요 기능이 들어 있습니다. 웹 클라이언트가 HTTP 요청을 웹 서비스에 보내면, 웹 서비스는 요청을 수락한 후 특정 HTTP 메소드 VI로 보냅니다. HTTP 메소드 VI와 웹 클라이언트는 네트워크를 통해 데이터를 교환합니다.

웹 리소스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 새 VI를 선택하여 새 HTTP 메소드 VI를 생성합니다.

웹 리소스 프로젝트 아이템 아래에 서브아이템을 추가하여 HTTP 메소드 VI를 구성할 수 있습니다. 이러한 서브폴더는 사용자 모음과 같은 동일한 리소스를 기반으로 작업을 수행하는 HTTP 메소드 VI들을 그룹으로 묶는 RESTful 설계에 유용합니다. RESTful이란 Representational State Transfer (REST) 구조를 지칭합니다.

또한 서브폴더의 이름은 클라이언트가 서브 폴더에 둔 HTTP 메소드 VI를 실행할 때 사용하는 URL에도 나타납니다. 예를 들어 다음 그림에서, Subtract.vi HTTP 메소드 VI의 URL에는 서비스 이름 뒤에 AdditionalOperations이 포함됩니다: `http://127.0.0.1/TutorialService/AdditionalOperations/Subtract`.



관련 정보

HTTP 메소드 VI 개발하기

URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기

HTTP 메소드 VI 의존성

HTTP 메소드 VI 의존성은 개발하려는 HTTP 메소드 VI의 필수 구성요소입니다. 의존성에는 SubVI, 공유 변수, DAQ 채널 및 HTTP 메소드 VI 안에 나타나는 그 외 구성요소들이 있습니다.

LabVIEW 웹 서비스를 배포할 때, LabVIEW 웹 서비스는 SubVI와 같은 HTTP 메소드 VI 의존성을 자동으로 묶습니다. 공개된 웹 서비스에는 공유 변수와 같은 HTTP 메소드 VI 의존성은 포함되지 않습니다.

시작 VI

시작 VI는 웹 서비스 어플리케이션의 일부로 실행되지만, HTTP 메소드 VI와는 달리 웹 클라이언트와 데이터를 교환하지는 않습니다. 시작 VI를 사용하여 웹 서비스에서 다음 중 하나 또는 하나 이상의 태스크를 실행할 수 있습니다:

- 파일에서 데이터를 로드하고 값을 초기화하기.
- 웹 서비스가 클라이언트로부터 받은 데이터를 처리하고 조작하기.
- 웹 서비스 실행 중 주기적으로 데이터를 파일에 저장하기.

웹 서비스가 포함된 LabVIEW 어플리케이션은 또한 이전 태스크를 수행할 수 있습니다. 그러나, 어플리케이션의 일부가 아닌 독립 웹 서비스의 경우 대신 시작 VI를 사용합니다.

시작 VI는 네트워크 리소스를 사용하지 않고 HTTP 메소드 VI와 데이터를 교환할 수 있어, 어플리케이션 효율성을 증가시킬 수 있습니다.

시작 VI 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 새 VI를 선택하여 새 시작 VI를 생성합니다. 각 웹 서비스에는 여러 개의 시작 VI가 포함될 수 있습니다.

관련 정보

HTTP 메소드 VI를 초기화하고 이와 통신하는 시작 VI 개발하기

정적 콘텐츠

웹 서비스에서 HTTP 메소드 VI와 더불어 정적 콘텐츠를 공개할 수 있습니다. 정적 콘텐츠 파일의 예는 HTML 파일, JavaScript 파일, 스타일 시트, 이미지 및 비디오입니다. WebVI를 사용하여 웹 서비스에 대한 사용자 인터페이스를 생성하는 경우, WebVI가 생성하는 웹 페이지 파일은 정적 콘텐츠입니다.

웹 서비스에서 마우스 오른쪽 버튼을 클릭한 후, 공개 콘텐츠 폴더 추가 또는 비공개 콘텐츠 폴더 추가를 선택하여 프로젝트에 폴더를 추가하고 디스크상의 정적 콘텐츠 폴더에 매핑합니다. 디스크상의 정적 콘텐츠를 프로젝트 트리의 구성과 같도록 공개 콘텐츠가 있는 폴더와 비공개 콘텐츠가 있는 두 개의 주요 폴더로 구성합니다:

- 비공개 콘텐츠—클라이언트는 HTTP 요청을 통해 비공개 폴더에 있는 파일에 접근할 수 없습니다. 비공개 파일은 URL 매핑을 갖지 않습니다. 오직 웹 서비스의 VI만 비공개 정적 콘텐츠를 사용할 수 있습니다. 예를 들어, 웹 서비스가 설정 파일을 사용하여 실행되는 경우, 해당 설정 파일을 비공개로 만들 수 있습니다.
- 공개 콘텐츠—클라이언트는 HTTP 요청을 통해 공개 정적 콘텐츠에 접근할 수 있습니다. 공개로 해야 할 파일의 예로는 HTML 페이지 및 이미지가 있습니다. LabVIEW는 자동으로 각 공개 파일에 URL 매핑을 할당합니다.

주요 폴더는 Public Folder\html 및 Public Folder\images와 같이 서브폴더를 가질 수 있습니다. 프로젝트 트리의 폴더들은 기본으로 자동 적용 폴더이며, 이는 디스크상의 폴더의 내용을 실시간으로 반영하며 업데이트된다는 것을 의미합니다.

관련 정보

웹 서비스에 정적 콘텐츠 통합하기

HTTP 메소드 VI 개발하기 (Real-Time, Windows)

상위 토픽: [웹 서비스 개발하기](#)



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

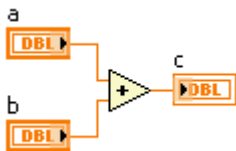
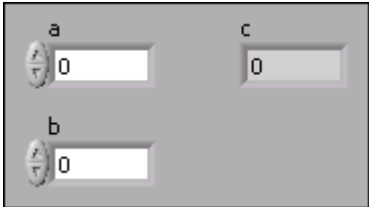
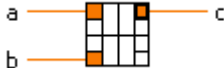
HTTP 메소드 VI를 생성하려면, 프로젝트 탐색기 윈도우의 웹 리소스 또는 웹 리소스 서버 아이템에서 마우스 오른쪽 버튼을 클릭한 후 새 VI를 선택합니다. LabVIEW는 HTTP 메소드 VI에서 일반적으로 사용하는 **LabVIEW 웹 서비스 요청** 컨트롤을 포함하는 새 VI를 엽니다.

다음 섹션의 가이드라인을 고려하여 HTTP 메소드 VI를 개발하십시오.

HTTP 메소드 VI와 웹 클라이언트 사이에서 데이터 전송하기

HTTP 메소드 VI는 커넥터 팬 터미널을 사용하여 데이터를 받고 전송합니다. 웹 클라이언트가 HTTP 메소드 VI에 HTTP 요청을 전송하면, VI는 커넥터 팬의 컨트롤에 값을 받고 그 값들을 처리한 후 클라이언트에게 응답을 반환합니다.

다음 그림은 웹 클라이언트로부터 두 개의 값을 받은 후, 이 값의 총합을 반환하는 HTTP 메소드 VI의 프런트패널, 블록 다이어그램, 커넥터 팬의 예를 보여줍니다.

블록 다이어그램	프런트패널	커넥터 팬
		

웹 서비스 내에서 값을 받고 반환하는데 사용하는 컨트롤 및 인디케이터의 라벨에는 오직 US ASCII 문자 세트에 있는 글자, 숫자, 하이픈, 밑줄 만이 포함될 수 있습니다. 라벨에는 공백 또는 특수 문자가 들어갈 수 없습니다.

관련 정보

URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기

컨트롤과 인디케이터에 터미널 지정하기

출력 데이터 포맷 선택하기

기본적으로 HTTP 메소드 VI는 웹 클라이언트에게 JSON 문자열로 데이터를 반환합니다. JSON 문자열에는 HTTP 메소드 VI의 커넥터 팬에 지정된 인디케이터 값에 대한 요약이 들어 있습니다. 또한 HTTP 메소드 VI를 설정하여 XML-, HTML-, 또는 일반 텍스트 포맷의 응답으로 데이터를 반환할 수도 있습니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 출력 포맷 타입을 설정합니다.

	노트 HTTP 메소드 VI는 커넥터 팬을 통해 클라이언트에게 데이터를 반환하지 않고, 사용자 정의된 HTML 페이지 또는 이미지 데이터와 같은 사용자 포맷으로 클라이언트에게 데이터를 스트리밍합니다.

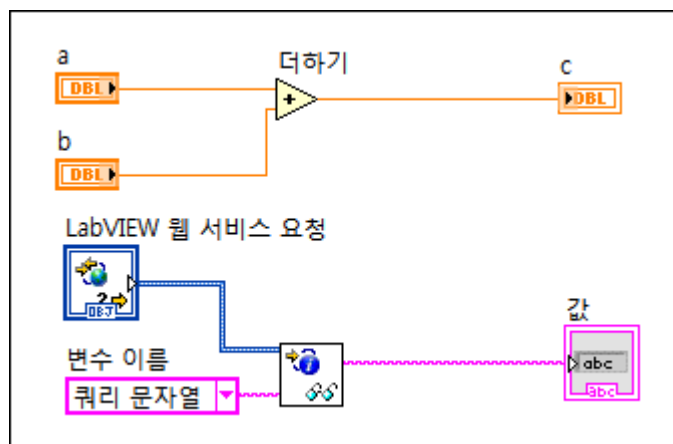
관련 정보

HTTP 헤더, 스트리밍, 버퍼 설정하기

웹 서비스 팔레트의 VI 사용하기

웹 서비스 팔레트의 VI는 POST data, form data, HTTP 세션, 쿠키와 기타 다른 웹 특정 기능을 처리할 수 있습니다. 웹 서비스 팔레트의 VI를 HTTP 메소드 VI에 포함시키는 경우, 각 VI의 LabVIEW 웹 서비스 요청 입력을 연결하여 현재 HTTP 요청을 균일하게 처리하도록 합니다. LabVIEW 웹 서비스 요청 입력은 LabVIEW의 참조 번호와 유사한 기능을 합니다.

다음 그림은 LabVIEW 웹 서비스 요청 입력을 사용하여 [요청 변수 읽기] VI가 데이터를 반환해야 할 요청을 식별하는 것을 보여줍니다.



노트 반드시 LabVIEW 웹 서비스 요청이 연결된 컨트롤을 최상위 레벨 HTTP 메소드 VI의 커넥터 팬에 할당해야 합니다.

HTTP 메소드 VI가 웹 서비스 팔레트에서 여러 개의 VI를 포함하는 경우, 첫 번째 웹 서비스 VI의 LabVIEW 웹 서비스 요청 입력에 하나의 컨트롤을 할당한 후, 다른 VI들을 각 VI의 LabVIEW 웹 서비스 요청 및 LabVIEW 웹 서비스 요청 출력 출력을 사용하여 함께 연결합니다.

내장된 다른 LabVIEW 노드 사용하기

HTTP 메소드 VI는 **지원되는 데이터 타입**을 사용하는 LabVIEW의 기능을 모두 포함할 수 있습니다.

관련 정보

- [POST Data, Form Data, 업로드된 파일 처리하기](#)
- [어플리케이션 웹 서버 사용 시 HTTP 세션 동안 값 저장하기](#)
- [HTTP 요청 정보 접근하기](#)

- HTTP 헤더, 스트리밍, 버퍼 설정하기

데이터 통신 방법 홈

POST Data, Form Data, **업로드된 파일 처리하기** (Real-Time, Windows)

상위 토픽: 웹 서비스 개발하기

웹 클라이언트가 웹 서비스에 HTTP 요청을 보내면, 웹 서비스는 요청을 수락한 후 URL에서 지정된 HTTP 메소드 VI로 보냅니다. HTTP 메소드 VI는 웹 서비스 팔레트의 VI와 커넥터 팬을 사용해서 요청을 처리하고, 응답을 반환합니다. 웹 클라이언트는 URL에 값을 첨부하거나, 표준 POST와 같이 HTTP 메소드에 설정된 HTML 형식을 사용해서 데이터를 보낼 수도 있습니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 웹 서비스 개요 및 길라잡이를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

HTTP 메소드 VI 설정하기

HTTP 메소드 VI가 POST Data를 수락하도록 하려면, HTTP 메소드를 POST로 설정합니다. LabVIEW 프로젝트에 있는 VI에서 마우스 오른쪽 버튼을 클릭한 후 POST를 선택합니다.

POST HTTP 메소드 VI 개발하기

기본으로, 웹 클라이언트는 HTTP 메소드 VI의 컨트롤에 직접 대응하는 값을 URL에 제출합니다. HTTP 메소드 VI는 커넥터 팬을 사용하여 컨트롤에 값을 보낼 수 있습니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 HTTP 메소드 VI가 스트리밍 출력 타입을 반환하도록 설정할 수 있습니다. 보다 고급 데이터의 경우, 웹 서비스 팔레트의 VI를 사용해서 웹 클라이언트가 요청한 데이터를 처리할 수 있습니다. 다음 VI를 사용해서 Form Data와 POST Data를 수락합니다:

- 모든 Form Data 읽기
- Form Data 읽기
- [Post Data 읽기]

LabVIEW 웹 서비스 요청 컨트롤을, POST Data를 수신하며 웹 서비스 팔레트의 VI를 포함하는 모든 HTTP 메소드 VI의 커넥터 팬에 연결해야 합니다.

웹 클라이언트로부터 업로드된 파일 받기

LabVIEW는 웹 클라이언트가 업로드한 각 파일의 정보를 클러스터의 배열에 저장합니다. 각각의 업로드된 파일에 대한 정보에는 클라이언트 파일 이름, 임시 위치의 경로, 내용 타입, 크기 (바이트) 등이 포함됩니다. HTTP 메소드 VI를 개발할 때에는 [업로드된 파일 정보 읽기] VI를 포함시켜, HTTP 요청과 관련된 모든 업로드된 파일에 대한 정보를 반환하도록 합니다. 이 정보를 사용하여 파일을 영구적인 위치로 옮기는 등의 작업을 업로드된 파일에 수행합니다.

LabVIEW 웹 서비스 요청 컨트롤을, 업로드된 파일을 받고 [업로드된 파일 정보 읽기] VI를 포함하는 모든 HTTP 메소드 VI의 커넥터 팬에 연결해야 합니다.



관련 정보

[웹 서비스 개발하기 \(Real-Time, Windows\)](#) (657 페이지 참조)

[URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기 \(Real-Time, Windows\)](#) (676 페이지 참조)

[POST HTTP 메소드 사용하기 \(Real-Time, Windows\)](#) (679 페이지 참조)

[개요: LabVIEW 어플리케이션과의 웹 기반 통신 \(Real-Time, Windows\)](#) (655 페이지 참조)

[길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 \(Windows\)](#) (1500 페이지 참조)

[웹 서비스를 사용하여 파일 업로드하기 \(Real-Time, Windows\)](#) (680 페이지 참조)

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

HTTP 요청 정보 접근하기 (Real-Time, Windows)

상위 토픽: HTTP 메소드 VI 개발하기

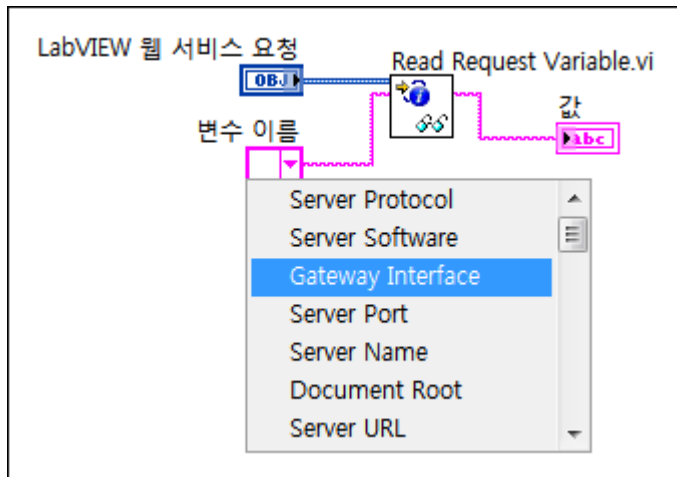
HTTP 환경 변수 혹은 요청 변수는 서버 라벨, 리모트 주소, 캐시 컨트롤을 포함하여 30 개가 넘는 웹 서비스 부분에 대한 정보를 제공합니다. [요청 변수 읽기] VI의 변수 이름 컨트롤을 사용하여 풀다운 메뉴에서 특정 요청 변수를 선택합니다. [모든 요청 변수 읽기] VI를 사용하여 현재 HTTP 요청에 대해 모든 요청 변수를 배열로 반환합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 웹 서비스 개요 및 길라잡이를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 그림은 [요청 변수 읽기] VI를 사용해서 HTTP 요청의 특성을 묘사하는 요청 변수에 접근하는 코드를 보여줍니다.



관련 정보

[HTTP 메소드 VI 개발하기 \(Real-Time, Windows\)](#) (661 페이지 참조)

개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows) (655 페이지 참조)

길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 (Windows) (1500 페이지 참조)

HTTP 헤더, 스트리밍, 버퍼링 설정하기 (Real-Time, Windows)

상위 토픽: HTTP 메소드 VI 개발하기



노트

이 토픽의 개념에 대한 더 자세한 정보는 웹 서비스 개요 및 길라잡이를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

데이터 스트리밍하기

출력 데이터를 HTTP 메소드 VI에서 클라이언트로 스트리밍할 수 있습니다. 스트림 출력 타입을 사용하여, HTTP 메소드 VI에서 사용자 정의된 포맷으로 데이터를 반환할 수 있습니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 HTTP 메소드 VI가 스트리밍 출력 타입을 사용하도록 설정할 수 있습니다.

[응답 쓰기] VI를 사용하여 사용자 정의된 HTML 페이지나 이미지 데이터와 같은 스트리밍 출력 데이터를 생성합니다.

[HTTP 응답 코드 설정] VI 및 [HTTP 응답 MIME 타입 설정] VI를 사용하여 스트리밍 출력 데이터의 응답 코드와 MIME 타입을 설정합니다.



노트 HTTP 메소드 VI는 사용자 정의된 HTML 페이지 또는 이미지 데이터와 같은 사용자 포맷으로 클라이언트에게 데이터를 스트리밍하지 않고 커넥터 팬을 통해 클라이언트에게 데이터를 반환할 수 있습니다.

HTTP 헤더 설정하기

HTTP 메소드 VI가 웹 클라이언트로 보내는 응답의 HTTP 헤더를 사용자 정의할 수 있습니다. [HTTP 헤더 설정하기] VI 및 [HTTP 다시 지정 설정] VI를 통해 스트림 출력 타입을 사용하는 웹 메소드 VI의 HTML 헤더를 사용자 정의합니다. 예를 들어, 웹 브라우저에 디스플레이되는 웹 페이지를 매 2초마다 새로 고치는 사용자 헤더를 생성할 수 있습니다. HTTP 메소드 VI의 HTTP 헤더를 사용자 정의하지 않는 경우, HTTP 메소드 VI는 응답에 대해 기본 헤더를 사용합니다.

스트리밍 데이터 버퍼하기

스트리밍 출력 데이터를 버퍼링하도록 HTTP 메소드 VI를 설정할 수 있습니다. [응답 쓰기] VI를 사용하여 버퍼되었거나, 버퍼되지 않은 스트리밍 출력 데이터를 생성합니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지를 사용하여 HTTP 메소드 VI가 스트리밍 출력 데이터를 버퍼하도록 설정합니다.



관련 정보

[HTTP 메소드 VI 개발하기 \(Real-Time, Windows\)](#) (661 페이지 참조)

[개요: LabVIEW 어플리케이션과의 웹 기반 통신 \(Real-Time, Windows\)](#) (655 페이지 참조)

[길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 \(Windows\)](#) (1500 페이지 참조)

[LabVIEW에서의 데이터 통신 방법](#) (603 페이지 참조)

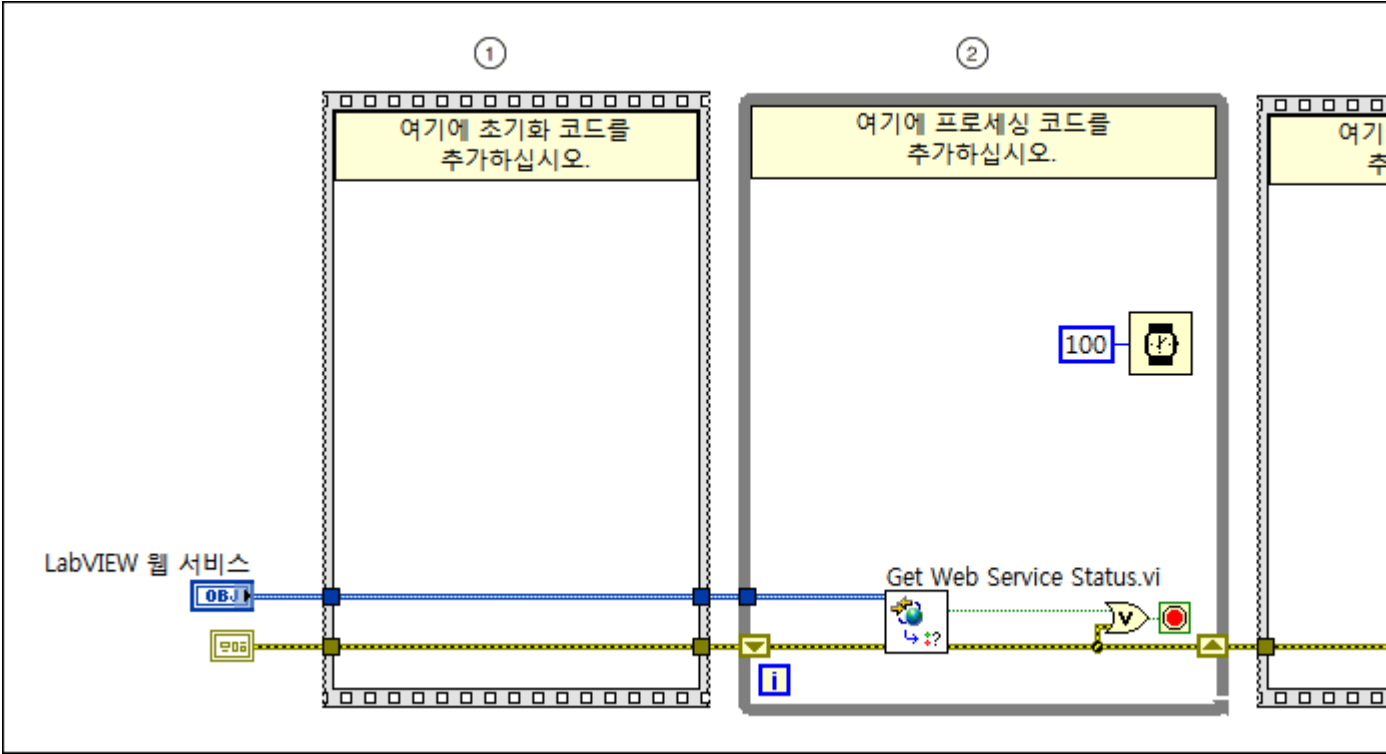
HTTP 메소드 VI를 초기화하고 통신하는 시작 VI 개발하기 (Real-Time, Windows)



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

시작 VI는 웹 서비스 어플리케이션의 일부로 실행되지만 웹 클라이언트와 데이터를 교환하지는 않습니다. 시작 VI는 웹 서비스가 로드될 때 한 번 실행되며, 웹 서비스가 공개되어 있는 동안 계속하여 실행됩니다. 시작 VI에 대한 다음 템플릿 블록 다이어그램을 고려하십시오:



①	초기화 프레임	한 번 실행되어 파 로드하기와 같은 값을 초기
②	프로세싱 루프	에러가 발생하거나 정지하여 [웹 서브 서비스 중지? 출 때까지 반복됩니 언기] vi가 웹 서 나타내면 곧 시작 v 확인합니다. 그 LabVIEW는 시작 종료 노트 LabVIEW 웹 서비스 팔라 서비스의 상 서비스에 다 것을 허용함 LabVIEW의 기능을 합니
③	정리 프레임	웹 서비스의 상태가 번 실행됩니다. 정 오랫동안 실행 중인 서버가 멈추는 것을

**팁**

이전 템플릿을 사용하는 시작 VI를 생성하려면, 프로젝트 탐색기 윈도우의 웹 서비스 프로젝트 아이템 아래의 시작 VI에서 마우스 오른쪽 버튼을 클릭한 후 새 VI를 선택합니다.

프로그래밍 가이드라인

어플리케이션의 필요에 맞도록 시작 VI 템플릿을 변경합니다. 예를 들어, 다음과 같은 변경 사항을 적용할 수 있습니다:

- 초기화 작업만을 수행하려는 경우, 프로세싱 및 정리 코드를 삭제할 수 있습니다.
- 프로세싱 루프의 [기다림 (ms)] 함수를 VI의 타이밍 요구조건에 맞게 삭제하거나 조정합니다. 또는 시작 VI가 큐로부터의 데이터를 기다리는 경우, 기다림 대신 타임아웃을 사용합니다.



데이터 통신 방법

웹 서비스 구조 및 파일 위치 이해하기 (Real-Time, Windows)

상위 토픽: [웹 서비스 개발하기](#)

이 토픽에서는 웹 서비스 구조와 웹 서비스에서 지원되는 데이터 타입, 또한 웹 서비스 파일 시스템에 대해 설명합니다.

**노트**

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

웹 서비스 구조

LabVIEW 웹 서비스는 Representational State Transfer (REST) 구조에 기반한 RESTful 웹 서비스 구조를 사용합니다. RESTful은 매우 다양한 HTTP 사용 클라이언트가 접근할 수 있는 간단한 프로토콜을 제공하고, 복잡한 메시지 분석을 필요로 하지 않습니다.

웹 서비스에서 지원되는 데이터 타입

웹 서비스가 지원하는 데이터 타입은 다음과 같습니다:

- 문자열
- 경로
- 불리언
- 숫자형 (복소수 제외)

- 열거형 (부호없는 8비트, 16비트, 32비트)
- 배열 (JSON XML 출력만)
- 클러스터 (JSON 및 XML 출력만)
- 디지털/아날로그 웨이브폼 (JSON 및 XML 출력만)
- 타임스탬프, 문자열로 포맷된 경우 (JSON 및 XML 출력만)

웹 서비스 파일 시스템 및 .LVWS 파일



노트 다음 섹션은 LabVIEW 독립 어플리케이션에 포함시킨 웹 서비스의 경우에만 해당됩니다.

LabVIEW는 .lvws 파일을 사용하여 웹 서비스를 호스트하는 웹 서버에 웹 서비스를 공개합니다. 웹 서비스를 공개할 때에는, LabVIEW는 필요한 모든 파일을 .lvws 확장자를 갖는 압축(Zip) 파일로 묶고 해당 파일을 타겟 웹 서버에 배포합니다. 그리고 나서 LabVIEW는 파일의 압축을 풀고, 웹 서비스 구성 요소를 타겟 웹 서버의 고유 디렉토리에 설치합니다.

웹 서비스 호스트하기 (Real-Time, Windows)

상위 토픽: [개요: LabVIEW 어플리케이션과의 웹 기반 통신](#)

LabVIEW는 두 가지 타입의 웹 서버 중 하나를 사용하여 네트워크 상에서 웹 서비스를 호스트합니다:

- NI 웹 서버—NI 웹 서버는 NI가 만든 LabVIEW 웹 서비스 및 SystemLink 서비스와 같이 사용자가 생성한 서비스를 호스팅할 수 있는 산업용급 웹 서버입니다. NI 웹 서버는 일반적인 웹 보안 위협으로부터 웹 어플리케이션을 보호하고 많은 엔터프라이즈 등급 데이터 서비스에 높은 확장성을 제공하며 디바이스 관리를 가능하게 합니다.
- 어플리케이션 웹 서버—어플리케이션 웹 서버는 빠르고 안전하며 컴팩트한 타사의 오픈 소스 웹 서버를 기반으로 개발됩니다.

웹 서비스에 대한 웹 서버 지원은 웹 서비스를 게시하거나 실행하는 방법에 따라 달라집니다. 다음 테이블은 각 타입의 웹 서버에 대해 지원되는 웹 서비스 공개 및 실행 방법을 보여줍니다.

공개/실행 방법	NI 웹 서버	어플리케이션 웹 서버
테스트 및 디버깅	✓	✓ (디버그 서버)
패키지를 통해 공개하기	✓	✗
설치 프로그램을 통해 공개하기	✗	✓

공개/실행 방법	NI 웹 서버	어플리케이션 웹 서버
독립 어플리케이션을 통해 공개하기	✓	✓ (어플리케이션 전용)
독립 웹 서비스로 공개하기	✗	✓

NI 웹 서버 이해하기

NI 웹 서버는 호스트 컴퓨터에서 실행되는 독립 웹 서비스 및 패키지 또는 독립 어플리케이션에 포함된 웹 서비스를 호스팅할 수 있습니다. LabVIEW Development System을 실행하지 않고, NI 웹 서버를 실행할 수 있습니다. NI 웹 서버를 한 번 활성화해 놓으면, 시스템을 부팅할 때마다 시작됩니다. 웹 서비스를 실행하거나 공개하기 전에 NI 웹 서버를 설정해야 할 수도 있습니다.



노트

NI 웹 서버에서 호스팅되는 웹 서비스의 상태 확인은 아직 지원되지 않습니다.

관련 정보

웹 서비스용 NI 웹 서버 설정하기

NI 웹 서버 사용 시 웹 서비스 보안 설정하기

어플리케이션 웹 서버 이해하기

어플리케이션 웹 서버에는 사용 방법에 따라 세 가지 모드가 있습니다.

- 시스템 전체 어플리케이션 웹 서버
- 어플리케이션 전용 어플리케이션 웹 서버
- 서버 디버깅하기

시스템 전체 어플리케이션 웹 서버

시스템 전체 어플리케이션 웹 서버는 설치 프로그램에 포함된 독립 웹 서비스 및 웹 서비스를 호스팅할 수 있습니다. 어플리케이션 웹 서버는 독립적인 시스템 서비스로 실행됩니다. LabVIEW Development System을 실행하지 않고, 어플리케이션 웹 서버를 활성화하고 실행할 수 있습니다. 어플리케이션 웹 서버를 한 번 활성화해 놓으면, 시스템을 부팅할 때마다 ApplicationWebServer.exe로 실행됩니다. 웹 서비스를 실행하거나 공개하기 전에 어플리케이션 웹 서버를 설정해야 할 수도 있습니다.

다음 단계에 따라 어플리케이션 웹 서버에 공개된 웹 서비스의 상태를 확인합니다.

- 1. 프로젝트 트리에 있는 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 어플리케이션 웹 서버>>웹 서버 관리를 선택합니다. 웹 브라우저에 NI 웹 기반 설정 및 모니터링 유틸리티가 열립니다.
- 2. 왼쪽 사이드바의 웹 서비스 관리 버튼  을 클릭합니다.
- 3. 공개된 웹 서비스 리스트에 공개된 모든 웹 서비스가 나타납니다. 새로 고침 버튼을 클릭하여 해당 리스트를 업데이트합니다.

관련 정보

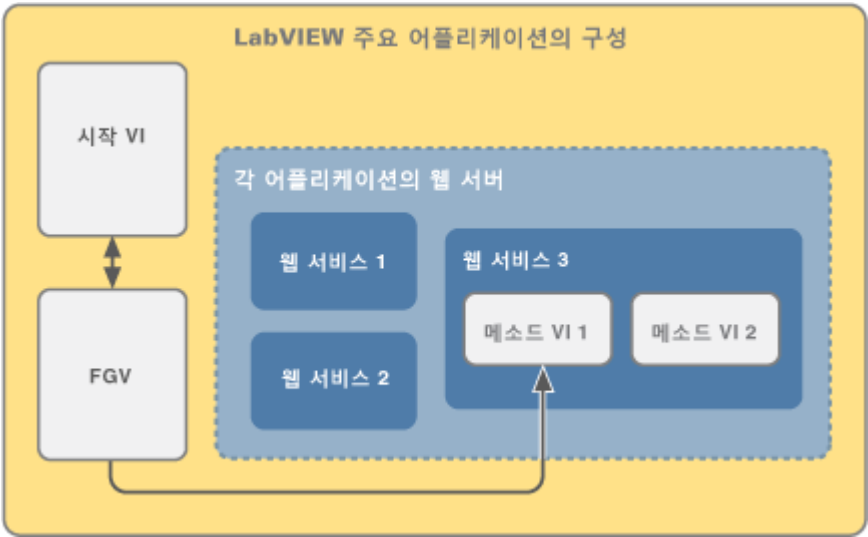
웹 서비스용 어플리케이션 웹 서버 설정하기

어플리케이션 웹 서버 사용 시 웹 서비스 보안 설정하기

어플리케이션 전용 어플리케이션 웹 서버

LabVIEW 독립 어플리케이션에 웹 서비스를 포함시키면, 어플리케이션은 웹 서비스를 직접 호스트하는 웹 서버와 같이 동작합니다. LabVIEW는 어플리케이션이 종료할 때 웹 서비스를 비공개합니다.

빌드된 LabVIEW 어플리케이션은 포함된 웹 서비스를 호스트하는 고유의 웹 서버를 시작합니다. 웹 서비스의 각 VI가 어플리케이션의 시작 VI와 동일한 상황에서 실행되기 때문에, 간단한 기능적인 글로벌 변수(FGV)를 사용하여 그 사이에서 데이터를 전달합니다. 다음 그림은 빌드된 LabVIEW 어플리케이션이 고유한 웹 서버를 시작하는 예를 보여줍니다.



어플리케이션 빌드 스펙에 웹 서비스를 포함하도록 지정할 때, 웹 서비스에 대한 요청 수락에 사용할 SSL 및 포트를 설정할 수 있습니다.

관련 정보

독립 어플리케이션을 통해 웹 서비스 공개하기

서버 디버깅하기

웹 서비스의 디버깅을 시작하면, LabVIEW는 웹 서비스를 디버깅 서버에 놓고, 웹 서비스는 디버깅 서버에서 마치 타겟에 공개된 것처럼 동작합니다.

디버깅 모드에서 웹 클라이언트가 웹 서비스로 요청을 보내는 기본 포트는 8001입니다. 다른 포트를 사용하여 디버깅을 하려면, 도구 » 옵션을 선택한 후 웹 서버 페이지를 탐색하여 HTTP 포트 디버그 옵션을 변경합니다.

관련 정보

어플리케이션 웹 서버에서 웹 서비스 테스트 및 디버깅하기

다른 LabVIEW 웹 서버

LabVIEW에는 또한 웹 서비스 내부 처리를 위한 시스템 웹 서버가 있습니다. 또한 시스템 웹 서버는 독립적인 시스템 서비스로 실행됩니다. LabVIEW를 설치한 후, 시스템 웹 서버는 시스템을 SystemWebServer.exe로 부팅할 때마다 시작됩니다. 시스템 웹 서버는 비활성화할 수 없습니다.

LabVIEW에는 **리모트 프론트 패널**을 호스트하는 별도의 웹 서버가 포함되어 있습니다.

관련 정보

웹 서비스 공개하기

웹 서비스 공개하기 (Real-Time, Windows)

웹 서비스를 호스트 컴퓨터 RT 타겟 또는 다른 Windows 컴퓨터에 공개하여 LabVIEW Development System에 의존하지 않고 웹 서비스를 실행할 수 있습니다. 웹 서비스를 공개하는 과정은 웹 서비스를 호스팅하는 데 사용하는 플랫폼 및 웹 서비스를 타겟에 배포하려는 방법에 따라 달라집니다. 또한 웹 서비스의 지속기간은 웹 서비스를 공개하기 위해 사용한 방법에 따라 다릅니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 **웹 서비스 개요** 및 **길라잡이**를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 테이블을 참조하여 웹 서비스를 공개하는 적절한 방법을 선택합니다.

용도	배포 타겟	호스팅 서버	공개 방법	실행 시작/중지	내용
타겟이 NI 패키지 관리자가 설치된 Windows 컴퓨터입니다.	Windows	NI 웹 서버	패키지(.nibkg)를 통해 웹 서비스를 공개합니다.	패키지를 설치/제거한 후	AQMP 웹 서비스 호스트 프로세스 안에 있는 각 웹 서비스에 대한 고유한 상황 정보
타겟이 NI 소프트웨어가 설치되어 있지 않은 Windows 컴퓨터입니다.	Windows	NI 웹 서버	패키지 설치 프로그램을 통해 웹 서비스를 공개합니다.	설치/제거 프로세스를 완료한 후	AQMP 웹 서비스 호스트 프로세스 안에 있는 각 웹 서비스에 대한 고유한 상황 정보
어플리케이션 웹 서버	설치 프로그램을 통해 웹 서비스를 공개합니다.	설치/제거 프로세스를 완료한 후	어플리케이션 웹 서버 프로세스 안에 있는 각 웹 서비스에 대해 고유한 상황 정보		
웹 서비스가 독립 어플리케이션과 통신해야 하며 어플리케이션 수명 안에서만 실행됩니다.	Windows	NI 웹 서버	독립 어플리케이션(.exe)을 통해 웹 서비스를 공개합니다.	어플리케이션이 실행을 시작/중지할 때	어플리케이션 시작 VI가 실행되는 주요 어플리케이션 인스턴스
Windows 및 Real-Time	어플리케이션 웹 서버				
타겟이 호스트 컴퓨터 또는 연결된 RT 타겟입니다. 웹 서비스가 독립 어플리케이션과 통신하지 않습니다.	Windows 및 Real-Time	어플리케이션 웹 서버	독립 웹 서비스로 공개합니다.	LabVIEW 프로젝트에서 웹 서비스를 공개/비공개할 때	어플리케이션 웹 서버 프로세스 안에 있는 각 웹 서비스에 대해 고유한 상황 정보



노트 NI 웹 서버는 Windows OS만을 지원합니다. 어플리케이션 웹 서버는 Windows OS 및 RT 타겟을 모두 지원합니다.

웹 서비스와 클라이언트 사이에서 데이터 전송하기 (Real-Time, Windows)

기본으로, 웹 클라이언트는 URL과 HTTP 메소드를 사용하여 HTTP 메소드 VI의 커넥터 팬의 컨트롤과 데이터를 교환합니다. 예를 들어, 커넥터 팬에 두 개의 숫자형 컨트롤이 지정된 HTTP 메소드 VI를 생성할 경우, 웹 클라이언트는 URL이나 HTTP 메소드를 사용하여 컨트롤에 값을 보냅니다. 그 후, HTTP 메소드 VI는 숫자형 컨트롤의 합계와 같은 응답을 반환할 수 있습니다. 기본 응답 포맷은 JSON 문자열입니다. JSON 문자열에는 HTTP 메소드 VI의 커넥터 팬에 지정된 인디케이터 값에 대한 요약이 들어 있습니다. 또한 응답 포맷을 설정하여 XML, HTML, 또는 일반 텍스트 포맷의 응답으로 데이터를 반환할 수도 있습니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 응답 포맷 타입을 설정합니다.



노트

HTTP 메소드 VI는 커넥터 팬을 통해 클라이언트에게 데이터를 반환하지 않고, 사용자 정의된 HTML 페이지 또는 이미지 데이터와 같은 사용자 포맷으로 클라이언트에게 데이터를 스트리밍합니다.

HTML 형식, 웹 브라우저, HTTP 클라이언트 VI와 HTTP를 사용하는 타사 어플리케이션은 HTTP 메소드를 사용하여 HTTP 메소드 VI와 데이터를 교환할 수 있습니다.



노트 브라우저를 새로 고침하는 경우와 같이 클라이언트가 HTTP 메소드 VI에 새롭게 접속할 때마다 클라이언트는 새로운 HTTP 요청을 전송합니다.

HTTP 메소드

웹 서비스는 웹 통신에서 사용되는 4개 표준 HTTP 메소드를 지원합니다.

- POST—데이터를 보내고, 파일을 업로드할 때 사용합니다. POST 메소드를 통해 일반적인 HTML 형식이나 다른 호환 가능한 클라이언트를 사용해서 POST data로 불리는 정보를 제출할 수 있습니다.
- GET—웹 서비스로부터 데이터를 불러오기 위해 사용합니다.
- PUT—파일을 업로드할 때 POST 대신 사용할 수 있는 메소드입니다. RESTful 설계에서, PUT와 POST의 용도는 다릅니다.
- DELETE—웹 서비스가 지정된 리소스를 삭제하도록 지시할 때 사용합니다.

POST, GET, PUT과 DELETE를 포함해 HTTP 메소드 정의에 대한 자세한 정보는 www.w3.org에서 World Wide Web Consortium을 참조하십시오.

관련 정보

- [웹 서비스에서 URL 맵핑 이해하기](#)
- [URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기](#)
- [POST HTTP 메소드 사용하기](#)
- [웹 서비스를 사용하여 파일 업로드하기](#)

• 웹 서비스 구조와 파일 위치 이해하기

데이터 통신 방법 홈

URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기 (Real-Time, Windows)

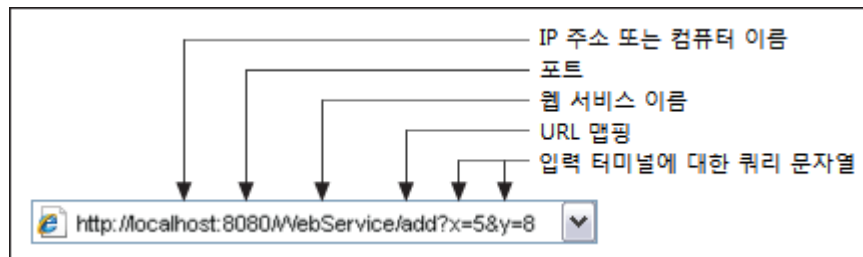
상위 토픽: 웹 서비스와 클라이언트 사이에서 데이터 전송하기



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

클라이언트는 URL과 HTTP 메소드를 사용하여 HTTP 메소드 VI의 커넥터 팬의 컨트롤에 데이터를 직접 전달하고, POST HTTP 메소드를 사용하여 POST Data와 같은 값을 전송합니다. HTTP 메소드 VI와 데이터를 교환하기 위해 웹 클라이언트가 사용하는 URL은 LabVIEW 웹 서비스를 생성할 때 결정한 다양한 값으로 구성됩니다. 다음 그림은 HTTP 메소드 VI와 데이터를 교환할 때 웹 클라이언트가 사용할 수 있는 URL의 예를 보여줍니다.



- IP 주소 또는 컴퓨터 이름 — 웹 클라이언트가 호스트 웹 서버를 실행하고 있는 시스템에서 웹 서비스에 접근하는 경우, localhost 값을 사용하십시오. 웹 클라이언트가 원격 타겟의 웹 서비스에 접근하는 경우, 타겟의 IP 주소 또는 시스템 이름을 사용하십시오. NI 웹 서버 설정을 사용하여 NI 웹 서버의 호스트 이름을 사용자 정의합니다. [NI 웹 기반 설정 및 모니터링](#)을 사용하여 어플리케이션 웹 서버의 호스트 이름을 사용자 정의합니다.
- 포트 — 호스트 웹 서버의 포트와 일치시킵니다. NI 웹 서버는 HTTP/HTTPS 프로토콜에 따라 80/443을 기본 포트로 사용합니다. 그러나 다른 어플리케이션이 해당 포트를 점유하면 포트 번호가 변경될 수 있습니다. NI Web Server Configuration에서 포트 번호를 사용자 정의할 수 있습니다. 어플리케이션 웹 서버는 8080을 기본 포트로 사용합니다. NI 웹 기반 설정 및 모니터링에서 포트 번호를 사용자 정의할 수 있습니다.
- 웹 서비스 이름 — 웹 서비스 프로퍼티 대화 상자의 서비스 셋팅 페이지에 있는 서비스 이름 옵션과 같습니다.
- URL 매핑 — 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에 있는 [URL 매핑](#)과 일치합니다. 각 HTTP 메소드 VI에 대해, LabVIEW는 해당하는 경우 VI가 속한 웹 리소스를 포함하는 기본 매핑을 정의하며, 이후 VI 이름이 나옵니다.
- 입력 터미널에 대한 쿼리 문자열 — HTTP 메소드 VI의 커넥터 팬에 할당된 컨트롤의 라벨과 일치합니다. 앞의 그림에서의 구문을 사용하여 URL 매핑에 이 쿼리문

자열을 추가합니다. 예를 들어, 쿼리 문자열 `?x=5&y=8`은 URL 매핑이 add인 HTTP 메소드 VI의 커넥터 팬의 x와 y 라벨이 있는 컨트롤에 데이터를 전달합니다. 라벨과 값의 쌍은 앰퍼샌드(&)를 사용하여 구분합니다.

관련 정보

웹 서비스에서 URL 매핑 및 쿼리 문자열 이해하기

웹 서비스 호스트하기

웹 서비스에서 URL 매핑 및 쿼리 문자열 이해하기 (Real-Time, Windows)

상위 토픽: [URL을 사용하여 공개된 어플리케이션으로 데이터 전송하기](#)



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

웹 클라이언트는 URL을 사용하여 공개된 웹 서비스와 데이터를 교환합니다. URL 매핑과 쿼리 문자열은 URL의 일부입니다:

- URL 매핑—호스트 웹 서버가 HTTP 요청을 특정한 HTTP 메소드 VI 또는 정적 파일로 보내는데 사용하는 식별자로 작용합니다. 각 HTTP 메소드 VI 및 정적 파일은 하나의 URL 매핑을 갖습니다.
- 쿼리 문자열—값을 식별하고 HTTP 메소드 VI의 커넥터 팬의 입력 파라미터에 할당합니다. 컨트롤에 할당할 물음표, 컨트롤 라벨, 등호 부호 및 값을 추가하여 URL 매핑의 끝에 입력 파라미터를 추가할 수 있습니다.

Add.vi라는 이름의 HTTP 메소드 VI에 대해서, LabVIEW는 자동으로 URL 매핑을 /Add라고 정의합니다. VI가 입력 a와 b를 포함하는 경우, URL 및 쿼리 문자열 /Add?b=2&a=3은 값을 입력에 할당합니다. 이 예에서, 클라이언트가 사용하는 URL은 `http://127.0.0.1:8080/WebService/Add?b=2&a=3`입니다.



노트 기본 URL 매핑을 사용하는 대신 [URL 매핑을 사용자 정의](#)할 수 있으며, 이는 HTTP 메소드 VI가 입력 파라미터에 값을 할당하는 방식에 영향을 끼칩니다.

쿼리 문자열에서 입력 파라미터 값 생략하기

클라이언트는 쿼리 문자열에서 어떠한 입력 파라미터도 생략할 수 있습니다. 클라이언트가 입력 파라미터 값을 생략하는 경우, 웹 서비스는 VI에 저장된 입력 터미널의 기본값을 사용합니다. 예를 들어, a와 b라는 라벨이 있는 컨트롤을 갖는 Add.vi라는 이름의 HTTP 메소드 VI의 경우, 다음 URL 매핑과 쿼리 문자열을 사용하는 HTTP 요청은 유효합니다:

- /Add?b=2&a=3
- /Add?a=3 (b 컨트롤의 기본값 사용)
- /Add?b=2 (a 컨트롤의 기본값 사용)
- /Add (두 컨트롤 모두의 기본값을 사용)

URL �핑 사용자 정의하기

보통 웹 서비스에는 기본 URL �핑을 사용할 수 있습니다. 그러나 고급 사용의 경우에는, HTTP 메소드 VI에 대한 URL �핑을 사용자 정의해야 할 필요가 있을 수 있습니다. 다음 단계에 따라 HTTP 메소드 VI에 대한 사용자 정의된 URL �핑을 설정합니다:

1. 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택합니다.
2. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지를 탐색합니다.
3. VI의 테이블에서 URL �핑을 사용자 정의하려는 HTTP 메소드 VI를 선택합니다.
4. URL �핑 탭에서, 표준 URL �핑 사용에서 확인 표시를 제거하여 URL 텍스트박스를 편집할 수 있도록 합니다. 이제 VI가 속한 웹 서비스 및 해당 웹 리소스의 이름 뒤에 나오는 URL의 부분을 편집할 수 있습니다. 즉, IP 주소, 포트, 웹 서비스 이름 또는 해당 웹 리소스 이름은 편집할 수 없습니다.

포워드 슬래쉬(/)로 시작하고 표준 HTTP 구문을 사용하는 한, URL �핑 문자열은 어느 값으로든 설정할 수 있습니다. 연결된 HTTP 메소드 VI에 포워드 슬래쉬(/), 콜론(:), 커넥터 팬 입력의 라벨을 추가하여 URL �핑의 끝에 입력 파라미터를 덧붙일 수 있습니다. 이 구문은 파라미터 값을 설정하기 위해 기본 URL �핑에 쿼리 문자열을 추가하는 것과 다릅니다.

/Add/:a/:b는 사용자 정의된 URL �핑 문자열의 예입니다. URL �핑 문자열 /Add는 URL �핑의 베이스이며 /:a/:b는 HTTP 메소드 VI의 입력 파라미터를 설정합니다. URL �핑에 HTTP 메소드 VI에 전달할 입력 파라미터가 포함되지 않은 경우, VI는 반드시 모든 입력 터미널의 기본값을 포함해야 합니다.

URL �핑을 사용하여 여러 부분 전달하기

HTTP 메소드 VI의 단일 문자열 입력 터미널에 여러 URL 부분을 할당하는 사용자 정의 브라우저 URL을 생성할 수 있습니다. 이 기법은 사용자 정의 브라우저 URL을 사용하여 파일 경로나 계층구조적인 값을 전달할 때 유용합니다.

예를 들어, /:x/*y가 URL �핑에 추가될 때 x라는 라벨이 붙은 컨트롤과 y라는 라벨이 붙은 컨트롤에 대한 입력 파라미터를 생성합니다. 그런 다음, URL �핑에 추가한 입력 파라미터 /:x/*y에 대응하는 사용자 브라우저 URL에 /test/C/directory와 같은 값을 추가할 수 있습니다. 이 예에서, x는 test에 할당되고 y는 C/directory에 할당됩니다. 1 개가 넘는 부분을 포함하는 입력 파라미터는 사용자 브라우저 URL의 마지막에 있어야 합니다.

POST HTTP 메소드 사용하기 (Real-Time, Windows)

상위 토픽: 웹 서비스에 HTTP 요청 보내기

POST [HTTP 메소드](#)를 사용하여 데이터를 HTTP 메소드 VI로 보낼 수 있습니다. 일반적인 HTML 형식인 HTTP 클라이언트 VI나 다른 호환 가능한 클라이언트를 사용해서 POST data를 수집하고, 제출할 수 있습니다. HTTP 메소드 VI에 대량의 데이터를 전송할 때에는 URL 맵핑 대신 POST 메소드를 사용합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

HTML 형식 사용해서 POST 요청 보내기

다음 예는 HTML 형식을 사용해서 웹 서비스에 POST data를 보내는 방법을 보여줍니다. 웹 서비스의 이름은 MyWebService입니다. [URL 맵핑](#)은 /add이고, POST data를 받도록 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 설정됩니다.

다음 HTML 코드는 두 개의 값 필드와 한 개의 제출 버튼을 생성합니다:

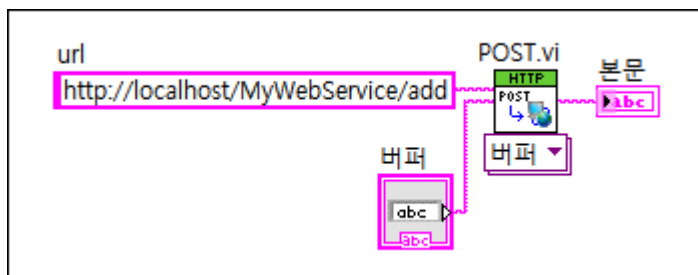
```
<form name="send" action="/MyWebService/add" method="POST">
<input type="text" name="x"><input type="text" name="y">
<input type="submit" value="Submit"></form>
```

2와 3을 각각 순서대로 x와 y로 라벨 붙여진 입력에 지정하고 제출 버튼을 클릭하면, HTML 형식은 다음의 POST data를 제출합니다:

POST /MyWebService/add/ x=2&y=3

HTTP 클라이언트 VI 사용해서 POST 요청 보내기

POST VI와 [POST 멀티파트] VI를 사용해서 LabVIEW에서 배포된 웹 서비스로 POST 요청을 보내는 웹 클라이언트를 생성할 수 있습니다. 다음 그림은 POST VI를 사용해서 이전의 예에 나왔던 HTML 형식과 유사한 HTTP 요청을 수행하는 코드를 보여줍니다.



웹 서비스를 사용하여 파일 업로드하기 (Real-Time, Windows)

상위 토픽: 웹 서비스에 HTTP 요청 보내기

POST 또는 PUT [HTTP 메소드](#)를 사용해서 웹 서비스에 파일을 업로드합니다. 웹 서비스는 파일을 .tmp 파일로 임시 디렉토리에 저장합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

HTML 형식을 사용하여 파일 업로드하기

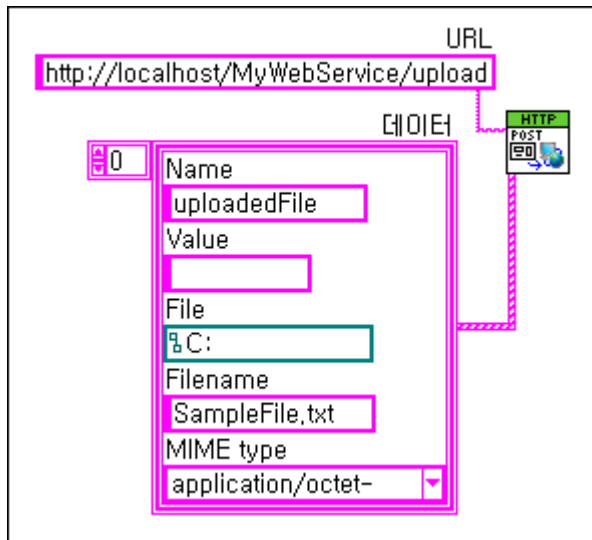
다음 HTML 코드는 POST 메소드를 사용하여 파일을 웹 서비스에 업로드하는 HTML 형식을 생성합니다. 웹 서비스의 이름은 MyWebService입니다. URL 매핑은 /upload입니다.

```
<form name="send" action="/MyWebService/upload"
  enctype="multipart/form-data" method="POST"> <input
  type="file" name="uploadedFile"> <input type="submit"
  value="Submit"> </form>
```

이 HTML 코드는 한 개의 값 필드와 한 개의 제출 버튼을 가진 형식을 생성합니다. SampleFile.txt와 같은 클라이언트 측 파일을 업로드된 파일라는 라벨을 가진 필드로 할당하고, 제출 버튼을 클릭할 경우, HTML 형식은 SampleFile.txt 파일을 웹 서비스에 제출합니다.

HTTP 클라이언트 VI를 사용하여 파일 업로드하기

POST, POST 멀티파트, PUT VI를 사용해서 웹 서비스에 파일을 업로드하는 웹 클라이언트를 LabVIEW에서 생성합니다. 다음 그림은 POST 멀티파트 VI를 사용해서 앞의 예에 나왔던 HTML 형식과 유사한 파일을 업로드하는 코드를 보여줍니다.



데이터 통신 방법

웹 서비스 반입하기

웹 서비스와 관련된 모든 복잡성을 관리하지 않고서도 웹 서비스를 사용할 수 있습니다. 웹 서비스 반입 마법사는 어떤 웹 서비스든지 VI의 라이브러리로 변환시켜서, 이후 사용자가 마치 로컬 컴퓨터에서 독립적으로 사용 가능한 것처럼 웹 서비스를 프로그래밍하는 데에 활용할 수 있습니다. 웹 서비스를 반입하려면, 웹 서비스 반입 마법사에서 웹 서비스 설명 언어 (WSDL)에 유효한 URL을 제공해야 하며, 해당 WSDL은 ASP.NET을 올바르게 검증해야 합니다. WSDL은 웹 서비스와 기능을 설명하는데 사용되는 XML 포맷의 언어입니다.



노트

- 웹 서비스 반입 마법사를 사용하려면 .NET Framework 4.0이 설치되어 있어야 합니다.
- 반입하려는 웹 서비스의 WSDL URL 변경하는 경우, 웹 서비스에 새로운 URL을 사용하도록 VI를 다시 생성하여야 합니다.

어플리케이션 웹 서버 (Real-Time, Windows)

상위 토픽: 개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows)

NI 웹 서버 또는 어플리케이션 웹 서버를 사용하여 웹 서비스를 호스팅할 수 있습니다. 이 문서의 토픽은 어플리케이션 웹 서버에 관한 것입니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 웹 서비스 개요 및 길라잡이를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.



관련 정보

개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows) (655 페이지 참조)

개요: LabVIEW 어플리케이션과의 웹 기반 통신 (Real-Time, Windows) (655 페이지 참조)

길라잡이: 어플리케이션 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 (Real-Time, Windows) (1509 페이지 참조)

어플리케이션 웹 서버 사용 시 HTTP 세션 동안 값 저장하기 (Real-Time, Windows)



노트

이 토픽의 개념에 대한 더 자세한 정보는 **웹 서비스 개요** 및 **길라잡이**를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

HTTP 세션은 웹 클라이언트와 웹 서비스 사이에서 생성된 모든 **HTTP 요청**의 모음입니다. HTTP 세션을 사용하여 하나의 웹 클라이언트에서 온 여러 개의 HTTP 요청에 인증이나 변수와 같은 데이터를 저장합니다. 예를 들어, 사용자는 HTTP 세션을 시작할 때 사용자 이름을 입력할 수 있고, 웹 서비스는 이 데이터를 저장해서 이 후의 HTTP 요청에 적용할 수 있습니다.



노트 웹 서비스는 영구 데이터를 클라이언트 시스템의 HTTP 쿠키 안에 세션 변수로 저장합니다. HTTP 세션을 사용하려면, 브라우저와 같은 웹 클라이언트 내부에 쿠키를 활성화해야 합니다.

아래 VI를 사용하여 HTTP 세션 생성 및 삭제, 세션 변수 쓰기, 읽기 및 삭제, 특정 웹 클라이언트에 현재 세션이 존재하는지 확인하기 등의 작업을 수행하십시오.

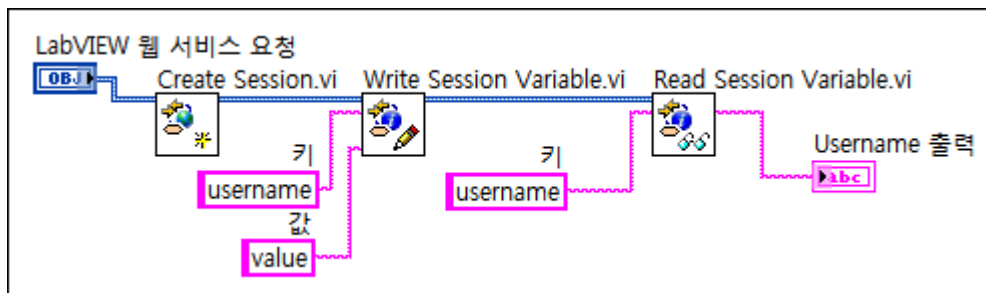
- 세션 존재하는지 확인
- 세션 생성
- 세션 변수 쓰기
- 세션 ID 쿠키 얻기
- 세션 변수 읽기
- 모든 세션 변수 읽기

- 세션 변수 삭제
- 세션 삭제

예를 들어 사용자 이름을 나타내는 세션 변수를 생성할 수 있습니다. 클라이언트는 HTTP 요청 초기에서 사용자 이름의 값을 전달할 수 있습니다. HTTP 메소드 VI는 세션 변수를 사용하여 클라이언트 시스템에 쿠키를 저장합니다. 그 후 클라이언트는 이 값을 다시 입력하지 않고도 HTTP 요청을 할 수 있습니다. HTTP 세션이나 세션 변수가 없으면, 클라이언트가 HTTP 요청을 할 때마다 사용자 이름 값이 다시 설정됩니다.

쿠키를 지원하는 웹 브라우저 또는 다른 클라이언트를 사용할 때, 쿠키를 사용하면 클라이언트가 여러 HTTP 요청에 대한 특정 HTTP 세션을 이어서 계속할 수 있습니다. [세션 생성] VI를 사용하여 새 HTTP 세션을 생성할 때 세션 ID 쿠키를 생성할 수 있습니다. [세션 ID 쿠키 얻기] VI를 사용하여 쿠키를 통해 특정 HTTP 세션을 재개할 수 있습니다.

다음 그림은 HTTP 세션을 만들고, 세션 변수 사용자 이름을 쓰고, 웹 클라이언트로부터 값을 받는 HTTP 메소드 VI의 코드를 보여줍니다. [세션 변수 읽기] VI는 사용자 이름의 값을 읽고, 이 값은 이 HTTP 메소드 VI 안에서 이후 오는 HTTP 요청에서 사용할 수 있습니다.



관련 정보

HTTP 메소드 VI 개발하기

웹 서버에서 프런트패널을 원격으로 보고 컨트롤하기

LabVIEW나 웹 브라우저로 LabVIEW에 내장된 웹 서버에 연결하여, VI의 프런트패널을 원격에서 보거나 컨트롤할 수 있습니다. 원격의 클라이언트에서 프런트패널을 열게 될 경우, 웹 서버는 클라이언트에게 프런트패널은 보내지만 블록다이어그램이나 모든 SubVI는 서버 컴퓨터에 남습니다. 블록다이어그램이 서버에서 실행되는 것을 제외하고 마치 VI가 클라이언트에서 실행되는 것처럼 동일하게 프런트패널과 상호 작용할 수 있습니다. 전체 프런트패널을 공개하거나 원격 어플리케이션을 안전하고 쉽고 빠르게 컨트롤하기 위해 이러한 기능을 사용하십시오.



노트

전체 VI를 컨트롤하려는 경우, LabVIEW 웹 서버를 사용합니다. **공유 변수**를 사용하여 VI의 단일 프런트패널에서 데이터를 읽고 씁니다.

클라이언트를 위하여 서버를 설정하기

클라이언트가 원격에서 LabVIEW나 웹 브라우저를 통해 프런트패널을 보거나 컨트롤하기 위해서는 먼저 서버 컴퓨터에 있는 사용자가 서버를 설정해야 합니다. 도구>> 옵션을 선택하고 항목 리스트에서 웹 서버 페이지를 선택하여 웹 서버를 설정합니다. 이 페이지를 사용하여 서버에 접근할 수 있는 브라우저를 컨트롤하고 원격에서 볼 수 있는 프런트패널이 어떤 것인지 지정합니다. 또한 여러 클라이언트가 VI를 컨트롤하려고 차례를 기다리는 경우 한 원격 클라이언트가 얼마간 컨트롤할 수 있는지에 대한 시간 제한을 이 페이지에서 설정할 수 있습니다.

웹 서버는 여러 클라이언트가 동시에 같은 프런트패널에 연결하도록 할 수 있도록 하지만, 한 번에 하나의 클라이언트만 프런트패널을 컨트롤하도록 합니다. 서버 컴퓨터의 사용자는 언제든지 VI의 제어권을 다시 가져올 수 있습니다. 제어권자가 프런트패널의 값을 바꾸게 되면, 모든 클라이언트의 프런트패널에도 그 내용이 전달됩니다. 하지만, 클라이언트의 프런트패널은 모든 변화를 반영하지는 않습니다. 일반적으로, 클라이언트의 프런트패널은 프런트패널 객체의 디스플레이의 변화에 영향을 받지 않고, 프런트패널 객체의 실제 값에만 영향을 받습니다. 예를 들어, 만약 제어권자가 맵핑 모드나 차트 스케일의 간격을 표시한 마커를 변경하거나, 차트의 스크롤 막대를 보이거나 숨기면, 이 모든 변화에 대해 제어권자의 프런트패널만 영향을 받습니다.

LabVIEW나 웹 브라우저로 프런트패널을 보고 컨트롤하기

클라이언트는 LabVIEW나 웹 브라우저를 통하여 원격에서 프런트패널을 보거나 컨트롤할 수 있습니다.

클라이언트와 서버 컴퓨터는 반드시 동일한 버전의 LabVIEW를 실행해야 합니다. 리모트 프런트패널을 보고 컨트롤하기 위해서 브라우저를 사용하는 경우, 반드시 서버 컴퓨터의 LabVIEW 버전과 호환되는 LabVIEW 런타임 엔진 버전을 사용해야 합니다. 또한, 서버 관리자와 연락하여 HTML 문서가 LabVIEW 런타임 엔진의 정확한 버전을 지정하는지를 확인하도록 합니다.



노트 원격에서 프런트패널을 보거나 컨트롤하기 전에 보고 컨트롤하려는 **VI나 어플리케이션**이 위치해 있는 서버 컴퓨터의 웹 서버를 활성화시켜야 합니다.

LabVIEW에서 프런트패널 보기와 컨트롤하기

LabVIEW를 클라이언트로 사용하여 리모트 프런트패널을 보려면, 수행>> 리모트 패널에 연결을 선택하여 리모트 패널에 연결 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 서버의 인터넷 주소 및 보려고 하는 VI를 지정합니다. 기본적으로 리모트 VI 프런트패널은 관찰자 모드로 초기화되어 있습니다. VI를 요청할 때 리모트 패널에 연결 대화 상자의 컨트롤 요청 확인란에 확인 표시를 하여 컨트롤을 요청할 수 있습니다. VI가 사용자 컴퓨터에 나타나면, 프런트패널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 요청을 선택합니다. 또는 프런트패널 윈도우의 아래쪽에 위치한 상태 표시를 클릭하여 이 메뉴에 접근할 수 있습니다. 현재 다른 클라이언트가 컨트롤 권한을 행사하고 있지 않을 경우, 현재 사용자에게 프런트패널의 컨트롤 권한이 부여됩니다. 현재 다른 클라이언트가 VI를 컨트롤하고 있는 경우, 서버는 다른 클라이언트가 컨트롤을 해제하거나 컨트롤 시간 제한이 타임아웃될 때까지 요청을 큐합니다. 서버 컴퓨터의 사용자만 도구>> 리모트 패널 연결 관리자를 선택하여 클라이언트 큐 리스트를 볼 수 있습니다. 원격 컴퓨터에서 실행되는 VI로 생성된 데이터를 저장하려는 경우, 리모트 프런트패널 대신 **TCP**를 사용하십시오.

클라이언트에서 관찰하거나 컨트롤하려고 하는 모든 VI는 서버 컴퓨터에서 메모리 상에 존재해야 합니다. 요청한 VI가 메모리에 상주할 경우, 서버는 요청한 클라이언트에게 이 VI의 프런트패널 데이터를 전송합니다. 요청한 VI가 메모리에 상주하지 않을 경우, 리모트 패널에 연결 대화 상자의 연결 상태 섹션이 에러 메시지를 디스플레이합니다.



노트 (Linux) NI는 Linux에서 리모트 프런트패널을 지원하지 않습니다.

웹 브라우저로 프런트패널을 보고 컨트롤하기

LabVIEW가 설치되어 있지 않은 클라이언트가 웹 브라우저로 프런트패널을 원격으로 보거나 컨트롤하기 위해서는, LabVIEW 런타임 엔진이 설치되어야 합니다. LabVIEW 런타임 엔진은 브라우저 플러그인 디렉토리에 설치되는 LabVIEW 브라우저 플러그인 패키지를 포함합니다. LabVIEW Platform 미디어에는 LabVIEW 런타임 엔진의 설치 프로그램이 포함되어 있습니다.



노트 NI는 버전 42부터 Google Chrome에 대한 리모트 프런트패널을 지원하지 않습니다. 더 자세한 정보는 NI 웹사이트를 참조하십시오.

(Linux) Firefox를 사용하여 원격으로 프런트패널을 보고 컨트롤합니다. 기본 위치에 Firefox를 설치한 경우, LabVIEW 설치 프로그램은 웹 브라우저를 사용하여 원격으로 프런트패널을 보고 컨트롤하는데 필요한 브라우저 플러그인을 자동으로 설치합니다.

클라이언트는 LabVIEW 런타임 엔진을 설치하고 그리고 서버 컴퓨터의 사용자는 클라이언트가 보고 컨트롤하는 VI를 참조하는 <OBJECT> 및 <EMBED> 태그를 포함하는 HTML 파일을 생성합니다. 이 태그는 VI의 URL 레퍼런스와 VI를 LabVIEW 브라우저 플러그인에 전달하도록 웹 브라우저에게 지시하는 정보를 가지고 있습니다. 웹 브라우저 윈도우 상단의 주소나 URL 필드에 웹 서버의 주소를 기입하여 클라이언트가 웹 서버에 접근할 수 있습니다. 이 플러그인은 웹 브라우저 윈도우에 프런트패널을 디스플레이하고, 클라이언트가 리모트 프런트패널과 상호작용할 수 있도록 웹 서버와 통신합니다. 클라이언트는 웹 브라우저의 리모트 프런트패널 윈도우의 아래쪽의 VI의 컨트롤 요청을 선택하거나 프런트패널 상의 임의의 위치에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 VI의 컨트롤 요청을 선택하여 컨트롤 권한을 요청할 수 있습니다.

현재 다른 클라이언트나 같은 연결의 웹 브라우저 윈도우가 컨트롤 권한을 행사하고 있지 않을 경우, 현재 사용자에게 프런트패널의 컨트롤 권한이 부여됩니다. 현재 다른 클라이언트가 VI를 컨트롤하고 있는 경우, 서버는 다른 클라이언트가 컨트롤을 해제하거나 컨트롤 시간 제한이 타임아웃될 때까지 요청을 큐합니다. 서버 컴퓨터의 사용자만 도구 >> 리모트 패널 연결 관리자를 선택하여 클라이언트 큐 리스트를 볼 수 있습니다.

리모트 프런트패널을 동시에 컨트롤하는 여러 클라이언트

여러 클라이언트가 동시에 원격으로 어플리케이션이나 VI를 컨트롤할 수 있습니다. VI의 동시 컨트롤을 허용하려면 VI가 **재호출**이어야 합니다. VI를 재호출로 만드려면, 파일 >> VI 프로퍼티를 선택하고 항목 리스트에서 실행을 선택한 후 공유 복제 재호출 실행 또는 미리 할당된 복제 재호출 실행 중 하나를 선택합니다. LabVIEW는 리모트 프런트패널의 각 클라이언트 요청에 대한 재호출 VI의 **복제**를 엽니다. 웹 서버:VI 접근 리스트 프로퍼티를 사용하여 프로그램적으로 리모트 프런트패널 연결시 이미 메모리에 있는 복제본에 대한 접근을 제한할 수 있습니다.

리모트 프론트패널 보기와 컨트롤에서 지원되지 않는 기능

다음 리스트에서는 리모트 프론트패널을 확인하고 컨트롤할 때 지원되지 않는 기능과 권장사항에 대해 설명합니다.

- 웹 브라우저의 제약 때문에, 프론트패널이 웹 페이지의 일부로 디스플레이될 때, 프론트패널의 넓이나 위치를 조작하는 사용자 인터페이스 어플리케이션은 잘 작동하지 않습니다. 웹 서버와 LabVIEW 브라우저 플러그인은 복잡한 사용자 인터페이스 어플리케이션도 LabVIEW에서 구현하는 것과 동일하게 실행하도록 만들지만, 특히 대화상자나 SubVI 윈도우가 있는 경우, 일부 어플리케이션은 웹 브라우저의 환경에서 잘 작동하지 않을 수 있습니다. National Instruments는 웹 브라우저에서 그와 같은 타입의 어플리케이션을 실행하지 않을 것을 권유합니다.
- 기다림 함수가 없는 While 루프가 포함된 VI의 사용은 삼가해 주십시오. 이러한 VI는 우선 순위가 낮은 태스크가 일정한 시간동안 수행하는 것을 방해하여 원격에서 프론트패널을 보고 컨트롤할 때 프론트패널이 반응을 하지 않을 수 있습니다.
- 특정 VI는 원격 컴퓨터에서 실행할 때와 로컬 컴퓨터에서 실행할 때 동일하게 수행되지 않을 수도 있습니다. 임베디드 .NET과 ActiveX 컨트롤은 LabVIEW에 상관없이 거의 독립적으로 그림을 그리거나 수행되므로 리모트 클라이언트에서는 나타나지 않습니다. VI가 표준 파일 대화 상자를 사용할 경우, 원격에서 파일 시스템을 탐색할 수 없으므로 컨트롤러는 에러를 받게 됩니다. 또한, 경로 컨트롤의 탐색 버튼은 리모트 패널에서 비활성화됩니다.
- 클라이언트가 원격에서 프론트패널을 볼 때, 이 프론트패널이 생성된 어플리케이션과 연결이 되어 있는지 아닌지에 따라서 다른 작동을 합니다. 특히, 생성된 어플리케이션의 프론트패널인 경우, 클라이언트가 이 프론트패널에 접속하기 전에 프로그램적으로 변경된 부분은 클라이언트의 컴퓨터에 적용되지 않습니다. 예를 들어, 클라이언트가 프론트패널에 접속하기 전에 프로퍼티 노드로 컨트롤의 캡션을 변경할 경우, 클라이언트는 변경된 캡션이 아닌 원래의 캡션을 보게 됩니다.
- VI 서버를 이용하여 다이나믹하게 열고 실행된 VI의 프론트패널 또는 호출될 때 프론트패널을 디스플레이하도록 설정된 subVI의 프론트패널을 컨트롤러만이 원격으로 볼 수 있습니다. VI를 컨트롤할 권한이 없는 클라이언트는 그 프론트패널을 볼 수 없습니다.
- 프론트패널 컨트롤의 프로퍼티를 폴링하여 사용자 인터페이스의 효과를 얻는 블록 다이어그램은 원격 컴퓨터에서 VI를 컨트롤할 때 성능이 저하될 수 있습니다. 이러한 VI의 성능은 [프론트패널 작업 기다림] 함수를 사용하여 향상시킬 수 있습니다.
- LabVIEW는 원격에서 보기 또는 컨트롤하려는 VI의 패널 닫기 이벤트를 생성할 수 없습니다. VI를 원격에서 보거나 컨트롤하는 경우, LabVIEW는 어플리케이션 또는 VI 클래스가 아닌 컨트롤 클래스에서만 이벤트를 생성할 수 있습니다.
- SubVI 호출 시 SubVI의 프론트패널을 디스플레이하는 VI를 원격으로 보거나 컨트롤하는 경우, 수동으로 SubVI의 프론트패널을 닫지 마십시오. 그렇지 않으면, 클라이언트 컴퓨터와 서버 컴퓨터에 있는 VI의 컨트롤을 잃을 수도 있습니다. 호출된 SubVI의 프론트패널을 확인해야 하는 경우에는, LabVIEW가 **SubVI의 실행 후 이를 닫도록** 설정하십시오.

TLS/SSL로 웹 서버 보안하기 (Windows)

리모트 프론트패널 웹 서버와 웹 서비스를 위한 어플리케이션 웹 서버에서 이전에는 SSL (Secure Sockets Layer)이라고 알려진 TLS (Transport Layer Security) 암호화를 활성화할

수 있습니다. TLS/SSL 암호화를 사용하여 클라이언트와 웹 서버 간의 데이터 전송 시 안전하고 암호화된 연결을 생성할 수 있습니다.



노트

TLS/SSL은 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

TLS/SSL에서의 인증서

TLS/SSL은 X.509 인증서를 사용하여 클라이언트와 서버 사이에 안전한 연결을 만들 수 있습니다.

인증서는 퍼블릭 키와 디지털 서명뿐만 아니라 서버에 대한 식별 정보를 포함하는 디지털 파일입니다. 식별 정보에 대한 예로 서버 이름, 유효기간, 로캘을 들 수 있습니다. 퍼블릭 키는 문자로 이루어진 문자열로 암호화를 생성하는데 쓰이며, 디지털 서명은 인증서의 진위 여부를 확인합니다. 디지털 서명은 **자체 서명한 인증서**인 인증서 생성자에 의해 추가되거나 디지털 인증서를 발행하는 신뢰할 수 있는 타사, 즉 **인증 기관 (CA: certificate authority)**에 의해 추가될 수 있습니다.

클라이언트가 서버에 안전하게 연결하려고 할 때, 서버가 이 인증서를 제공합니다. 클라이언트는 주요 웹 브라우저를 사용하여 인증서의 진위 여부를 확인합니다. 웹 브라우저는 자동으로 인증서의 진위 여부를 인증 기관(CA)의 루트 인증서 리스트와 비교하여 인증서를 수락하거나 거부하는 입력요청을 디스플레이합니다. 클라이언트가 인증서를 수락하면, 클라이언트는 퍼블릭 키를 기반으로 고유의 암호와 코드를 사용하여 서버에게 응답 통신합니다. 그러면, 서버는 인증서와 함께 생성된 프라이빗 키를 사용하여 암호를 해제합니다. 이 단계에 이르면 클라이언트와 서버는 보안되고 암호화된 연결을 생성한 것입니다.



주의 프라이빗 키는 웹 서버를 실행 중인 시스템에 남고, 클라이언트나 인증 기관 모두에게 공개되지 않습니다. 프라이빗 키는 암호화된 연결을 보전하는데 필수적이며, 초기 인증서를 생성한 시스템에 남아있어야 합니다. 프라이빗 키가 노출된 경우, 관련된 인증서의 사용을 중단하십시오.

TLS/SSL 보안 활성화하기

어플리케이션 웹 서버의 웹 설정 페이지를 사용해서 **웹 서비스의 TLS/SSL 보안을 활성화**할 수 있습니다. 옵션 대화 상자에서 **리모트 프론트패널의 TLS/SSL 보안을 활성화**할 수 있습니다. 웹 클라이언트가 TLS/SSL 보안이 활성화된 웹 서비스나 리모크 프론트패널에 연결할 때는 반드시 `https://` 프로토콜을 사용해야 합니다. 예를 들어, `https://localhost:443`는 TLS/SSL 보안화가 활성화된 웹 서버를 포트 443에서 연결합니다.

자체 서명한 TLS/SSL 인증서 생성하기

자체 서명한 인증서를 사용하면 타사 인증 기관에서 사용자의 인증서에 디지털 서명을 할 필요없이 즉시 TLS/SSL을 사용할 수 있습니다. 자체 서명한 인증서는 인증서를 만든 사람에 의해 서명되며, 이는 LabVIEW에서 TLS/SSL을 사용하는 가장 빠른 방법입니다. 또한, 인증 기관이 서명한 인증서를 받기 이전에 시스템을 테스트할 때 자체 서명한 인증서를 사용할 수 있습니다. 클라이언트가 주 웹 브라우저를 사용하여 웹 서버에 접근할 때, 자체 서명한 인증서는 추가적인 보안 요청 사항을 생성할 수도 있습니다.

LabVIEW가 TLS/SSL을 활성화할 때 생성하는 **기본 LabVIEW 자체 서명한 인증서**를 사용하거나, **NI 웹 기반 설정 및 모니터링**을 통해 사용자 자체 서명한 인증서를 생성할 수 있습니다.

LabVIEW 기본 자체 서명한 인증서

LabVIEW는 TLS/SSL을 활성화할 때 기본 자체 서명한 인증서를 생성합니다. 이때, 자체 서명한 인증서를 새로 생성하거나 다시 인증서 서명 요청 (CSR:certificate signing request)을 할 필요가 없습니다. TLS/SSL을 활성화할 때 사용자 인증서를 선택하지 않는 경우, LabVIEW는 기본 인증서를 사용합니다.

기본 자체 서명한 인증서는 생성한 날로부터 10년간 유효합니다.

(Windows 7) C:\ProgramData\National Instruments\certstore\server_certs\server_0.cer 디렉토리에서 자체 서명한 기본 인증서 파일을 찾을 수 있습니다.

NI 웹 기반 설정 및 모니터링을 TLS/SSL과 함께 사용하기

NI 웹 기반 설정 및 모니터링에서는 웹 서버 설정 페이지에 있는 SSL 인증서 관리 탭을 사용하여 자체 서명한 인증서 및 인증서 서명 요청을 생성 및 관리합니다. 다음 과정에 대한 더 자세한 정보는 NI 웹 기반 설정 및 모니터링에 포함된 도움말을 참조하십시오:

- 자체 서명한 인증서 생성하기
- 인증서 서명 요청 생성하기
- 인증서 서명 요청으로부터 인증서 생성하기

인증서 서명 요청 생성하기

인증서 서명 요청을 생성하여 인증서의 진위여부를 디지털로 인증하는 신뢰할 수 있는 타사, 즉 인증 기관의 디지털 서명으로 확인된 인증서를 받을 수 있습니다. 기업의 IT 부서와 같은 네트워크 관리자가 디지털 서명을 발행할 수도 있습니다.

인증서 서명 요청은 식별 정보와 인증서의 퍼블릭 키 구성요소를 포함합니다. 그 후 신뢰할 수 있는 인증 기관에 인증서 서명 요청을 보내면, 인증 기관에서 디지털 서명을 추가하고 완성된 인증서를 반환합니다.

인증 기관 사용하기

인증서 서명 요청을 생성할 때, 이 인증서 서명 요청을 인증 기관에 보내서 디지털 서명을 받아야 합니다. 네트워크 관리자가 선호하는 인증 기관이 있거나 디지털로 CSR을 서명할 수도 있습니다. 인증 기관은 웹 서버에 사용할 수 있도록 유효한 인증서를 생성합니다.

관련 정보

웹 서비스 보안 설정하기

웹 브라우저에서 원격 디바이스 모니터 및 설정하기

LabVIEW를 TCP, UDP 및 기타 하위 레벨 통신 어플리케이션과 함께 사용하기

LabVIEW는 컴퓨터간에 통신하기 위해 사용하는 여러가지 하위 레벨 프로토콜을 지원합니다.

각 프로토콜은 원격 어플리케이션의 네트워크 위치를 참고하는 방식이 다릅니다. 각 프로토콜은 다른 프로토콜과 일반적으로 호환될 수 없습니다. 예를 들어, macOS와 Windows 사이에서 통신하려면, TCP와 같이 양쪽 플랫폼에서 모두 동작하는 프로토콜을 사용해야 합니다.

TCP와 UDP

TCP(Transmission Control Protocol)와 **UDP**(User Datagram Protocol)는 LabVIEW가 지원하는 모든 플랫폼에서 사용 가능합니다. TCP는 신뢰성 높은, 연결을 기반으로 한 프로토콜입니다. 에러 검출을 제공하며 데이터가 순서대로 중복없이 전송되도록 보장합니다. 이러한 이유로, TCP는 일반적으로 네트워크 어플리케이션 중 가장 현명한 선택입니다.

UDP는 TCP보다 향상된 성능을 보이며 연결이 꼭 필요하지는 않지만, 전송이 이루어지는지는 보장할 수 없습니다. 일반적으로, UDP는 전송에 대한 보증이 필요하지 않는 어플리케이션에서 사용됩니다. 예를 들어, 목적지까지 데이터 전송이 충분히 빈번히 이루어지는 어플리케이션의 경우, 약간의 데이터 손실은 문제되지 않습니다.

[UDP 열기] 함수 대신 [UDP 멀티캐스트 열기] VI를 사용하여 멀티캐스트 IP 주소에 대해 UDP 데이터를 읽기, 쓰기, 또는 읽기 및 쓰기가 가능한 연결을 엽니다. 멀티캐스트 IP 주소는 멀티캐스트 그룹을 정의합니다. 멀티캐스트 IP 주소는 224.0.0.0에서 239.255.255.255의 범위 내에 있습니다. 클라이언트가 멀티캐스트 그룹에 가입하고자 하는 경우, 클라이언트는 그룹의 멀티캐스트 IP 주소를 등록합니다. 클라이언트가 멀티캐스트 그룹에 등록하게 되면 멀티캐스트 IP 주소로 보내어지는 데이터를 받을 수 있습니다.

AppleScript 코드 실행하기 (macOS)

AppleScript라는 스크립트 언어를 사용하면 macOS 어플리케이션과 그 어플리케이션의 여러 구성 요소를 컨트롤할 수 있습니다. Run AppleScript Code VI를 사용하여 LabVIEW 외부에 있는 AppleScript 코드를 사용하는 macOS 어플리케이션에 작업을 요청하거나 해당 어플리케이션으로부터 정보를 반환할 수 있습니다.

Pipe VI (Linux)

Pipes VI를 사용하여 pipes라고 이름 지어진 Linux를 열고, 닫고, 읽고, 씁니다. LabVIEW와 관계없는 프로세스 사이의 통신에는 이름 있는 파이프를 사용합니다.

시스템 레벨 명령 실행하기(Windows와 Linux)

[시스템 실행] VI를 사용하여 기타 Windows 기반 어플리케이션, 또는 Linux 명령 라인 어플리케이션을 실행 또는 시작합니다. 시스템 실행 VI를 사용하면, 시작하려는 어플리케이션이 지원하는 파라미터를 포함할 수 있는 시스템 레벨의 명령 라인을 실행합니다.

데이터 통신 방법

TCP/IP 및 UDP와 함께 LabVIEW 사용하기

Internet Protocol(**IP**), User Datagram Protocol(**UDP**), Transmission Control Protocol(**TCP**)은 네트워크 통신을 위한 기본 도구입니다. TCP/IP란 이름은 인터넷 프로토콜에서 가장 잘 알려진 프로토콜인 Transmission Control Protocol과 Internet Protocol에서 유래합니다.

TCP/IP를 사용하여 단일 네트워크 또는 상호연결된 네트워크와 통신을 할 수 있습니다. 지형적으로 멀리 떨어져 있는 각각의 네트워크도 연결할 수 있습니다. TCP/IP는 한 네트워크에서 다른 네트워크, 또는 인터넷이 연결된 컴퓨터에서 다른 컴퓨터로 데이터를 보냅니다. 대부분의 컴퓨터에서 TCP/IP를 사용할 수 있기 때문에, 다양한 시스템 사이에 정보를 전송할 수 있습니다.

LabVIEW와 TCP/IP

모든 플랫폼에서 TCP/IP 프로토콜을 LabVIEW와 함께 사용할 수 있습니다. LabVIEW는 사용자가 클라이언트 또는 서버 VI를 생성하는데 사용할 수 있는 TCP와 UDP VI 및 함수를 포함합니다.

IP

IP는 컴퓨터 사이에 데이터를 이동하는 하위 레벨 서비스를 수행합니다. IP는 데이터를 데이터그램으로 불리는 구성요소로 묶습니다. 데이터그램에는 데이터 및 소스와 대상 주소를 나타내는 헤더가 포함되어 있습니다. IP는 데이터그램의 정확한 경로를 결정하여 네트워크 또는 인터넷을 통하여 데이터를 지정된 대상에 보냅니다.

IP는 전달을 보장하지 못합니다. 실제로 데이터그램이 전송에서 복사된 경우, IP는 한 번 이상 단일 데이터그램을 전달할 수도 있습니다. 프로그램은 IP를 거의 사용하지 않고 대신 TCP 또는 UDP를 사용합니다.

UDP

UDP는 컴퓨터의 프로세스 사이의 단순한 하위 레벨 통신을 제공합니다. 프로세스는 데이터그램을 대상 컴퓨터 또는 포트에 보내어 통신합니다. 포트는 사용자가 데이터를 보

내는 위치입니다. IP는 컴퓨터와 컴퓨터 사이의 전달을 처리합니다. 데이터그램이 대상 컴퓨터에 도달한 후, UDP는 데이터그램을 대상 포트에 이동시킵니다. 대상 포트가 열려 있지 않은 경우, UDP는 데이터그램을 삭제합니다. UDP에는 IP와 같은 전달 문제가 없습니다.

UDP는 신뢰도가 그다지 중요하지 않은 어플리케이션에서 많이 사용됩니다. 예를 들어, 정보 데이터를 빈번하게 전송하는 어플리케이션의 경우, 약간의 데이터 손실은 문제가 되지 않습니다.

LabVIEW에서 UDP 사용하기

UDP는 TCP와 같은 연결 기반 프로토콜이 아니므로, 데이터를 보내거나 받기 전에 대상과 연결할 필요는 없습니다. 대신 각 데이터그램을 보낼 때 데이터의 대상을 지정합니다. OS는 전송 에러를 보고하지 않습니다.

[UDP 열기] 함수를 사용하여 포트에 UDP 소켓을 엽니다. 동시에 열리는 UDP 포트의 수는 OS에 따라 다릅니다. [UDP 열기] 함수는 고유하게 UDP 소켓을 식별하는 네트워크 연결 참조 번호를 반환합니다. 이 연결 참조 번호를 사용하여 다음 VI 호출에서 이 소켓을 참조하십시오.

[UDP 쓰기] 함수를 사용하여 데이터를 대상에 보내고, [UDP 읽기] 함수를 사용하여 해당 데이터를 읽습니다. 각 쓰기 작업에는 대상 주소와 포트가 필요합니다. 각 읽기 작업에는 소스 주소와 포트가 포함됩니다. UDP는 사용자가 보내는 각 명령에 지정하는 데이터그램 바이트를 보존합니다.

UDP 데이터그램의 크기는 작동 환경에 따라 다르지만 최대 65535 바이트입니다. National Instruments는 1 KB 이하의 데이터그램을 전송할 때 UDP를 사용하여 기본 IP 패킷이 패킷 손실의 가능성을 크게 증가시키는 전송 중 분리 또는 결합의 가능성을 최소화시키도록 권장합니다.

포트에서 모든 통신을 마치면, [UDP 닫기] 함수를 사용하여 시스템 리소스를 해제합니다.

UDP 멀티캐스트

UDP 함수를 사용하여 단일 클라이언트(단일캐스트) 또는 브로드캐스트를 통하여 서브네트의 모든 컴퓨터와 통신할 수 있습니다. 여러 특정한 컴퓨터와 통신하려는 경우, UDP 함수를 설정하여 클라이언트의 리스트를 반복해야 합니다. 이렇게하면 중복된 네트워크 트래픽이 생성됩니다. LabVIEW가 데이터 복사본을 각 클라이언트에 보내어 데이터를 받는 클라이언트의 리스트를 유지하기 때문입니다.

멀티캐스팅을 사용하면 전송자가 클라이언트의 리스트를 유지하거나 각 클라이언트에 데이터의 여러 복사본을 보내지 않고도 네트워크의 단일 전송자와 여러 클라이언트 사이의 통신을 수행할 수 있습니다. 멀티캐스트 전송자로부터 데이터 브로드캐스트를 받으려면, 모든 클라이언트가 멀티캐스트 그룹에 참여해야 합니다. 전송자는 데이터를 보내기 위해 그룹에 참가할 필요가 없습니다. 전송자는 멀티캐스트 그룹을 정의하는 멀티캐스트 IP 주소를 지정합니다. 멀티캐스트 IP 주소는 224.0.0.0에서 239.255.255.255의 범위 내에 있습니다. 클라이언트가 멀티캐스트 그룹에 가입하고자 하는 경우, 클라이언트는 그룹의 멀티캐스트 IP 주소를 등록합니다. 멀티캐스트 그룹에 등록한 후, 클라이언트는 멀티캐스트 IP 주소로 보내지는 데이터를 받게 됩니다.

LabVIEW에서 멀티캐스트를 하려면, [UDP 멀티캐스트 열기] VI를 사용하여 UDP 데이터를 읽거나, 쓰거나, 읽고 쓰기가 가능한 연결을 엽니다. 데이터 쓰기를 위해 time-to-live (TTL), 데이터 읽기를 위해 멀티캐스트 주소, 데이터 읽기 및 쓰기를 위해 멀티캐스트 포트 번호를 지정합니다. 기본 TTL은 1이며, 이는 LabVIEW가 데이터그램을 로컬 서브네트에만 보내는 것을 의미합니다. 라우터가 멀티캐스트 데이터그램을 받으면, 데이터그램 TTL이 감소됩니다. TTL이 1보다 큰 경우, 라우터는 데이터그램을 다른 라우터로 전달합니다. 다음 테이블은 time-to-live 파라미터에 값을 지정할 때 멀티캐스트 데이터그램에 어떤 동작이 발생하는지 설명합니다.

0	데이터그램은 호스트 컴퓨터에 남아있습니다.
1	해당 IP 주소를 사용하는 같은 로컬 서브넷의 모든 클라이언트에 보내진 데이터그램. 허브/반복기와 브리지/스위치는 데이터그램을 전달합니다. TTL이 1인 경우, 라우터는 데이터그램을 전달하지 않습니다.
>1	데이터그램은 보내지고, 라우터가 TTL-1 레이어를 통해서 이를 전달합니다.

UDP 멀티캐스팅을 사용하는 예제는 다음 VI를 참조하십시오:

- UDP Multicast - Receiver VI: labview\examples\Data Communication\Protocols\UDP\UDP Multicast\UDP Multicast.lvproj
- UDP Multicast - Sender VI: labview\examples\Data Communication\Protocols\UDP\UDP Multicast\UDP Multicast.lvproj

TCP

TCP는 네트워크를 통해 신뢰할 수 있는 전송을 하고 에러, 손실, 또는 중복 없이 데이터를 시퀀스로 전달합니다. TCP는 승인을 받을 때까지 데이터그램을 다시 전송합니다.

LabVIEW에서 TCP 사용하기

TCP는 사이트가 데이터를 전송하기 전에 반드시 연결되어야 하는 연결 기반 프로토콜입니다. 데이터 전송은 **클라이언트**와 **서버** 사이에서 일어납니다. TCP는 동시에 여러 개의 연결을 허용합니다.



노트

단일 포트에서 최대 1024개의 TCP 연결을 생성할 수 있습니다.

들어오는 연결을 기다리거나 능동적으로 지정된 주소와의 연결을 찾아 연결을 초기화합니다. TCP를 연결할 때, 주소와 이 주소의 포트를 지정해야 합니다. 주소가 같아도 포트가 다르면 그 주소에 대해 다른 서비스로 인식합니다.

[TCP 연결 열기] 함수를 사용하여 능동적으로 특정한 주소 및 포트와의 연결을 만듭니다. 연결에 성공하면, 함수는 고유하게 해당 연결을 식별하는 네트워크 연결 참조 번호를 반환합니다. 이 연결 참조 번호를 사용하여 다음 VI 호출의 연결을 참조합니다.

다음 기술을 사용하여 들어오는 연결을 기다릴 수 있습니다:

- [TCP 리스] VI를 사용하여 리스너를 생성하고 지정된 포트에서 TCP 네트워크 연결이 받아들여지기를 기다립니다. 연결이 성공하면, VI는 연결 참조 번호, 주소, 원격 TCP 클라이언트의 포트를 반환합니다.
- [TCP 리스너 생성] 함수를 사용하여 리스너를 생성하고, [TCP 리스너 기다림] 함수를 사용하여 새 연결을 듣고 받습니다. [TCP 리스너 기다림] 함수는 사용자가 함수에 연결한 것과 동일한 리스너 ID를 반환합니다. 새 연결에 대한 기다림을 마치면 [TCP 연결 닫기] 함수를 사용하여 리스너를 닫습니다. 리스너로부터 읽거나 리스너에 쓸 수 없습니다.

두번째 기술을 사용하는 것의 이점은 [TCP 연결 닫기] 함수를 사용하여 타임아웃을 사용하지 않고 연결을 기다릴 때 유용한 대기 동작을 취소할 수 있다는 것입니다. 그러나 다른 조건이 참이면 대기를 취소합니다. [TCP 리스] VI는 언제든지 닫을 수 있습니다.

연결을 설정하면, [TCP 읽기] 함수와 [TCP 쓰기] 함수를 사용하여 원격 어플리케이션에서 데이터를 읽고 씁니다.

[TCP 연결 닫기] 함수를 사용하여 원격 어플리케이션과의 연결을 닫습니다. 읽지 않은 데이터가 남아있는 상태에서 연결을 닫는 경우, 데이터를 잃을 수 있습니다. 컴퓨터에 상위 레벨 프로토콜을 사용하여 연결을 언제 닫을지 결정합니다.

TCP와 UDP 선택하기

TCP는 사용자가 신뢰성 있는 데이터 전송을 원할 때 가장 적합한 프로토콜입니다. UDP는 연결 없이 사용할 수 있는 고성능 프로토콜이나, 신뢰성 있는 데이터 전송을 보장하지는 못합니다.

타임아웃과 에러

네트워크 어플리케이션을 설계할 때, 에러가 발생하면 어떤 일이 일어나는지 주의깊게 고려하십시오. 예를 들어, 서버가 충돌하는 경우, 각 클라이언트 VI가 그것을 어떻게 처리할지 결정합니다.

한 가지 솔루션은 각 클라이언트 VI가 타임아웃을 가지도록 하는 것입니다. 일정 시간 후에 결과가 나오지 않는 경우, 클라이언트는 계속 실행됩니다. 실행 중에 클라이언트는 연결을 재설정 시도하며 에러를 보고할 수 있습니다. 필요한 경우, 클라이언트 VI는 어플리케이션을 닫을 수 있습니다.

TCP와 UDP 예제

TCP와 UDP VI 및 함수를 사용하는 예제는 다음 VI를 참조하십시오:

- labview\examples\Data Communication\Protocols\TCP\Simple TCP\Simple TCP.lvproj
- labview\examples\Data Communication\Protocols\UDP\Simple UDP\Simple UDP.lvproj



LabVIEW와 OPC 시스템

LabVIEW를 사용하여 OPC 시스템에 연결하기(Windows에서만 가능)

실시간 데이터 전송 시 서로 다른 제조업체가 만든 컨트롤 디바이스 사이의 상호 운용성을 확보하기 위해 OPC Foundation이 개발한 표준 인터페이스 세트. OPC Foundation은 컨트롤 시스템, 인스트루멘테이션 시스템, 프로세스 컨트롤 시스템의 제조업체 대표로 구성되어 있으며 OPC 스펙의 개발과 관리를 관장합니다.

OPC 스펙

OPC Foundation은 OPC 스펙을 개발하여 서로 다른 제조업체의 디바이스를 사용하고 해당 디바이스들과 통신하는 코드를 변경하지 않고도 디바이스 사이의 상호 변화를 할 수 있도록 해줍니다. 또한 이러한 스펙을 사용하여 공장에서 프로세스를 컨트롤하고 데이터를 수집한 후 수집한 데이터를 전송하여 기업 전체에서 사용할 수 있습니다.

OPC Foundation은 현재 여덟 개의 OPC 스펙을 관리하고 있습니다:

- OPC 데이터 접근(DA)
- OPC 알람과 이벤트
- OPC 배치
- OPC 데이터 교환
- OPC 히스토리 데이터 접근
- OPC 보안
- OPC XML-DA
- OPC Unified Architecture (UA)



노트

LabVIEW는 OPC 데이터 접근 스펙을 지원합니다. **타사 OPC 클라이언트의 LabVIEW OPC 서버에 연결**할 수 있습니다. 또한 **LabVIEW에서 OPC 클라이언트를 개발**하여 다른 OPC 서버에 연결할 수 있습니다.



노트 LabVIEW OPC UA Toolkit은 OPC UA 스펙을 지원합니다. [OPC UA] VI를 사용하여 OPC UA 서버 또는 OPC UA 클라이언트 어플리케이션을 생성할 수 있습니다.

OPC Foundation은 OPC 스펙의 주요 버전을 세 번 출시했습니다. 현재 LabVIEW가 지원하는 OPC DA 스펙 버전은 다음과 같습니다:

- Version 1.0 — DataSocket Client.
- Version 2. x — DataSocket Client, Variable Engine OPC Server, (DSC Module) DSC OPC Client.

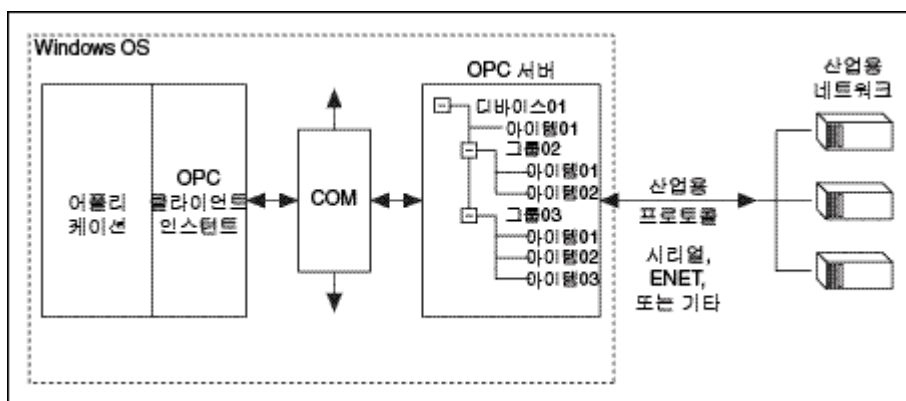
- Version 3.0 — Variable Engine OPC Server, (DSC Module) DSC OPC Client.

OPC 스펙에 대한 더 자세한 정보는 OPC Foundation 웹 사이트를 참조하십시오.

로컬 OPC 시스템

로컬 OPC 시스템의 경우, Windows를 실행하는 같은 컴퓨터에 OPC 서버와 OPC 클라이언트를 모두 설치합니다.

다음 설명에서 보이는 것처럼, 플랜트는 시리얼 또는 ENET과 같은 산업용 프로토콜을 사용하여 OPC 서버와 통신하는 컨트롤 디바이스로 구성되는 산업용 네트워크를 포함합니다.



OPC 서버가 산업 네트워크에서 데이터를 받을 때 OPC 서버는 데이터를 표준 OPC 포맷으로 변환하고 데이터가 OPC 클라이언트에 접근할 수 있도록 합니다. OPC 클라이언트는 OPC 서버를 통하여 데이터를 읽거나 새 데이터를 디바이스에 쓸 수 있습니다. OPC 서버는 데이터를 그룹과 아이템으로 구성하고 표준 인터페이스를 통해 데이터를 노출합니다. 그룹은 OPC 클라이언트가 데이터를 구성하고 업데이트 알림을 위한 그룹 레벨 요청을 하도록 허용합니다. 아이템은 OPC 서버가 공개하는 데이터에 대한 정보를 제공합니다.

OPC 서버는 Windows 기반의 기술인 COM을 따르는 여러 인터페이스를 구현하여 OPC 서버와 데이터를 교환합니다. 그렇기 때문에 OPC는 Windows 플랫폼에서만 사용 가능합니다. OPC 클라이언트가 OPC 서버에 연결되면 클라이언트는 어떤 데이터 아이템을 구독할 것인지와 얼마나 자주 업데이트를 받을 것인지 지정합니다.

원격 OPC 시스템

원격 OPC 시스템의 경우, 다른 컴퓨터에 OPC 서버와 OPC 클라이언트를 설치합니다. 두 컴퓨터 모두 Windows를 실행해야 하며 이더넷을 사용하여 컴퓨터를 연결해야 합니다.

OPC 서버에서 DCOM 설정을 적절히 설정하여 OPC 서버/클라이언트 시스템이 올바르게 작동하도록 해야 합니다. DCOM은 도메인에서 특정한 컴퓨터에 접근할 수 있는 권한을 제한하며, 보안 환경은 OPC 클라이언트가 원격 OPC 서버와 통신하는 것을 방지할 수 있습니다. 그러므로 로컬 OPC는 원격 OPC보다 설정하기 쉬운 경우가 많습니다.

또한 보안, 방화벽, OPC 서버와 OPC 클라이언트 소프트웨어 같은 여러 Windows 설정을 설정해야 합니다. Windows에서 보안, DCOM, 방화벽을 설정하는데 대한 정보는 Windows 도움말 및 지원 센터를 참조하십시오. OPC 서버와 클라이언트를 설정하는데 대한 정보는 OPC 서버 및 OPC 클라이언트 소프트웨어 문서를 참조하십시오.

가장 좋은 결과를 얻으려면 다음 순서로 설정하십시오:

- 사용자, 그룹, 각 접근 권한을 포함하는 Windows 보안
- DCOM
- 방화벽
- OPC 서버
- OPC 클라이언트

타사 OPC 클라이언트에서 LabVIEW OPC 서버에 연결하기(Windows에서만 가능)

LabVIEW 8.0 또는 이후 버전에는 **공유 변수 엔진(SVE)**이라고 불리는 OPC 서버가 포함되어 있습니다. 공유 변수 엔진(SVE)은 **OPC 데이터 접근 2.x 및 OPC 데이터 접근 3.0**을 지원합니다. **공유** 변수를 사용하여 공유 변수 엔진(SVE)의 데이터를 공개할 수 있습니다.

타사 OPC 클라이언트에서 공유 변수 엔진에 연결하려면, ProgID National Instruments Variable Engine을 사용합니다. OPC 클라이언트가 OPC 서버를 탐색하도록 허용하는 경우, 클라이언트가 지원하는 버전에 따라 OPC 버전 2.x 또는 3.0 아래에서 National Instruments Variable Engine을 찾을 수 있습니다.

타사 OPC 클라이언트에서는 공유 변수 엔진이 공개한 모든 데이터 항목을 탐색할 수 있습니다. 공유 변수 엔진을 사용하면 스칼라와 스칼라 배열에는 연결할 수 있으나, 클러스터와 같은 복잡한 데이터 타입의 데이터 항목은 인식할 수 없습니다. 해당 디바이스 항목 아래서 FieldPoint 및 DAQ 데이터 항목을 탐색할 수 있습니다.

LabVIEW에서 OPC 클라이언트 개발하기(Windows에서만 가능)

LabVIEW에서 **OPC 시스템**에 연결할 수 있는 OPC 클라이언트를 개발하는 데에는 두 가지 옵션이 있는데, 하나는 LabVIEW 개발 시스템에 포함된 LabVIEW DataSocket이고 다른 하나는 LabVIEW DSC(DataLogging and Supervisory Control) Module입니다.

LabVIEW DataSocket

DataSocket을 사용하면 opc 프로토콜로 OPC 서버에 공개된 **데이터 항목으로 연결** 열기가 가능합니다. 이 옵션은 다음 조건을 갖춘 어플리케이션에 잘 작동합니다:

- 데이터를 자동으로 로그하거나 알람, 스케일링, 보안을 추가할 필요가 없습니다.
- 하나 또는 두 개의 OPC 데이터 항목만을 참조하는 작은 어플리케이션을 가지고 있습니다.
- 업데이트 속도나 불감 대역과 같은 설정을 설정할 필요가 없습니다.

- OPC 버전 1.0 또는 2.x를 지원하는 OPC 서버에 연결해야 합니다. OPC 버전 3.0을 사용해야 하는 경우, DSC Module을 사용해야 합니다.



노트

LabVIEW 64비트는 OPC 서버에 DataSocket를 사용한 연결을 지원하지 않습니다.

LabVIEW DSC Module

DSC Module은 추가적인 기능으로 배포된 측정, 컨트롤, 다채널 카운트 모니터링 어플리케이션의 신속한 개발을 위해 LabVIEW 그래픽 개발 환경을 확장할 수 있습니다. 또한, DSC Module은 LabVIEW 공유 변수의 데이터 서비스 부분을 확장시켜서, 이를 OPC 서버에 공개된 데이터 아이템에 연결할 수 있습니다.

DSC Module은 다음 조건을 갖춘 어플리케이션에 이상적입니다:

- 데이터를 자동으로 로그하거나 알람, 스케일링, 보안을 활성화할 필요가 없습니다.
- OPC 데이터 아이템이 많습니다. DSC Module은 컴퓨터나 네트워크에 설치된 모든 OPC 서버를 찾을 수 있으며 서버 성능과 데이터 아이템에 대한 정보를 직접 OPC 서버에서 읽을 수 있습니다. 그 후 데이터 아이템을 탐색하고 업데이트 속도나 불감 대역과 같은 셋팅을 설정할 필요가 없습니다.
- 공유 변수에서 OPC 아이템에 접근해야 합니다.
- OPC 버전 3.0을 지원하는 OPC 서버에 연결해야 합니다.

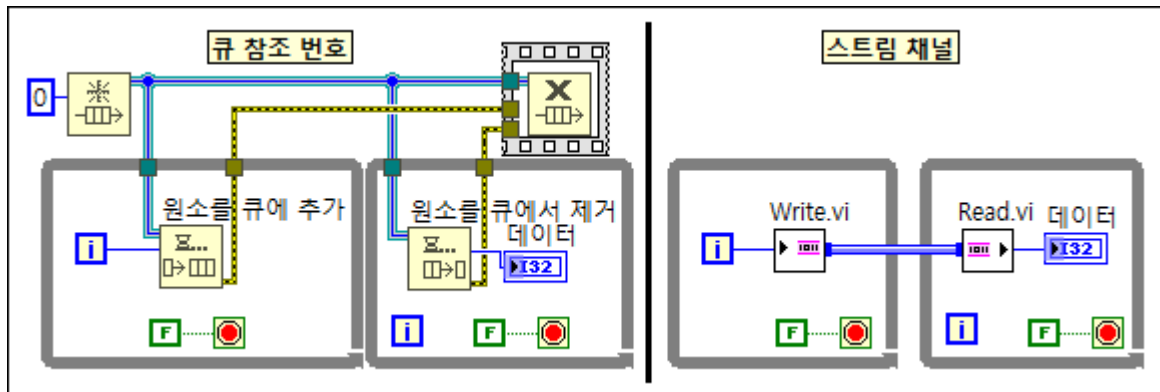
DSC Module을 OPC 클라이언트로 사용함으로써 DSC Module이 제공하는 NI OPC 서버를 포함하여 OPC 서버에 접근할 수 있습니다. 시작 >> 프로그램 >> National Instruments >> NI OPC Servers >> NI OPC Servers를 선택하여 사용할 수 있는 NI OPC Servers 어플리케이션을 통해 산업용 디바이스의 시스템으로부터 호스트 머신에 있는 클라이언트 어플리케이션으로 데이터를 전송할 수 있습니다. NI OPC Servers 어플리케이션은 인간-컴퓨터 인터페이스 소프트웨어, 생산 실행 시스템, ERP 어플리케이션과 같은 다양한 어플리케이션들 사이에서 데이터를 공유하는 것을 가능하게 합니다.

DSC Module 및 구매 정보는 ni.com/labviewdsc를 참조하십시오.

채널 와이어를 사용하여 병렬 코드 섹션 사이에서 데이터 통신하기

채널 와이어란 실행 순서를 강제 적용하지 않고 두 개의 병렬 코드 섹션 사이에서 비동기적 통신을 수행하는 LabVIEW의 와이어입니다. 채널 와이어를 사용하는 것은 한 루프에서 참조 번호나 변수를 쓰고 다른 루프에서 이를 읽도록 하는 것과 비슷합니다. 참조 번호나 변수를 사용하는 대신, 와이어를 사용하여 데이터 전송을 그래픽적으로 표현합니다.

다음 그림을 보면 채널 와이어를 사용하여 만든 블록다이어그램이 큐 참조 번호를 사용할 때와 똑같은 기능을 수행할 수 있다는 것을 확인할 수 있습니다.



모든 채널에는 최소 두 개의 끝포인트가 있어야 하며, 이는 발신자와 수신자가 됩니다. 끝포인트는 코드에서 데이터를 교환하는 곳입니다. 발신자 끝포인트는 채널에 데이터를 씁니다. 수신자 끝포인트는 채널로부터 데이터를 읽습니다. 채널 와이어를 사용하여 발신자 끝포인트를 수신자 끝포인트에 연결하면, 병렬 코드 섹션 사이에서 비동기적 데이터 통신이 활성화됩니다.

LabVIEW는 어플리케이션 빌드에 사용할 수 있는 채널 템플릿을 다음과 같이 제공합니다. 각각의 템플릿을 다른 통신 프로토콜을 사용합니다. 사용 중인 통신 방법 및 어플리케이션의 필요에 따라 다음 템플릿 중 하나를 선택할 수 있습니다.

LabVIEW	Real-Time Module	FPGA Module*
스트림 태그 메신저 어큐물레이터 태그 이벤트 메신저 고속 스트림 손실있는 스트림 단일 원소 스트림	스트림 태그 메신저 어큐물레이터 태그 이벤트 메신저 고속 스트림 손실있는 스트림 단일 원소 스트림 리얼타임 스트림	스트림 태그 어큐물레이터 태그 손실있는 스트림 단일 원소 스트림
*지원되는 각 채널 템플릿에서 FPGA Module은 끝포인트의 부분만 지원합니다.		

터미널 또는 와이어에서 마우스 오른쪽 버튼을 클릭한 후 생성 >> 채널 발신자 또는 생성 >> 채널 수신자를 선택하여 채널 발신자 끝포인트 또는 채널 수신자 끝포인트를 생성할 수 있습니다. 이때 나타나는 채널 끝포인트 선택 대화 상자에서, 읽기/쓰기 전환을 클릭하면 발신자 또는 수신자 끝포인트 사이에서 전환할 수 있습니다. 터미널에서 끝포인트를 생성하면, 채널 템플릿에서 해당 데이터 타입을 전송 타입으로 하는 인스턴스가 생성됩니다.

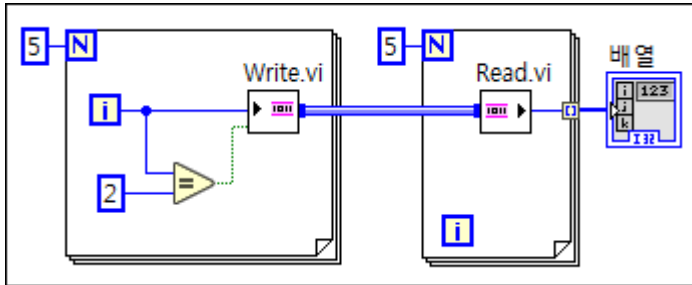
모든 스트림 채널은 분기할 수 없으며, 이는 한 개 발신자 끝포인트를 한 개 수신자 끝포인트와 연결한다는 것을 의미합니다. 다른 모든 채널은 분기할 수 있으며, 따라서 한 채널에 여러 발신자 또는 여러 수신자가 있을 수 있습니다.

채널의 수명 이해하기

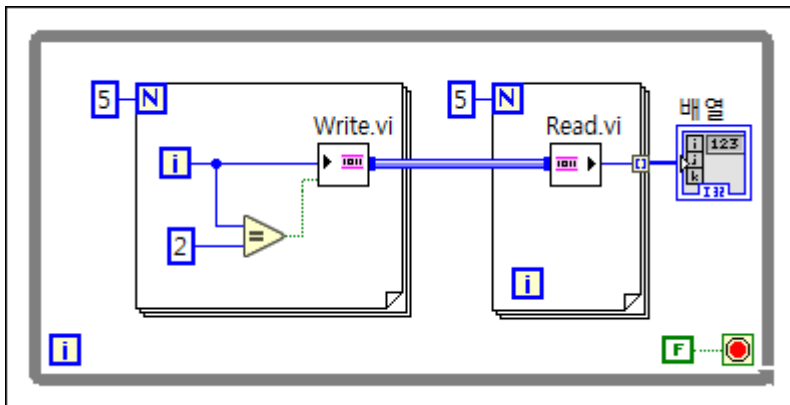
채널의 호출 블록 다이어그램이 채널 실행을 제어합니다. 호출 블록 다이어그램은 채널의 모든 끝포인트 또는 모든 끝포인트를 호출하는 모든 노드가 포함된 가장 낮은 레벨의 블록 다이어그램입니다.

채널은 채널의 호출 블록다이어그램이 실행되기 시작하면 초기화됩니다. 채널의 한 끝 포인트에서 마지막 원소? 또는 강제 종료 입력이 참이면 채널이 종료됩니다. 채널이 종료되면, 이후 채널에 작성되는 데이터는 무시됩니다. 채널의 호출 블록다이어그램이 다시 실행되기 시작하면 채널이 다시 초기화됩니다. 호출 블록다이어그램은 구조 노드, 또는 VI의 최상위 레벨 블록다이어그램이거나, 현재 최상위 레벨 VI를 호출하는 더 높은 레벨의 블록다이어그램일 수 있습니다.

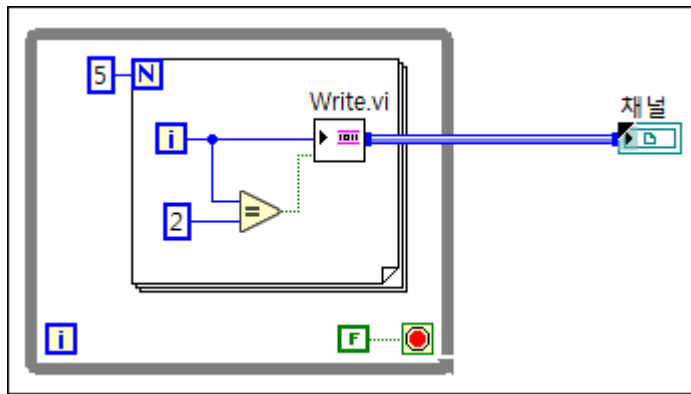
다음 그림에서, 채널의 호출 블록다이어그램은 이 VI의 최상위 다이어그램입니다. 채널은 호출 블록다이어그램이 실행되기 시작하면 초기화됩니다. 이는 최상위 레벨 VI가 실행되기 시작하거나 최상위 레벨 VI가 SubVI로 호출될 때마다 채널이 초기화된다는 것을 의미합니다.



다음 그림에서, 채널의 호출 블록다이어그램은 While 루프입니다. 채널은 While 루프가 실행되기 시작하면 초기화됩니다. 이는 While 루프가 반복할 때마다 채널이 초기화된다는 것을 의미합니다.



다음 그림에서, 채널의 호출 블록다이어그램은 최상위 VI가 아니고 현재 최상위 블록다이어그램보다 더 높은 레벨의 블록다이어그램입니다. 채널은 더 높은 레벨의 블록다이어그램이 실행되기 시작하면 초기화됩니다. 이는 더 높은 레벨의 블록다이어그램이 실행되기 시작하거나 더 높은 레벨의 블록다이어그램이 SubVI로 호출될 때마다 채널이 초기화된다는 것을 의미합니다. While 루프가 채널의 끝포인트를 모두 포함하고 있지 않기 때문에 While 루프가 반복될 때 채널은 초기화되지 않습니다.



위의 모든 채널 인스턴스에서, 쓰기 끝포인트는 처음 세 개 값을 채널에 쓰고 마지막 두 개 값은 무시하는데, 이는 배열에서 작성되지 않고 남은 원소의 개수가 2일 때 마지막 원소? 입력이 참으로 되기 때문입니다.

복제 끝포인트 정지하기

채널 수신자 또는 발신자를 생성할 때, 강제 종료를 선택하여 불리언 입력 강제 종료와 함께 끝포인트를 생성합니다. 참이면, 강제 종료가 모든 끝포인트에게 알려 강제 종료됨? 출력을 생성하도록 합니다. 모든 끝포인트는 강제 종료 신호를 보낼 수 있습니다. 채널 와이어가 포함된 루프가 있는 경우, 모든 루프를 끝포인트의 강제 종료됨? 출력에 연결하여 끝포인트가 강제 종료 신호를 보내면 루프를 정지시킵니다. 이 메커니즘에는 로컬이 필요하지 않습니다. 복제 끝포인트를 사용하는 경우, 원래 채널이 강제 종료 값을 제공해야 합니다. 복제 끝포인트는 내부 루프를 정지하고 복제된 채널에 강제 종료 값을 전달합니다.

참조 번호 및 변수와 비교해 볼 때 채널 와이어의 장점

채널 와이어를 사용하면 다음과 같은 장점이 있습니다.

- 채널 와이어를 사용하면 큐와 같은 참조 번호를 생성 및 삭제할 필요가 없습니다.
- 채널 와이어를 사용하면 큐를 사용할 때보다 블록 다이어그램의 노드 수가 줄어듭니다.
- 채널 와이어는 데이터가 어디에서 어디로 흐르는지 보여주며, 따라서 참조 번호 및 변수보다 데이터 흐름이 보다 시각적이고 디버깅이 보다 용이합니다.
- 채널 와이어는 정지 및 강제 종료 신호를 데이터와 통합하기 때문에, 참조 번호 및 변수를 사용하여 그러한 신호를 생성할 때보다 에러 발생 가능성이 낮습니다.
- 채널 와이어를 사용하면 다른 프로그래밍 언어에서 클로저의 역할을 수행하는 공통 루틴을 생성할 수 있습니다. 채널 와이어를 사용하여 공통 루틴을 생성하는 예는 `labview\examples\Channels\Replacing The Function At The Heart Of An Algorithm At Run Time\Replacing The Function At The Heart Of An Algorithm At Run Time.lvproj`를 참조하십시오.

무선 디바이스와 함께 LabVIEW 사용하기

IrDA 기술을 이용하여 디바이스는 무선 적외선 인터페이스를 통하여 통신할 수 있습니다. LabVIEW에서 서로 다른 두 컴퓨터에서 실행 중인 두 VI는 내장된 IrDA 기능을 사용하여 통신할 수 있습니다. 예를 들어, 한 데스크탑 컴퓨터에서 테스트를 수행하고 결과를 수집하는 VI를 만들 수 있습니다. 이후 적외선 빔을 통해 데이터를 다른 컴퓨터로 전송할 수 있습니다.

블루투스 기술은 라디오 주파수를 통해서 블루투스 디바이스 간에 유연하고 낮은 동력의 통신을 제공합니다. LabVIEW에서 서로 다른 컴퓨터 또는 Mobile 디바이스에서 실행 중인 VI는 블루투스 기능을 사용하여 통신할 수 있습니다.

IrDA 클라이언트/서버 네트워크의 개요

IrDA 네트워크는 네트워크의 각 주소가 고유한 이상 사용자가 IP 주소를 임의로 지정할 수 있는 고립된 TCP/IP 네트워크와 비슷합니다. IrDA 네트워크가 다이나믹이며 디바이스가 자주 네트워크에 출입할 수 있으므로, 클라이언트가 서버와 통신을 설정하기 위해 찾을 수 있는 고정된 IrDA 주소가 없습니다. 네트워크가 컴퓨터를 탐지하면, 네트워크는 각 디바이스를 이름(일반적으로 사용자가 지정)으로 식별하며 다이나믹하게 고유한 32비트 ID를 생성합니다.

무선 네트워크에서 디바이스 사이에 통신을 설정하려면, IrDA 디바이스는 네트워크에 통신을 만들고자 하는 디바이스를 찾아 네트워크를 모니터링하는 서버로서 작동합니다. 디바이스가 네트워크에 연결되는지 결정하기 위해 주소를 지정하여 연결을 개통하는 것과 반대로 서버는 리스너를 생성하여 네트워크에 들어오는 모든 디바이스에 대해서 응답할 수 있습니다. 리스너는 TCP에서 포트를 여는 것과 비슷하게 정보 접근 서비스(IAS) 데이터베이스로 불리는 서버 데이터베이스의 독립 엔트리에 접근하여 서비스를 설정합니다. 데이터베이스는 최대 128 엔트리를 포함할 수 있습니다. IAS 데이터베이스에 만들어진 각 서비스는 0부터 127 범위의 숫자인 Logical Service Access Point Selector(LSAP-SEL)와 서비스를 식별하는 문자열에 상응하는 서비스 ID에 지정됩니다.

클라이언트는 서비스 ID로 데이터베이스를 쿼리하여 LSAP-SEL 번호를 찾습니다. LSAP-SEL 번호가 확정된 후 디바이스 사이의 통신이 시작됩니다.

예를 들어, 서비스 ID 온도로 서비스를 식별할 수 있습니다. 그래서 서버가 클라이언트와 연결을 설정하면 서버는 온도의 수집을 클라이언트에 보냅니다. 그 후 서버는 클라이언트가 서비스 ID 온도를 요청하는 것을 듣습니다. 클라이언트가 네트워크에 연결되면, 서비스 온도를 서버에 보내고, 서버는 다시 서비스에 LSAP-SEL 번호를 설정합니다. 그 후, 클라이언트는 서버를 쿼리하여 온도 서비스에 대응하는 LSAP-SEL 번호를 찾습니다. LSAP-SEL 번호가 설정된 후 서버는 온도 데이터를 클라이언트에 보냅니다.

IrDA 기술에 대한 추가적인 정보는 국제 적외선 통신 데이터 협회 웹 사이트를 참조하십시오.

LabVIEW에서 IrDA 통신 모델

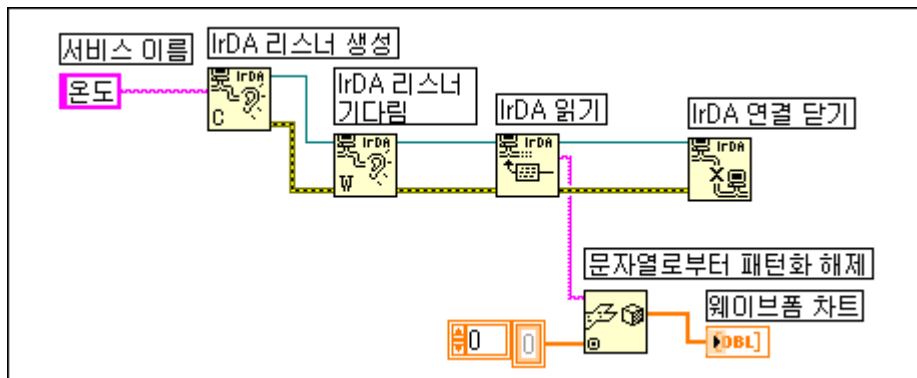
컴퓨터에 IrDA 표준에 맞는 적외선 디바이스를 설치하고 올바르게 작동하는 것을 확인한 후, IrDA 어플리케이션 만들기를 시작할 수 있습니다.

서로 다른 컴퓨터에서 실행되는 VI 사이에 무선 통신 링크를 설정하려면, IrDA 함수를 사용하여 IrDA 서버를 생성합니다. 서버는 원격 컴퓨터와의 통신 라인을 설정합니다. 서버

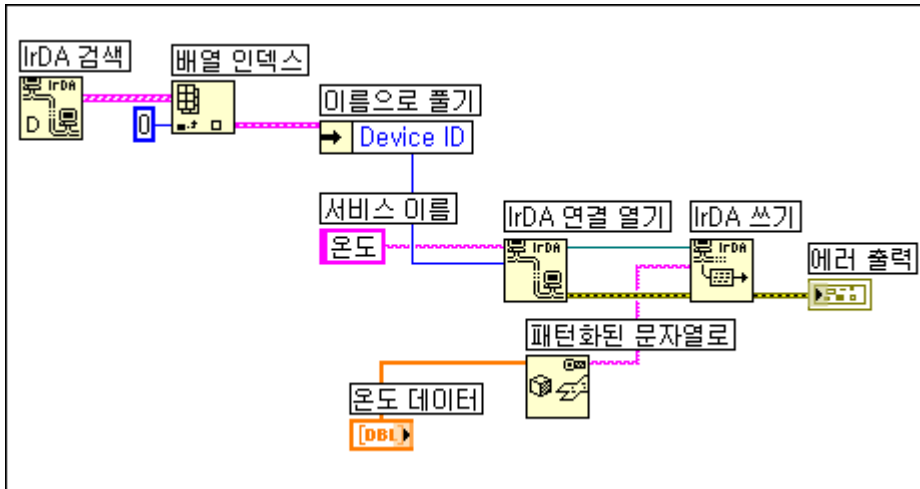
는 원격 컴퓨터로부터의 전송을 리슨하고 검색합니다. 서버는 원격 컴퓨터와 데이터를 주고 받은 후에 연결을 닫습니다.

- [IrDA 리스너 생성] 함수를 사용하여 서비스를 설정합니다. 또한, 이 함수를 사용하여 사용자의 VI와 원격 컴퓨터의 VI가 LSAP-SEL 번호를 설정하는데 사용하는 서비스 이름을 지정합니다.
- [IrDA 리스너 기다림] 함수를 사용하여 원격 컴퓨터의 인식을 기다립니다. 컴퓨터에 부착된 적외선 센서가 서비스 ID를 전달하는 다른 컴퓨터를 감지하면, 서버는 자신과 클라이언트 사이에 통신을 설정합니다. 한 번에 하나의 컴퓨터와만 통신할 수 있습니다.
- [IrDA 쓰기] 함수를 사용하여 데이터를 원격 컴퓨터의 VI에 씁니다. 쓰는 데이터는 문자열이어야 합니다. [패턴화된 문자열로] 함수를 사용하여 문자열이 아닌 모든 데이터를 문자열 포맷으로 변환합니다. 그 후, 원격 컴퓨터에서 문자열을 패턴화 해제해야 합니다. 또한, [패턴화된 XML로] 함수를 사용하여 XML 포맷으로 데이터를 변환할 수 있습니다.
- [IrDA 읽기] 함수를 사용하여 원격 컴퓨터의 VI로부터 데이터를 읽습니다. 원격 컴퓨터는 데이터가 다른 데이터 타입인 경우에도 데이터를 문자열로 보냅니다. [문자열로부터 패턴화 해제] 함수를 사용하여 데이터를 올바른 데이터 타입으로 변환합니다. 또한, **[XML로부터 패턴화 해제]** 함수를 사용하여 XML 포맷으로부터 데이터를 변환할 수 있습니다. 어떤 종류의 데이터를 받는지 알아야 합니다.
- [IrDA 연결 닫기] 함수를 사용하여 서비스를 닫습니다.

다음 그림의 VI는 온도라고 불리는 서비스를 생성하고, 해당 서비스를 요청하는 원격 컴퓨터를 듣고, 디바이스가 수집한 온도 데이터를 읽고, 문자열을 숫자 배열로 패턴화 해제하고, 그 데이터를 차트에 플롯하고, 연결을 닫습니다.



다음 그림의 VI는 원격 서버의 디바이스 ID를 찾고, 서비스 온도에 연결을 설정하고, 숫자 배열을 문자열로 패턴화하고, 온도 데이터를 서버에 씁니다.



블루투스 클라이언트/서버 네트워크의 개요

블루투스는 2.4 GHz의 라디오 주파수를 사용하여 디바이스의 통신을 가능하게 하는 무선 기술입니다. 블루투스 연결의 범위는 디바이스와 주변 상황에 따라 30에서 40피트 사이입니다.

LabVIEW 블루투스 VI와 함수는 Winsock 인터페이스가 노출하는 연결 프로토콜인 RFCOMM을 사용합니다. RFCOMM은 시리얼 통신과 유사한 단순 전달 프로토콜입니다. RFCOMM 인터페이스는 블루투스 서버와 클라이언트를 정의합니다.

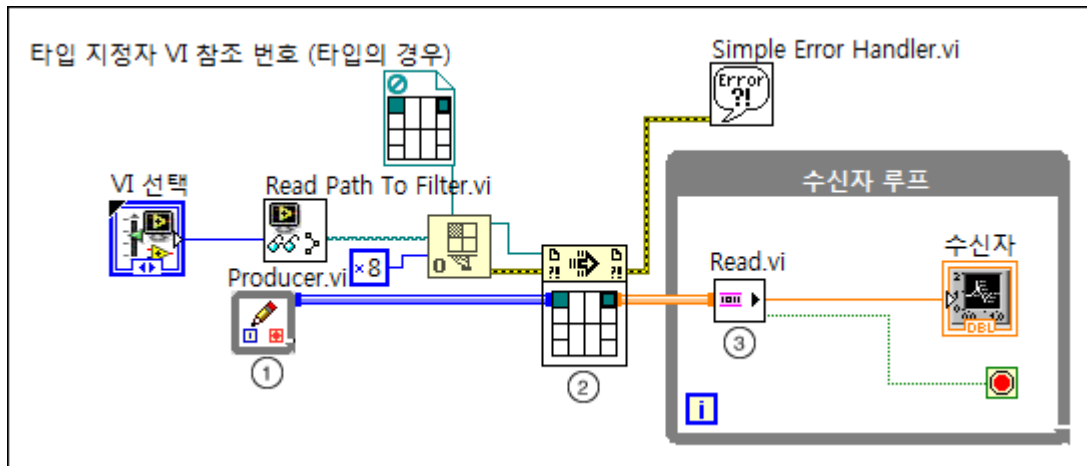
LabVIEW에서 블루투스 서버와 클라이언트 어플리케이션 생성은 TCP 통신에서의 서버 및 클라이언트 어플리케이션 생성과 유사합니다. 블루투스 서버는 Service Discovery Protocol(SDP)을 사용하여 서버가 포함하는 서비스가 사용 가능한지 알리고 들어오는 연결을 듣습니다. 클라이언트는 서버에 나가는 RFCOMM 연결을 생성합니다. 일단 클라이언트와 서버가 서로 연결되면, 클라이언트 또는 서버가 연결을 끊거나 연결이 사라질 때까지 서로 데이터를 교환합니다.

LabVIEW는 Microsoft Bluetooth 드라이버를 사용하는 블루투스 디바이스를 지원합니다. Microsoft Bluetooth 드라이버를 지원하는 블루투스 디바이스에 대해서는 Microsoft 웹사이트를 참조하십시오. 대부분의 블루투스 디바이스는 기본적으로 고유의 블루투스 드라이버를 사용합니다. LabVIEW에서 디바이스를 사용하려면 반드시 Microsoft Bluetooth 드라이버로 바뀌어야 합니다.

블루투스 기술에 대한 더 자세한 정보는 Bluetooth 및 Microsoft 웹사이트를 참조하십시오.

참조에 의한 호출 노드와 동적으로 연결하기

채널 와이어는 병렬 코드 섹션 사이에서 정적 연결을 보여주고, 참조에 의한 호출 노드와 함께 작동하는 경우 동적 연결을 보여줍니다. 다음 그림에서는 채널 와이어가 참조에 의한 호출 노드와 동적으로 연결되어 있습니다.



다음은 위의 다이어그램에서 중요한 사항을 상세히 설명한 것입니다.



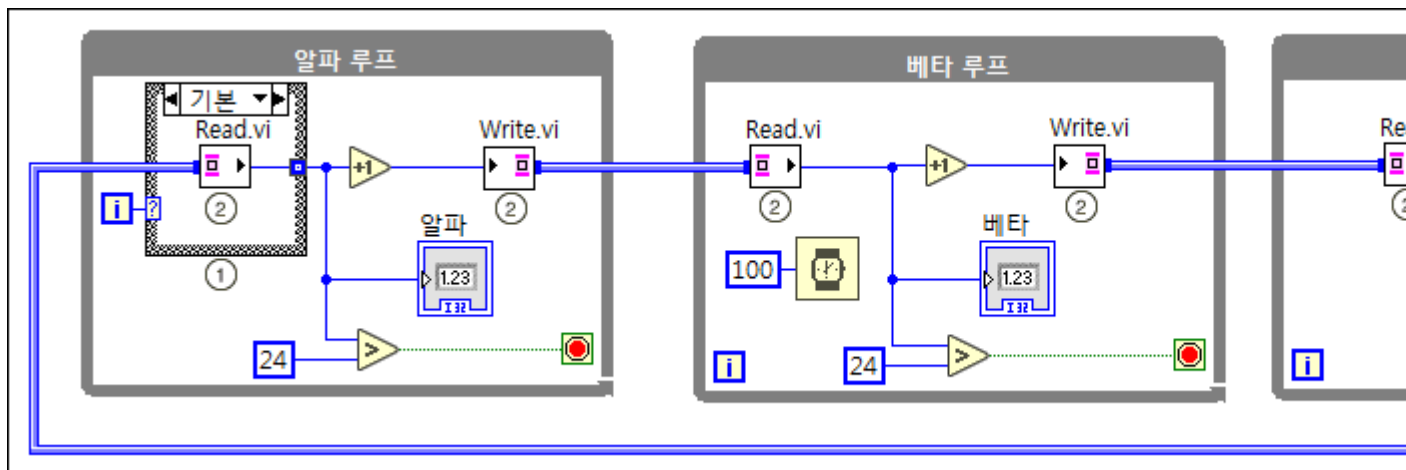
노트

발신자 끝포인트와 참조에 의한 호출 노드를 연결하는 채널 와이어의 색은 참조에 의한 호출 노드와 수신자 끝포인트를 연결하는 채널 와이어의 색과 다릅니다. 이는 채널 와이어가 데이터 패턴의 색이 아닌, 전송 데이터 타입의 색을 디스플레이하기 때문입니다.

노드 간 사이클 생성하기

사이클은 데이터 흐름이 노드의 출력에서 시작되고 같은 노드의 입력으로 끝나는 경우 발생합니다. 채널 와이어 이외에 다른 와이어를 사용하는 경우, 이러한 사이클에 시작점을 제공하도록 [피드백 노드]를 추가하지 않으면 사이클이 유효하지 않아 VI가 깨지게 됩니다. 채널 와이어가 없으면, 채널 와이어로 인해 끝포인트 간 데이터 흐름 의존성이 발생되지 않기 때문에 [피드백 노드]를 추가하지 않고 사이클을 생성할 수 있습니다. 모든 끝포인트는 시작점의 역할을 합니다.

이러한 기능은 서로 의존하는 여러 루프 사이에서 메시지를 주고받아 때때로 상태를 업데이트해야 할 때 유용합니다. 다음 그림은 채널 와이어를 사용하여 만들 수 있는 사이클을 보여줍니다. 3개의 루프가 병렬로 실행되고 정수의 값을 돌아가며 전달합니다. 이때 정수는 각 루프를 지날때마다 증가합니다.



다음은 위의 다이어그램에서 중요한 사항을 상세히 설명한 것입니다.



노트

또한 어느 한 끝포인트에 타임아웃을 추가하거나, 아니면 태그 채널과 같이 기다리지 않는 템플릿 채널을 사용하여 데이터 흐름 의존성을 제거할 수 있습니다.

NI Service Locator

NI Service Locator는 등록된 서비스를 저장하고 서비스를 요청하는 클라이언트에 데이터를 반환합니다. 클라이언트는 서비스 이름을 사용하여 NI Service Locator를 쿼리하고 해당 서비스 이름과 연결된 포트 번호를 얻을 수 있습니다. 그렇기 때문에, 클라이언트는 서버 어플리케이션의 포트 번호를 알 필요가 없습니다. NI Service Locator는 배경 서비스로 실행되며 사용자와의 상호작용은 필요하지 않습니다.

NI Service Locator로 서비스 등록하기

[TCP 리스너 생성] 함수 및 [UDP 열기] 함수의 서비스 이름 입력을 사용하여, NI Service Locator로 서비스 이름과 포트 번호를 등록할 수 있습니다. 또한, 옵션 대화 상자의 VI 서버 페이지를 사용하여 포트 번호 또는 서비스 이름을 설정할 수 있습니다. 서버:포트 및 서버:서비스 이름 프로퍼티를 사용하여 프로그램적으로 포트 번호 또는 서비스 이름을 설정할 수 있습니다.

서비스 이름을 등록하는 함수 사용에 대한 예제는 다음 VI를 참조하십시오:

- labview\examples\Data Communication\Protocols\TCP\Simple TCP\Simple TCP.lvproj

NI Service Locator를 쿼리하여 등록된 서비스 찾기

NI Service Locator로 서비스를 등록한 후, [어플리케이션 참조 열기] 함수, [TCP 연결 열기] 함수, [UDP 쓰기] 함수를 사용하여 지정한 서비스 이름과 연결된 포트 번호를 NI Service Locator에서 쿼리할 수 있습니다.

NI Service Locator에 대한 추가적인 정보는 기술지원 데이터베이스를 참조하십시오.

멀티태스킹, 멀티스레딩, 멀티프로세싱

멀티태스킹은 OS가 진행 중인 여러 작업 사이를 신속하게 오고 가며 작업을 하여, 마치 여러 작업을 동시에 실행하는 것처럼 보이는 것을 의미합니다. 예를 들어, Windows 3.1에서 일반적으로 작업은 Microsoft Word, Microsoft Excel, 또는 LabVIEW와 같은 전체 어플리케이션입니다. 각 어플리케이션은 다음 어플리케이션이 실행되기 전까지 짧은 시간 동안 실행됩니다. Windows 3.1은 협력형 멀티태스킹으로 알려진 기술을 사용합니다. 즉, OS는 실행 중인 어플리케이션에 의존하여 프로세서의 컨트롤을 정기적으로 OS에게 넘겨줍니다. Windows는 선점형 멀티태스킹에 의존합니다. 즉, 현재 실행되는 어플리케이션의 상태와 관계 없이 OS가 언제나 프로세서의 컨트롤을 가져올 수 있습니다. 선점형 멀티태스킹을 사용하면 사용자와 많은 데이터 처리량에 보다 잘 대응할 수 있습니다.

멀티스레딩은 멀티태스킹의 개념을 어플리케이션까지 확대한 것으로, 단일 어플리케이션 내의 특정 작업을 개별적인 스레드로 나눌 수 있어, 여러 작업이 이론적으로 병렬 실행될 수 있습니다. 그 후, OS는 서로 다른 어플리케이션 뿐만 아니라 한 어플리케이션 내의 각 스레드 사이에도 처리 시간을 나눌 수 있습니다. 예를 들면, LabVIEW 멀티스레드 프로그램에서 어플리케이션은 세 스레드로 나눌 수 있습니다 — 사용자 인터페이스 스레드, 데이터 수집 스레드, 인스트루먼트 컨트롤 스레드 — 각 스레드는 우선순위가 지정될 수 있으며 독립적으로 작동할 수 있습니다. 그러므로, 멀티스레드 어플리케이션에는 다른 어플리케이션과 병렬로 진행되는 여러 태스크가 있을 수 있습니다.

멀티프로세싱(또는 멀티코어 프로그래밍)은 한 컴퓨터에 두 개 또는 이상의 프로세서가 있어서 각각 별도의 스레드를 동시에 실행할 수 있음을 의미합니다. 그러므로 싱글 스레드 어플리케이션은 한 번에 한 프로세서만 실행할 수 있습니다. 멀티스레드 어플리케이션에서는 여러 스레드가 여러 프로세서에서 동시에 실행될 수 있습니다. LabVIEW 멀티스레드 예제에서, 사용자 인터페이스 스레드가 한 프로세서에서 실행되는 동안 데이터 수집 스레드가 다른 프로세서에서 실행될 수 있습니다. 단일 스레드 어플리케이션은 시스템 성능에 심각한 영향을 미칠 수 있습니다.



노트

LabVIEW Desktop Execution Trace Toolkit 을 사용하여 스레드 사용과 메모리 손실, 기타 LabVIEW 프로그래밍의 영역을 모니터링할 수 있습니다.

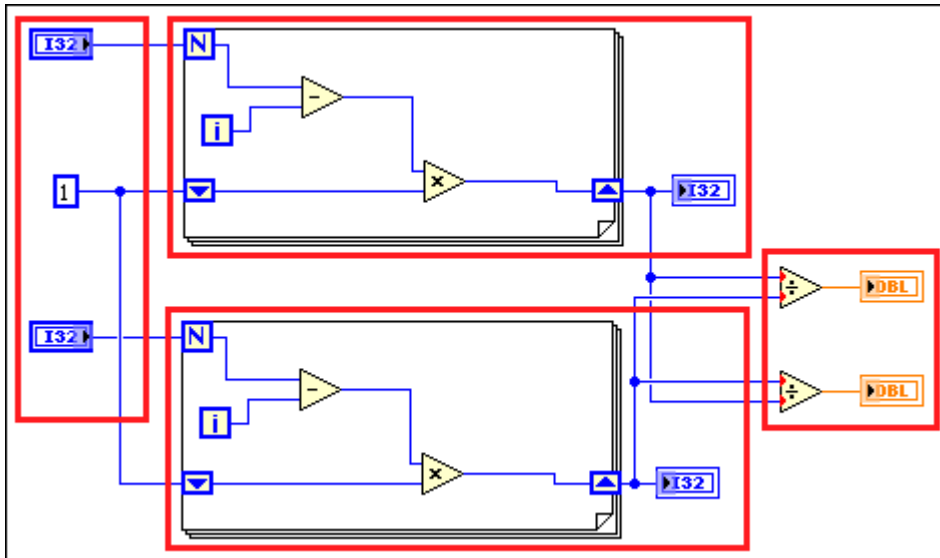
LabVIEW 스레딩 모델

스레드 관리의 모든 복잡한 작업이 LabVIEW 실행 시스템 안에 내장되어 있습니다. 텍스트 기반 프로그래머는 멀티스레드 어플리케이션을 생성하기 위해 새롭고 복잡한 프로그래밍 방법을 배워야 합니다. 그러나 모든 LabVIEW 어플리케이션은 코드 수정 없이도 자동적으로 멀티스레드가 됩니다.

OS의 선점형 멀티태스킹과 더불어 LabVIEW는 협력형 멀티태스킹 시스템을 사용합니다. 편집하는 동안, LabVIEW는 VI를 분석하여 클럼프 내에서 함께 실행될 수 있는 노드의 그룹을 찾습니다. 각 우선순위와 실행 시스템의 조합은 클럼프가 함께 실행될 수 있도록 유지하는 실행 큐 데이터 구조를 가집니다. 실행 시스템이 스레드를 활성화하면, 실행 시스템은 실행 큐에서 클럼프를 가져와 실행합니다. 실행 시스템이 실행을 마치면, 실행 큐의 입력 조건을 충족하는 부가적인 클럼프를 저장하며, 이에 따라 블록다이어그램은 4개의 사용 가능한 실행 스레드 모든 스레드에서 실행 가능합니다. 블록다이어그램이 충분한 병렬 구조를 포함하는 경우, 모든 스레드 안에서 동시에 실행될 수 있습니다.

LabVIEW는 특정한 스레드에 영구히 코드의 클럼프를 지정하지 않습니다. LabVIEW는 다음에 VI를 실행할 때 다른 스레드를 사용하여 클럼프를 실행할 수 있습니다.

다음의 블록다이어그램에서 빨간색 사각형은 LabVIEW가 어떻게 클럼프에서 노드를 그룹화하는지의 예를 보여줍니다.



이 예에서, 컨트롤의 클럼프는 인디케이터의 클럼프와 같기 때문에 블록 다이어그램은 3개의 다른 클럼프를 가지게 됩니다. 이에 따라, 이 클럼프는 두 번의 다른 시간대에 실행됩니다. 클럼프는 실행 전과 후, 휴면하거나 지연합니다. 왼쪽 첫번째 클럼프는 실행 후 For 루프의 다른 두 개의 클럼프가 실행되는 동안 휴면합니다. VI가 인디케이터의 클럼프에서 작업을 실행할 준비가 되면, 오른쪽과 왼쪽 클럼프는 휴면을 멈춥니다.

LabVIEW에서 멀티태스킹

LabVIEW는 선점형 및 협력형 멀티스레딩을 사용합니다. 선점형 멀티스레딩을 가진 OS와 프로세서는 한정된 수의 스레드를 사용하므로, 특정한 경우에는 이런 시스템이 협력형 멀티스레딩 사용으로 돌아갑니다.

실행 시스템은 스레드를 사용하여 선점적으로 VI를 멀티태스킹합니다. 그러나 사용 가능한 스레드 수는 한정되어 있습니다. 완전한 병렬 어플리케이션의 경우, 실행 시스템은 유효한 스레드가 사용 중일 때 협력형 멀티태스킹을 사용합니다. 또한, OS는 어플리케이션과 다른 작업 사이의 선점형 멀티태스킹을 다룹니다.

기본 실행 시스템

기본 실행 시스템은 활성화 작업의 큐를 유지합니다. 예를 들어 세 루프가 병렬로 실행되는 경우, 어느 시점에서나 한 작업이 실행되고 다른 두 작업은 큐에서 기다립니다. 모든 작업의 우선순위가 같다고 가정하면, 작업 중 하나가 특정한 시간동안 실행됩니다. 그 후 해당 작업은 큐의 끝으로 이동하고 다음 작업이 실행됩니다. 작업이 완료되면 실행 시스템은 큐에서 작업을 삭제합니다.

실행 시스템은 VI의 생성된 코드를 호출하여 큐의 첫 원소를 실행합니다. 일정 포인트에서, 해당 VI의 생성된 코드는 실행 시스템을 확인하여 실행 시스템이 다른 작업에 실행을 지정했는지 확인합니다. 지정하지 않은 경우, VI의 코드는 계속 실행됩니다.

동기화/차단 노드

블록다이어그램의 몇몇 노드 또는 아이템은 동기화되어 있습니다. 즉 다른 노드와 멀티태스킹하지 않습니다. 멀티스레드 어플리케이션에서 이런 노드는 완료까지 실행되며, 노드가 실행되는 스레드는 해당 작업이 완료될 때까지 독점됩니다.

컨트롤 또는 어플리케이션 참조에 사용되는 [프로퍼티 노드]와 [인보크 노드], 컨트롤 참조의 값 프로퍼티도 비스레드와 동기화됩니다. 그러므로 많은 양의 데이터를 그래프에 디스플레이하는 경우와 같이 비스레드가 사용 중인 경우, 프로퍼티 노드 또는 인보크 노드는 비스레드가 현재 작업을 완료할 때까지 실행되지 않습니다.

공유 라이브러리 호출과 계산 함수는 동기적으로 실행됩니다. 대부분의 분석 VI와 데이터 수집 VI는 공유 라이브러리 호출을 포함하고 있으므로 동기적으로 실행됩니다. 단일 비재호출 VI를 설정하여 공유 라이브러리를 호출함으로써 한 번에 하나의 스레드만이 비재호출 공유 라이브러리를 호출하도록 합니다. 하나 이상의 재호출되지 않은 VI가 공유 라이브러리를 호출하는 경우, LabVIEW는 예상치 않은 결과를 반환할 수 있습니다.

대부분의 다른 노드는 비동기화되어 있습니다. 예를 들어 구조, I/O 함수, 타이밍 함수, subVI는 비동기적으로 실행됩니다.

어커런스 기다림, 타이밍, GPIB, VISA 함수와 시리얼 VI는 스레드를 점유하지 않고 작업이 완료되기를 기다릴 수 있습니다. 실행 시스템은 작업이 완료될 때까지 해당 작업을 큐에서 빼 놓습니다. 작업이 완료되면 실행 시스템은 해당 작업을 큐의 맨 끝에 놓습니다. 예를 들어, 사용자가 [세 개 버튼 대화 상자] VI가 디스플레이하는 대화 상자의 버튼을 클릭할 때 실행 시스템은 작업을 큐의 맨 끝에 놓습니다.

교착

교착은 둘 이상의 작업이 같은 시스템 리소스를 놓고 경쟁하여 어느 작업도 완료되지 못할 때 발생합니다. 교착의 예는 두 어플리케이션이 동시에 파일을 인쇄해야 하는 경우입니다. 스레드 1에서 실행되는 한 어플리케이션은 파일을 수집하여 다른 어플리케이션이 접근 못하도록 잠급니다. 스레드 2에서 실행되는 다른 어플리케이션도 프린터에 같은 작업을 합니다. OS가 리소스에 개입하거나 리소스를 해제하지 않는 비 선점형 환경에서는, 두 어플리케이션 모두 상대 어플리케이션이 리소스를 해제하기를 기다리지만, 이미 가지고 있는 리소스는 해제하지 않습니다. 교착을 피하기 위한 한 방법은 VI를 설정하여 같은 순서대로 공통 리소스에 접근하도록 하는 것입니다.

실행 시스템에서 사용자 인터페이스 관리하기

VI를 실행하는 것 외에, 실행 시스템은 사용자 인터페이스와의 상호작용을 조정해야 합니다. 버튼을 클릭하거나, 윈도우를 이동하거나, 슬라이드 컨트롤의 값을 변경할 때, 실행 시스템은 해당 작업을 관리하고 VI가 배경에서 계속 실행되도록 해야 합니다.

싱글스레드 실행 시스템은 사용자 상호작용과 VI 실행에 번갈아가며 대응함으로써 멀티태스킹합니다. 실행 시스템은 처리가 필요한 사용자 인터페이스 이벤트가 있는지 확인합니다. 처리가 필요한 이벤트가 없는 경우, 실행 시스템은 VI로 돌아가서 큐의 다음 작업을 받습니다.

버튼 또는 풀다운 메뉴를 클릭할 때, LabVIEW가 배경에서 VI를 실행하고 있기 때문에 실행한 작업은 완료까지 약간의 시간이 걸릴 수 있습니다. LabVIEW는 컨트롤이나 메뉴와의 상호작용 및 VI 실행에 번갈아가며 대응합니다.

LabVIEW 실행 시스템

LabVIEW에는 6개의 실행 시스템이 있는데, 파일»VI 프로퍼티를 선택한 후 VI 프로퍼티 대화 상자에서 실행을 선택하여 실행 시스템을 지정할 수 있습니다. 다음 실행 시스템 중 에서 선택할 수 있습니다.

- 사용자 인터페이스—사용자 인터페이스를 처리합니다. 멀티스레드 어플리케이션에서도 싱글스레드 어플리케이션의 경우와 똑같이 작동합니다. VI는 사용자 인터페이스 스레드에서 실행되나, 실행 시스템은 협력형 멀티태스킹과 사용자 인터페이스 이벤트 응답 사이를 번갈아가며 작동합니다.
- 표준—사용자 인터페이스와 별도의 스레드에서 실행됩니다.
- 인스트루먼트 I/O—VISA, GPIB, 시리얼 I/O가 다른 VI를 방해하지 못하게 합니다.
- 데이터 수집—데이터 수집이 다른 VI를 방해하지 못하게 합니다.
- 기타 1과 기타 2—어플리케이션의 작업이 자신의 스레드를 요구할 때 사용 가능합니다.
- 호출자와 같음—subVI의 경우, subVI를 호출한 VI와 같은 실행 시스템에서 실행됩니다. 최상위 VI에서, 호출자와 동일한 실행 시스템을 표준으로 설정하는 것과 같은 효과를 갖습니다.

이 실행 시스템은 VI가 다른 VI와 독립적으로 실행되어야 하는 대략적인 기준을 제공합니다. 기본적으로 VI는 표준 실행 시스템에서 실행됩니다.

인스트루먼트 I/O와 데이터 수집이라는 이름은 이 실행 시스템 내에서 이루어지는 작업 타입에 대한 제안입니다. I/O와 데이터 수집은 다른 시스템에서는 작동되나, 이 라벨을 사용하여 어플리케이션을 나누고 구조를 이해할 수 있습니다.

실행 시스템은 사용자 인터페이스 관리를 담당하지 않습니다. 이 실행 시스템 중 하나에 있는 VI가 컨트롤을 업데이트하는 경우, 실행 시스템은 책임을 사용자 인터페이스 실행 시스템에 넘겨줍니다. 사용자 인터페이스 실행 시스템을 다수의 프로퍼티 노드를 포함하는 VI에 지정하십시오.

LabVIEW는 머신에 있는 프로세서의 수에 비례하여 할당하는 스레드 개수를 정합니다. 각 스레드는 작업을 처리합니다. 예를 들어, VI가 공유 라이브러리를 호출하는 경우, 별도의 스레드가 해당 실행 시스템 내에서 다른 VI를 계속 실행합니다. 각 실행 시스템이 갖고 있는 스레드의 개수가 한정되어 있기 때문에, 스레드가 사용 중인 경우 태스크는 지연됩니다.

생성한 VI가 표준 실행 시스템에서는 정상적으로 실행되더라도 다른 실행 시스템 사용을 고려하십시오. 예를 들어 인스트루먼트 드라이버를 개발하는 경우, 인스트루먼트 I/O 실행 시스템 사용을 고려하십시오.

표준 실행 시스템을 사용하더라도 사용자 인터페이스는 자신의 스레드에 분리되어 있습니다. 프런트패널에 그리기, 마우스 클릭에 응답하기 등 사용자 인터페이스에서 실행되는 모든 작업은 블록 다이어그램 코드의 실행 시간을 방해하지 않고 실행됩니다. 마찬가지로, 긴 계산 루틴을 실행해도 사용자 인터페이스가 마우스 클릭이나 키보드 데이터 입력에 응답하는 것을 막지는 않습니다.

프로세서가 여러 개 있는 컴퓨터는 멀티스레딩으로 더욱 큰 혜택을 얻습니다. 단일 프로세서 컴퓨터에서, OS는 스레드를 선취하여 프로세서의 각 스레드에 시간을 분배합니다. 멀티프로세서 컴퓨터에서, 스레드는 동시에 여러 프로세서에서 실행될 수 있으므로 둘 이상의 작업이 동시에 발생할 수 있습니다.

멀티스레드 어플리케이션의 장점

멀티스레딩을 이용하는 어플리케이션은 다음을 포함한 많은 장점을 가지고 있습니다.

- 더 효율적인 CPU 사용
- 더 나은 시스템 신뢰도
- 멀티프로세서 컴퓨터에서 향상된 성능

더 효율적인 CPU 사용

많은 LabVIEW 어플리케이션에서, 사용자는 인스트루먼트 또는 데이터 수집 디바이스로 동기화된 수집 호출을 합니다. 이런 호출은 종종 완료까지 시간이 많이 걸릴 수 있습니다. 싱글스레드 어플리케이션에서, 이런 동기화 호출은 수집이 완료될 때까지 LabVIEW 어플리케이션 내의 다른 태스크가 실행되는 것을 실질적으로 막거나 방해합니다. LabVIEW에서, 멀티스레딩은 이런 방해 작업을 예방합니다. 동기화 수집 호출이 한 스레드에서 실행되는 동안, 데이터 분석과 파일 I/O와 같이 수집과 관련이 없는 프로그램의 다른 부분은 다른 스레드에서 실행됩니다. 그러므로 수집이 완료될 때까지 기다리지 않고 수집 중에도 어플리케이션의 실행은 진행됩니다.

이런 방식으로, 멀티스레드 어플리케이션은 프로세서의 효율성을 최대화합니다. 어플리케이션의 스레드 중 하나라도 실행할 준비가 된 경우 프로세서가 작동하기 때문입니다. 파일을 읽고 쓰거나, I/O를 수행하거나, 작업을 위해 사용자 인터페이스를 폴링하는 모든 프로그램은 멀티스레딩을 사용할 경우 여러 이점이 있습니다. 이러한 동기화 작업을 수행하는 동안 CPU를 더 효율적으로 사용할 수 있기 때문입니다.

더 나은 시스템 신뢰도

프로그램을 다른 실행 스레드에서 분리함으로써 프로그램의 한 작업이 다른 중요한 작업에 나쁜 영향을 미치지 않도록 할 수 있습니다. 가장 일반적인 예는 사용자 인터페이스가 더욱 시간에 결정적인 작업에 영향을 미치는 경우입니다. 화면 업데이트 또는 사용자 이벤트에 대한 응답은 종종 프로그램의 실행 속도를 줄이기도 합니다. 예를 들어, 윈도우를 이동하거나, 크기를 조정하거나, 다른 윈도우를 열면, 프로세서가 사용자 인터페이스 이벤트에 응답하는 동안 프로그램 실행이 사실상 중단됩니다.

LabVIEW 멀티스레드 어플리케이션에서는 사용자 인터페이스 작업이 지정된 사용자 인터페이스 스레드에 할당되며 프로그램의 데이터 수집, 분석, 파일 I/O 부분은 다른 스레드에서 실행될 수 있습니다. 사용자 인터페이스 스레드에 다른 더욱 시간에 결정적인 작업보다 낮은 우선순위를 주어 사용자 인터페이스 작업이 CPU가 컴퓨터 기반의 인스트루먼트에서 데이터를 수집하는 등의 더욱 중요한 작업을 실행하는 것을 방해하지 못하게 합니다. 사용자 인터페이스에 낮은 우선순위를 두면 데이터 수집과 프로세싱, LabVIEW의 성능, 전반적인 컴퓨터 성능을 포함한 전체 시스템 신뢰도가 향상됩니다. 예를 들어, 연산자가 윈도우를 이동하여도 데이터를 잃지 않습니다.

멀티스레딩이 더 나은 시스템 신뢰도를 제공하는 다른 예는 초고속 데이터 수집을 수행하고 결과를 디스플레이할 때입니다. 화면 업데이트는 연속적인 초고속 데이터 수집과 같은 다른 작업에 비해 느린 경우가 많습니다. 단일스레드 어플리케이션에서 많은 양의 데이터를 초고속으로 수집하고 수집된 모든 데이터를 그래프에 디스플레이하려는 경우, 프로세서가 화면 업데이트에 너무 많은 시간을 소비하게 되므로 데이터 버퍼가 오버플로우될 수 있습니다. 데이터 버퍼가 오버플로우되면 데이터를 잃게 됩니다.

그러나 사용자 인터페이스가 해당 스레드에 분산된 LabVIEW 멀티스레드 어플리케이션에서는 데이터 수집 작업이 다른 상위 우선순위 스레드에 상주할 수 있습니다. 이 경우, 데이터 수집과 디스플레이는 독립적으로 실행되므로 데이터 수집은 연속적으로 실행되고 데이터를 중단 없이 메모리로 보낼 수 있습니다. 디스플레이는 최대한 빨리 실행되며, 실행 시 메모리에서 찾는 모든 데이터를 그립니다. 수집 스레드가 디스플레이 스레드보다 앞서 실행되므로 화면이 업데이트될 때 데이터를 잃지 않습니다.

멀티프로세서 컴퓨터에서 향상된 성능

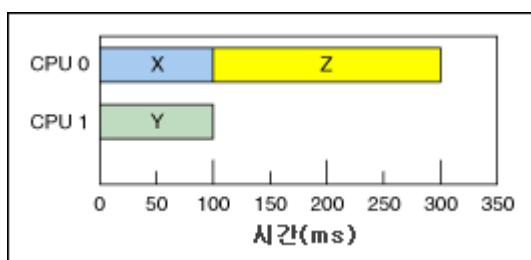
멀티스레딩의 가장 큰 장점 중 하나는 멀티프로세서 컴퓨터의 성능을 이용할 수 있다는 것입니다. 현재 상당수의 고성능 컴퓨터는 추가적인 계산 능력을 위해 둘 이상의 프로세서를 제공합니다. 멀티스레드 어플리케이션은 이러한 컴퓨터의 장점을 최대한 이용할 수 있습니다. 여러 스레드가 동시에 실행되는 멀티스레드 어플리케이션에서, 각 프로세서는 다른 스레드를 실행할 수 있습니다. 멀티프로세서 컴퓨터에서는 어플리케이션이 실제로 병렬 작업 실행을 할 수 있으므로 전반적인 시스템 성능이 향상됩니다.

반대로 단일스레드 어플리케이션은 하나의 프로세서에서만 실행되므로 여러 프로세서를 활용하여 성능을 향상시킬 수 없습니다. 그러므로 멀티스레드 OS 및/또는 멀티프로세서 컴퓨터에서 최대한의 성능을 얻으려면 멀티스레드 어플리케이션을 사용해야 합니다.

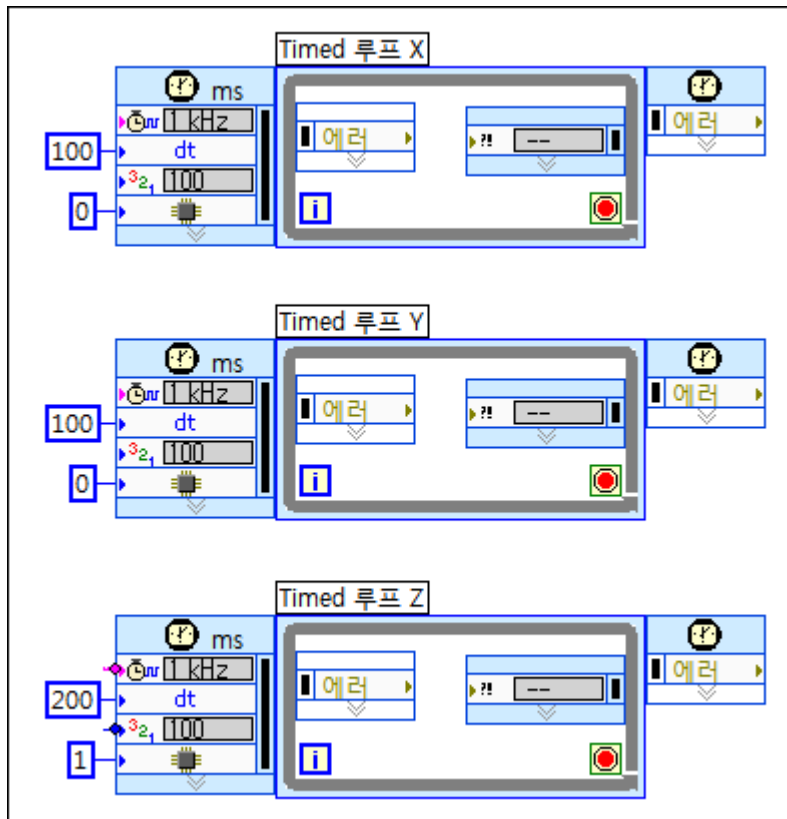
수동으로 CPU 할당하기

LabVIEW로는 **병렬** 또는 **파이프라인**을 사용하여 쉽게 여러 CPU(멀티코어, 멀티프로세서 또는 SMP 시스템으로도 불림)를 활용할 수 있습니다. 일부 경우, 특정한 프로세서에 특정한 스레드를 **수동으로 할당**하여 CPU 효율성을 더욱 높일 수도 있습니다.

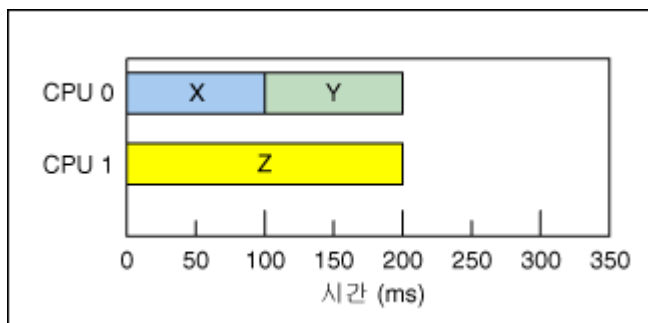
(Real-Time, Windows) Timed 루프를 사용하여 수동으로 CPU 할당을 컨트롤할 수 있습니다. 예를 들어 2 개의 CPU가 있는 시스템에서 병렬 Timed 루프 X, Y, Z를 가진 어플리케이션을 가정해봅시다. Timed 루프는 실행하는데 각각 100 ms, 100 ms, 200 ms가 소요됩니다. Timed 루프가 모두 기본 우선순위로 설정된 경우, CPU 스케줄러는 Timed 루프 X와 Z가 하나의 CPU에서, Timed 루프 Y가 다른 하나의 CPU에서 작동하도록 할당할 수 있습니다. 그 결과 다음 그림과 같이 전체 실행 시간은 300 ms가 됩니다.



다음 블록다이어그램에서 보이는 것처럼 수동으로 Timed 루프 X와 Y를 하나의 CPU에 할당하고, Timed 루프 Z를 다른 하나의 CPU에 할당하여 VI를 최적화할 수도 있습니다.



이 경우, 수동 CPU 할당은 다음 설명에서 보이는 것처럼 전체 실행 시간을 200 ms로 단축 시킵니다.



노트

Real-Time Module은 기본 프로세서(CPU 0)에서 시스템 스레드를 실행합니다. 정확성을 가지는 Timed 루프를 기본 CPU가 아닌 전용 CPU에 할당하여 지터를 줄일 수도 있습니다. 또한 다른 CPU 세트를 수동 CPU 할당을 위해 예약하는 동안 하나의 CPU 세트에 자동 로드 균형을 사용할 수 있습니다. Real-Time Module CPU 스케줄러는 Timed 구조 스레드를 시스템 스레드와 개별적으로 처리하며, 기본적으로 두 스레드 모두에 모든 CPU의 자동 로드 균형을 수행합니다. 각 스레드 타입의 자동 로드 균형에 사용할 수 있는 CPU 세트를 지정하는 것에 대한 더 자세한 정보는 National Instruments 웹사이트를 참조하십시오.

여러 CPU를 가진 시스템의 파이프라인

여러 CPU를 가진 시스템(멀티코어, 멀티프로세서 또는 SMP 시스템으로도 불림)의 주요 장점은 멀티스레드가 병렬로 실행될 수 있다는 점입니다. 그러므로 어플리케이션이 주로 단일 연속 프로세스로 구성되어 있는 경우, 여러 프로세서 시스템을 활용하기 어렵습니다. 그러나 파이프라인 구조를 구현하면 여러 CPU를 활용하여 단일 연속 프로세스의 작업량을 향상시킬 수 있습니다. 파이프라인은 연속적인 데이터 흐름을 보존하면서 여러 CPU에서 병렬 실행을 활용합니다.

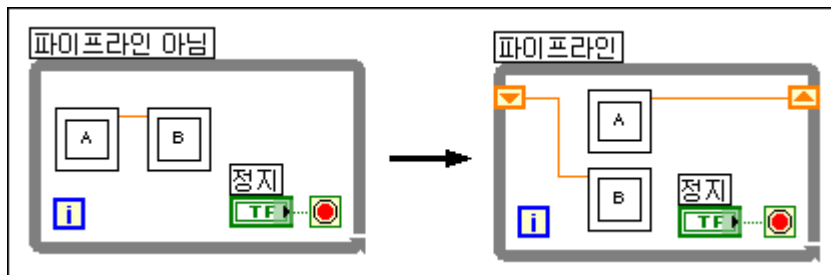


노트

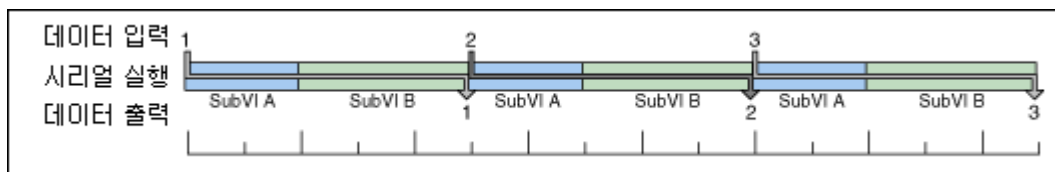
어플리케이션에서 파이프라인을 이용하려면 CPU 사이에서 데이터 전송이 이루어져야 합니다. 이는 단순히 하나의 CPU에서 다음 작업에 데이터를 전달하는 것보다 시간이 오래 걸립니다. 그러므로 파이프라인을 구현할 경우, 병렬 프로세스로 절감되는 시간이 CPU 사이의 데이터 전송에 소요되는 시간보다 클 경우에만 VI 실행 스피드가 증가하게 됩니다.

파이프라인 구현

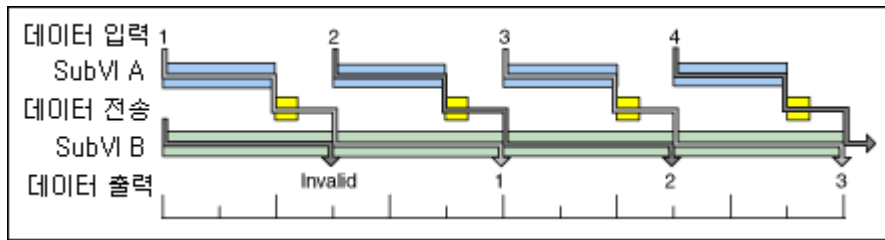
LabVIEW VI에서 파이프라인을 구현하려면 코드를 이산 단계 또는 subVI로 나누십시오. 예를 들어 한 프로세스의 처음 반을 하나의 subVI에 담고 나머지 반을 다른 subVI에 담아 복잡한 연산 프로세스를 두 개의 이산 단계로 나눌 수 있습니다. 다음 블록 다이어그램에서 왼쪽은 표준 연속 구조, 오른쪽은 파이프라인 구조를 보여줍니다.



왼쪽 구조는 여러 CPU를 활용하지 않습니다. 다음 설명에서 보이는 것처럼 LabVIEW가 두 subVI 모두를 단일 CPU에서 순차적으로 실행되어야 하는 단일 실행 스레드에서 실행하기 때문입니다.



그러나 두 subVI를 시프트 레지스터를 통해 함께 연결하면 LabVIEW는 subVI를 파이프라인합니다. 이제 VI가 하나 이상의 CPU를 가진 시스템에서 실행될 때, SubVI는 다음 설명에서 보이는 것처럼 병렬로 실행됩니다.



노트 파이프라인을 구현할 때 파이프라인의 단계가 공유 리소스를 사용하지 않도록 하십시오. 공유 리소스를 동시에 요청하게 되면, 병렬 실행이 방해받으며 여러 CPU 시스템을 사용하는 성능상의 장점이 사라지게 됩니다.

파이프라인의 단계를 실행하는 CPU가 다른 태스크를 실행할 필요가 없으며 CPU 활용을 최소화하려는 경우, 파이프라인 단계의 균형을 맞춰 각 단계가 거의 같은 시간만큼 실행되도록 합니다. 하나의 파이프라인 단계가 다른 단계보다 더 오래걸릴 경우, 시간이 적게 걸리는 단계를 수행하고 있는 CPU는 시간이 많이 걸리는 단계가 끝날 때까지 기다려야 합니다.

파이프라인 데이터 흐름

파이프라인은 연속적인 데이터 흐름 의존성을 보존하면서 병렬 프로세싱을 활용합니다. 이전 예제의 경우, 루프의 첫번째 반복에서 subVI A가 입력 1을 처리하는 동안 subVI B는 시프트 레지스터의 기본값을 처리하므로 유효하지 않은 출력이 나옵니다. 두번째 루프 반복에서 subVI B가 첫번째 루프 반복에서 나온 subVI A의 출력을 처리하는 동안 subVI A는 입력 2를 처리합니다. SubVI B의 출력은 파이프라인이 다 채워질때까지 유효하지 않습니다. 파이프라인이 채워지면, 이후의 모든 루프 반복은 1 루프 반복만큼 계속 뒤쳐져서 유효한 출력을 냅니다.



노트 파이프라인 실행의 초기에 발생하는 무효한 출력으로 인한 원하지 않는 동작을 방지하기 위해 주의해야 합니다. 예를 들어 케이스 구조를 사용하여 Timed 루프가 N번 반복된 후에만 작동기를 활성화할 수 있습니다.

일반적으로 마지막 파이프라인 단계의 출력은 파이프라인의 단계 개수만큼 입력에 뒤처지며 출력은 파이프라인이 꽉 찰 때까지 각 루프 반복에서 사용할 수 없습니다. 파이프라인의 단계 개수는 파이프라인 깊이라고 부르며 루프 반복으로 측정되는 파이프라인의 지연은 이 깊이에 해당합니다. 깊이가 N인 파이프라인의 경우, N번째 루프 반복까지 결과는 유효하지 않으며 각 유효한 루프 반복은 N-1 루프 반복만큼 입력에 뒤처집니다.



노트 병렬로 실행될 수 있는 파이프라인 단계의 개수는 사용 가능한 CPU의 갯수로 제한됩니다.

Timed 루프로 파이프라인 사용

(Real-Time, Windows) 각 파이프라인 단계를 특정 CPU에 지정하려 하는 경우, Timed 루프를 사용하여 파이프라인을 구현할 수 있습니다. LabVIEW는 각 Timed 루프를 단일 스레드에 맵핑하기때문에 Timed 루프 안에 시프트 레지스터를 사용하여 파이프라인을 구현할 수는 없습니다. Timed 루프로 파이프라인을 구현하려면, 각 파이프라인 단계를 병렬

Timed 루프에 넣고 큐, 로컬 변수, 또는 글로벌 변수를 사용하여 Timed 루프 사이에 데이터를 전달해야 합니다.



노트 로컬이나 글로벌 변수를 사용하여 파이프라인을 구현하는 것은 어렵습니다. 로컬 및 글로벌 변수는 새로운 데이터를 기다리지 않으므로 입력을 건너뛰거나 하나의 입력에서 반복적으로 출력 결과를 낼 수도 있습니다.



노트 RT 어플리케이션에서 파이프라인을 사용하는데 대한 정보는 LabVIEW Real-Time Module 도움말을 참조하십시오.

병렬 작업의 우선순위 정하기

전략적으로 [기다림(ms)] 함수를 사용하거나 VI 프로퍼티 대화 상자의 실행항목에서 우선순위 설정을 변경하여 병렬 작업의 우선순위를 정합니다.

대부분의 경우, VI의 기본 우선순위를 변경해서는 안 됩니다. 우선순위를 사용하여 실행 순서를 컨트롤해도 반드시 원하는 결과가 나오지는 않습니다. 올바르게 사용되지 않는 경우, 낮은 우선순위의 작업은 완전히 배제될 수 있습니다. 높은 우선순위의 작업이 오랫동안 실행되도록 설계된 경우, 낮은 우선순위의 작업은 높은 우선순위의 작업이 주기적으로 대기하지 않는 한 실행되지 않습니다.

기다림 함수 사용하기

기다림 함수를 사용하여 우선순위가 낮은 작업이 낮은 빈도로 실행되도록 할 수 있습니다. 예를 들어, 몇 개의 루프가 병렬로 존재하며 그 중 일부를 더 자주 실행하고자 하는 경우, 낮은 우선순위의 작업에 기다림 함수를 사용하십시오. 이렇게 하면 더 많은 시간이 다른 작업에 할애됩니다. 많은 경우, 기다림 함수를 사용하면 충분합니다. 파일 >> VI 프로퍼티를 선택하고 VI 프로퍼티 대화 상자에서 실행을 선택하여 우선순위를 변경하지 않아도 될 것입니다.

동기화/차단 노드 섹션에서 설명했듯이, 블록다이어그램이 기다릴 때 컴퓨터는 블록다이어그램을 큐에서 제거하여 다른 작업이 실행될 수 있도록 합니다.

기다림 함수는 사용자 인터페이스를 폴링하는 루프에 가장 유용합니다. 100에서 200ms 기다림은 거의 눈에 띄지 않으나, 어플리케이션을 작업에서 해제하여 다른 작업을 더 효율적으로 처리하게 해줍니다. 또한, 기다림은 OS를 해제하여 다른 스레드 또는 어플리케이션에 더 많은 시간을 할당하도록 합니다. 블록다이어그램의 우선순위가 낮은 섹션에 기다림을 추가하여 다른 작업에 더 많은 시간을 할당하는 것을 고려하십시오.

기다림 (ms) 함수를 사용하면, LabVIEW는 사용자가 지정한 시간만큼 기다립니다. 다음 ms 배수까지 기다림 함수를 사용하면, LabVIEW는 밀리초 카운터를 모니터하고, 카운터가 사용자가 지정한 수의 배수가 될 때까지 기다립니다.

우선순위 변경

또한 파일 » VI 프로퍼티를 선택한 후 VI 프로퍼티 대화 상자에서 실행을 선택하여 우선순위를 변경할 수 있습니다. 최하위에서 최상위 순서로 나열된 다음 우선순위 레벨 중에서 선택할 수 있습니다.

- 배경 우선순위(최하위)
- 표준 우선순위
- 표준 우선순위 이상
- 높은 우선순위
- 시간에 결정적인 우선순위(최상위)
- 서브루틴 우선순위

처음 다섯 우선순위는 작동이 유사하나(최하위에서 최상위), 서브루틴 우선순위는 추가적인 특징이 있습니다. 다음 두 섹션은 서브루틴 레벨을 제외한 모든 우선순위에 적용됩니다.

사용자 인터페이스 실행 시스템에서의 우선순위

멀티스레드 어플리케이션의 사용자 인터페이스 실행 시스템에서 실행 시스템 큐는 여러 개의 입력 포인트를 갖습니다. 실행 시스템은 큐에서 상위 우선순위 VI를 하위 우선순위 VI 앞에 둡니다. 상위 우선순위 작업이 실행 중이고 큐에 하위 우선순위 작업만이 있는 경우, 상위 우선순위 VI가 계속해서 실행됩니다. 예를 들어, 실행 큐에 각 우선순위 레벨의 두 VI가 있는 경우, 시간에 결정적인 VI는 두 VI가 완료될 때까지 독점적으로 실행 시간을 공유합니다. 그 후, 높은 우선순위 VI가 두 VI가 완료될 때까지 독점적으로 실행 시간을 공유합니다. 그러나, 높은 우선순위 VI가 **기다리는 함수**를 호출하는 경우, 실행 시스템은 기다림 또는 I/O가 완료될 때까지 큐에서 높은 우선순위 VI를 제거하고, 다른 작업(낮은 우선순위를 가진)의 실행을 지정합니다. 기다림 또는 I/O가 완료될 때, 실행 시스템은 대기 중인 작업을 큐의 하위 우선순위 작업 앞에 다시 끼워 넣습니다.

또한, 우선순위가 높은 VI가 우선순위가 낮은 SubVI를 호출하는 경우, 호출 VI가 실행 중이 아닌 경우에도 해당 SubVI는 호출하는 VI의 가장 높은 우선순위를 상속합니다. 결과적으로, subVI의 우선순위 레벨을 올리기 위해 VI가 호출하는 subVI의 우선순위 레벨을 변경할 필요가 없습니다.

다른 실행 시스템과 멀티스레드 어플리케이션에서 우선순위

각 실행 시스템은 서브루틴 우선순위 레벨과 사용자 인터페이스 실행 시스템은 제외하고, 각 우선순위 레벨에 대해 별도의 실행 시스템을 갖습니다. 각 우선순위 시스템에는 고유의 큐와 해당 큐의 블록 다이어그램을 처리하기 위한 두 개의 스레드가 있습니다.

6개의 실행 시스템을 가지는 대신, 우선순위와 관계없이 사용자 인터페이스 시스템을 위해 한 개의 실행 시스템이 있고 기타 시스템을 위해 25개의 실행 시스템이 있습니다. 이것은 5개의 실행 시스템에 5개의 우선순위 레벨 중 각각에 대해 5개를 곱한 것입니다.

OS는 이 분류에 기초하여 각 실행 시스템의 스레드에 OS 우선순위 레벨을 지정합니다. 그러므로, 일반적인 실행에서 상위 우선순위 작업은 하위 우선순위 작업보다 더 많은 시간을 얻습니다. 사용자 인터페이스 실행 시스템에서의 우선순위와 같이, 하위 우선순위 작업은 상위 우선순위 작업이 주기적으로 기다리지 않는 한 실행되지 않습니다.

몇몇 OS는 주기적으로 하위 우선순위 작업의 우선순위 레벨을 올려 이 문제를 피하려 합니다. 이런 OS에서는 상위 우선순위 작업이 계속 실행되려하는 경우에도 하위 우선순위 작업이 주기적으로 실행될 기회를 얻습니다. 그러나 이런 작동은 OS마다 다릅니다. 일부 OS에서는 이런 동작과 작업의 우선순위를 조정할 수 있습니다.

사용자 인터페이스 실행 시스템은 단일 스레드형입니다. 사용자 인터페이스 스레드는 다른 실행 시스템의 표준 우선순위를 사용합니다. 그러므로 표준 우선순위 이상의 표준 실행 시스템에서 VI가 실행되도록 설정하는 경우, 사용자 인터페이스 실행 시스템은 실행되지 않을 수 있으며, 이로 인해 사용자 인터페이스가 느려지거나 응답을 하지 않을 수 있습니다. 마찬가지로, VI를 배경 우선순위에서 실행되도록 설정하는 경우 사용자 인터페이스 실행 시스템보다 낮은 우선순위로 실행됩니다.

이전에 설명한 바와 같이, VI가 낮은 우선순위의 subVI를 호출하는 경우, 호출 기간 동안 subVI는 호출자와 같은 우선순위 레벨로 올라갑니다.

서브루틴 우선순위 레벨

서브루틴 우선순위 레벨은 VI가 최대한 효율적으로 실행되도록 합니다. 서브루틴 우선순위를 설정한 VI는 다른 VI와 실행 시간을 공유하지 않습니다.

VI가 서브루틴 우선순위 레벨에서 실행될 때, VI는 실행되는 스레드를 효율적으로 컨트롤하며 호출자와 같은 스레드에서 실행됩니다. 다른 VI가 서브루틴 우선순위 레벨에 있는 경우에도 서브루틴 VI가 실행을 마칠 때까지 다른 VI는 해당 스레드에서 실행될 수 없습니다. 단일 스레드 어플리케이션에서, 다른 VI는 실행되지 않습니다. 실행 시스템에서, 서브루틴을 실행하는 스레드는 다른 VI를 처리하지 않으나 실행 시스템의 두 번째 스레드는 다른 실행 시스템과 함께 계속 VI를 실행할 수 있습니다.

다른 VI와 시간을 공유하지 않는 것에 덧붙여서, 서브루틴 VI 실행은 간소화되어 프론트 패널 컨트롤과 인디케이터는 서브루틴이 호출될 때 업데이트되지 않습니다. 서브루틴 VI 프론트패널은 실행에 대해 아무 정보도 보여주지 않습니다.

서브루틴 VI는 다른 서브루틴 VI를 호출할 수 있으나, 다른 우선순위의 VI를 호출할 수는 없습니다. 서브루틴 우선순위 레벨은 단순한 계산을 수행하는 subVI의 오버헤드를 최소화하려는 경우 사용합니다.

또한, 서브루틴은 실행 큐와 상호작용하도록 설계되지 않았기 때문에 LabVIEW가 큐에서 서브루틴을 빼도록 하는 함수는 호출할 수 없습니다. 즉, 기다림, GPIB, VISA, 또는 대화 상자 함수는 모두 호출할 수 없습니다.

서브루틴은 시간에 결정적인 어플리케이션에서 유용한 추가적인 특징을 가지고 있습니다. SubVI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 사용중이면 서브루틴 호출 건너뛰기를 선택하는 경우, 서브루틴이 현재 다른 스레드에서 실행되고 있는 경우 실행 시스템은 호출을 건너뛵니다. 이것은 실행 시스템이 안전하게 서브루틴이 실행하는 작업을 건너뛰는 시간에 결정적인 루프에 유용합니다. 또한, subVI가 작업을 끝내기를 기다리는 시간을 피하려는 경우에도 유용합니다. SubVI의 실행을 건너뛰는 경우, subVI의 모든 출력은 subVI 프론트패널 인디케이터의 기본값이 됩니다.

실행 시스템과 우선순위 사용에 대한 제안

대부분의 어플리케이션에서, 우선순위 레벨 또는 자동적으로 VI의 멀티태스킹을 처리하는 표준 실행 시스템 외의 실행 시스템을 사용할 필요가 없습니다. 기본적으로 모든 VI는

표준 우선순위일 경우 표준 실행 시스템에서 실행됩니다. 멀티스레드 어플리케이션에서, 별도의 스레드가 사용자 인터페이스 작업을 처리하여 VI가 사용자 인터페이스 상호작용으로부터 차단되도록 합니다. 단일 스레드 어플리케이션의 경우에도, 실행 시스템은 사용자 인터페이스 상호작용과 VI의 실행을 번갈아 처리하므로 비슷한 결과가 나옵니다.

일반적으로, 실행의 우선순위를 정하는 가장 좋은 방법은 어플리케이션에서 기다림 함수를 사용하여 낮은 우선순위 루프의 속도를 줄이는 것입니다. 100에서 200ms의 지연은 거의 사용자의 눈에 띄지 않으므로 이는 특히 사용자 인터페이스 VI를 위한 루프에 유용합니다.

우선순위를 사용하는 경우 신중하게 사용하십시오. 상당 기간동안 동작하는 상위 우선순위 VI를 설계하는 경우, 이 코드의 비교적 시간에 결정적이지 않은 섹션의 VI에 기다림을 추가하여 하위 우선순위 작업과 시간을 공유할 수 있도록 고려하십시오.

다른 작업이 변경할 수 있는 글로벌 변수, 로컬 변수, 또는 다른 외부 리소스를 사용할 때는 주의하십시오. **기능적인 글로벌 변수** 또는 **세마포어**와 같은 동기화 기술을 사용하여 이 리소스에 대한 접근을 보호할 수 있습니다.

글로벌 및 로컬 변수와 외부 리소스 접근 동기화하기

실행 시스템이 여러 작업을 병렬로 실행할 수 있으므로, 글로벌과 로컬 변수 및 리소스가 올바른 순서로 접근되도록 해야합니다.

경합 조건 방지하기

다음 여러가지 방법 중 하나로 경합 조건을 피할 수 있습니다. 가장 간단한 방법은 글로벌 변수가 전체 어플리케이션에서 변경되는데 이 글로벌 변수를 단지 한 곳에서만 가지는 것입니다.

단일 스레드 어플리케이션에서, 서브루틴 우선순위 VI를 사용하여 경합 조건을 발생시키지 않고 글로벌 변수를 읽거나 쓸 수 있습니다. 서브루틴 우선순위 VI는 다른 VI와 실행 스레드를 공유하지 않기 때문입니다. 멀티스레드 어플리케이션에서, 서브루틴 우선순위 레벨은 글로벌 변수로의 독점적인 접근을 보장하지 않습니다. 다른 스레드에서 실행되는 다른 VI가 동시에 글로벌 변수에 접근할 수 있기 때문입니다.

기능적인 글로벌 변수

기능적인 글로벌 변수는 초기화되지 않은 시프트 레지스터가 있는 루프를 사용하여 글로벌 데이터를 얻는 재호출이 아닌 VI입니다. 기능적인 글로벌 변수는 일반적으로 VI가 어떤 작업을 수행할지 지정하는 동작 입력 파라미터를 가지고 있습니다. 기능적인 글로벌 변수를 사용하여 변수에서 작업에 접근하는 코드의 중요 섹션을 보호할 수 있으며, 글로벌 변수와 관련된 경합 조건을 방지할 수 있습니다. 글로벌 변수를 사용하면 변수의 값을 읽고 변경한 후 다시 값을 쓸 때 **경합 조건**이 발생할 수 있습니다. 이는 동시에 하나의 변수 값을 읽은 병렬 코드의 서로 다른 두 부분이 다른 부분에도 도입된 변경 사항을 덮어 쓸 수 있기 때문입니다. 기능적인 글로벌 변수를 사용하면 데이터를 변경하는 작업을 보호할 수 있습니다. 예를 들어, 기능적인 글로벌 변수는 읽기, 증가 및 메모리에 데이터 쓰기 작업 또는 데이터베이스 업데이트 또는 파일 변경 작업과 같은 작업을 포함하는 중요 섹션에 대한 경합 조건을 방지할 수 있습니다.

수에 쓰는 코드의 각 섹션은 코드의 중요한 섹션입니다. 파일 및 디바이스 I/O는 중요한 섹션을 구성하는 코드의 대표적인 예입니다.

2. 중요한 섹션이 공유하도록 세마포어 생성—[세마포어 참조 얻기] VI를 사용하여 새 세마포어를 생성하거나 기존에 존재하는 세마포어를 이름으로 찾아 불러옵니다. 새로 생성한 세마포어는 한 개의 태스크에 의해서만 사용되도록 기본 설정되어 있습니다. 여러 태스크가 이 세마포어에 동시에 접근할 수 있도록 하려면, [세마포어 참조 얻기] VI의 크기 입력에 태스크의 수를 지정합니다.
3. 각 중요한 섹션 전에 세마포어에 접근—[세마포어 얻기] VI를 사용하여 세마포어에 대한 접근을 요청합니다. 다른 태스크가 세마포어를 이미 사용하고 있는 경우, 데이터 흐름은 이 태스크가 세마포어를 더 이상 사용하지 않을 때까지 [세마포어 얻기] VI에 멈추어 있습니다. 코드의 한 섹션이 세마포어를 수집하면, 다른 태스크는 이 코드가 세마포어를 해제할 때까지 세마포어를 수집할 수 없습니다. 그러므로, [세마포어 얻기] VI와 [세마포어 해제] VI 사이에서 보호되어 있는 코드는 항상 다른 보호된 코드 섹션이 실행을 시작하기 전에 실행을 완료하도록 보장됩니다.
4. 각 중요한 섹션 실행 후 세마포어에 대한 접근 해제—[세마포어 해제] VI를 사용하여 세마포어에 대한 접근을 해제하여 대기 중인 다른 태스크가 이 세마포어를 수집하고 코드의 중요한 섹션을 실행하도록 합니다.
5. 세마포어에 대한 참조 해제—[세마포어 참조 해제] VI를 사용하여 세마포어에 대한 참조를 해제합니다. 시스템 리소스를 효율적으로 사용하려면 이 세마포어를 사용하는 모든 태스크가 완료된 것을 확인한 후에 세마포어 참조를 해제합니다.

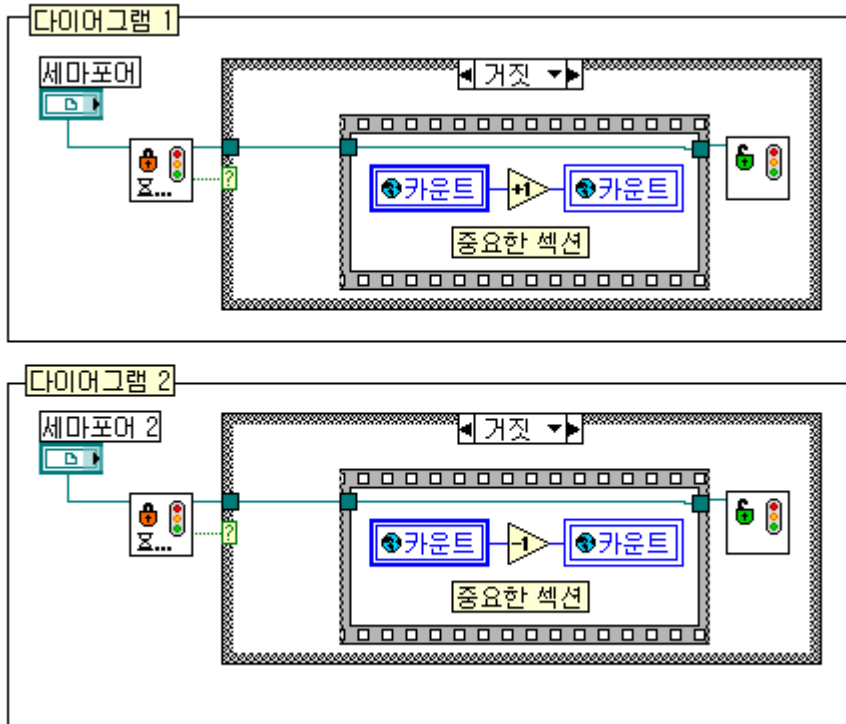


노트

더 숙련된 방법으로 세마포어를 사용하면 세마포어의 수집과 해제를 코드의 같은 섹션에서 하지 않아도 됩니다.

[세마포어 참조 얻기] VI와 [세마포어 참조 해제] VI를 사용하는 예제는 labview \examples\Synchronization\Semaphore 디렉토리에서 Simple Semaphore VI를 참조하십시오.

다음 그림은 세마포어를 사용하여 중요한 섹션을 보호하는 방법을 보여줍니다. 다른 VI가 세마포어를 생성하여 참조를 이 SubVI로 전달했습니다. 이 세마포어의 크기는 1이므로, 한 번에 한 개의 태스크만 세마포어에 접근할 수 있습니다.



이전의 블록다이어그램은 각각 같은 공유 변수인 카운트에 접근하는 중요한 섹션을 포함합니다. 그러므로, 이러한 블록다이어그램은 같은 세마포어를 공유합니다. 각 블록다이어그램은 중요한 섹션을 실행하기 전에, [세마포어 얻기] VI를 호출하여 다른 블록다이어그램이 이 세마포어를 이미 수집했는지 감지합니다. 세마포어를 사용 중인 경우, [세마포어 얻기] VI는 세마포어가 사용 가능할 때까지 기다립니다. 세마포어가 사용 가능하게 되면, [세마포어 수집] VI는 타임 아웃에 거짓을 반환하여 블록다이어그램이 중요한 섹션을 실행할 권한이 있음을 나타냅니다. 블록다이어그램이 중요한 섹션의 실행을 종료할 때 [세마포어 해제] VI는 세마포어를 해제하여, 대기 중인 다른 블록다이어그램이 실행을 재개할 수 있도록 허용합니다.

세마포어 지속기간

유휴 상태가 아닌 최상위 VI가 세마포어에 대한 참조를 소유하고 있는 한 세마포어는 메모리에 남아있습니다. 최상위 VI가 유휴 상태가 되면, LabVIEW는 최상위 VI의 SubVI 안에 존재하는 참조를 포함하여 해당 VI가 소유하는 모든 세마포어 참조를 해제합니다. LabVIEW가 이름있는 세마포어에 대한 마지막 참조를 해제하면, LabVIEW는 세마포어를 제거합니다. 이름없는 세마포어에 대한 참조는 오직 한 개만을 가질 수 있으므로, LabVIEW는 최상위 VI가 유휴 상태가 될 때 이름없는 세마포어를 제거합니다. 여러 최상위 VI에서 세마포어를 사용하려면, 세마포어에 이름을 붙이고 각 최상위 VI에서 [세마포어 참조 얻기] VI를 호출하여 각 VI가 자체적으로 세마포어에 대한 고유의 참조를 얻을 수 있도록 합니다.

LabVIEW에서의 멀티프로세싱과 하이퍼스레딩

하이퍼스레드를 지원하는 컴퓨터는 단일 프로세서를 가지지만 멀티프로세서를 가진 컴퓨터와 같이 동작합니다. (Windows) 하이퍼스레딩은 일부 Intel Pentium 4 및 이후 버전에서 지원하는 기능입니다. 하이퍼스레드를 지원하는 컴퓨터에서 Windows 작업 관리자를

실행하고 성능 탭을 클릭하면, Windows 작업 관리자는 두 개의 CPU의 사용 히스토리를 디스플레이합니다. (macOS) Activity Monitor를 시작하여 성능을 점검합니다.

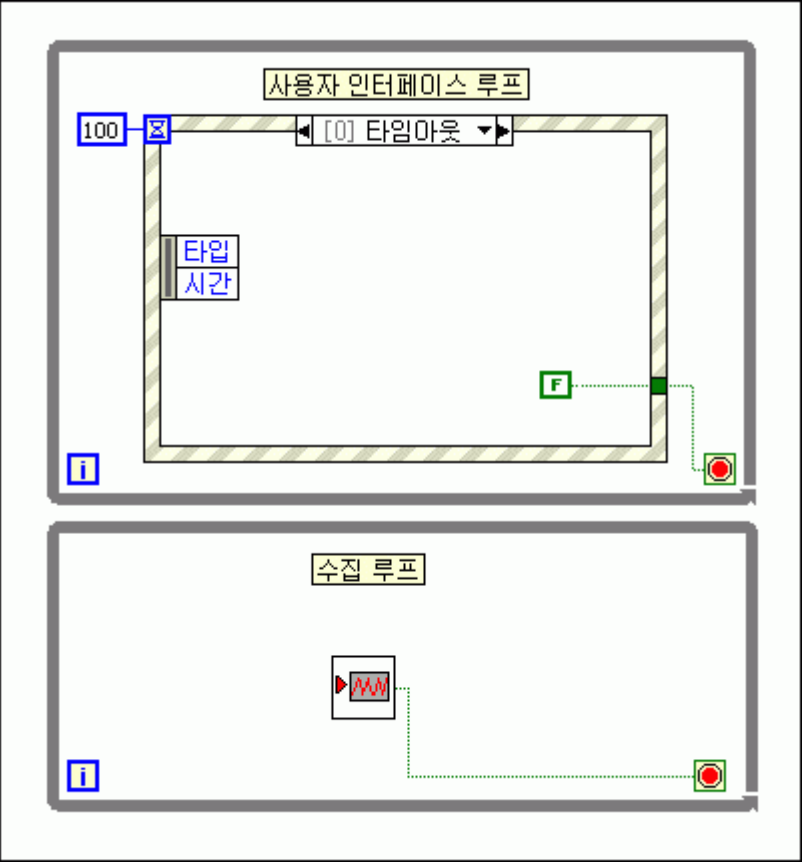
하이퍼스레드를 지원하는 프로세서는 같은 마이크로칩에 내장된 여러 프로세서처럼 동작합니다. 레지스터 세트와 같은 칩의 일부 리소스는 복사됩니다. 실행 유닛 및 캐시와 같은 다른 리소스는 공유됩니다. 작은 작업을 저장하는 버퍼와 같은 일부 리소스는 여러 부분으로 분할되며, 각 논리적 프로세서가 한 부분씩을 받습니다.

하이퍼스레딩을 이용한 어플리케이션의 최적화는 멀티프로세서 시스템(멀티코어, 여러 CPU 또는 SMP 시스템으로도 불림)의 어플리케이션 최적화와 유사하지만, 일부 차이점이 있습니다. 예를 들면 하이퍼스레딩 컴퓨터는 실행 단위를 공유하며, 두 개의 프로세서를 가진 컴퓨터는 두 개의 완전한 실행 단위를 가집니다. 그러므로, 부동소수로 실행 단위가 제한된 모든 어플리케이션의 경우 실행 단위를 공유할 필요가 없기 때문에 멀티프로세서를 가진 컴퓨터쪽이 성능이 좋습니다. 캐시 쟁탈에도 같은 원리가 적용됩니다. 두 개의 스레드가 캐시에 접근하려 하는 경우, 각 프로세서가 각각 완전한 크기의 캐시를 가지고 있는 멀티프로세서 컴퓨터가 성능이 더 좋습니다.

LabVIEW 실행 개요

LabVIEW 실행 시스템은 이미 멀티프로세싱을 지원하도록 만들어져 있습니다. 텍스트 기반의 프로그래밍 언어에서 멀티스레드를 사용한 어플리케이션을 만들기 위해서는 멀티스레드를 생성하고 스레드간의 통신을 위한 코드를 작성해야 합니다. 하지만 LabVIEW는 VI에서 멀티스레딩 기회를 인식할 수 있으며, 실행 시스템이 대신 멀티스레딩 통신을 처리해줍니다.

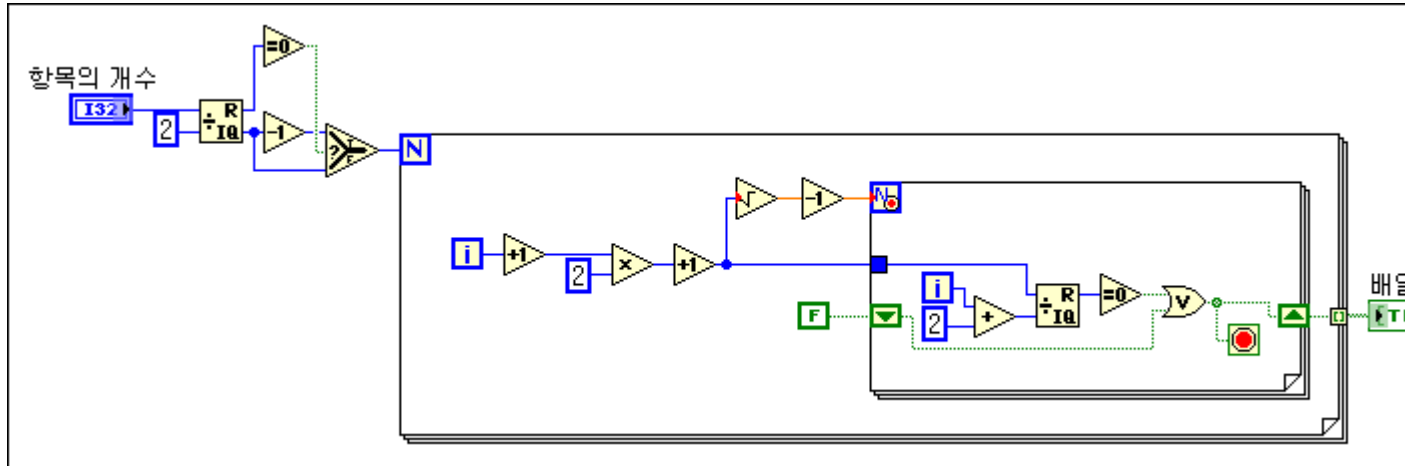
다음 예제는 LabVIEW 멀티스레드 실행 시스템을 이용합니다.



이 VI에서, LabVIEW는 멀티프로세싱 또는 하이퍼스레딩 환경에서 두 개의 루프를 동시에 독립적으로 실행할 수 있다는 것을 인식합니다.

소수 병렬 예제

다음 예제는 2보다 큰 소수를 계산합니다.

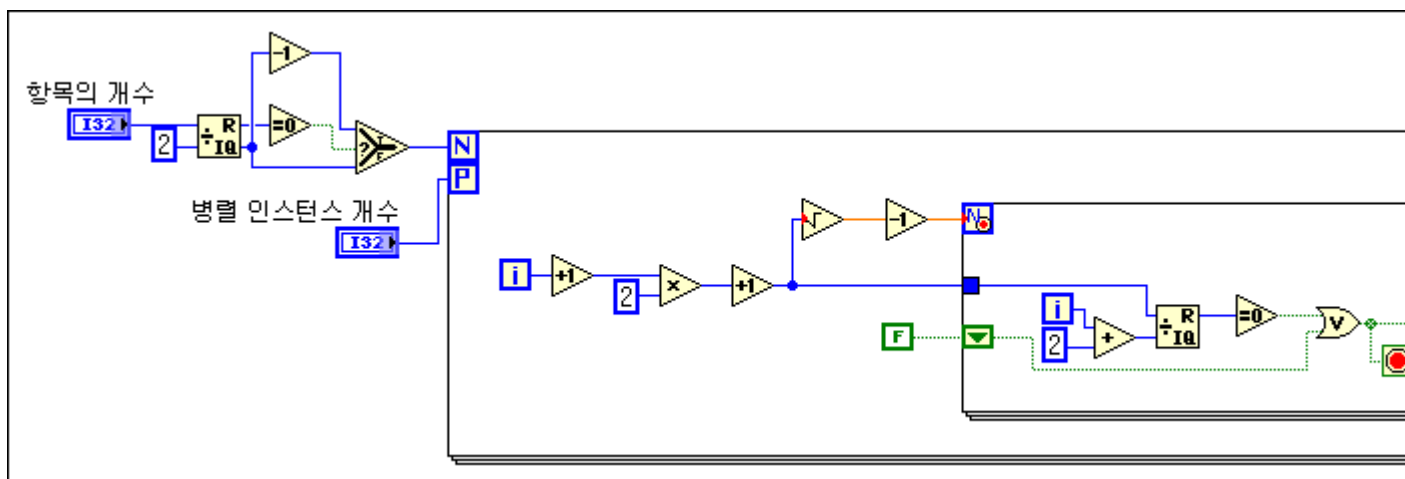


블록다이어그램은 3과 숫자 항 사이의 모든 홀수를 계산하고 소수인지 결정합니다. 안쪽의 For 루프는 임의의 숫자를 나눈 나머지가 제로일 때 참값을 반환합니다.

바깥쪽 For 루프의 반복은 서로 의존하지 않습니다. For 루프의 반복이 순서에 관계없이 실행될 수 있는 경우, For 루프에서 **루프 반복 병렬 처리**를 활성화해서 성능을 향상시킵니다. 예를 들어 바깥쪽의 For 루프는 각각의 반복루프마다 많은양의 연산을 처리하기 때문에 반복 병렬 처리를 하기에 좋은 대상입니다.

이 VI에서 병렬 처리를 활성화하려면, For 루프에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 반복 병렬 처리 설정을 선택합니다. 이렇게 하면 For 루프 반복 병렬 처리 대화 상자가 나타납니다.

멀티프로세서 컴퓨터에서 병렬 루프 반복을 사용하면, LabVIEW가 여러 프로세서를 활용하여 For 루프를 더 빨리 실행하도록 할 수 있습니다. 이 예의 병렬 버전은 LabVIEW가 동시에 여러 다른 코드를 실행할 수 있기 때문에 더욱 효율적입니다.



이러한 두 개의 예제 VI에는 명확한 스레드 관리를 위한 코드를 가지지 않습니다. LabVIEW 데이터 흐름 프로그래밍 패러다임은 LabVIEW 실행 시스템이 반복 루프 인스턴스를 다른 스레드에서 실행하도록 허용합니다.

하이퍼스레드 또는 멀티프로세서 시스템을 위한 프로그래밍

하이퍼스레드를 지원하는 컴퓨터를 위한 어플리케이션의 성능 최적화는 멀티프로세서 컴퓨터의 성능 최적화와 거의 같습니다. 그러나, 하이퍼스레드를 지원하는 컴퓨터는 두 개의 논리적인 프로세서 사이에 캐시 및 실행 단위와 같은 일부 리소스를 공유하기 때문에 차이점은 존재합니다. 하이퍼스레드를 지원하는 컴퓨터의 공유된 리소스가 어플리케이션에 제한이 된다고 생각하는 경우, Intel VTune과 같은 고급 샘플링 성능 분석기로 어플리케이션을 테스트합니다.

LabVIEW의 메모리 관리

성능과 메모리 관리하기

이 섹션은 VI 성능에 영향을 미치는 요소를 설명하고 가장 뛰어난 성능을 얻는 데 도움이 되는 기술을 제공합니다. 이 섹션은 VI의 실행 시간과 메모리 사용에 대한 데이터를 디스플레이하는 성능과 메모리 프로파일 윈도우를 설명합니다. 또한 LabVIEW가 사용하는 데이터 흐름 모델의 메모리 관리도 설명하며 효율적으로 메모리를 사용하는 VI 생성에 관한 팁도 제공합니다.



노트

LabVIEW VI Analyzer Toolkit을 이용하여 VI의 성능과 효율성 및 기타 LabVIEW 프로그래밍 영역을 대화식 또는 프로그램적으로 점검할 수 있습니다.

성능과 메모리 관리에 대한 예제는 labview\examples\Performance 디렉토리를 참조하십시오.

관련 예제 찾기



관련 정보

[성능과 메모리 프로파일 윈도우 사용하기](#) (725 페이지 참조)

[VI 실행 속도](#) (191 페이지 참조)

[VI 메모리 사용](#) (199 페이지 참조)

[VI 코드 복잡도](#) (233 페이지 참조)

[재호출: 같은 SubVI에 동시에 여러 호출을 허용](#) (234 페이지 참조)

[VI 실행 시간과 메모리 사용 프로파일](#) (911 페이지 참조)

[어플리케이션의 실행 속도 최적화하기](#) (912 페이지 참조)

[편집기 응답과 VI 실행 속도 중 선택하기](#) (914 페이지 참조)

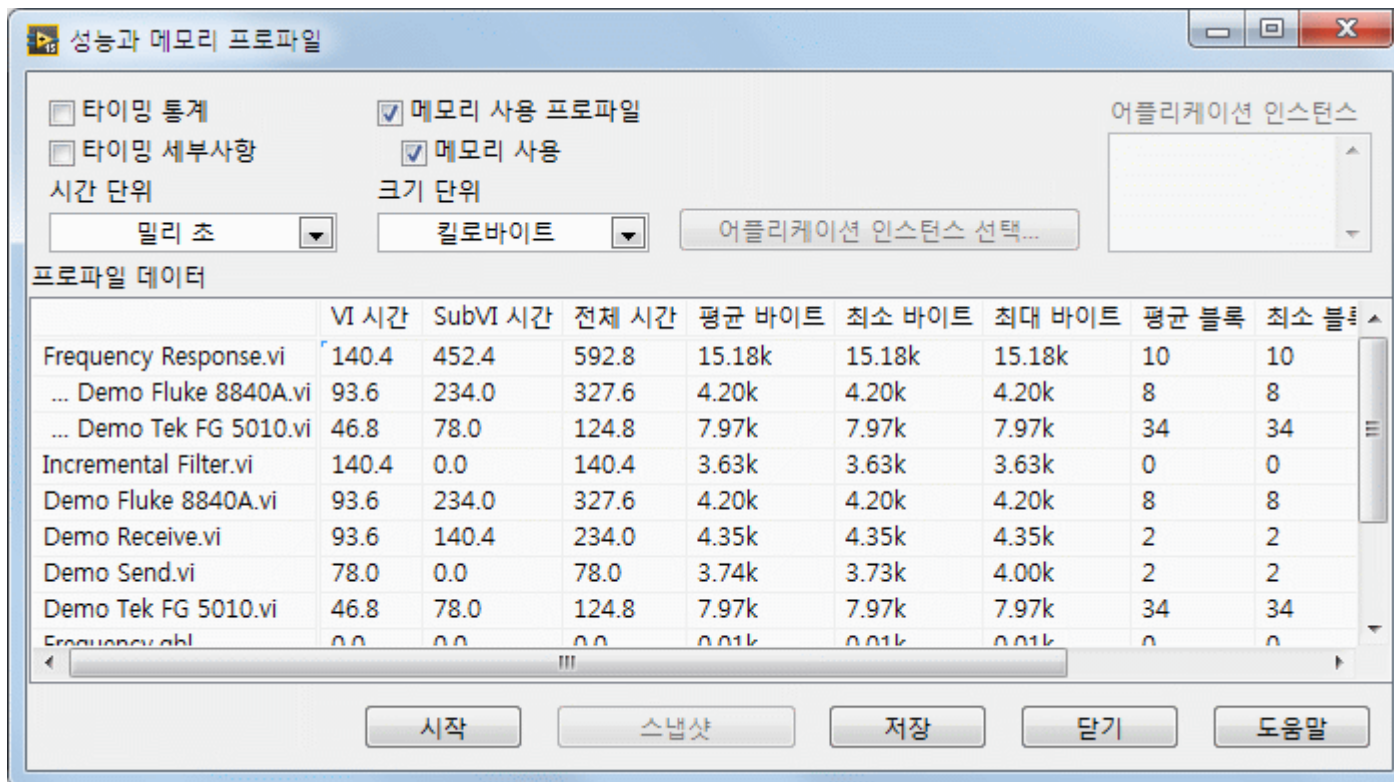
[VI를 컴파일할 때 발생하는 메모리 부족 에러 해결하기](#) (916 페이지 참조)

성능과 메모리 프로파일 윈도우 사용하기

성능과 메모리 프로파일 윈도우는 어플리케이션이 어디에서 그 시간을 소요하고 어떻게 메모리를 사용하는가에 대해 결정하는 강력한 도구입니다. 성능과 메모리 프로파일 윈도우에는 대화식 테이블 디스플레이가 있어 시스템에 있는 각 VI의 시간과 메모리 사용을 보여줍니다. 테이블의 각 행은 특정한 VI의 정보를 가지고 있습니다. 각 VI에 의해 사용된 시간은 몇 개의 항목으로 나누어지고 요약됩니다. 성능과 메모리 프로파일 윈도우는 VI 실행당 사용된 최소, 최대, 평균 시간을 계산합니다.

이 대화식 테이블 디스플레이를 사용하여 이 정보의 전체 또는 일부를 볼 수 있고, 다른 항목별로 정렬할 수 있고, 특정한 VI로부터 호출되는 SubVI의 성능 데이터를 볼 수 있습니다.

도구 >> 프로파일 >> 성능과 메모리를 선택하여 성능과 메모리 프로파일 윈도우를 디스플레이합니다. 다음 그림은 이미 사용 중에 있는 윈도우의 예를 보여줍니다.



메모리 사용 정보 수집을 하게 되면 수집 과정에서 VI의 실행 시간에 상당량의 오버헤드가 추가될 수 있기 때문에 이는 옵션 사항입니다. 메모리 사용 프로파일 확인란을 적절히 선택하여 성능과 메모리 프로파일 윈도우를 시작하기 전에 데이터 수집 여부를 선택해야 합니다. 이 확인란은 프로파일 세션이 일단 시작된 후에는 변경할 수 없습니다.

그 결과를 보기

테이블에 정보의 일부만 디스플레이하도록 선택할 수 있습니다. 일부 기본 데이터는 항상 볼 수 있지만, 성능과 메모리 프로파일 윈도우의 적절한 확인란에 확인 표시를 하거나 제거하여 통계, 세부사항, 메모리 사용(활성화된 경우)을 디스플레이하도록 선택할 수 있습니다.

성능 정보는 글로벌 VI를 위해 디스플레이됩니다. 그러나, 이 정보는 때때로 아래의 항목별 섹션에서 설명된 것과 해석이 다른 경우도 있습니다.

테이블 디스플레이에서 SubVI의 이름을 더블 클릭하여 SubVI를 위한 성능 데이터를 볼 수 있습니다. 이렇게 할 때, 새로운 행이 VI의 이름 바로 아래에 나타나고 그 SubVI 각각에 대한 성능 데이터를 가지게 됩니다. 글로벌 VI의 이름을 더블 클릭할 때, 새로운 행이 프런트패널의 각 개별 컨트롤에 대해 나타납니다.

원하는 열 헤더에서 클릭하여 테이블 디스플레이에 데이터의 행을 정렬할 수 있습니다. 현재 정렬된 열이 굵은체의 헤더 제목으로 나타납니다.

VI의 타이밍은 VI가 실행을 끝마치는데 경과된 시간의 양과 반드시 일치하지는 않습니다. 이것은 여러 스레드 실행 시스템이 2개 이상의 VI의 실행을 끼워넣을 수 있기 때문입니다. 또한, 사용자가 대화 상자에 응답하는데 걸리는 시간, 블록 다이어그램의 [기다림 (ms)] 함수에서 소모된 시간, 마우스 클릭을 확인하는데 걸리는 시간과 같은 VI와 관련이 없는 상당한 양의 오버헤드가 있습니다.

타이밍 정보

타이밍 통계 확인란에 확인 표시가 있을 때, VI의 타이밍에 대한 추가적인 상세 설명을 볼 수 있습니다.

타이밍 세부사항 확인란에 확인 표시가 있을 때, VI에 의해 소모된 총 시간을 여러 타이밍 항목으로 분류한 내역을 볼 수 있습니다. 많은 사용자 인터페이스를 가지고 있는 VI에 대해, 이 항목은 어떤 작동이 대부분의 시간을 차지하는지 결정하도록 도와줍니다.

메모리 정보

프로파일 과정을 시작하기 전에 메모리 사용 프로파일 확인란에 확인 표시를 하면 이용할 수 있는 메모리 사용 확인란에 확인 표시를 하면, VI가 메모리를 어떻게 사용하는지에 대한 정보를 볼 수 있습니다. 이 값은 VI를 위한 데이터 공간에 의해 사용된 메모리의 추정이고 모든 VI를 위해 필요한 부수적인 데이터 구조를 포함하지 않습니다. VI의 데이터 공간은 프런트패널 컨트롤에 의해 외부적으로 사용되는 데이터뿐만 아니라 컴파일러가 내부적으로 생성한 임시적인 버퍼도 가지고 있습니다.

메모리 크기는 VI 실행의 결과로 추정되며 정확한 전체 사용을 반영하지 않는 경우도 있습니다. 예를 들면, VI가 그 실행동안 큰 배열을 생성하지만 VI가 끝나기 전에 그 크기를 줄이면, 디스플레이된 크기는 중간의 배열 크기는 반영되지 않습니다.

이 섹션은 사용된 바이트 수와 관련된 데이터와 사용된 블록 수와 관련된 데이터의 두가지 데이터 세트를 디스플레이합니다. 블록은 단일 데이터를 저장하기 위해 사용된 메모리의 연속적인 부분입니다. 예를 들면, 정수의 배열은 멀티바이트의 길이를 가질 수 있지만, 오직 하나의 블록을 차지합니다. 실행 시스템은 배열, 문자열, 경로, 그림을 위한 독립된 메모리의 블록을 사용합니다. 사용자 어플리케이션의 메모리 히프에 블록이 많으면 실행뿐만 아니라 전체적으로 성능을 저하시킬 수 있습니다.

코드 복잡도가 컴파일 시간에 미치는 영향

LabVIEW가 **VI를 컴파일**할 때 마다 컴파일러는 VI의 궁극적인 실행 속도를 최대화하기 위해 블록 다이어그램에 특정 최적화를 적용합니다. 이러한 최적화를 수행하려면, 컴파일러는 기존 코드를 분석하여 최적화가 가능한 부분 및 최적화로 인해 이득을 얻는 부분을 찾아야 합니다. 이러한 분석은 VI의 **코드 복잡도**에 비례하는 양의 메모리와 시간을 필요로 합니다. 그러므로, VI 안에 있는 코드의 양이 늘어날수록, 컴파일러가 최적화를 수행하는데 걸리는 메모리와 시간도 증가합니다.

극히 복잡한 VI를 최적화하는데 소요되는 메모리 때문에, 컴파일러가 이러한 VI를 컴파일할 때, 메모리를 소진하거나, 상당한 지연이 발생할 수 있습니다. 상당수의 VI에서 이러한 문제를 방지하기 위해 컴파일러는 각 VI의 복잡도를 컴파일러 최적화 임계점과 비교하며, 이때 컴파일러 최적화 임계점은 VI의 복잡도가 일정 수치를 초과하여 컴파일러가 많은 메모리를 사용하는 최적화를 제한하기 시작하는 지점입니다. 그러나, 최적화를 제한하면 VI가 최대 실행 속도를 얻지 못하게 됩니다. 그러므로, 복잡한 VI에서 컴파일러는 편집기 응답 및 VI 실행 속도 중 하나를 선택하고 나머지 하나를 희생하게 됩니다.

VI를 여러 개의 SubVI로 나누어 이를 방지할 수 있습니다. 이렇게하면 편집기 응답이 향상되고, 실행 속도가 최적화되며, 코드의 관리가 용이해지고 가독성이 향상됩니다. 그러나 효과적으로 여러 SubVI로 나눌 수 없는 VI의 경우, LabVIEW는 사용자가 **임계점을 조정**하여 다음과 같은 문제를 해결하도록 합니다:

- VI를 컴파일하거나 어플리케이션을 만드는 도중 LabVIEW의 메모리가 소진됩니다.
- VI를 컴파일하거나 어플리케이션을 만드는 도중 LabVIEW에 상당한 지연이 나타납니다.
- 편집을 마친 후, VI 또는 어플리케이션을 최대 실행 속도로 컴파일하려고 합니다.

LabVIEW가 언제 VI를 컴파일하는지 알아내기

VI를 실행하려면, LabVIEW는 VI의 코드를 LabVIEW 런타임 엔진이 실행할 수 있는 형태로 번역해야 하며, 이 작업을 컴파일이라고 합니다. VI를 변경한 후에는 LabVIEW는 반드시 코드를 다시 컴파일해야 합니다. 그러나 컴파일에는 시간과 메모리가 소요되기 때문에, LabVIEW는 사용자가 VI를 편집하는 즉시 코드를 다시 컴파일하지는 않습니다. 대신, LabVIEW는 사용자가 다음 중 하나의 동작을 수행하려 할 때 VI를 다시 컴파일합니다:

- VI 저장하기
- VI 실행하기
- 이 VI를 필요로 하는 어플리케이션 만들기
- VI를 리얼타임 OS에 배포하기
- LabVIEW 이전 버전 또는 다른 플랫폼에서 저장된 VI를 로드하기



노트

오래된 버전의 LabVIEW 툴킷을 새 버전의 LabVIEW와 함께 사용하도록 설치하는 경우, LabVIEW는 자동으로 툴킷 VI를 로드하기 전에 다시 컴파일합니다.

LabVIEW에서 VI의 재컴파일을 필요로 하는 변경사항

LabVIEW는 사용자가 다음과 같은 동작을 수행할 때, 다시 컴파일이 필요한 VI로 표시합니다:

- 블록 다이어그램 수정하기
- 컨트롤 또는 인디케이터 추가하기
- 컨트롤 또는 인디케이터 모양 수정하기 그러나 컨트롤의 값을 변경할 때는 LabVIEW에서 이 VI를 다시 컴파일하지 않아도 됩니다.
- VI 프로퍼티 대화 상자에서 VI의 특정 프로퍼티 변경하기
- LabVIEW가 SubVI를 다시 컴파일해야 하도록 SubVI 변경하기 VI 또는 SubVI에서 컴파일된 코드를 분리하는 경우, LabVIEW에서 VI를 다시 컴파일하지 않아도 됩니다.
- 위의 동작을 프로그램적으로 수행하기

관련 정보

코드 복잡도가 컴파일 시간에 미치는 영향

편집기 응답과 VI 실행 속도 중 선택하기

파일에서 컴파일된 코드 분리하기

큰 데이터 세트에 대한 메모리 관리

LabVIEW는 자동으로 **메모리 할당**을 처리합니다. 자동적인 프로세스이기 때문에, LabVIEW에서 데이터를 안전하게 처리하려면 사전에 주의를 기울여야 합니다. 이는 LabVIEW가 자주 데이터를 복사해야 한다는 것을 의미합니다. 프로그램에 큰 데이터 세트를 다루는 경우, 작은 대규모 데이터 복사는 메모리 부족 에러를 야기할 수 있습니다. 다음 가이드라인에 따라 메모리 관련 문제를 방지하고 프로그램에서 큰 데이터 세트를 처리하는 과정을 최적화하십시오.



노트

성능과 메모리는 성능과 메모리 프로파일 윈도우를 디스플레이합니다. 이를 사용하여 VI의 실행 시간과 메모리 사용에 대한 정보를 얻고 디스플레이할 수 있습니다. 이 윈도우를 사용하여 메모리 관련 문제를 일으키는 VI를 찾아낼 수 있습니다.

큰 데이터 세트 복사본 개수 줄이기

LabVIEW는 데이터 흐름 언어이기 때문에, VI가 더 많은 데이터를 필요로 할 때, 한 개의 와이어가 두 개의 와이어로 될 때처럼 복사본을 생성합니다. LabVIEW는 통상적으로 언제 새 복사본을 만들어야 하는지 가늠할 수 있지만, 가늠이 불가능한 경우에는 만약의 경우에 대비해서 복사본을 만듭니다.



노트 LabVIEW Real-time Module을 사용하는 경우, 메모리 복사본은 Real-time 어플리케이션의 결정성에 영향을 미칩니다.

버퍼 할당 보이기 윈도우를 사용하여 LabVIEW가 어디에 메모리를 할당하는지 구분할 수 있습니다. 이 윈도우는 데이터 버퍼를 구분해내며, LabVIEW는 이 데이터 버퍼를 사용하여 데이터의 복사본을 저장합니다.

다음의 가이드라인을 따라 LabVIEW가 만들어야 하는 복사본의 개수를 줄이십시오.



노트 다음의 팁은 **좋은 LabVIEW 디자인 사례**에 있는 내용과 상충될 수도 있으며, 비정상적으로 큰 데이터 세트 작업을 할 때 메모리의 소모를 줄이는 용도로만 활용하는 것이 좋습니다.

- 가능한 가장 작은 데이터 타입을 사용합니다. 예를 들어, 배정도 부동소수 대신에 16-비트 정수를 사용합니다.
- 단순한 배열을 사용합니다. LabVIEW는 웨이브폼 또는 다이나믹 데이터에서의 프로세스를 위해 데이터 배열을 추출할 때 데이터 복사본을 추가적으로 하나 더 만듭니다.
- 큰 블록 다이어그램 또는 인라인 SubVI를 생성하여 **오버헤드를 줄입니다**. LabVIEW는 SubVI를 호출할 때 데이터의 복사본을 생성하므로, SubVI를 호출하는 것을 피하십시오.
- SubVI를 통해 데이터를 경로를 설정할 경우에 블록 다이어그램 터미널들이 구조나 루프의 밖에 위치하도록 하십시오. LabVIEW는 터미널이 케이스 구조 또는 루프 안에 있을 때 더 많은 데이터 복사본을 생성합니다.
- 필요한 경우를 제외하면, 가능한 한 데이터가 루프를 통과하지 않도록 주의합니다. 데이터가 루프를 통과해야 하는 경우, 터널 대신 시프트 레지스터를 사용합니다. 터널을 사용하는 경우, LabVIEW는 루프가 반복될 때마다 매번 데이터 복사본을 만듭니다.
- 가능한 경우, **필수 입력을 사용**합니다. LabVIEW는 기본값을 생성할 때 데이터 복사본을 생성합니다.
- 플랫 시퀀스 구조 대신에 In Place 원소 구조를 사용합니다.



팁 최신 LabVIEW 버전이 자동으로 만드는 복사본의 개수는 이전 버전에 비해 적습니다. 따라서 최신 LabVIEW 버전일수록 큰 데이터 처리 성능이 좋습니다.

큰 데이터 세트 전송하기

데이터 복사본이 생성되는 것을 막을 수 없을 때에는 각 복사본의 크기를 더 작게 만드는 방법이 있습니다. 이를 위해, 데이터를 다른 위치로 전송할 때 큰 데이터 세트를 더 작은 세트로 나눌 수 있습니다. 이 방법은 그룹으로 나누기(chunking)라고도 불립니다. 데이터를 더 작은 그룹으로 나눌 때 LabVIEW가 만드는 복사본은 메모리의 사용을 저해하지 않습니다. 복사본 자체는 작업량 자체에 부정적인 영향을 미치지 않기 때문에, 그래도 복사본을 최소화하는 것이 권장됩니다. 다음의 예는 이와 관련된 개념을 설명해주고 있습니다.

512 MB의 데이터를 디스크에 저장해야 합니다. 단 한번의 호출로 모든 데이터를 불러와서 디스크에 저장할 수 있습니다. 하지만, 복사본의 개수를 1 개로 최소화하더라도, 원본 데이터와 하나의 추가 복사본을 전송하고 있는 것입니다. 즉, LabVIEW로부터 1 GB의 메모리를 요청하는 것입니다. 따라서, 한 번에 500 KB 씩 데이터를 가져오도록 반복적인 루프를 만들어서 디스크에 스트리밍하는 것이 더 좋은 방법입니다. 이제 메모리는 대부분의 컴퓨터가 처리할 수 있는 1 MB, 즉 원본 데이터 500 KB와 복사본용 500 KB로 줄게 됩니다.

이를 통해 얻게되는 추가적인 이점은 LabVIEW가 대규모의 메모리 블록을 할당하는데 소비하게 될 상당량의 시간을 절약할 수 있다는 것입니다. 최신형 컴퓨터에서 250 MB의 데이터를 스트리밍하는데 15 초 내외의 시간이 걸립니다. LabVIEW가 1 GB RAM을 할당하는데에만 해도 그 정도의 시간이 필요할 것입니다.

RAID(Redundant Array of Independent Disks)에 대규모 데이터 세트를 스트리밍하거나 데이터를 저장하려는 경우, **버퍼링** 없이 데이터를 전송하여 데이터 복사를 줄이고 데이터 전송 속도를 높일 수 있습니다. 버퍼링을 비활성화하려면, [파일 열기/생성/대체] 함수의 버퍼링 비활성화 입력에 참 값을 연결합니다.

큰 데이터 세트 디스플레이하기

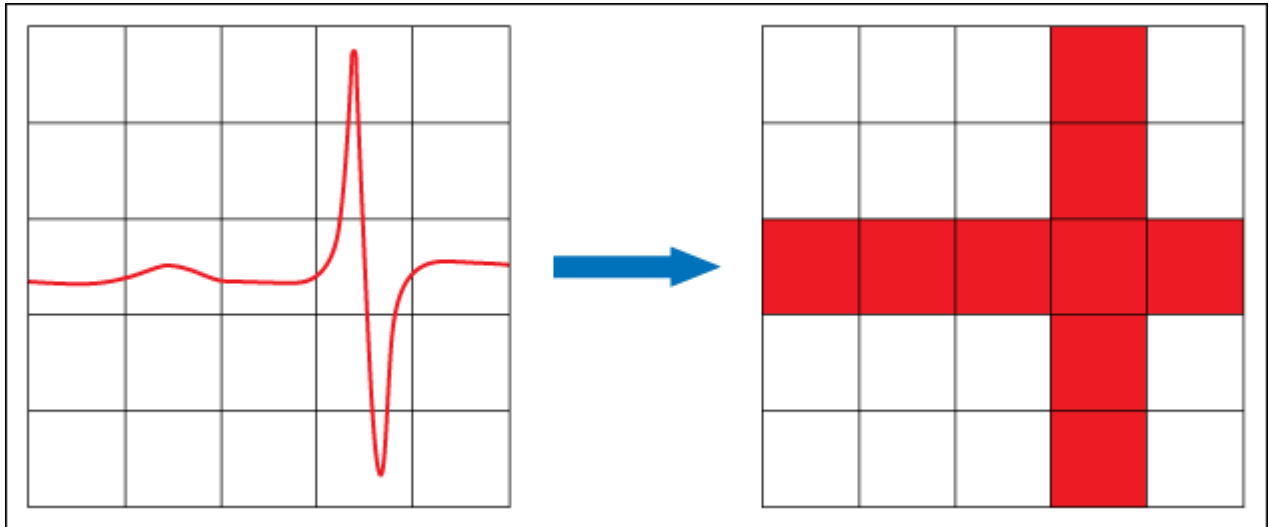
많은 대화식 어플리케이션에서는 다른 사람이 데이터를 디스플레이하는 것이 가장 중요한 작업입니다. 5백만 개의 데이터 포인트를 디스플레이해야 하는 경우도 있지만, 이는 대부분의 디스플레이에서는 지원할 수 없을 정도의 데이터 크기입니다. 평균적으로 LabVIEW 그래프는 폭이 대략 300에서 1000 픽셀입니다. 5백만 포인트는 실제 웨이브 폼 그래프에서 볼 수 있는 크기의 3 배가 넘습니다. 이때 데이터 부분제거(Data decimation)를 통해 문제를 해결할 수 있습니다.

예를 들어, 그래프 상에서 정밀한 픽셀로 대규모 데이터 세트를 나타내고자 한다고 가정 해봅시다. 5백만 포인트 버퍼에서 결함이 하나 있는 경우, 플롯은 폭이 1 픽셀인 스파이크 하나가 있는 수평선을 나타낼 것입니다. 데이터가 사인파이고 스크린의 픽셀 폭보다 더 큰 주기를 갖고 있는 경우, 그래프는 앨리어스가 없는 상태에서 스크린 전체에 깔끔한 곡선형으로 표시될 것입니다. 최대-최소 부분제거 알고리즘이 이 두 경우에 적용될 수 있습니다.

최대-최소 부분제거는 각 부분제거 간격의 최대 및 최소 데이터 포인트를 사용하는 부분 제거입니다. 단순한 부분제거는 부분제거 간격의 데이터 포인트로 각 부분제거 간격의 첫 포인트를 사용합니다. 단순한 부분제거를 사용하면 대상을 작위적으로 앨리어스하게 되므로, 시간이 중요하고 정확도는 중요하지 않는 경우가 아니라면 사용하지 말아야 합니다.

최대-최소 부분제거를 적용하려면, 먼저 그래프의 픽셀 폭을 알아야 합니다. 그래프의 플롯 영역:크기:폭 프로퍼티에서 이를 찾을 수 있습니다. 작위적인 결과와 나오지 않도록, 하나의 픽셀 폭당 최소한 2 개의 부분제거 구간이 필요합니다. 따라서 그래프 픽셀 폭을 2로 곱하여, 데이터를 나눌 구간의 개수를 구합니다. 데이터 길이를 이 숫자로 나눈 후, 올림하여 가장 가까운 정수를 구합니다. 이는 부분제거의 그룹 크기입니다. 각 그룹에서 최대 및 최소 포인트를 찾고, 이를 데이터 세트에서 발생한 순서대로 정렬합니다. 마지막 그룹이 갖고 있는 포인트의 개수가 나머지 그룹보다 적더라도 문제가 되지 않습니다. 그룹이 1 픽셀 폭보다 작은 경우에는 컴퓨터 스크린에서 볼 수 없기 때문에 문제가 될 수는 있습니다. 모든 최대 및 최소 데이터를 연결하여 플롯합니다. 스크린 상에서 한 픽셀 폭당 4 개의 포인트가 있을 것입니다. 이 덕분에, 옆에 있는 다른 픽셀과 겹쳐지지 않고 스파이크가 한 픽셀 폭에서 표시될 수 있습니다. 최대-최소 부분제거를 통해 항상 데이터의 피크를 볼 수 있고, 고주파수의 사인파가 만드는 곡선이 바르게 표시될 수 있습니다. 그래프에 플롯하는 데이터 숫자는 더 적은 상태에서 더 빠른 속도로 이 작업이 모두 수행될 수 있습니다.

다음 그림에서 왼쪽의 데이터를 최대-최소 부분 제거를 사용하여 처리하면, LabVIEW가 오른쪽에 그래프를 생성합니다.



큰 데이터 세트 저장하기

큐나 **데이터 값 참조**를 사용하여 많은 메모리를 사용하지 않고 큰 데이터 세트를 메모리에 저장합니다. 데이터를 포함하는 단일 원소로 큐를 생성합니다. 데이터를 사용하고 싶을 때마다, 원소를 큐에서 제거합니다. 이는 프로그램의 다른 부분이 동시에 접근할 수 없도록 합니다. 데이터에서 작업하려면, 다시 원소로 큐를 생성합니다. 이때 오직 큐 참조만 전달하면 됩니다. LabVIEW는 큐를 전달할 때에는 필요한 데이터만을 복사합니다. 큐에 이름을 붙이면, 큐 얻기 함수에서 이름을 지정함으로써 언제든지 큐에 대한 참조를 얻을 수 있습니다. 여러 데이터 객체를 생성하는 것은 여러 큐를 생성하는 것 만큼 쉽습니다.

데이터 값 참조를 사용하여 여분의 데이터 복사본을 생성하지 않고 데이터를 저장할 수 있습니다. 데이터 값 참조는 큐보다는 효율성이지만, 타임아웃 옵션은 없습니다. 데이터를 큐에 전달하지 않고, 데이터에 대한 참조를 생성하여 참조를 전달할 수 있습니다. 데이터에서 작업하려면, In Place 원소 구조를 사용합니다. 데이터 값 참조 원소 읽기/쓰기 경계 노드는 데이터 값 참조를 입력으로 받고, 사용자가 In Place 원소 구조 안에 있는 데이터에서 작업할 수 있도록 하며, 원래 메모리 공간에 있는 데이터를 대체합니다.



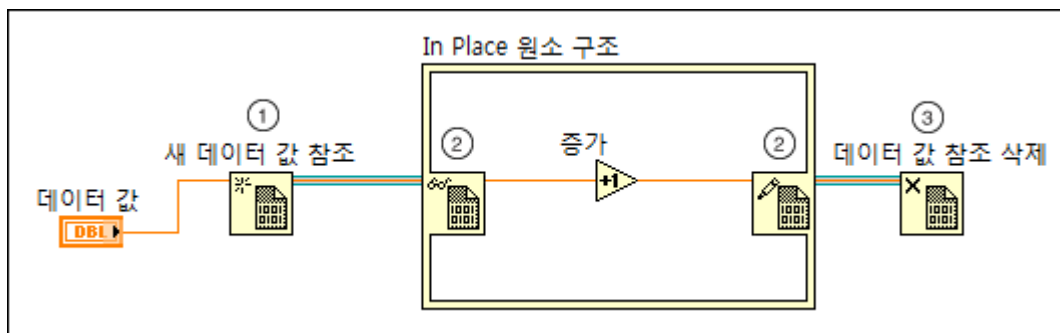
노트 클래스 프로퍼티 대화 상자의 상속 페이지에서 LabVIEW 클래스의 멤버 vi만이 클래스에 대한 데이터 값 참조를 생성하도록 할 수 있습니다.

또한 **기능적인 글로벌 변수**를 사용하여 큰 데이터 세트를 메모리에 저장할 수 있습니다. 기능적인 글로벌 변수를 사용하면, LabVIEW는 데이터를 그룹으로 저장하고 사용함으로써 메모리를 많이 소모하지 않으면서 데이터를 전달할 수 있습니다. 초기화되지 않은 데이터를 지니고 있을 도구로 시프트 레지스터를 사용합니다. 배열 함수를 사용하여 데이터를 읽고, 쓰고, 크기 조정합니다. 배열 함수는 in-place로 작동하며, 데이터 복사본을 생성하지 않습니다. 이 접근법은 일반적으로 큐 접근법보다 더 느립니다.

값 참조를 사용하여 데이터를 저장하고 데이터 복사 줄이기

LabVIEW가 데이터를 자주 복사하기 때문에, 큰 데이터 세트를 가진 어플리케이션에서는 실행 속도가 느려지고, 메모리 부족 에러가 발생할 수 있습니다. 데이터 값 참조를 사용해서 **큰 데이터 세트를 저장**하는 방법으로 **메모리를 관리**하고 데이터를 지나치게 자주 복사하는 것을 피할 수 있습니다.

다음 블록 다이어그램은 데이터 값 참조가 가리키는 위치에 저장되어 있는 데이터에 접근하고, 이 데이터를 처리하는 방법을 보여줍니다:



노트

구조 오른쪽의 경계 노드에서 마우스 오른쪽 버튼을 클릭하고 병렬 읽기 전용 권한 허용을 선택하여 데이터 값 참조의 읽기 전용 접근을 활성화할 수 있습니다. 오른쪽의 경계 노드가 연결되지 않은 경우, LabVIEW는 여러 개의 동시 읽기 전용 작업을 허용하고 데이터 값 참조를 변경하지 않습니다.

외부 데이터 저장하고 복사본 줄이기

외부 데이터 값 참조는 데이터 스트리밍 디바이스를 제어하는 디바이스 드라이버의 메모리 직접 참조(DMA: direct memory access) 버퍼와 같이, LabVIEW 밖의 구성요소가 소유하는 데이터와 직접적으로 상호 작용하는 점을 제외하면 데이터 값 참조와 비슷합니다. 외부 데이터를 LabVIEW로 불러오기 위해 복사본을 만들지 않고, 이 외부 데이터를 사용할 수 있습니다. 디바이스 드라이버가 외부 데이터 값 참조를 지원하는 경우, 데이터 값 참조와 상호 작용하는 방법과 비슷한 방법으로 이 외부 데이터와 상호 작용할 수 있습니다. 또한 고급 TDMS VI 및 함수를 사용하여 외부 데이터 값 참조와 **상호 작용**할 수 있습니다.



데이터 통신 방법

LabVIEW가 메모리에 데이터를 저장하는 방법

LabVIEW는 객체의 데이터 타입에 따라 컨트롤, 인디케이터, 와이어, 기타 객체를 위해 메모리에 데이터를 저장합니다.

불리언 데이터

LabVIEW는 불리언 데이터를 8비트 값으로 저장합니다. 값이 제로이면 불리언 값은 거짓입니다. 0이 아닌 모든 값은 참 (TRUE)을 나타냅니다.

바이트 정수

바이트 정수 숫자는 8비트 포맷의 부호 있는 정수와 부호 없는 정수를 가지고 있습니다.

워드 정수

워드 정수 숫자는 16비트 포맷의 부호 있는 정수와 부호 없는 정수를 가지고 있습니다.

롱 정수

롱 정수 숫자는 32비트 포맷의 부호 있는 정수와 부호 없는 정수를 가지고 있습니다.

쿼드 정수

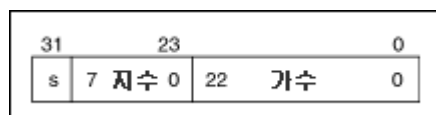
쿼드 정수 숫자는 64비트 포맷의 부호 있는 정수와 부호 없는 정수를 가지고 있습니다.

고정 소수점

고정 소수점 수는 64비트 포맷의 부호 있는 정수와 부호 없는 정수를 가지고 있습니다. LabVIEW는 확장된 비트가 유효숫자 값이 아니더라도 고정 소수점 수를 64비트로 부호 확장합니다.

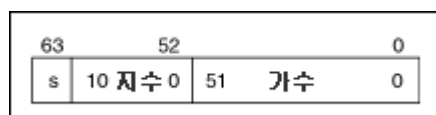
단정도

단정도 부동소수는 32비트 IEEE 단정도 포맷을 가지고 있습니다.



배정도

배정도 부동소수는 64비트 IEEE 배정도 포맷을 가지고 있습니다.

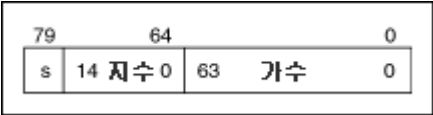


확장형

확장형 부동소수는 80비트 IEEE 확장형 포맷을 가지고 있습니다.

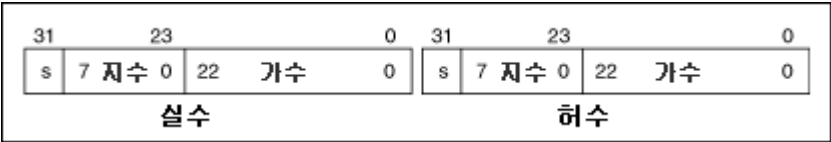


노트
어떤 경우에는 컴퓨터 프로세서에 따라 확장형 부동소수가 64, 96, 128 비트 IEEE 확장형 포맷을 가질 수 있습니다. 대부분의 Intel 및 AMD 프로세서는 80비트 포맷을 사용합니다.



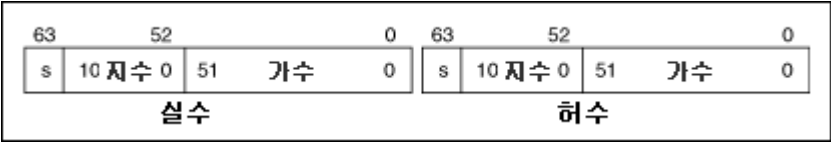
단정도 복소수

단정도 복소수 부동소수는 32비트 IEEE 단정도 포맷의 실수와 허수로 구성되어 있습니다.



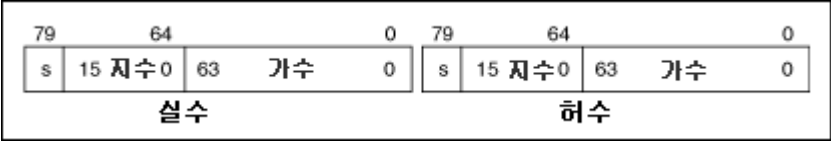
배정도 복소수

배정도 복소수 부동소수는 64비트 IEEE 배정도 포맷의 실수와 허수로 구성되어 있습니다.



확장형 정밀도 복소수

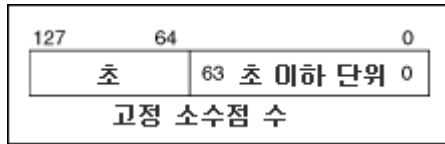
확장형 복소수 부동소수는 IEEE 확장형 포맷의 실수와 허수로 구성되어 있습니다.



부동소수와 복소수에서, s는 부호 비트(양의 값의 경우 0, 음의 값의 경우 1)이고, exp는 밑이 2인 편중 지수이고, 가수는 [0,1] 범위에 있는 숫자입니다.

타임스탬프

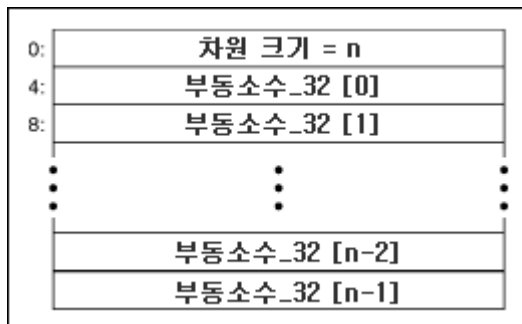
LabVIEW는 타임스탬프를 두 개 배정도 수의 클러스터로 저장합니다. 첫 번째 배정도 수는 표준 시간대와 관계없는, 세계시 [01-01-1904 00:00:00], 1904년 1월 1일 금요일 오전 12시 이후부터 경과된 초의 수를 나타냅니다. 두 번째 배정도 수는 초의 소수 단위를 나타냅니다.



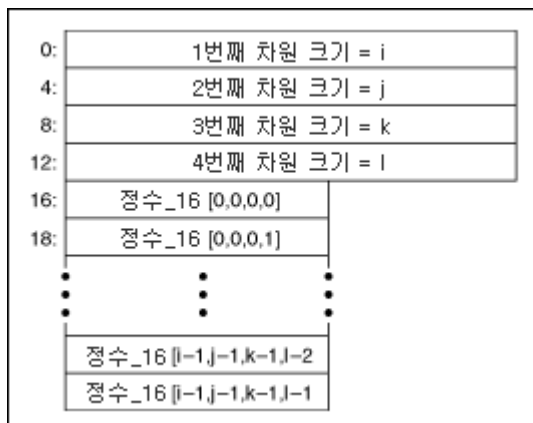
배열

LabVIEW는 배열을 핸들 또는 포인터에 대한 포인터로 저장하는데, 이 핸들이나 포인터는 데이터 앞에 배열의 각 차원의 크기를 32비트 정수로 가지고 있습니다. 핸들이 0이면 배열은 비어 있습니다. 특정 플랫폼의 정렬 제한 때문에, 데이터의 첫 번째 원소가 올바르게 정렬되도록 차원 크기 뒤에 몇 바이트가 채워지는 경우도 있습니다. LabVIEW 배열을 사용하는 공유 라이브러리를 쓰는 경우, **핸들의 크기를 조정**해야 할 수도 있습니다.

다음 그림은 단정도 부동소수의 1D 배열을 보여줍니다. 왼쪽의 십진수는 메모리에서 배열이 시작하는 곳의 바이트 오프셋 위치를 나타냅니다.



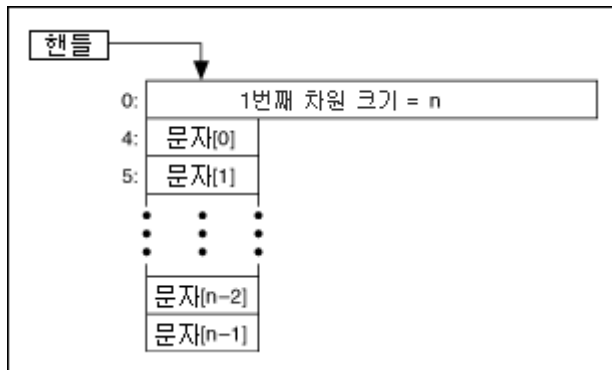
다음 그림은 16비트 정수의 4D 배열을 보여줍니다.



LabVIEW에서는 1D와 2D 배열이 메모리에 정렬됩니다. 이는 선형 대수 연산 및 행렬 데이터 타입과 연관된 연산 수행에 도움을 줍니다.

문자열

다음 그림에서 보는 것처럼, LabVIEW는 문자열을 4바이트 길이 값 뒤에 바이트 정수(8비트 문자)의 1D 배열이 포함된 구조의 포인터로 저장합니다. 핸들이나 구조의 포인터가 널인 경우 LabVIEW는 문자열을 빈 문자열로 다루는데, 이것은 길이 값이 제로인 문자열과 같습니다. LabVIEW는 문자열의 끝을 결정할 때 종료하는 문자 대신 길이 값을 사용하기 때문에, 문자열의 어느 포인트에나 널 문자, ASCII 문자 0을 포함한 모든 문자를 내장할 수 있습니다. LabVIEW 문자열을 C 문자열을 기다리는 외부 코드에 전달할 때, 내장된 널 문자는 해당 코드가 문자열을 첫 번째 널 문자에서 끝나는 것으로 해석하도록 합니다.



경로

LabVIEW 경로는 경로 타입과 경로 구성요소를 포함하는 불투명한 데이터 구조에 대한 포인터입니다. 경로 타입은 절대 경로의 경우 0, 상대 경로의 경우 1, 범용 명명 규칙(UNC) 경로의 경우 3입니다. UNC 경로는 Windows에서만 적용되는데 그 첫 번째 구성요소로서 드라이브 글자가 아닌 \\<machine name>\<share name>\을 씁니다. 경로 타입의 모든 다른 값은 유효하지 않은 경로를 나타냅니다.

다음 함수를 사용하여 경로에 대한 정보에 접근합니다:

- FDepth()
- FDirName()
- FIsAPath()
- FIsAPathOfType()
- FIsEmptyPath()
- FNamePtr()
- FVolName()

다음 함수를 사용하여 경로에 원소를 추가합니다:

- FAddPath()
- FAppendName()

다음 함수를 사용하여 새 경로를 생성합니다:

- FNotAPath()
- FPathCpy()
- FEmptyPath()
- FMakePath()
- FRelPath()

FDestroyPath() 함수를 사용하여 경로를 삭제합니다.

다음 함수를 사용하여 경로를 텍스트 형태로부터 또는 텍스트 형태로 변환합니다:

- FFileSystemStringToPath()
- FPathToFileSystemDSString()
- FPathToText()
- FTextToPath()
- ConvertPathToPlatformIndependentText()
- ConvertPlatformIndependentTextToPath()

다음 함수를 사용하여 두 경로를 비교합니다:

- FPathCmp()
- FPathCmpLexical()

관련 정보

Using LabVIEW Manager Functions in Shared Libraries

클러스터

LabVIEW는 클러스터 순서에 따라 여러가지 데이터 타입의 클러스터 원소를 저장합니다. 클러스터의 경계에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 클러스터 내의 컨트롤 순서 재설정을 선택하여 클러스터 순서를 보고 **클러스터의 순서를 수정**할 수 있습니다. LabVIEW는 스칼라 데이터를 클러스터에 직접 저장하고, 이는 간접적으로 배열, 문자열, 경로를 클러스터에 저장합니다. 특정 플랫폼의 정렬 제한 때문에, LabVIEW는 다음 클러스터의 시작 부분이 바르게 정렬되도록 배열에 있는 클러스터의 끝부분에 몇 바이트를 추가할 수도 있습니다. 클러스터의 배열을 사용하는 공유 라이브러리를 쓰는 경우, 배열에 대해 **행들의 크기를 조정**할 때 이 추가되는 부분을 고려해야 합니다.

클러스터의 메모리 레이아웃은 실행 중인 플랫폼에 따라 다릅니다. LabVIEW는 클러스터 원소 사이를 채워서 해당 원소가 특정 주소 경계에 맞게 정렬되도록 만들 수도 있습니다. 주소 경계는 자연스러운 정렬의 개념과도 관련 있습니다. 데이터가 시작되는 주소가 데이터 크기의 배수인 경우, 그 데이터 조각은 본래의 자연스러운 경계에 정렬되었다고 봅니다. 예를 들어, 4 바이트 정수가 4의 배수인 주소에서 시작하는 경우에는 자연스럽게 정렬되어 있는 것입니다. 대부분의 컴파일러는 이 경계 상에서 상위 리미트를 갖고 있으

나, 이에 대한 제한은 관대한 편입니다. 예를 들어, 8 바이트 정수가 4 바이트 경계에 정렬될 수도 있습니다. LabVIEW가 실행되는 플랫폼과 관련된 정렬 제한은 다음과 같습니다:

- (Phar Lap ETS) 데이터는 오직 1 바이트 경계에 정렬됩니다.
- (macOS 64비트 및 Linux 64비트) 데이터는 최대 8 바이트 경계까지 자연스럽게 정렬됩니다.
- (Windows 64비트 및 VxWorks) 데이터는 최대 8 바이트 경계까지 자연스럽게 정렬됩니다.

모든 플랫폼에서, 클러스터는 가장 엄격하게 정렬된 원소의 정렬을 따릅니다.

다음 테이블은 16비트 정수, 배정도 부동소수, 확장형 부동소수, 배열, 8 비트 부호없는 정수를 포함하는 클러스터를 나타냅니다. 각 행은 메모리 주소 및 이에 대응하는 원소와 원소의 크기를 포함합니다.

- (Phar Lap ETS) 데이터가 1 바이트 경계에만 정렬되기 때문에, 다음 클러스터의 메모리 주소는 17입니다.

메모리 주소	원소	크기(바이트)
I16 정수	2	
확장형 부동소수	10	
배열 핸들	4	
U8 정수	1	

- (Windows 64비트, macOS 64비트 및 Linux 64비트) 데이터가 최대 8 바이트 경계까지 자연스럽게 정렬되기 때문에, LabVIEW는 U8 정수 다음에 7 바이트를 채워서 다음 클러스터의 메모리 주소가 32가 되도록 만듭니다.

메모리 주소	원소	크기(바이트)
I16 정수	2	
확장형 부동소수	10	
채우기	4	
배열 핸들	8	
U8 정수	1	
채우기	7	

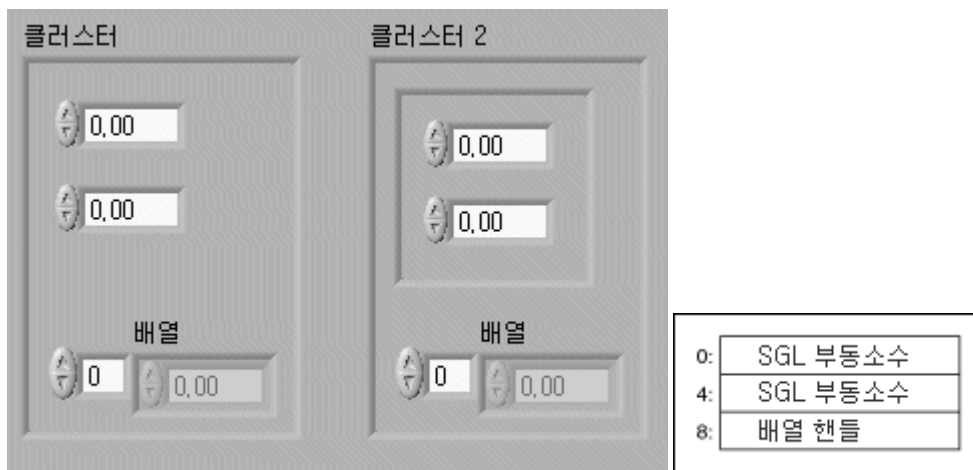
- (VxWorks) 데이터가 최대 8 바이트 경계까지 자연스럽게 정렬되기 때문에, LabVIEW는 U8 정수 다음에 7 바이트를 채워서 다음 클러스터의 메모리 주소가 40이 되도록 만듭니다. VxWorks는 확장형 부동소수를 롱 배정도로 구현합니다.

메모리 주소	원소	크기(바이트)
I16 정수	2	

메모리 주소	원소	크기(바이트)
채우기	6	
확장형 부동소수	16	
배열 핸들	8	
U8 정수	1	
채우기	7	

LabVIEW는 다른 클러스터 내에 임베디드된 클러스터를 직접 일렬로, 간접 지정(indirection) 없이 저장합니다. LabVIEW는 배열, 문자열, 경로만을 간접적으로 저장합니다.

다음 그림은 데이터를 같은 방식으로 저장하는 두 개의 다른 클러스터를 보여줍니다.



웨이브폼

LabVIEW는 웨이브폼을 클러스터같이 저장합니다.

참조 번호

LabVIEW는 부호 있는 32비트 정수로 참조 번호를 저장합니다.

배리언트

LabVIEW는 LabVIEW 내부 데이터 구조에 핸들로 배리언트를 저장합니다. 배리언트 데이터는 4 바이트로 이루어져 있습니다.

관련 개념

패턴화된 데이터

타입 설명자

패턴화된 데이터

LabVIEW는 메모리의 포맷으로부터 온 데이터를 파일에 쓰거나 파일로부터 읽는데 더 적합한 형태로 변환합니다. 이 더 적합한 형태를 패턴화된 데이터라고 부릅니다.

LabVIEW는 문자열, 배열, 경로를 핸들(메모리의 분리된 영역에서 포인터에 대한 포인터)에 저장하기 때문에, 이 문자열과 배열을 가지고 있는 클러스터는 인접해 있지 않습니다. 일반적으로, LabVIEW는 데이터를 트리 형태로 저장합니다. 예를 들면, LabVIEW는 클러스터를 배열도 부동소수로 저장하고, 문자열을 4바이트 핸들이 뒤에 따라오는 8바이트 부동소수로 저장합니다. LabVIEW는 메모리에 확장형 부동소수에 인접한 문자열 데이터를 저장하지 않습니다. 그러므로 디스크에 클러스터 데이터를 쓰려면 LabVIEW는 두 개의 다른 장소로부터 데이터를 얻어야 합니다. 많은 문자열, 배열 및/또는 경로를 가지고 있는 클러스터와 함께, LabVIEW는 데이터를 여러 다른 장소에 저장합니다.

데이터를 파일에 저장할 때, LabVIEW는 데이터를 저장하기 전에 단일 문자열로 패턴화합니다. 데이터를 패턴화하면 임의의 복소수 클러스터로부터의 데이터가 여러 장소에 저장되지 않고 인접한 데이터가 됩니다. LabVIEW가 파일에서 데이터를 로드할 때, LabVIEW는 단일 문자열을 읽고 그 문자열을 그 내부에서 가능한 인접하지 않은 형태로 패턴화 해제하는 반대 작업을 수행해야 합니다.

LabVIEW는 표준 형태로 패턴화된 데이터를 정규화하여 VI가 실행되는 플랫폼에 관계없이 데이터를 사용할 수 있도록 합니다. LabVIEW는 패턴화된 숫자 데이터를 빅 엔디언 형태(최상위 바이트 먼저)로 저장하고 패턴화된 확장형 부동소수를 16 바이트로 저장합니다.

LabVIEW로 만들지 않은 어플리케이션에서 사용되는 파일에 데이터를 쓸 때 또는 LabVIEW로 만들지 않은 어플리케이션에서 생성된 파일로부터 데이터를 읽을 때, 데이터를 패턴화한 후나 패턴화 해제하기 전에 리틀 엔디언(최하위 바이트 먼저)이나 빅 엔디언 형태로 변환할 수 있습니다. Windows 기반의 어플리케이션은 일반적으로 숫자 데이터가 리틀 엔디언의 형태가 될 것을 예상합니다.

[패턴화된 문자열로]와 [문자열로부터 패턴화 해제] 함수를 사용하여 LabVIEW가 데이터를 저장하고 로드할 때 내부적으로 LabVIEW가 하는 것처럼 데이터를 패턴화하고 패턴화 해제합니다.

데이터의 패턴화된 형태는 데이터의 타입을 인코딩하지 않습니다. LabVIEW는 이 정보를 **타입 설명자**에 저장합니다. [문자열로부터 패턴화 해제] 함수를 사용하려면 데이터 타입을 입력으로 연결하여 그 함수가 문자열을 적절히 디코드할 수 있도록 해야 합니다.

배리언트 데이터 타입을 사용하여 메모리에 쓸 때 데이터를 패턴화하고 메모리에서 읽을 때 데이터를 패턴화 해제하는 대신에 데이터 타입에 독립적으로 데이터를 다룰 수 있습니다. 배리언트 함수를 사용하여 배리언트 데이터를 생성하고 처리합니다.



노트

배리언트 데이터에서 [패턴화된 문자열로] 함수를 사용하는 경우, LabVIEW는 배리언트와 그 속성을 포함하는 모든 내용을 패턴화합니다. 그러나, [배리언트를 패턴화된 문자열로] 함수를 사용하는 경우, LabVIEW는 오직 배리언트만을 패턴화하고 모든 속성을 삭제합니다.

불리언과 숫자형 객체

모든 숫자 타입과 불리언 타입의 패턴화된 형태는 빅 엔디언 포맷으로만 데이터를 저장합니다. 예를 들면, -19의 값을 가진 32비트 정수는 FFFF FFED로 패턴화됩니다. 1/4와 같은 값을 가진 배정도 부동소수는 3FD0 0000 0000 0000으로 패턴화됩니다. 참인 불리언 값은 제로가 아닌 모든 값입니다. 거짓의 불리언 값은 00입니다.

확장형 수의 패턴화된 형태는 128비트의 확장형 부동소수 포맷입니다. 확장형 수를 디스크에 저장할 때, LabVIEW는 그 수를 이 포맷으로 저장합니다.

문자열과 경로

문자열과 경로가 여러가지 크기를 가지기 때문에, 그 길이를 바이트로 기록하는 패턴화된 32비트 정수가 패턴화된 형태의 앞에 옵니다. 예를 들면, ABC 값을 가진 문자열 타입은 0000 0003 4142 43으로 패턴화됩니다. 문자열의 경우, 패턴화된 포맷은 문자열이 메모리에서 취한 포맷과 비슷합니다.

그러나, LabVIEW가 메모리에 경로를 저장할 때 경로는 앞에 길이 값을 가지고 있지 않으므로, 이 값은 메모리의 데이터의 실제 크기로부터 얻고 LabVIEW가 데이터를 패턴화할 때 그 값이 앞에 옵니다. 이 길이의 앞에 4개의 문자인 PTH0가 옵니다.

예를 들면 C:\File의 값을 가진 경로는 5054 4830 0000 000B 0000 0002 0143 0466 696C 65로 패턴화됩니다.

5054 4830은 PTH0을 나타냅니다. 0000 000B은 전체 11 바이트를 나타냅니다. 0000은 타입입니다. 0002은 구성요소의 개수입니다. 0143은 파스칼 문자열로 글자 C를 나타냅니다. 0466 696C 65은 파스칼 문자열로 단어 File을 나타냅니다.

배열

배열의 각 차원 크기를 원소로 기록하는 패턴화된 32비트 정수는 패턴화된 배열을 위한 데이터의 앞에 옵니다. 차원 크기가 메모리에 저장되는 것처럼, 가장 느리게 변하는 차원이 먼저 오고, 더 빠르게 변하는 차원이 연속해서 옵니다. 패턴화된 데이터는 LabVIEW가 메모리에 저장하는 것과 같은 순서로 이 차원 크기 바로 다음에 옵니다. 다음 예제는 6개의 8비트 정수의 2D 배열을 보여줍니다.

{ {1, 2, 3}, {4, 5, 6} }은 0000 0002 0000 0003 0102 0304 0506로 패턴화됩니다.

다음 예제는 불리언 변수의 패턴화된 1D 배열을 보여줍니다.

{T, F, T, T}는 0000 0004 0100 0101로 패턴화됩니다. 참에 대해 선호하는 값은 01입니다.

클러스터

패턴화된 클러스터는 그 원소의 패턴화된 데이터를 클러스터 순서로 연결하는 것입니다. 예를 들면, 십진수로 4인 값의 16비트 정수와 12인 값의 32비트 정수의 패턴화된 클러스터는 0004 0000 000C입니다.

문자열 ABC의 패턴화된 클러스터와 4의 16비트 정수의 패턴화된 클러스터는 0000 0003 4142 4300 04입니다.

7의 16비트 정수의 패턴화된 클러스터, 8과 9의 16비트 정수의 패턴화된 클러스터는 0007 0008 0009입니다.

웨이브폼

LabVIEW는 웨이브폼을 클러스터같이 패턴화합니다.

참조 번호

LabVIEW는 대부분의 패턴화된 참조 번호를 **패턴화된 32비트 정수**로 저장하는데, 이것은 LabVIEW 내부 데이터 구조를 나타냅니다.

그 참조 번호 **타입 코드**로 나머지 참조 번호를 분류할 수 있습니다.

타입 코드인 0xE, 0xF, 0x15는 데이터를 **패턴화된 문자열**로 저장하는 참조 번호입니다. 이 문자열은 참조 번호 태그의 값을 가지고 있으며 비어 있는 경우(제로의 4 바이트)도 있습니다.

타입 코드인 0x1A, 0x1C, 0x1D는 다음 순서로 연결됩니다:

1. 참조 번호 태그에 있는 이름의 패턴화된 문자열. 참조 번호가 태그를 가지고 있지 않다면 이 문자열은 비어 있습니다(제로의 4 바이트).
2. 참조 번호에 특정한 정보를 가지고 있는 패턴화된 문자열. 이 문자열은 비어 있을 수 있습니다(제로의 4 바이트).
3. 참조 번호에 특정한 정보를 가지고 있는 패턴화된 문자열. 이 문자열은 비어 있을 수 있습니다(제로의 4 바이트).
4. 참조 번호에 특정한 정보를 가지고 있는 패턴화된 32비트 부호있는 정수.
5. 참조 번호에 특정한 정보를 가지고 있는 패턴화된 문자열. 이 문자열은 비어 있을 수 있습니다(제로의 4 바이트).

클래스

LabVIEW는 다음 일반 포맷에 따라 LabVIEW 클래스를 패턴화합니다:

계층구조 안의 레벨	클래스 이름	버전 리스트	프라이빗 데이터

- 계층구조 안의 레벨 — 클래스 계층구조에서 클래스가 발생하는 레벨의 수를 나타내는 4바이트 부호 없는 정수. 예를 들어 이 값이 2이라면, 이 클래스와 LabVIEW 객체 사이에는 한 개의 상위 클래스가 있습니다.

노트 계층구조 안의 레벨이 0인 경우, 객체는 LabVIEW 객체의 인스턴스이며 LabVIEW는 문자열에서 다른 3개의 필드를 제외합니다.

- 클래스 이름 — 클래스의 정규화된 이름의 길이를 나타내는 단일 바이트 뒤 따라 오는 파스칼 문자열의 시퀀스. 각 파스칼 문자열, 즉 첫 번째 바이트가 문자열의 길이를 나타내고 나머지 바이트가 문자열의 텍스트를 나타내는 바이트의 시퀀스는 정규화된 이름의 한 부분을 나타냅니다. 파스칼 문자열의 끝은 제로의 길이 바이트입니다. 패턴화된 문자열의 이 섹션에는 클래스 이름 섹션을 4 바이트의 배수로 증가시키는데 충분한 제로 패드 바이트가 포함되어 있습니다.
- 버전 리스트 — 각 클러스터가 버전 번호를 나타내는 4개의 2바이트 부호 없는 정수인 클러스터의 시리즈. 이 리스트의 첫 번째 클러스터는 클래스 이름의 버전, 두 번째는 상위 클래스의 버전을 나타내며 뒤의 숫자도 이와 같이 진행됩니다. 이 리스트는 계층구조 안의 각 레벨에 대해 한 개의 버전 번호를 포함합니다.

노트 계층구조 안의 레벨이 1이고 버전이 0인 경우, 패턴화된 데이터는 이 클래스의 기본 데이터를 나타내며 LabVIEW는 문자열의 나머지 필드를 제외합니다.

- 프라이빗 데이터 — 계층구조의 각 레벨의 프라이빗 데이터를 나타내는 패턴화된 클러스터 시리즈. 버전 리스트와는 반대 순서로, 이 시리즈의 첫 번째 클러스터는 최상위 클래스와 상응합니다. 각각의 패턴화된 클러스터는 뒤에 오는 데이터의 바이트 수를 나타내는 4바이트 부호 있는 정수로 시작합니다. 첫 번째 수가 0인 경우, 패턴화된 클러스터는 계층구조의 상응하는 클래스의 기본 데이터를 나타냅니다. 그렇지 않으면, 다음 데이터는 이 토픽 앞 부분에서 설명된 패턴화된 클러스터 표준형을 사용합니다. 각 패턴화된 클러스터는 클러스터를 4 바이트의 배수로 증가시키는데 충분한 패드 바이트로 끝납니다.

LabVIEW의 .NET Framework

LabVIEW에서 .NET 사용하기

.NET Framework를 사용하면 성능 모니터, 이벤트 로그, 파일 시스템과 같은 Windows 서비스 뿐만 아니라, 음성 인식 및 생성 서비스와 같은 고급 Windows API에 접근할 수 있습니다. 또한 .NET Framework는 SOAP, WSDL, UDDI와 같은 웹 프로토콜에 대한 접근도 제공합니다.

LabVIEW를 .NET 클라이언트로 사용하여 .NET 서버와 관련된 객체, 프로퍼티, 메소드, 이벤트에 접근할 수 있습니다. 또한 VI의 프런트패널에서 .NET 사용자 인터페이스 컨트롤

을 호스트할 수도 있습니다. LabVIEW는 .NET 서버가 아니지만 COM 객체에 대한 .NET 지원을 통해 ActiveX 서버 인터페이스가 있는 LabVIEW와 원격으로 통신할 수 있습니다.



노트

LabVIEW에서 .NET 객체를 생성하고 이 객체와 통신하려면 LabVIEW와 함께 설치되는 .NET CLR 4.0이 필요합니다. .NET 2.0 혼합 모드 어셈블리를 로드하려면 .NET 2.0 설정 파일을 사용해야 합니다. LabVIEW에서 .NET 제한에 대한 더 자세한 정보는 LabVIEW에서 .NET를 사용할 때의 요구조건을 참조하십시오. NI는 LabVIEW 프로젝트 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

올바른 위치에 .NET 어셈블리를 저장하여 예상대로 로드 및 동작할 수 있도록 점검하십시오.



관련 정보

[LabVIEW에서 ActiveX 사용하기](#) (351 페이지 참조)

[LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건](#) (745 페이지 참조)

[LabVIEW에서 .NET 2.0, 3.0, 3.5 어셈블리 로드하기](#) (1435 페이지 참조)

[.NET 객체 생성하기](#) (746 페이지 참조)

[LabVIEW 프로젝트에서 라이브러리 사용하기](#) (296 페이지 참조)

[.NET 어셈블리 로드하기](#) (750 페이지 참조)

.NET 환경

.NET 환경의 구성요소를 이해하면 컴퓨터가 LabVIEW와 .NET을 함께 사용하도록 하는데 도움이 될 뿐만 아니라 .NET Framework가 어떻게 작동하는지 이해하는데 도움이 됩니다. .NET 환경의 기본 구성요소에 대해 이해한 후, 사용자 컴퓨터의 .NET 환경이 [LabVIEW와 .NET을 함께 사용하는데 필요한 최소 요구조건](#)을 충족하는지 확인합니다.

.NET 환경을 구성하는 아이템은 다음과 같습니다.

- .NET Framework — Common Language Runtime에 대한 총칭 및 특정 버전의 .NET을 구성하는 클래스 라이브러리
- Common Language Runtime(CLR) — .NET 어셈블리의 로드와 실행을 담당하는 실행 시스템 CLR은 언어 통합, 보안 집행, 메모리, 자투리 모으기, 프로세스 관리, 스레드 관리와 같은 런타임 서비스를 제공합니다. 각종 프로그램과 .NET 사이의 통신을 돕기 위해 CLR은 여러 프로그래밍 언어 및 OS를 지원하는 데이터 타입 시스템을 제공합니다. 따라서 개발자는 이 시스템을 메모리와 스레드의 집합이 아닌 데이터 타입과 객체의 집합으로 볼 수 있습니다. CLR은 정보를 CLR intermediate language (IL) 메타데이터 포맷으로 생성하기 위해 컴파일러와 링커를 필요로 합니다. 모든 .NET 프로그래밍 언어 컴파일러는 어셈블리 코드 대신 CLR IL 코드를 생성합니다.
- 클래스 라이브러리—입출력, 문자열 처리, 보안 관리, 네트워크 통신, 스레드 관리, 텍스트 관리, 사용자 인터페이스 디자인과 같은 기본 기능을 제공하는 클래스의 세트입니다. 이러한 클래스는 Win32/COM 시스템과 동일한 기능을 제공합니다. .NET Framework에서는 하나의 .NET 언어에서 생성한 클래스를 다른 .NET 언어에서 사용할 수 있습니다.

- **어셈블리**—DLL, OCX, 또는 COM의 구성요소 실행 파일 등과 비슷한 배포 단위입니다. 어셈블리는 .NET 컴파일러를 사용하여 만든 DLL과 실행 파일입니다. 어셈블리는 단일 파일 또는 여러 파일로 구성될 수 있습니다. 어셈블리에는 어셈블리 이름, 버전, 작성자의 보안, 어셈블리를 구성하는 파일, 의존적인 어셈블리, 리소스, 반출된 데이터 타입에 대한 정보를 담은 목록이 포함됩니다.
- Global Assembly Cache (GAC)—시스템에서 사용 가능한 퍼블릭 어셈블리 목록입니다. GAC는 COM이 사용하는 레지스트리와 유사합니다.

LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건

LabVIEW에서 .NET 객체를 생성하고 조작하려는 경우이든 또는 LabVIEW를 사용하여 다른 어플리케이션이 사용할 .NET interop 어셈블리를 빌드하려는 경우이든, 다음의 기본 사양과 셋팅을 반드시 따라야 합니다.

호환가능한 Common Language Runtime (CLR)

LabVIEW에서 .NET 객체를 생성 및 조작하려면 .NET CLR 4.0을 사용해야 합니다. 이 버전의 CLR은 LabVIEW와 함께 설치되는 .NET Framework 4.0의 일부에 속합니다.

LabVIEW에서 CLR 4.0 용도:

- 어셈블리 로드하기 — LabVIEW는 .NET CLR 4.0 또는 이전 버전을 대상으로 .NET 어셈블리를 로드할 수 있습니다. 그러나 LabVIEW는 .NET CLR 4.0에서 모든 어셈블리를 로드합니다.
- 독립형 어플리케이션, 공유 라이브러리, 묶음 프로젝트 라이브러리, 소스 배포 및 .NET 기능을 사용하는 VI를 포함하는 압축(ZIP) 파일 만들기 또는 .NET interop 어셈블리 만들기 — LabVIEW에서 이러한 어플리케이션은 .NET CLR 4.0을 대상으로 합니다.

호환가능한 클래스 라이브러리

LabVIEW는 LabVIEW와 함께 설치되는 .NET CLR 4.0과 연관된 클래스 라이브러리 어셈블리에 접근이 가능합니다. .NET Framework의 클래스 라이브러리 어셈블리를 통해 어떤 .NET 기능이 LabVIEW 어플리케이션에서 사용 가능한지 결정됩니다. LabVIEW 프로젝트나 어셈블리에 의존하는 VI에서 **어플리케이션을 빌드**하는 경우, 어플리케이션에 대한 타겟 컴퓨터에는 반드시 .NET Framework 4.0이 설치되어 있어야 합니다.

LabVIEW에서 지원되지 않는 .NET 기능

LabVIEW는 .NET Framework 4.0의 언어 기능의 대부분을 지원합니다. 그러나 .NET Framework 4.0에서는 LabVIEW가 지원하지 않는 몇 가지 새로운 기능이 .NET CLR에 도입됩니다. LabVIEW가 지원하지 않는 .NET 기능의 목록은 ni.com에서 기술지원 데이터베이스를 참조하십시오.

디스크에서 .NET 어셈블리 저장 위치

LabVIEW로부터 호출하려는 .NET 어셈블리는 반드시 LabVIEW가 찾을 수 있는 위치에 저장해야 합니다. 다음 가이드라인에 따라 어디에 어셈블리를 저장할지 결정합니다.



노트

National Instruments는 .NET 객체를 사용하는 VI를 항상 프로젝트에 포함할 것을 적극 권장합니다.

경로 옵션 .NET 어셈블리 로드하기



팁 특정 어셈블리를 로드하거나 어셈블리에서 객체에 접근할 때 문제가 발생하는 경우, [어셈블리 호출을 디버그](#)하거나 메모리의 .NET 어셈블리 대화 상자를 사용하여 메모리의 어셈블리가 올바른 버전인지 확인합니다.

.NET 객체 생성하기

프론트패널 또는 블록다이어그램에 .NET 객체를 생성할 수 있습니다. 컨스트럭터 노드를 사용하여 블록다이어그램에 .NET 객체를 생성하고 프로퍼티, 메소드를 설정하거나 객체의 이벤트를 처리합니다. 예를 들어, 블록다이어그램에 .NET 컨테이너를 놓은 후 타사 브라우저에 접근하는 컨테이너의 내부에 버튼을 놓습니다. .NET 컨테이너를 사용하여 프론트패널에 .NET 컨트롤을 생성합니다. 또한 추후 사용을 위해 .NET 컨트롤을 컨트롤 팔레트에 추가할 수 있습니다.



노트

ActiveX 컨테이너에서와는 달리 .NET 컨테이너에서는 디자인 모드와 실행 모드를 전환할 수 없습니다. .NET 컨테이너의 동작은 디자인 모드의 ActiveX 컨테이너와 유사합니다.

.NET 객체가 사용자에게 보이거나 보이지 않을 수도 있습니다. 예를 들어, 버튼, 윈도우, 그림, 문서, 대화 상자는 보입니다. 어플리케이션은 안 보입니다.

.NET 컨트롤은 자동으로 사용자 인터페이스 실행 시스템을 사용합니다.

.NET 함수를 사용하여 .NET 객체 및 관련 프로퍼티와 메소드에 접근할 수 있습니다.



관련 정보

[블록다이어그램에서 .NET 객체 생성하기](#) (1432 페이지 참조)

[.NET 컨테이너에서 .NET 컨트롤 생성하기](#) (1431 페이지 참조)

[.NET 서브팔레트 생성하기](#) (797 페이지 참조)

[LabVIEW에서 ActiveX 사용하기](#) (351 페이지 참조)

[LabVIEW에서 멀티태스킹](#) (707 페이지 참조)

[.NET 프로퍼티와 메소드](#) (747 페이지 참조)

.NET 프로퍼티와 메소드

.NET 객체는 다른 어플리케이션이 접근할 수 있는 프로퍼티와 메소드를 공개합니다. 어플리케이션과 관련된 객체에 접근하고 해당 객체의 프로퍼티를 설정하거나 메소드를 실행하여 어플리케이션에 접근할 수 있습니다.

프런트패널이나 블록다이어그램 상에서, .NET 객체에서 마우스 오른쪽 버튼을 클릭한 후 생성>>프로퍼티 노드 또는 생성>>인보크 노드를 선택한 후 바로 가기 메뉴에서 프로퍼티나 메소드를 선택하여 객체의 프로퍼티를 설정하거나 메소드를 실행합니다. 또한 .NET 객체에 대해 .NET 특성의 프로퍼티를 선택하거나 .NET 특성의 메소드를 실행할 수 있습니다. 블록다이어그램에서 .NET 객체에서 마우스 오른쪽 버튼을 클릭한 후 생성>>프로퍼티 노드 또는 생성>>인보크 노드를 선택한 후 바로 가기 메뉴에서 .NET 특성의 프로퍼티나 메소드를 선택합니다.



노트

ActiveX처럼 프로퍼티 탐색기에서 .NET 프로퍼티를 보거나 설정할 수는 없습니다. 반드시 프로퍼티 노드를 사용하여 .NET 프로퍼티를 설정해야 합니다.

.NET 이벤트

.NET 이벤트는 마우스 클릭, 키 누름, 메모리 부족이나 작업 완료 등의 알림을 받는 것과 같이 .NET 객체에 행해지는 동작입니다. 이러한 객체에 대한 동작이 발생하면, 객체는 이를 알리기 위해 .NET 컨테이너와 이벤트 특성의 데이터 모두에게 이벤트를 보냅니다. .NET 객체는 객체에 사용 가능한 이벤트를 정의합니다.

어플리케이션에서 .NET 이벤트를 사용하려면 **이벤트를 등록하고 이벤트가 발생하면 이를 처리**해야 합니다. .NET 이벤트 등록은 **다이나믹 이벤트 등록**과 유사합니다. 그러나 .NET 이벤트 VI의 구조는 이벤트 핸들링 VI의 구조와 다릅니다. 일반적인 .NET 이벤트 VI는 다음의 구성요소로 구성됩니다.

- 이벤트를 생성하고자 하는 .NET 객체.
- 생성하려는 이벤트의 타입을 지정하고 등록하기 위한 [이벤트 콜백 등록] 함수. [이벤트 콜백 등록 함수]는 [이벤트 등록] 함수와 비슷하게 여러 이벤트를 처리할 수 있는 크기 조정이 가능한 노드입니다.
- **콜백 VI** (174 페이지 참조) 지정한 이벤트를 핸들링하기 위해 작성한 코드를 포함하는 **콜백 VI**.

.NET 객체를 [이벤트 콜백 등록] 함수에 연결하고 해당 객체에 생성하려는 이벤트를 지정하면 해당 이벤트를 위한 .NET 객체가 등록됩니다. 이벤트 등록이 끝나면, 이벤트를 핸들링하기 위해 작성한 코드를 포함하는 **콜백 VI를 생성**합니다. 서로 다른 이벤트는 다른 이벤트 데이터 포맷을 가지고 있을 수 있으므로, 콜백 VI를 생성한 후 이벤트를 변경하면 블록다이어그램의 와이어가 깨질 수 있습니다. 콜백 VI를 생성하기 전에 이벤트를 선택하십시오.

컨테이너 안의 .NET 컨트롤에서 이벤트를 처리할 수 있습니다. 예를 들어, .NET 컨테이너에 달력 컨트롤을 놓은 후 달력에 디스플레이되는 아이템에 대해 더블 클릭 이벤트를 처리하도록 지정할 수 있습니다.

.NET 데이터 타입

LabVIEW가 데이터를 읽고 해석할 수 있도록 LabVIEW는 .NET 프로퍼티, 메소드, 컨스트럭터 파라미터의 데이터 타입을 LabVIEW 데이터 타입으로 변환합니다. LabVIEW는 변환할 수 없는 데이터 타입을 .NET 참조 번호로 디스플레이합니다. 다음 테이블은 .NET 데이터 타입과 이에 대응하는 LabVIEW 데이터 타입을 나열합니다.

.NET 타입	LabVIEW 타입
System.Int64, System.UInt64, System.Int32, System.UInt32, System.Int16, System.UInt16	 ,  ,  ,  ,  , 
System.String	
System.Boolean	
System.Byte, System.Char, System.UByte	
System.Single, System.Double, System.Decimal	
System.Array	대응하는 타입의 배열로 디스플레이됩니다.
Enumeration	대응하는 타입의 정수를 가진 링 컨트롤, 상수 또는 인디케이터로 표시됩니다.
DateTime	
기타 .NET 객체	

.NET 클라이언트 어플리케이션 설정하기

LabVIEW에서 .NET 클라이언트 어플리케이션은 .NET 어셈블리에서 정의된 객체를 생성하고 조작하는 LabVIEW 프로젝트 및 VI입니다. 어플리케이션이 .NET 객체를 사용할 때 LabVIEW는 **기본 동작** 세트를 사용하여 어떤 버전의 특정 .NET 어셈블리를 참조할 것인지 그리고 어떤 버전의 Common Language Runtime (CLR)으로 해당 어셈블리를 로드할 것인지 결정합니다. 그러나 .NET 설정 파일을 생성하여 특정 프로젝트 또는 모든 프로젝트에 대해 LabVIEW가 .NET 객체를 로드하고 .NET 객체와 상호작용하는 기본 방식을 다시 정의할 수 있습니다. 설정 파일을 사용해야 하는 작업은 다음과 같습니다:

- [어셈블리 호출 디버그하기](#)
- [.NET 2.0 어셈블리 로드하기](#)
- [특정 버전의 어셈블리 로드하기](#)
- [원격 .NET 접근 설정하기](#)
- [로깅 옵션 설정하기](#)
- [보안 환경 설정하기](#)

.NET 구성 파일에 대한 요건

.NET 설정 파일은 다음 요건을 충족해야 합니다:

- 특정 LabVIEW 프로젝트, 공유 라이브러리 또는 독립 어플리케이션에 대응
- 파일 이름이 설정하는 파일과 일치하고 여기에 덧붙여 .config 확장자를 포함. 예를 들어, MyApp.lvproj에 대한 설정 파일은 이름이 MyApp.lvproj.config, MyApp.exe에 대한 설정 파일은 이름이 MyApp.exe.config, 이런 식으로 파일 이름이 정해져야 합니다.
- 설정하는 어플리케이션과 같은 디렉토리에 위치



노트

National Instruments는 **LabVIEW 프로젝트**에 속한 VI에 대해서만 .NET 객체를 사용할 것을 적극 권장합니다. 프로젝트에 속하지 않은 VI는 LabVIEW.exe와 같은 디렉토리에 위치한 LabVIEW.exe.config 파일만 참조할 수 있습니다.

- 설정 목적에 맞는 특정 XML 내용을 포함합니다. 원격 .NET 접근 설정, 로깅 옵션 설정, 보안 설정을 위한 설정 파일 생성에 관한 정보는 MSDN(Microsoft Developer Network)을 참조하십시오.

설정된 프로젝트에서 어플리케이션 만들기

.NET 설정 파일을 사용하는 LabVIEW 프로젝트에서 독립 어플리케이션을 만들 때 만든 어플리케이션도 설정 파일의 복사본을 필요로 합니다. **설정 파일 요건**을 충족하려면 독립 어플리케이션에 대한 설정 파일이 어플리케이션의 이름을 반영해야 하고 2개의 파일이 같은 디렉토리에 위치해야 합니다.



노트 설정 파일의 배포에 관한 자세한 정보는 **.NET 클라이언트 어플리케이션으로 .NET 어셈블리 배포하기**를 참조하십시오.

LabVIEW에서 .NET 어셈블리 사용하기

어셈블리는 DLL, OCX, 또는 COM의 구성요소 실행 파일 등과 유사한 배포 단위입니다. 어셈블리는 .NET 컴파일러를 사용하여 만든 DLL과 실행 파일입니다. 어셈블리는 단일 파일 또는 여러 파일로 구성될 수 있습니다. 어셈블리에는 어셈블리 이름, 버전, 작성자의 보안, 어셈블리를 구성하는 파일, 의존적인 어셈블리, 리소스, 반출된 데이터 타입에 대한 정보를 담은 목록이 포함됩니다. 단일 파일 어셈블리는 목록과 필요한 모든 리소스를 포함한 전체 데이터를 단일 파일에 가지고 있습니다. 여러 파일 어셈블리는 비트맵, 아이콘, 사운드 파일 등과 같은 외부 리소스를 가지고 있거나, 핵심 코드를 위한 파일 하나와 도움 라이브러리를 위한 파일 하나를 가질 수 있습니다.

어셈블리는 퍼블릭 또는 프라이빗이 될 수 있습니다. 프라이빗 어셈블리는 일반적으로 어플리케이션과 동일한 디렉토리에 위치합니다. 퍼블릭 어셈블리는 Global Assembly Cache(GAC)에 있습니다. 일반적으로 어플리케이션 개발자는 그 어플리케이션이 사용하는 프라이빗 어셈블리를 생성합니다. 또한, 어셈블리 개발자는 어셈블리의 버전을 결정합니다. 어셈블리 이름은 목록을 포함하는 파일의 파일명에서 확장자를 제외한 부분입니다.

Common Language Runtime(CLR)은 **설정 셋팅**을 사용하여 어플리케이션에 로드할 어셈블리의 버전을 결정합니다. 이 설정 셋팅은 사용자의 요청하는 특정 어셈블리 버전보다 우선될 수 있습니다. 예를 들어, 시스템 관리자가 사용자 컴퓨터를 설정하여 버전 1.0.0.0 대신 1.0.0.1를 로드하도록 할 수 있습니다. 그 후 사용자가 버전 1.0.0.0을 로드하려고 하면, CLR는 어셈블리 버전을 1.0.0.1로 업그레이드합니다. LabVIEW는 또한 버전이 상향 조정되었음을 알려줍니다.

.NET 어셈블리 로드하기

VI의 프런트패널 또는 블록 다이어그램에서 .NET 객체를 참조하는 경우, LabVIEW가 해당 객체에 대한 .NET 어셈블리를 로드할 수 있는지 확인합니다. Common Language Runtime(CLR)은 사용자가 호출하는 .NET 어셈블리를 찾습니다. CLR가 어셈블리의 위치를 찾는 방법에 대한 추가적인 정보는 MSDN(Microsoft Developer Network) 웹 사이트를 참조하십시오. CLR이 어셈블리를 찾지 못하는 경우, LabVIEW는 찾을 수 없는 VI를 검색하는 방법과 같은 방식으로 어셈블리를 검색합니다. LabVIEW 옵션 대화 상자의 경로 페이지에서 지정한 디렉토리 안에서 찾을 수 없는 VI를 검색합니다. LabVIEW는 프런트패널 또는 블록 다이어그램에서 직접 참조한 .NET 객체의 .NET 어셈블리를 못 찾을 경우, 로드 타임 에러를 발생시킵니다. LabVIEW가 런타임에서 필요한 의존성 어셈블리를 로드하지 못하는 경우, 런타임 에러를 발생시킵니다.

CLR은 프라이빗 .NET 어셈블리를 로드할 때 동작 중인 실행 파일의 디렉토리를 기본 검색 경로로 사용합니다. LabVIEW 프로젝트에 속하지 않는 VI로부터 .NET 객체를 참조하는 경우, CLR은 LabVIEW.exe를 동작 중인 실행 파일로 인식합니다. 따라서 CLR은 LabVIEW.exe 파일이 위치하는 디렉토리에서 프라이빗 어셈블리를 검색합니다. LabVIEW 프로젝트에 속하는 VI로부터 .NET 객체를 참조하는 경우, CLR은 프로젝트를 동작 중인 실행 파일로 인식합니다. 따라서 CLR은 프로젝트 디렉토리에서 프라이빗 어셈블리를 검색합니다. VI로부터 .NET 어셈블리를 참조하고 해당 어셈블리가 .NET Framework에 속하지 않는 경우, LabVIEW.exe 파일이 위치한 디렉토리에 파일을 놓지 않도록 VI를 프로젝트에 저장할 것을 National Instruments는 권장합니다.

프로젝트에 속하지 않는 VI로부터 .NET 어셈블리를 호출하는 경우, 호출하는 VI와 같은 디렉토리에 어셈블리를 저장할 수 있습니다. LabVIEW는 CLR가 기본적으로 로드할 수 없는 어셈블리를 호출하는 VI 디렉토리를 포함하여 특정 VI 디렉토리에서 검색합니다. 하지만 호출 어셈블리를 이 위치에 저장하면 이름이 충돌하고 그 외 기대하지 않은 .NET 동작을 야기할 수 있습니다. 따라서 National Instruments는 이 위치에 어셈블리를 저장하지 않는 것을 권장합니다.

메모리에 있는 어셈블리 변경 감지하기

LabVIEW가 어셈블리를 메모리에 로드한 후, 어셈블리를 로드한 **어플리케이션 인스턴스**를 사용자가 닫을 때까지 해당 어셈블리는 메모리에 상주합니다. 어셈블리가 메모리에 있는 동안, 디스크에서 어셈블리에 변경이 있어도 LabVIEW는 감지하지 못합니다. 그러므로 LabVIEW가 어셈블리의 변경에 접근할 수 있기 전에, **메모리에 있는 어셈블리 버전을 업데이트**해야 합니다.

어셈블리 변경 후 VI 로드하기

.NET Framework SDK에서 제공되는 Microsoft Visual Studio .NET 및 기타 개발 도구는 어셈블리에 강력한 이름을 지정할 수 있습니다. 같은 강력한 이름을 가진 어셈블리들은 동일한 것으로 여겨집니다.

.NET 어셈블리의 경로가 변경되었거나 강력한 이름의 어셈블리의 버전 번호 또는 컬처 문자열이 변경된 VI를 로드할 때, LabVIEW는 이를 알리는 경고 대화 상자를 실행합니다. 일단 로드되면, VI는 제목 표시줄과 윈도우 메뉴에 디스플레이된 열린 VI 리스트에 별표를 포함합니다. VI를 저장하면, 새로 변경할 때까지 별표는 사라집니다.

.NET 어셈블리의 타임스탬프에 변경사항이 있는 VI를 로드할 때 LabVIEW는 경고 대화 상자를 실행하지 않지만 해당 VI의 제목 표시줄에 별표를 디스플레이합니다.



팁

메모리의 .NET 어셈블리 대화 상자를 참조하여 LabVIEW가 올바른 버전의 어셈블리를 로드하는지 확인합니다.

특정한 버전의 .NET 어셈블리를 선택하고 로드하는 방법에 대한 추가적인 정보는 ni.com에서 기술지원 데이터베이스를 참조하십시오.

.NET 클라이언트 어플리케이션으로 .NET 어셈블리 배포하기

LabVIEW에서 .NET 클라이언트 어플리케이션을 생성한 후, 독립 어플리케이션, 공유 라이브러리, 묶음 프로젝트 라이브러리, 소스 배포, 압축(ZIP) 파일 또는 .NET interop 어셈블리로 **해당 프로젝트나 VI를 빌드**할 수 있습니다. 그런 다음, 어플리케이션에서 .NET 객체의 특성과 동작을 정의하는 관련 어셈블리 및 설정 파일을 제공하는 한 어플리케이션을 다른 컴퓨터로 배포할 수 있습니다.



노트

National Instruments는 배포된 어플리케이션에 대한 타겟 컴퓨터의 버전이 LabVIEW가 어플리케이션을 만드는데 사용한 .NET Framework 설치 버전과 같거나 이후 버전이 되도록 권장합니다. 타겟 컴퓨터의 .NET Framework 요구조건에 대한 추가 정보는 [호환가능한 클래스 라이브러리](#)를 참조하십시오.

배포할 관련 파일

.NET 클라이언트 어플리케이션에서 만든 어플리케이션과 함께 다음 파일을 배포해야 합니다.



노트 배포한 어플리케이션에 대해 관련 파일을 포함하는 **설치 프로그램을 만들 수** 있습니다.

- 어플리케이션의 data 디렉토리—어플리케이션을 생성하면 LabVIEW는 Global Assembly Cache(GAC)에 없는 모든 필수 .NET 어셈블리를 이 디렉토리로 자동 복사합니다.
- GAC에 있는 프라이빗 .NET 어셈블리—이 어셈블리는 LabVIEW가 data 디렉토리로 복사하지 않으므로 사용자 자신이 배포해야 합니다.
- **.NET 설정 파일**—프로젝트나 VI가 .NET 설정 파일에 의존하는 경우, 해당 프로젝트나 VI에서 만드는 모든 어플리케이션은 유사한 설정 파일을 필요로 합니다. 다음과 같은 상황에서는 설정 파일을 배포할 필요가 없습니다:

- 원본 프로젝트 또는 VI의 경우 설정 파일이 필요하지 않습니다.
- .NET interop 어셈블리를 만듭니다.

타겟 컴퓨터에서 배포된 파일을 저장하는 위치

배포된 .NET 어플리케이션이 타겟 컴퓨터에서 실행되려면 관련 파일이 타겟 컴퓨터의 다음 위치에 저장되어야 합니다.

- data 디렉토리—어플리케이션과 같은 디렉토리에 이 디렉토리를 저장합니다.
- GAC의 프라이빗 .NET 어셈블리—어셈블리를 GAC나 CLR이 검색하는 다른 위치에 저장합니다.
- .NET 설정 파일—어떤 타입의 어플리케이션을 배포하려는지에 따라 설정 파일의 위치가 달라집니다.
 - 독립 어플리케이션—어플리케이션과 같은 디렉토리에 설정 파일을 저장합니다.
 - 공유 라이브러리, 묶음 프로젝트 라이브러리, 소스 배포 또는 압축(ZIP) 파일—공유 라이브러리, 묶음 프로젝트 라이브러리, 소스 배포 또는 압축(ZIP) 파일을 사용하는 어플리케이션의 디렉토리에 설정 파일을 저장합니다.

LabVIEW의 터치 패널

Creating a Project Using the Touch Panel Project Template

Complete the following steps to use the Touch Panel Project template to create a new project.

1. Select File»Create Project to display the Create Project dialog box.
2. Search for the Touch Panel Project template and click Next.
3. On the Configure the new project page, complete the following steps to configure details of the project.
 1. Specify the name of the .lvproj file and directory to save the project in Project Name and Project Root, respectively.
 2. (Optional) In the Modify VI Icons section, edit the appearance of VI icons.
 3. Click Browse to open the Add Targets and Devices dialog box. Select the touch panel target to add to the project and click OK.
 4. Select a **touch panel VI template** from the VI template pull-down menu.
4. Click Finish to save the project and open the Project Explorer window.

The project appears in the Project Explorer window. Double-click the VI under the target to open the front panel and block diagram windows. Refer to the comments on the block diagrams of VIs in the project for more information about how to modify the project. Also refer to the Project Documentation folder in the Project Explorer window for more information about how to modify the project.

Navigation Controls

Use the navigation controls located on the System and Modern palettes to navigate through multiple front panels in an application. Navigation controls include the Previous Button, Next Button, and Jump Button controls.

Previous Button and Next Button Navigation Controls

The Previous Button and Next Button controls navigate through and display the front panels of VIs in a project library. The controls navigate through the VIs in a project library based on the order the VIs appear in the project library. If you want to change the order of the VIs, you must [change the order in the project library](#).



NOTE

You must [reference a VI in a LabVIEW project library](#) to use the Previous Button and Next Button controls. When you add a Previous Button or Next Button control to the front panel of a VI that is not referenced by a LabVIEW project library file, the control displays an error glyph.

When a user selects the Next Button control for the last VI in a project library, the Next Button control displays the front panel for the first VI listed in the project library. You only can access VIs in the same project library with these controls.



Tip Select File»New to display the New dialog box, and select Other Files»Multi-panel Application to display a template that uses the Previous Button and Next Button controls.

You can set the string text on the Previous Button and Next Button programmatically by using the Button Text property. Right-click a navigation control from the front panel, select Create»Property Node, and select the Button Text property from the shortcut menu. LabVIEW creates the Property Node on the block diagram. The Property Node links implicitly to the navigation control.

Jump Button Navigation Control

You can use the Jump Button control to navigate to a specific VI. When you add a Jump Button control to the front panel, the Configure Jump Button dialog box appears. Use this dialog box to select the target VI and the text to display on the button. You can specify any VI on disk as the target VI, even if the VI is not referenced by a project library. You may want to use a Jump Button control if you have a VI that functions as the main panel in an application or references multiple project libraries.



NOTE

Use the Button Text property and Target VI property to set the string text and target VI, respectively, on the Jump Button programmatically.

Protecting Data Using Write Filters

Write filters protect data from unwanted modifications by redirecting write operations to another location or overlay. Use the Enhanced Write Filter (EWF) VIs to protect volumes and redirect writes to a disk location on another volume or to the RAM. Use the File-Based Write Filter (FBWF) VIs to protect files and folders in a volume and redirect writes to the memory cache. The Write Filters VIs are available for VIs under a **Touch Panel target**. Refer to the MSDN Web site at msdn.microsoft.com to learn more about write filters.

LabVIEW supports the Write Filters VIs, which you use to enable, disable, and configure write filters, for touch panel devices. You must target a touch panel device to use the Write Filters VIs.

Supported Touch Panel Targets

LabVIEW supports the following touch panel targets:

 NOTE

The LabVIEW Run-Time Engine is required to run touch panel applications.

Target	Processor	Use
NI TPC-2215	x86	Use with NI TPC-2215 hardware
NI TPC-2212	x86	Use with NI TPC-2212 hardware
NI TPC-2206	x86	Use with NI TPC-2206 hardware
Windows Embedded Standard Device	x86	Use with third-party touch panel computers running Windows Embedded Standard



NOTE

- LabVIEW supports Windows Embedded Standard 7 only.
- Windows Embedded Standard is a componentized version of Windows, which allows you to customize the operating system image. National Instruments tests and certifies LabVIEW to work with the Windows Embedded Standard OS image that ships with National Instruments touch panel computers. If you use third-party touch panel hardware with Windows Embedded Standard and you do not install the correct components, the LabVIEW application may not function properly. Please contact National Instruments technical support at ni.com/support if you experience unexpected behavior executing a LabVIEW touch panel application with a third-party Windows Embedded Standard touch panel computer. However, National Instruments does not officially support compatibility issues with third-party Windows Embedded Standard touch panel computers.
- National Instruments touch panel computers come with an English version of the Windows Embedded Standard operating system. The English version supports VI front panels that use Western European languages, including English, French, and German. For support in other languages, contact National Instruments technical support at ni.com/support.

Using Touch Panel VI Templates

Use a touch panel VI template as a starting point for developing touch panel VIs. LabVIEW includes portrait and landscape templates whose user interfaces are preset to the touch panel device. These templates include controls and indicators that you commonly use in touch panel applications.

You can add a touch panel VI template by right-clicking a touch panel target in the Project Explorer window and selecting New VI Template. The Select a template dialog box appears. Select the VI template to use with the target.

LabVIEW 사용법

LabVIEW 사용법

이 모음에서는 LabVIEW로 다양한 태스크를 수행하기 위한 지침을 제공합니다.

LabVIEW 설정하기

LabVIEW 평가판 및 정품인증 정보 (Windows)

LabVIEW 플랫폼 미디어를 사용하여 LabVIEW, LabVIEW 모듈 및 툴킷을 설치하고, 설치한 제품을 정해진 기간 동안 무료로 사용해 볼 수 있습니다. 이 토픽에서는 LabVIEW, 모듈 또

는 툴킷의 평가판을 사용할 때 예상되는 부분과 평가판을 사용한 후 제품을 정품인증하는 방법을 설명합니다.

LabVIEW, 모듈, 툴킷 평가하기

무료 평가판 모드를 통해 LabVIEW Professional Development System에서 사용가능한 모든 도구, VI 및 함수, 도움말, 예제와 대부분의 모듈 또는 툴킷에 접근할 수 있습니다. 평가 기간이 만료되면, 청품 인증을 받아야 계속해서 LabVIEW를 사용할 수 있습니다.

LabVIEW 평가판 사용하기

LabVIEW를 평가판 모드에서 실행하는 경우, LabVIEW에는 다음과 같은 제한사항이 있습니다:

- 평가판 버전이 만료될 때까지 시간 제한—정품인증되지 않은 동안, LabVIEW는 매번 제품을 시작할 때마다 정품인증하도록 입력요청합니다. 계속해서 LabVIEW를 평가판 모드에서 실행할 수 있습니다.
- 시간 제한동안 평가 버전 표시 워터마크—모든 사용자 VI와 컨트롤에 평가판 워터마크가 있으며 비정품인증 모드에서 생성한 모든 VI(VI의 서브패널에서의 SubVI 제외)에는 모든 디스플레이된 프론트패널에 워터마크가 포함되어 있습니다.

LabVIEW의 평가판 기간이 만료된 후에는, 더이상 LabVIEW를 사용할 수 없습니다.

모듈과 툴킷 평가판 사용하기

모듈 또는 툴킷의 평가 기간이 만료될 때, 더이상 수행할 수 없는 작업은 다음과 같습니다:

- 모듈 또는 툴킷에서 VI를 실행합니다.
- 모듈 또는 툴킷의 VI를 사용하는 VI를 실행합니다.
- 모듈 또는 툴킷의 VI를 사용하는 VI를 실행 파일 또는 공유 라이브러리에 빌드합니다.
- 모듈 또는 툴킷에 의해 설치된 메뉴, 도구, 팔레트에 접근합니다.

평가판으로 모듈이나 툴킷을 설치할 때, 해당 모듈 또는 툴킷의 도움말이 설치됩니다. 설치된 모듈 또는 툴킷의 도움말 파일은 평가판 기간 이후에도 남아있게 됩니다. 또한, NI 예제 탐색기는 평가판 기간 이후에도 모든 모듈 또는 툴킷 예제를 디스플레이합니다. 해당 도움말과 예제를 지우려면 모듈 또는 툴킷을 제거해야 합니다.

LabVIEW를 평가판 모드에서 실행하는 경우, Professional Development System의 기능이 제공됩니다. 이후에 LabVIEW 라이선스를 구입하고 정품인증하는 경우, 구입한 개발 시스템의 모든 기능을 완전히 사용할 수 있습니다. 예를 들어, Professional Development System에서만 사용가능한 기능을 사용하여 VI를 생성한 후에 Full Development System을 구입하고 정품인증한 경우, Professional Development System을 사용하여 생성한 VI는 깨집니다.

평가판 사용 후 LabVIEW, 모듈 또는 툴킷 정품인증하기

LabVIEW 라이선스를 정품인증하려면, 소프트웨어 키트에 포함된 소유권 인증서에 있는 시리얼 번호를 사용하고 정품인증 방법을 따릅니다.

모듈 또는 툴킷과 같은 애드온 소프트웨어를 구매하는 경우, NI 라이선스 관리자를 사용하여 소프트웨어를 정품인증합니다. NI 라이선스 관리자에서 소프트웨어가 보이지 않으면 LabVIEW 플랫폼 미디어에서 이 소프트웨어를 설치할 수 있습니다.

LabVIEW를 설치할 때 LabVIEW가 어플리케이션 빌더를 설치합니다. LabVIEW Professional Development System를 정품인증하는 경우, 자동으로 어플리케이션 빌더가 정품인증됩니다. LabVIEW Base Development System 또는 Full Development System을 정품인증하는 경우, 별도로 어플리케이션 빌더를 구매하고 정품인증해야 합니다.

평가판을 유료 버전으로 업그레이드하기

평가판을 유료 버전으로 업그레이드하는 경우, 소프트웨어를 다시 설치할 필요가 없습니다. 어플리케이션을 시작할 때 나타나는 입력요청을 따르거나, NI 라이선스 관리자를 사용하여 소프트웨어를 정품인증합니다.

유료 버전 사이에서 이동하기

LabVIEW Base Development System을 LabVIEW Professional Development System으로 업그레이드하는 것과 같이 차후에 다른 제품 패키지로 바꾸려는 경우, 추가적인 소프트웨어를 설치하지 않고도 새 패키지를 정품인증할 수 있습니다. NI 라이선스 관리자를 사용하여 소프트웨어를 정품인증합니다.

LabVIEW 사용자 이름 설정하기

다음 단계에 따라 LabVIEW 사용자 이름을 설정하거나 변경합니다.

1. 도구 >> 사용자 이름을 선택하여 사용자 로그인 대화 상자를 디스플레이합니다.
2. 기본적으로, OS 사용자 이름이 선택됩니다. LabVIEW 사용자 이름을 변경하려면, 풀다운 메뉴에서 이름을 선택합니다. 또한 풀다운 메뉴에서 기타를 선택하고 사용자 이름 텍스트 박스에서 새 사용자 이름을 입력해도 됩니다.
3. 확인 버튼을 클릭합니다.

LabVIEW에서 로그인하기 및 로그아웃하기 (Windows)

다음 단계를 따라 LabVIEW에 로그인합니다.

1. 도구 >> 보안 >> 로그인을 선택하여 NI 보안 로그인 대화 상자를 디스플레이합니다.
2. 사용자 이름 텍스트 박스에 사용자 계정 이름을 입력합니다.
3. 암호 텍스트 박스에 계정 암호를 입력합니다.

4. NI 도메인 리스트박스에서 사용자 계정이 정의된 보안 도메인을 선택합니다.



노트
만약 선택할 수 있는 보안 도메인이 없다면 도메인 계정 관리자를 사용해서 하나를 정의할 수 있습니다.

5. 확인을 클릭하여 로그인합니다.

LabVIEW에서 로그아웃하려면, 도구 » 보안 » 로그아웃을 선택합니다.



노트 아무도 로그인하지 않은 경우, LabVIEW는 기본 nobody 계정을 사용해서 사용자의 접근 권한을 정의합니다. nobody 계정에 지정된 암호는 없습니다.

키보드 바로 가기 키

다음 테이블은 LabVIEW 환경에서 키보드 바로 가기를 나열합니다. 또한, VI 메뉴 아이템의 사용자 바로 가기 메뉴를 생성할 수 있습니다.



노트
이 바로 가기 키에서 <Ctrl> 키는 (macOS) <Option> 또는 <Command> 키 또는 (Linux) <Alt> 키에 상응합니다.

키보드 바로 가기 키	설명
객체/이동	
<Shift>-클릭	여러 개의 객체를 선택하고; 현재 선택에 객체를 추가합니다.
화살표 키	선택된 객체를 한 번에 한 픽셀씩 이동합니다.
<Shift>-화살표 키	선택된 객체를 한 번에 여러 픽셀씩 이동합니다.
<Shift>-클릭(끌기)	선택된 객체를 한 축에 따라 이동합니다.
<Ctrl>+K	다층으로 쌓여있는 객체에서 선택한 객체를 한 위치 높은 곳으로 이동하여 객체의 순서를 재설정합니다.
<Ctrl>+J	다층으로 쌓여있는 객체에서 선택한 객체를 한 위치 낮은 곳으로 이동하여 객체의 순서를 재설정합니다.
<Ctrl>+<Shift>+K	다층으로 쌓여있는 객체에서 선택한 객체를 가장 높은 곳으로 이동하여 객체의 순서를 재설정합니다.
<Ctrl>+<Shift>+J	다층으로 쌓여있는 객체에서 선택한 객체를 뒤로 이동하여 객체의 순서를 재설정합니다.
<Ctrl>-클릭(끌기)	선택한 객체를 복사합니다.
<Ctrl>-<Shift>-클릭(끌기)	선택한 객체를 복사하여 축에 따라 이동합니다.

키보드 바로 가기 키	설명
<Shift>-크기 조정	외형의 비율을 유지하면서 선택된 객체의 크기를 조정 합니다.
<Ctrl>-크기 조정	중심점을 유지하면서 선택된 객체의 크기를 조정 합니다.
<Ctrl>-<Shift>-크기 조정	중심점과 외형의 비율을 유지하면서 선택된 객체의 크기를 조정 합니다.
빈 공간 쪽으로 <Ctrl> 키 끌기	프론트패널이나 블록다이어그램에 보다 많은 작업 공간 을 추가합니다.
빈 공간 쪽으로 <Ctrl+Alt> 키 끌기	프론트패널이나 블록다이어그램에서 필요없는 여분의 작업 공간 을 줄입니다.
<Ctrl>-A	모든 프론트패널 또는 블록다이어그램 항목을 선택합니다.
<Ctrl>-<Shift>-A	마지막 객체 정렬 작업을 수행합니다.
<Ctrl>-D	마지막 객체 간격 조절 을 수행합니다.
빈 영역에서 더블 클릭	자동 도구 선택이 활성화되어 있을 경우, 프론트패널 또는 블록다이어그램에 독립 라벨 을 추가합니다.
<Ctrl>-마우스 휠	케이스, 이벤트, 시퀀스 구조의 서브다이어그램을 스크롤 합니다.
<Spacebar>	선택 사각형을 생성 할 때 여러 선택 모드 사이에서 전환합니다.
스페이스 바(끌기)	미리 지정된 정렬 위치 는 객체 라벨 또는 캡션을 옮길 때 비활성화됩니다.
<Ctrl>-U	블록다이어그램에서 자동으로 기존 와이어를 모두 재정리하고 객체를 재배치합니다.
LabVIEW 환경 탐색하기	
<Ctrl>-F	객체나 텍스트를 찾습니다.
<Ctrl>-<Shift>-F	이전 검색 결과와 함께 검색 결과 윈도우나 프로젝트 항목 찾기 대화 상자를 디스플레이합니다.
<Ctrl>-G	객체나 텍스트의 다음 인스턴스의 VI를 검색합니다.
<Ctrl>-<Shift>-G	VI, 객체, 또는 텍스트의 이전 인스턴스의 VI를 검색합니다.
<Ctrl>-<Tab>	윈도우가 화면에 나타나도록 LabVIEW 윈도우를 돌아가면서 띄웁니다. (Linux) 윈도우의 순서는 사용하는 윈도우 관리자에 따라 다릅니다.
<Ctrl>-<Shift>-<Tab>	LabVIEW 윈도우들을 반대 방향으로 돌아가면서 띄웁니다.
<Ctrl>-L	에러 리스트 윈도우를 디스플레이합니다.
<Ctrl>-<Shift>-E	프로젝트 탐색기 윈도우에 현재 VI를 디스플레이합니다.

키보드 바로 가기 키	설명
<Ctrl>-<Shift>-W	모든 윈도우 대화 상자를 디스플레이합니다.
<Ctrl>-<Shift>-B	클래스 탐색기 윈도우를 디스플레이합니다.
프런트패널과 블록다이어그램 탐색하기	
<Ctrl>-E	블록다이어그램이나 프런트패널 윈도우를 디스플레이합니다.
<Ctrl>-<Space>	빠른 탐색 대화 상자를 디스플레이합니다. 중국어 키보드에서는, <Ctrl-Shift-Space> 키를 누릅니다. (macOS) <Command-Shift-Space> 키를 누릅니다.
<Ctrl>-#	눈금 정렬을 실행하거나 실행 중지합니다. 프랑스어 키보드에서는, <Ctrl-"]> 키를 누르십시오. (macOS) <Command-*> 키를 누릅니다.
<Ctrl>-/_	윈도우를 최대로 키우거나 다시 줄입니다.
<Ctrl>-T	프런트패널과 블록다이어그램 윈도우를 나란히 세웁니다.
<Ctrl>-<Shift>-N	탐색 윈도우를 디스플레이합니다.
<Ctrl>-I	VI 프로퍼티 대화 상자를 디스플레이합니다.
<Ctrl>-Y	히스토리 윈도우를 디스플레이합니다.
VI 계층구조 윈도우 탐색하기	
<Ctrl>-D	VI 계층구조 윈도우를 다시 그립니다.
<Ctrl>-A	VI 계층구조 윈도우에 있는 모든 VI를 보여줍니다.
<Ctrl>-VI 클릭	VI 계층구조에서 선택한 VI를 구성하는 subVI와 기타 노드를 디스플레이합니다.
<Enter>	VI 계층구조 윈도우에 입력하여 검색을 시작한 후 검색 문자열과 일치하는 다음 노드를 찾습니다.
<Shift>-<Enter>	VI 계층구조 윈도우에 입력하여 검색을 시작한 후 검색 문자열과 일치하는 이전 노드를 찾습니다.
디버깅	
<Ctrl>-아래쪽 방향 화살표	노드에서 단계별 실행을 들어갑니다.
<Ctrl>-오른쪽 방향 화살표	노드에서 단계별 실행을 건너웁니다.
<Ctrl>-위쪽 방향 화살표	노드에서 단계별 실행을 나갑니다.
파일 작업	

키보드 바로 가기 키	설명
<Ctrl>-N	새 VI를 엽니다.
<Ctrl>-O	기존의 VI를 엽니다.
<Ctrl>-W	VI를 닫습니다.
<Ctrl>-S	VI를 저장합니다.
<Ctrl>-<Shift>-A	파일을 열고 닫습니다.
<Ctrl>-P	윈도우를 프린트합니다.
<Ctrl>-Q	LabVIEW를 종료합니다.
기본 편집	
<Ctrl>-Z	최근 작업을 취소합니다.
<Ctrl>-<Shift>-Z	최근 작업을 다시 실행합니다.
<Ctrl>-X	선택한 객체를 잘라내기합니다.
<Ctrl>-C	선택한 객체를 복사합니다.
<Ctrl>-V	최근에 잘라내기하거나 복사한 객체를 붙여넣기합니다.
도움말	
<Ctrl>-H	기본 도움말 윈도우를 디스플레이합니다. (macOS) <Command-Shift-H> 키를 누릅니다.
<Ctrl>-<Shift>-L	기본 도움말 윈도우를 고정해 놓습니다.
<Ctrl>-? 또는 <F1>	LabVIEW 도움말을 디스플레이합니다.
도구 및 팔레트	
<Ctrl>	다음으로 가장 많이 사용되는 도구로 바꿉니다.
<Shift>	위치 도구로 바꿉니다.
빈 공간에서 <Ctrl>-<Shift>	스크롤 도구로 바꿉니다.
<Spacebar>	자동 도구 선택이 비활성화되어 있으면 2개의 가장 잘 사용하는 도구 사이에서 토글합니다.
<Shift>-<Tab>	자동 도구 선택을 활성화합니다.
<Tab>	자동 도구 선택 버튼을 클릭하여 자동 도구 선택을 비활성화한 경우 가장 일반적인 4가지 도구를 돌아가면서 선택합니다. 그렇지 않으면, 자동 도구 선택을 사용하십시오.
화살표 키	임시 컨트롤과 함수 팔레트를 탐색합니다.
<Enter>	임시 팔레트로 들어갑니다.

키보드 바로 가기 키	설명
<Esc>	임시 팔레트에서 빠져나옵니다.
<Shift>-마우스 오른쪽 버튼 클릭	커서의 위치에서 도구 팔레트의 임시 버전을 디스플레이합니다.
SubVI	
SubVI 더블 클릭	SubVI의 프론트패널을 디스플레이합니다.
<Ctrl>-SubVI 더블 클릭	SubVI의 블록다이어그램과 프론트패널을 디스플레이합니다.
블록다이어그램으로 VI 아이콘을 끌어서 놓기	블록다이어그램의 subVI에서 그 VI를 사용 합니다.
<Shift>-블록다이어그램으로 VI 아이콘 끌기	블록다이어그램에 subVI로 그 VI를 놓고 기본값이 없는 컨트롤에 연결된 상수와 함께 사용합니다.
<Ctrl>-블록다이어그램에서 마우스 오른쪽 버튼을 클릭하고 팔레트에서 VI 선택	해당 VI의 프론트패널을 엽니다.
실행	
<Ctrl>-R	VI를 실행합니다.
<Ctrl>-.	VI가 실행되고 있는 동안 사용될 때 VI를 정지합니다.
<Ctrl>-M	실행 모드나 편집 모드로 변경합니다.
<Ctrl>-아래쪽 방향 화살표	VI가 실행되는 동안 사용될 때, 배열이나 클러스터 안으로 키 포커스를 움직 입니다.
<Ctrl>-위쪽 방향 화살표	VI가 실행되는 동안 사용될 때, 배열이나 클러스터 밖으로 키 포커스를 움직 입니다.
탭	VI가 실행되는 동안 사용될 때, 탭 순서 에 따라 컨트롤이나 인디케이터를 탐색합니다.
<Shift>-<Tab>	VI가 실행되는 동안 사용될 때, 컨트롤이나 인디케이터를 역순으로 탐색합니다.
와이어링	
<Ctrl>-B	VI의 모든 깨진 와이어 를 제거합니다. 깨진 와이어가 있는 구조 또는 블록다이어그램의 영역을 선택하는 경우, 이 바로 가기 키는 선택한 영역에서만 깨진 와이어를 삭제합니다.
<Esc>, 마우스 오른쪽 버튼 클릭, 또는 터미널 클릭	와이어 연결하는 동안, 진행 중인 와이어링을 취소합니다.
와이어 한번 클릭	한 와이어 선분을 선택합니다.

키보드 바로 가기 키	설명
와이어 더블 클릭	와이어 결가지를 선택합니다.
와이어 트리플 클릭	와이어 전체를 선택합니다.
A	연결하는 동안 일시적으로 자동 와이어 경로 설정을 비활성화합니다.
W	연결하는 동안, 구조에 들어오거나 나가는 객체가 구조 안의 객체와 연결되어 있는 경우, 이 키를 사용하여 자동 와이어 연결을 토글합니다.
더블 클릭	와이어를 연결하는 동안, 연결하지 않고 와이어를 고정합니다.
<Spacebar>	와이어를 연결하는 동안, 와이어의 방향을 수평과 수직으로 서로 바꿉니다.
<Spacebar>	객체를 이동하는 동안, 자동 와이어링을 토글 합니다.
<Shift>-클릭	가장 최근에 연결한 와이어를 취소합니다.
<Ctrl>-두 개의 입력을 가진 함수에서 입력을 클릭하기	두 입력 와이어를 서로 바꿉니다.
텍스트	
더블 클릭	문자열에서 한 개의 단어를 선택합니다.
트리플 클릭	전체 문자열을 선택합니다.
<Ctrl>-오른쪽 방향 화살표	서양 문자 세트와 같이 단일 바이트를 사용하는 문자를 갖는 텍스트를 사용할 때 문자열에서 한 단어 앞으로 이동합니다. 아시아 문자 세트와 같이 멀티 바이트를 사용하는 문자를 갖는 텍스트를 사용할 때 문자열에서 한 문자 앞으로 이동합니다.
<Ctrl>-왼쪽 방향 화살표	서양 문자 세트와 같이 단일 바이트를 사용하는 문자를 갖는 텍스트를 사용할 때 문자열에서 한 단어 뒤로 이동합니다. 아시아 문자 세트와 같이 멀티 바이트를 사용하는 문자를 갖는 텍스트를 사용할 때 문자열에서 한 문자 뒤로 이동합니다.
Home	문자열에서 현재 라인의 시작으로 이동합니다.
<End>	문자열에서 현재 라인의 마지막으로 이동합니다.
<Ctrl>-<Home>	전체 문자열의 시작으로 이동합니다.
<Ctrl>-<End>	전체 문자열의 마지막으로 이동합니다.

키보드 바로 가기 키	설명
<Shift>-<Enter>	열거형 타입의 컨트롤과 상수, 링 컨트롤과 상수, 혹은 케이스 구조에서 아이템을 입력할 때 새로운 항목을 추가합니다. 문자열 상수에서 활성화되어 있는 자동 크기 조정 기능을 비활성화하기 위해 키보드에서 <Shift-Enter> 키를 누릅니다. 자동 크기 조정 기능이 비활성화될 경우, 상수에 스크롤 막대를 나타내려면 키보드에서 <Shift-Enter> 키를 누르십시오.
<Ctrl>-<Shift>-<Enter>	케이스 선택자 라벨의 텍스트를 편집할 때 케이스 구조의 보이는 서브다이어그램을 복사합니다.
<Esc>	현재 진행중인 문자열 편집을 취소합니다.
<Ctrl>-<Enter>	텍스트 입력을 종료합니다.
<Ctrl>-=	현재 폰트 크기를 키웁니다.
<Ctrl>--	현재 폰트 크기를 줄입니다.
<Ctrl>-0	폰트 스타일 대화 상자를 디스플레이합니다.
<Ctrl>-1	폰트 스타일 대화 상자에서 어플리케이션 폰트로 변경합니다.
<Ctrl>-2	폰트 스타일 대화 상자에서 시스템 폰트로 변경합니다.
<Ctrl>-3	폰트 스타일 대화 상자에서 대화 상자 폰트로 변경합니다.
<Ctrl>-4	폰트 스타일 대화 상자에서 현재 폰트로 변경합니다.
<Ctrl>-B	텍스트를 굵은체로 나타냅니다.
<Ctrl>-I	텍스트를 이탤릭체로 나타냅니다.
<Ctrl>-U	텍스트에 밑줄을 긋습니다.
배열	
<Ctrl>-<Delete>	현재 원소를 삭제하고 이후 원소를 위로 이동합니다.
<Ctrl>-<Shift>-<Delete>	현재 원소 앞의 원소를 삭제합니다.
<Ctrl>-<Enter>	현재 원소 뒤에 새 원소를 삽입합니다.
<Ctrl>-<Shift>-<Enter>	현재 원소 앞에 새 원소를 삽입합니다.
<Ctrl>-<PgUp>	배열에서 보이는 원소를 한 페이지 위로 스크롤합니다.
<Ctrl>-<PgDn>	배열에서 보이는 원소를 한 페이지 아래로 스크롤합니다.
<Ctrl>-<Home>	배열의 첫번째 원소로 스크롤합니다.
<Ctrl>-<End>	배열의 마지막 원소로 스크롤합니다.
<Tab>	자동 도구가 켜져 있고 잠겨 있는 경우, 다음 원소로 이동합니다.

키보드 바로 가기 키	설명
<Shift>-<Tab>	자동 도구가 켜져 있고 잡겨 있는 경우, 이전 원소로 이동합니다.

빠른 탐색 바로 가기 키

다음 테이블에는 빠른 탐색 대화 상자가 활성화되어 있는 동안 사용할 수 있는 키보드 바로 가기 키가 나열되어 있습니다. 빠른 탐색 대화 상자를 열려면, <Ctrl-Space> 키를 누릅니다. 중국어 키보드에서는, <Ctrl-Shift-Space> 키를 누릅니다. (macOS) <Command-Shift-Space> 키를 누릅니다. (Linux) <Alt-Space> 키를 누릅니다.

또한 빠른 탐색 설정 대화 상자를 사용하여 빠른 탐색 키보드 바로 가기 키를 사용자 정의할 수 있습니다. 빠른 탐색 대화 상자에서 설정 버튼을 클릭하여 빠른 탐색 설정 대화 상자를 디스플레이합니다.



노트

특정 블록 다이어그램이나 프런트패널 객체에 바로 가기를 적용하려면, 반드시 빠른 탐색 대화 상자를 열기 전 바로 가기 키를 적용할 객체를 선택해야 합니다.

키보드 바로 가기 키	설명
<Ctrl>-D	선택한 블록 다이어그램 객체에서 와이어로 연결되지 않은 모든 입력 및 출력에 대해 컨트롤과 인디케이터를 생성합니다.
<Ctrl>-<Shift>-D	선택한 블록 다이어그램 객체에서 와이어로 연결되지 않은 모든 입력에 대해 상수를 생성합니다.
<Ctrl>-F	프런트패널에서, 커넥터 팬 정렬 방식과 일치하도록 컨트롤과 인디케이터를 정렬하고, 프런트패널의 크기를 변경 및 이동하여 주요 모니터에서 일관되게 왼쪽 상단에 위치하도록 합니다. 블록 다이어그램에서, 가장 위쪽 및/또는 가장 왼쪽 블록 다이어그램 객체를 기준으로 합당한 위치가 되도록 블록 다이어그램을 스크롤하고, 블록 다이어그램의 크기를 변경 및 이동하여 주요 모니터에서 일관되게 왼쪽 상단에 위치하도록 합니다.
<Ctrl>-W	선택한 블록 다이어그램 객체들의 1개의 행 또는 여러 개의 평행한 행을 연결합니다.
<Ctrl>-<Shift>-W	선택한 블록 다이어그램 객체들의 1개의 행 또는 여러 개의 평행한 행을 연결하고 선택한 객체들을 정리합니다.
<Ctrl>-R	선택한 블록 다이어그램 객체 및 선택한 객체와 연결된 모든 와이어와 상수를 제거하고, 제거한 객체의 입력과 출력에 연결되어 있던 동일한 데이터 타입의 와이어를 서로 연결합니다.

키보드 바로 가기 키	설명
<Ctrl>-T	최상위 프런트패널 및 블록다이어그램 객체의 보이는 라벨 및 캡션의 위치를 옵션 대화 상자에서 지정한 기본 라벨 위치와 일치하도록 조정합니다. 또한 블록다이어그램에서 여러 객체를 선택하고 빠른 탐색 대화 상자를 디스플레이한 후, <Ctrl-T>를 눌러 선택한 객체의 라벨만 옮길 수도 있습니다.
<Ctrl>-<Shift>-T	서브다이어그램에 포함된 터미널을 포함하여, 최상위 프런트패널 및 블록다이어그램 객체의 보이는 라벨 및 캡션의 위치를 조정하여, 옵션 대화 상자에서 지정한 기본 라벨 위치와 일치하도록 합니다.
<Ctrl>-P	선택한 프런트패널 또는 블록다이어그램 객체를 빠른 탐색 대화 상자에서 선택한 객체로 대체합니다.
<Ctrl>-I	다이어그램에서 선택한 와이어에 빠른 탐색 대화 상자에서 선택한 객체를 삽입합니다.
<Ctrl>-<Shift>-I	선택한 여러 개의 와이어에 빠른 탐색 대화 상자에서 선택한 객체의 단일 인스턴스를 삽입합니다.
<Ctrl>-B	선택한 프로퍼티 노드, 인보크 노드, 클래스 지정자 상수의 VI 서버 클래스를 빠른 탐색 윈도우에서 입력한 클래스로 변경합니다.
<Ctrl>-<Shift>-B	선택한 프로퍼티 노드, 인보크 노드의 프로퍼티나 메소드를 각각 빠른 탐색 윈도우에서 입력한 프로퍼티나 메소드로 변경합니다.

프로젝트 생성하기

LabVIEW 프로젝트 생성하기

프로젝트를 사용하여 LabVIEW 파일과 비LabVIEW 파일을 하나로 그룹화하고, 빌드 스펙을 생성하고, 파일을 타겟에 배포하거나 다운로드합니다.

어플리케이션과 공유 라이브러리를 만들려면 반드시 프로젝트를 사용해야 합니다. 또한 Windows Embedded Standard, RT 또는 FPGA 타겟으로 작업하려면 프로젝트를 사용해야 합니다. LabVIEW Real-Time, FPGA Module은 LabVIEW 개발 시스템에 기능을 추가하며, 별도로 구입해야 합니다.

다음 단계를 따라 프로젝트를 생성합니다:

1. 파일>>프로젝트 생성을 선택하여 프로젝트 생성 대화 상자를 디스플레이합니다. 프로젝트 생성 대화 상자에는 템플릿과 샘플 프로젝트의 리스트가 포함되어 있으며, 이를 사용하여 신뢰할 수 있는 디자인과 프로그래밍 방법으로 프로젝트를 생성할 수 있습니다.
2. 프로젝트의 목적을 달성하도록 지원하는 템플릿 또는 샘플 프로젝트를 검색하십시오. 다음 기능을 사용하여 템플릿 및 샘플 프로젝트를 검색합니다:
 - 필터—특정 타겟을 위한 템플릿 또는 샘플 프로젝트와 같이 특정 타입의 결과만 나타냅니다.

- 추가 검색—필터된 결과의 키워드, 제목 및 설명을 검색합니다.
 - 추가 정보—아이템의 도움말 파일을 엽니다. 도움말 파일을 참조하여 템플릿 또는 샘플 프로젝트가 프로젝트의 목적을 달성하도록 지원하는지 확인하십시오.
3. 다음 또는 마침 버튼을 클릭하여 프로젝트 저장 방법과 같은 프로젝트의 세부사항을 설정합니다. 프로젝트 설정을 마친 후, LabVIEW는 프로젝트를 저장하고 프로젝트 탐색기 윈도우를 엽니다.
 4. 프로젝트 탐색기 윈도우를 사용하여 프로젝트를 변경합니다. 프로젝트를 변경하는 방법에 대한 더 자세한 정보는 프로젝트의 VI의 블록 다이어그램에 있는 주석을 참조하십시오. 또한 프로젝트 탐색기 윈도우에서 프로젝트 문서를 참조하여 프로젝트를 변경하는 방법에 대한 더 자세한 정보를 얻을 수 있습니다.

또한 프로젝트:새로 만들기 메소드를 사용하여 비어있는 새 프로젝트를 프로그램적으로 생성할 수 있습니다.



노트

사용자 정의된 템플릿 및 샘플 프로젝트를 만들어 LabVIEW 프로젝트 개발에 사용할 수 있습니다. 프로젝트 생성 대화 상자에서 사용자 정의된 템플릿 및 샘플 프로젝트를 생성하는 방법에 대한 더 자세한 정보는 기술지원 데이터베이스를 참조하십시오.

컨트롤 선택하기

컨트롤 팔레트에서 컨트롤 선택을 클릭하거나 아이콘 보기에서 컨트롤 선택 아이콘을



(...) 클릭하여, 컨트롤 팔레트에 나타나지 않는 컨트롤을 프런트패널에 추가합니다. 적절한 .ctl 파일을 선택하여 해당 컨트롤을 프런트패널에 추가합니다.

프로젝트에 폴더 추가하기

프로젝트 탐색기 윈도우에서 폴더를 추가하여 **LabVIEW 프로젝트**의 아이템에 대한 **체계적 구조**를 생성합니다.

프로젝트에 자동 적용한 폴더 추가하기

자동 적용한 폴더 추가하기는 디스크의 디렉토리를 프로젝트에 추가합니다. LabVIEW는 폴더를 계속 모니터링하고 프로젝트와 디스크에서의 변경사항에 따라 업데이트합니다. 노란색 원기둥이 있는 파란색 폴더 아이콘은 이같은 타입의 폴더를 나타냅니다.

다음 단계를 따라 자동 적용한 폴더를 프로젝트에 추가합니다.

1. 아이템 페이지에서 타겟이나 타겟 아래의 폴더를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 추가 >> 폴더(자동 적용)를 선택하여 파일 대화 상자를 디스플레이합니다. 또한 타겟을 선택한 후 프로젝트 >> 프로젝트에 추가 >> 폴더(자동 적용)를 선택할 수 있습니다.

2. 프로젝트를 추가할 디스크에서의 디렉토리를 검색하고 선택합니다.
3. 폴더 선택 버튼을 클릭하여 현재 폴더를 프로젝트에 추가합니다. 자동 적용한 폴더가 프로젝트가 나타납니다. LabVIEW가 자동으로 폴더의 내용을 프로젝트에 추가합니다. 디스크 폴더의 아이템이 이미 프로젝트에 있으면, 아이템은 자동 적용한 폴더내로 이동합니다.
4. 파일 페이지에서, 자동 적용한 폴더로부터 파일을 이동하고, 이름을 변경하고, 다시 구성하고 제거할 수 있습니다. 파일 페이지에서의 프로젝트 관리는 디스크에 들어있는 내용을 반영하고 업데이트합니다. 마찬가지로, 디스크 디렉토리를 변경하는 경우, LabVIEW가 프로젝트의 폴더를 업데이트합니다. LabVIEW 프로젝트에서 파일을 업데이트하면, LabVIEW가 파일 이름 변경을 감지하여 호출자 VI를 자동적으로 업데이트할 수 있습니다. 이같은 특징은 디스크가 아닌 프로젝트에서 파일을 변경할 때의 장점입니다.



노트

(macOS and Linux) LabVIEW는 macOS나 Linux OS에서 자동 적용 폴더에 있는 파일에는 기호형 링크의 사용을 지원하지 않습니다.

자동 적용 폴더로 변경 메소드를 사용하여 자동 적용한 폴더를 프로젝트에 프로그램적으로 추가할 수 있습니다.



팁 아이템 페이지에서 아이템을 정렬할 수 있습니다. 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 배치>>이름을 선택하여 아이템을 알파벳 순서로 배치합니다. 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 배치>>타입을 선택하여 아이템을 파일 타입으로 배치합니다. 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 배치>>경로를 선택하여 아이템을 파일 경로 기준으로 배치합니다.

자동 적용한 폴더를 디스크로부터 연결 해제하려면 아이템 페이지의 자동 적용한 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 자동 적용 정지를 선택합니다. LabVIEW는 디스크의 해당 폴더에서 폴더를 연결 해제합니다. 이 옵션은 최상위 폴더에서만 사용 가능하며 자동 적용한 폴더의 서브폴더까지 반복적으로 적용됩니다.

또한 자동 적용 정지 메소드를 사용하여 자동 적용한 폴더를 디스크에서 프로그램적으로 연결 해제할 수 있습니다.



노트 이 프로젝트는 디스크 구성이 아닌 라이브러리 계층구조 기준으로 **프로젝트 라이브러리** 내용을 디스플레이합니다. 예를 들어, VI가 프로젝트 라이브러리 파일 .lvlib에 있으면 VI가 라이브러리 파일 아래의 프로젝트에서 나타납니다. 이 VI는 디스크의 위치에 있는 프로젝트에서는 나타나지 않습니다. 비슷하게 자동 적용한 폴더는 버추얼 폴더가 라이브러리 계층구조내에 있지 않으면 버추얼 폴더를 포함할 수 없습니다.

프로젝트에 버추얼 폴더 추가하기

버추얼 폴더는 프로젝트 아이템을 구성하는 프로젝트의 폴더로 디스크의 파일을 나타내지는 않습니다. 은색 폴더 아이콘은 이같은 타입의 폴더를 식별합니다.

다음 단계를 따라 버추얼 폴더를 프로젝트에 추가합니다.

1. 아이템 페이지에서, 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 추가>>폴더(스냅샷)를 선택하여 파일 대화 상자를 디스플레이합니다. 또한 타겟을 선택한 후 프로젝트>>프로젝트에 추가>>폴더(스냅샷)를 선택할 수 있습니다.
2. 프로젝트를 추가할 디스크에서의 디렉토리를 검색하고 더블 클릭합니다.
3. LabVIEW는 프로젝트에 디스크의 디렉토리와 같은 이름을 가진 새 버추얼 폴더를 생성합니다. 또한 LabVIEW는 서브 디렉토리의 파일과 내용을 포함한 전체 디렉토리의 내용을 나타내는 프로젝트 아이템을 생성합니다.



노트 버추얼 폴더는 디스크의 디렉토리를 변경하여도 프로젝트에서 업데이트되지 않습니다.

버추얼 폴더를 자동 적용 폴더로 변환할 수 있습니다. 가상 폴더에서 마우스 오른쪽 버튼을 클릭한 후 자동 적용 폴더로 변경을 선택하여 파일 대화 상자를 디스플레이합니다. 자동 적용할 디스크의 폴더를 선택합니다. 자동 적용 폴더가 프로젝트에 나타납니다. 자동으로 LabVIEW가 가상 폴더의 이름을 디스크 폴더와 같게 변경하고 디스크 폴더의 모든 내용을 프로젝트에 추가합니다. 디렉토리의 아이템이 이미 프로젝트에 있는 경우, 아이템은 자동 적용 폴더 내에서 이동합니다. 디스크 상의 디렉토리에 존재하지 않는 없는 버추얼 폴더의 아이템은 타겟으로 이동합니다.

또한 버추얼 폴더를 **LabVIEW 프로젝트 라이브러리**로 변환할 수 있습니다. 타겟 아래의 버추얼 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 라이브러리로 변환을 선택합니다.

타겟에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기>>버추얼 폴더를 선택하여 프로젝트 아이템을 구성할 새 버추얼 폴더를 생성합니다.

또한 아이템 추가 메소드를 사용하여 프로그램적으로 폴더를 프로젝트에 추가할 수도 있습니다.

프로젝트에 타겟 폴더 추가하기

타겟 폴더는 LabVIEW 프로젝트 안에서 한 개 또는 이상의 타겟과 그 아이템을 구성합니다. 은색 컨트롤러가 있는 노란색 폴더 아이콘으로 이러한 타입의 폴더를 나타냅니다.

LabVIEW 프로젝트에 타겟 폴더를 추가하려면, 설치된 타겟을 지원하는 모듈이나 드라이버를 가지고 있어야 합니다.

프로젝트에 타겟 폴더를 추가하려면, 프로젝트 탐색기 윈도우의 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기>>타겟 폴더를 선택합니다. 타겟 폴더가 프로젝트 탐색기 윈도우에 나타납니다.

프로젝트에 하이퍼링크 추가하기

프로젝트 탐색기 윈도우를 사용하여 하이퍼링크를 **LabVIEW 프로젝트**에 추가할 수 있습니다. 하이퍼링크를 사용하여 로컬 컴퓨터에는 없지만 인터넷이나 사용자의 로컬 네트

워크에서는 접근 가능한 파일이나 디렉토리로 링크할 수 있습니다. 네트워크, 로컬, HTTP, FTP, mailto 주소 등에 대응하는 하이퍼링크를 추가할 수 있습니다.

다음 단계를 따라 프로젝트에 하이퍼링크를 추가합니다.

1. 타겟 또는 타겟 아래의 폴더나 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 추가»하이퍼링크를 선택합니다. 하이퍼링크 프로퍼티 대화 상자가 나타납니다. 또한 타겟을 선택하고 프로젝트»프로젝트에 추가»하이퍼링크를 선택하여 하이퍼링크 프로퍼티 대화 상자를 디스플레이할 수 있습니다.
2. 이름 텍스트 박스에 하이퍼링크 이름을, 주소 텍스트 박스에 해당 주소를 입력합니다.
3. 확인 버튼을 클릭하여 변경사항을 저장하고 하이퍼링크 프로퍼티 대화 상자를 닫습니다. 하이퍼링크가 프로젝트 탐색기 윈도우에 아이템으로 나타납니다.

하이퍼링크를 더블 링크하거나 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 열기를 선택하여 브라우저 창에서 하이퍼링크를 엽니다.

기존 하이퍼링크에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 기존 하이퍼링크를 편집할 수 있습니다.

또한 아이템 추가 메소드와 하이퍼링크 주소 프로퍼티를 사용하여 하이퍼링크를 LabVIEW 프로젝트에 추가하고 해당 이름과 주소를 프로그램식으로 설정할 수 있습니다.

프로젝트에 아이템 추가하기

프로젝트 탐색기 윈도우를 사용하여 LabVIEW 특정 파일이 아닌 텍스트 파일, 스프레드시트 등의 파일뿐만 아니라 VI 및 **라이브러리**와 같은 LabVIEW 파일을 **LabVIEW 프로젝트**의 타겟에 추가할 수 있습니다. 프로젝트에 파일을 추가할 때, LabVIEW는 프로젝트 탐색기 윈도우에 아이템을 생성하여 디스크의 파일을 나타냅니다. 프로젝트의 아이템에 대한 **체계적 구조**를 생성할 수 있습니다.

한 아이템은 타겟 아래에 한번만 나타납니다. 예를 들어, 디스크에 있는 디렉토리의 파일을 내 컴퓨터 타겟에 추가한 후 내 컴퓨터 타겟에 전체 디렉토리를 추가하면 LabVIEW는 해당 파일을 다시 포함시키지 않습니다.

프로젝트 탐색기 윈도우의 아이템 페이지에서, 다음과 같은 방법으로 프로젝트의 타겟 아래에 아이템을 추가할 수 있습니다:

- 타겟에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기»VI를 선택하여 새 VI를 추가합니다. 또한 파일»새 VI 또는 프로젝트»프로젝트에 추가»새 VI를 선택하여 새 VI를 추가할 수 있습니다. 또한 프런트패널이나 블록 다이어그램 윈도우의 오른쪽 위 가장자리에 있는 VI 아이콘을 선택한 후 아이콘을 타겟으로 끌어 VI를 추가할 수 있습니다. VI를 프로젝트에 추가하면, LabVIEW는 자동으로 전체 계층구조를 프로젝트 탐색기 윈도우의 의존성 아래에 추가합니다.
- 타겟에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기를 선택합니다. 그 후 프로젝트에 추가할 아이템을 선택합니다.
- 타겟에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기»버추얼 폴더를 선택하여 타겟 아래에 버추얼 폴더를 추가합니다. 버추얼 폴더는 프

로젝트 아이템을 구성하는 프로젝트의 폴더로 디스크의 파일을 나타내지는 않습니다. 또한, 버추얼 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기»버추얼 폴더를 선택하여 서브폴더를 생성할 수 있습니다.

- 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 추가»파일을 선택하고, 대화 상자에서 추가하려는 파일을 선택합니다. 또한, 타겟을 선택하고 프로젝트»프로젝트에 추가»파일을 선택한 후 대화 상자에서 추가하려는 파일을 선택할 수 있습니다.
- 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 추가»폴더(스냅샷)를 선택하고, 대화 상자에서 추가하려는 파일을 선택합니다. 또한, 타겟을 선택하고 프로젝트»프로젝트에 추가»폴더(스냅샷)를 선택한 후 대화 상자에서 추가하려는 파일을 선택할 수 있습니다.
- 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 추가»폴더(자동 적용)를 선택하여 파일 대화 상자를 디스플레이합니다. 또한 타겟을 선택한 후 프로젝트»프로젝트에 추가»폴더(자동 적용)를 선택할 수 있습니다.
- 타겟 또는 타겟 아래의 폴더나 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 추가»하이퍼링크를 선택합니다. 하이퍼링크 프로퍼티 대화 상자가 나타납니다. 또한 타겟을 선택하고 프로젝트»프로젝트에 추가»하이퍼링크를 선택하여 하이퍼링크 프로퍼티 대화 상자를 디스플레이할 수 있습니다.
- 파일 추가, 아이템 추가 또는 메모리의 아이템 추가 메소드를 사용하여 프로그램 식으로 아이템을 프로젝트에 추가합니다. 하이퍼링크 주소 프로퍼티를 사용하여 하이퍼링크 아이템의 주소를 설정합니다.
- (Windows와 macOS) 디스크의 아이템 또는 디렉토리를 선택하여 타겟으로 끌고 옵니다.



노트

프로젝트 탐색기 윈도우의 아이템을 끌어서 파일 시스템에 놓을 수 없습니다.

또한, 새로 만들기 대화 상자로부터 새 LabVIEW 파일을 프로젝트에 추가할 수 있습니다. 파일»새로 만들기 또는 프로젝트»프로젝트에 추가»새로 만들기를 선택하여 새로 만들기 대화 상자를 디스플레이합니다. 새로 만들기 대화 상자에서 추가하려는 아이템을 선택하고 프로젝트에 추가 확인란에 확인 표시를 합니다. 여러 프로젝트가 열려있는 경우, 프로젝트 리스트에서 아이템을 추가하려는 프로젝트를 선택합니다.

프로젝트 탐색기 윈도우에 추가한 아이템은 아이콘을 포함할 수 있습니다.

프로젝트에 LLB 추가하기

다음 단계를 완료하여 LabVIEW 프로젝트에 LLB를 추가하고 프로젝트 탐색기 윈도우 상에서 폴더나 파일로 나타냅니다.

LLB를 폴더로 프로젝트에 추가하기

1. 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 추가»폴더(스냅샷)를 선택하여 프로젝트에 LLB를 폴더로 추가합니다. 파일 대화 상자가 나타납니다.

2. 추가하려는 LLB를 탐색합니다.
3. 파일 대화 상자에서 열기 버튼을 클릭합니다. 폴더가 프로젝트 탐색기 윈도우에 나타납니다. LabVIEW는 LLB의 파일 이름을 사용하여 폴더의 이름을 붙이고, LLB 내의 VI를 새 폴더 안에 아이템으로 추가합니다.

프로젝트 탐색기 윈도우의 버추얼 폴더에 아이템을 **추가**하거나 **제거**해도 디스크의 .llb 파일에는 영향을 주지 않습니다. LLB 관리자 윈도우 또는 **자동 적용 폴더**를 사용하여 디스크 상에서 LLB를 편집합니다. 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 추가 >> 폴더(자동 적용)를 선택하여 프로젝트에 LLB를 자동 적용 폴더로 추가합니다.

LLB를 파일로 프로젝트에 추가하기

1. 타겟이나 타겟 아래의 폴더에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 추가 >> 폴더(스냅샷)를 선택하여 프로젝트에 LLB를 파일로 추가합니다. 파일 대화 상자가 나타납니다.
2. 추가하려는 LLB를 탐색합니다.
3. 파일 대화 상자에서 파일 추가 버튼을 클릭합니다. LLB 탐색기가 열립니다.
4. LLB 탐색기에서 단일 마침표와 함께 VI라는 라벨이 붙어있는 열린 폴더 모양의 아이콘()을 클릭합니다.
5. 선택 버튼을 클릭하여 파일을 선택하고 LLB 탐색기를 닫습니다. 파일이 프로젝트 탐색기 윈도우에 나타납니다. LLB를 파일로 추가하는 경우, LLB 안에 있는 VI는 프로젝트 탐색기 윈도우에 나타나지 않습니다.

또한 아이템 추가 메소드를 사용하여 프로그램적으로 LLB를 폴더나 파일로 프로젝트에 추가할 수도 있습니다.

프로젝트에 타겟 추가하기(Windows)

LabVIEW 프로젝트에 타겟을 추가하려면, 설치된 타겟을 지원하는 모듈이나 드라이버를 가지고 있어야 합니다.

다음 단계를 통해서 프로젝트에 타겟 또는 디바이스를 추가합니다.

1. **프로젝트를 생성**합니다.
2. 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 타겟과 디바이스를 선택하여 타겟과 디바이스 추가 대화 상자를 디스플레이합니다. 프로젝트의 타겟이 다른 타겟을 지원하는 경우, 타겟에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 타겟과 디바이스를 선택하여 기존 타겟 아래에 타겟을 추가할 수도 있습니다. 예를 들어, 컴퓨터에 NI PCI 디바이스가 설치되어 있는 경우, 디바이스를 내 컴퓨터 아래에 추가할 수 있습니다.

3. 추가하려는 타겟 또는 디바이스 타입을 선택한 후 확인 버튼을 클릭합니다. 타겟을 나타내는 아이템이 프로젝트 탐색기 윈도우에 나타납니다.



노트

추가하려는 타겟이 다른 타겟과 상충할 때 타겟과 디바이스 추가 대화 상자는 타겟 및 디바이스 리스트 옆에 있는 옵션 열을 디스플레이합니다. 옵션 열의 선택한 충돌 옆의 아래 방향 화살표를 클릭하고 사용 가능한 옵션에서 선택하여 충돌을 해결합니다. 한 번에 하나의 타겟의 충돌만을 해결할 수 있습니다. 선택한 타겟에 대해서 옵션 열에 있는 모든 충돌을 해결할 때까지 LabVIEW는 확인 버튼을 비활성화합니다.

타겟 아래에 **아이템을 추가**할 수 있습니다. 또한 타겟 폴더를 사용하여 LabVIEW 프로젝트에서 한 개 또는 이상의 타겟을 구성할 수 있습니다. 프로젝트에 타겟 폴더를 추가하려면, 프로젝트 탐색기 윈도우의 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기»타겟 폴더를 선택합니다. 타겟 폴더가 프로젝트 탐색기 윈도우에 나타납니다.

프로젝트에서 아이템 찾기

다음 단계를 따라 열린 **프로젝트 탐색기**의 목차 트리에서 아이템을 검색합니다.

1. 프로젝트 탐색기 윈도우에서 편집»프로젝트 아이템 찾기를 선택하여 프로젝트 아이템 찾기 대화 상자를 디스플레이합니다. 특정 폴더, 프로젝트 라이브러리, LabVIEW 클래스 또는 X컨트롤에서 LabVIEW 프로젝트 아이템을 검색하려면, 프로젝트 탐색기 윈도우의 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로젝트 아이템 찾기를 선택합니다.
2. 검색할 텍스트 입력 텍스트 박스에서 찾으려는 텍스트를 입력합니다. 프로젝트 아이템 찾기 대화 상자의 제목 표시줄은 검색의 루트 아이템을 디스플레이합니다. 검색은 이 루트 아이템의 모든 아이템을 포함합니다.



노트

LabVIEW는 LabVIEW 프로젝트, 프로젝트 라이브러리, LabVIEW 클래스, 또는 X컨트롤의 멤버인 아이템만 검색합니다. 예를 들어, 검색하려는 LabVIEW 프로젝트가 Foo.vi를 포함하며, Foo.vi가 Bar.vi라는 이름의 SubVI를 포함하는 경우, SubVI의 검색 결과를 받으려면 Bar.vi를 LabVIEW 프로젝트에 추가해야 합니다.

1. (옵션) 검색하려는 텍스트를 입력한 후, 대소문자 구분, 모든 단어 일치, 정규식 옵션 중에서 선택하여 검색 범위를 설정합니다.
2. 찾기 버튼을 클릭합니다. 검색 결과 리스트는 LabVIEW가 찾은 텍스트의 모든 인스턴스를 포함합니다.



노트 하나의 아이템만이 검색 문자열에 일치하는 경우, LabVIEW는 자동으로 프로젝트 탐색기 윈도우에서 검색 결과를 하이라이트하고 프로젝트 아이템 찾기 대화 상자를 닫습니다.

1. 검색 결과를 선택하고 이동 버튼을 클릭합니다. LabVIEW는 프로젝트 탐색기 윈도우에서 아이템을 하이라이트합니다. 또한, 검색 결과 리스트에서 엔트리를 더블 클릭하여 프로젝트 탐색기 윈도우에서 아이템을 하이라이트할 수 있습니다.
2. (옵션) 편집»다음 찾기 또는 편집»이전 찾기를 선택하여 프로젝트 탐색기 윈도우에서 다른 검색 결과를 하이라이트합니다.

LabVIEW나 프로젝트 탐색기 윈도우가 닫힐 때까지, LabVIEW는 이전 검색 결과를 유지합니다. 편집»검색 결과 보이기를 선택하여 프로젝트 아이템 찾기 대화 상자와 이전 검색 결과의 리스트를 디스플레이하여 이전 검색 결과를 봅니다.

또한 독립 프로젝트 라이브러리 윈도우 또는 클래스 윈도우에서 프로젝트 아이템 찾기 대화 상자를 열 수 있습니다.

프로젝트에서 아이템 정렬하기

정렬 옵션을 사용하여 프로젝트 내의 파일을 구성할 수 있습니다. 선택한 정렬 옵션은 디스크 상에 있는 프로젝트를 재구성하지 않습니다. 따라서 프로젝트에서 더 적은 변경 사항을 통합하고 비교할 수 있습니다. 정렬 옵션은 정렬된 프로젝트에 추가한 모든 새 아이템에 자동으로 적용됩니다.

또한 **자동 적용 폴더** 내에서 잠기지 않은 **프로젝트 라이브러리**와 내용을 정렬할 수 있습니다.

다음 단계를 따라 **LabVIEW 프로젝트** 내에 있는 아이템을 정렬합니다.

1. 프로젝트 탐색기 윈도우의 정렬하려는 아이템에서 마우스 오른쪽 버튼을 클릭합니다.
2. 배치를 선택하고 바로 가기 메뉴에서 사용 가능한 정렬 항목을 선택합니다.

다음 정렬 옵션 가운데 하나를 사용하여 프로젝트 내의 아이템을 정렬합니다:

- 이름—아이템을 파일 이름에 따라 정렬합니다.
- 타입—아이템을 파일 타입에 따라 정렬합니다.
- 경로—아이템을 경로 이름에 따라 정렬합니다.
- 사용자—아이템을 사용자가 정한 기준에 따라 정렬합니다. 다른 정렬 항목과는 달리 사용자 정렬은 디스크 내 프로젝트의 레이아웃을 변경합니다.
- 상위 아이템과 같음—상위 아이템과 같은 방식으로 아이템을 정렬합니다. 상위 아이템과 같음으로 정렬하면 자동으로 아이템이 상위 폴더의 정렬 항목을 따르게 됩니다. 상위 아이템이 있는 아이템만 상위 아이템과 같음 항목을 사용하여 정렬할 수 있습니다.

이러한 정렬 항목은 프로젝트 내의 타겟, 파일, 폴더에 적용됩니다.



노트

기본적으로 LabVIEW는 모든 새 프로젝트와 프로젝트 라이브러리를 이름에 따라 정렬합니다. 이름, 타입, 경로, 상위 아이템과 같음으로 정렬하면 디스크 상 프로젝트의 구조를 변경하지 않습니다. 그러나 정렬 옵션을 사용하지 않고 프로젝트 내의 아이템을 이동하는 경우, LabVIEW는 자동으로 사용자 정렬로 변경되며 디스크 내 프로젝트의 구조가 수정됩니다. 프로젝트 라이브러리를 타입이나 경로로 정렬한 경우, 다음에 LabVIEW에서 프로젝트 라이브러리를 열 때 이름으로 다시 자동 정렬됩니다.

프로젝트에서 아이템 삭제하기

프로젝트 탐색기 윈도우를 사용하여 LabVIEW 프로젝트에서 아이템을 삭제할 수 있습니다. 다음과 같은 방법으로 아이템을 삭제할 수 있습니다:

- 삭제하려는 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로젝트에서 제거를 선택합니다.
- 삭제하려는 아이템을 선택하고 표준 도구 모음의 프로젝트에서 제거 버튼을 누릅니다.
- 아이템에서, 삭제하려는 아이템을 선택하고 <Delete> 키를 누릅니다.



노트

프로젝트의 아이템이 다른 아이템을 참조할 수 있습니다. 예를 들어, 프로젝트의 다른 VI가 호출하는 프로젝트에서 SubVI를 제거하려면, SubVI가 의존성으로 이동하게 됩니다.

삭제 메소드를 사용하여 프로그램적으로 프로젝트에서 참조를 통해 아이템을 삭제할 수도 있습니다.

프로젝트에서 의존성 제거하기

다음 방법으로 LabVIEW 프로젝트의 **의존성** 아래의 아이템을 제거할 수 있습니다.



노트

이전에 프로젝트 탐색기 윈도우에서 의존성을 숨긴 경우, 프로젝트 탐색기 윈도우에서 해당 아이템에 접근하려면 다시 **아이템을 디스플레이**해야 합니다.

- 아이템을 의존성에서 타겟 노드로 끌어서 타겟 노드 아래에 아이템을 추가합니다. 또한, 타겟에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 추가>> 파일을 선택하고, 타겟 아래에 추가하려는 아이템을 선택합니다.
- 아이템을 호출하는 VI를 제거하고, 의존성에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 고침을 선택합니다.

의존성 새로 고침 메소드를 사용하여 프로그램적으로 의존성을 새로 고칠 수도 있습니다.

프로젝트 저장하기

다음 방법으로 **LabVIEW 프로젝트**를 저장할 수 있습니다:

- 파일>>저장을 선택합니다.
- 프로젝트>>프로젝트 저장을 선택합니다.
- 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 저장을 선택합니다.
- 프로젝트 도구 모음에서 모두 저장(이 프로젝트) 버튼을 클릭합니다. 또한 파일>>모두 저장(이 프로젝트)을 선택하거나 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모두 저장(이 프로젝트)을 선택합니다.
- 저장 메소드를 사용하여 프로젝트를 프로그램적으로 저장합니다.

프로젝트를 저장하기 전에 반드시 새로운, 저장되지 않은 파일을 프로젝트에 저장해야 합니다.



노트

프로젝트에 주요한 개정이 있을 때 또는 LabVIEW의 새로운 버전으로 업그레이드 하기 전에 프로젝트의 백업 복사본을 만듭니다.

어플리케이션 인스턴스 선택하기

LabVIEW는 **LabVIEW 프로젝트**에 각 타겟에 대한 어플리케이션 인스턴스를 생성합니다. 프로젝트 탐색기 윈도우에서 VI를 열 때, VI는 타겟의 어플리케이션 인스턴스에서 열립니다. 또한, LabVIEW는 주요 어플리케이션 인스턴스를 생성합니다. 여기에는 프로젝트의 일부가 아닌 열린 VI와 프로젝트에서 열지 않은 VI가 포함됩니다. 어플리케이션 인스턴스 바로 가기 메뉴를 사용하여 VI에 대한 어플리케이션 인스턴스를 선택할 수 있습니다.

다음 단계를 따라 VI에 대한 어플리케이션 인스턴스를 선택합니다.

1. 프런트패널 윈도우나 블록다이어그램의 왼쪽 아래 코너를 탐색합니다. 어플리케이션 인스턴스 선택 바로 가기 메뉴는 VI의 현재 어플리케이션 인스턴스를 디스플레이합니다.
2. 바로 가기 메뉴에서 마우스 오른쪽 버튼을 클릭하고 VI를 실행할 인스턴스를 선택합니다. VI가 선택한 어플리케이션 인스턴스에서 실행됩니다.

또한 어플리케이션:기본:어플리케이션 프로퍼티를 사용하여 기본 어플리케이션 참조를 프로그램적으로 반환할 수 있습니다. 어플리케이션 프로퍼티를 사용하여 타겟 어플리케이션 인스턴스를 프로그램적으로 엽니다.

프로젝트 라이브러리에서 기본 팔레트 파일 설정하기

프로젝트 라이브러리에 팔레트 파일(.mnu)을 포함하는 경우, 이 파일을 프로젝트 라이브러리가 소유하는 모든 VI의 기본 팔레트 파일로 설정할 수 있습니다. 다음 단계에 따라 **LabVIEW 프로젝트 라이브러리**의 기본 팔레트 파일을 설정합니다.

1. **프로젝트 탐색기** 윈도우에서 기본 팔레트 파일로 설정할 팔레트 파일 .mnu 위의 프로젝트 라이브러리 아이콘을 찾습니다. .mnu 파일이 없으면, **팔레트 파일을 추가**합니다.
2. 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **프로퍼티**를 선택하여 프로젝트 라이브러리 프로퍼티 대화 상자를 디스플레이합니다.
3. 일반 셋팅 페이지의 기본 팔레트 폴다운 메뉴에서 기본으로 설정할 팔레트 파일을 선택합니다. 또한 목차 트리에서 .mnu 파일을 선택하여 **아이템 셋팅** 페이지에서 기본 팔레트 파일을 설정하고 기본 팔레트 확인란에 확인 표시를 할 수 있습니다.
4. 확인 버튼을 클릭하여 프로젝트 라이브러리 프로퍼티 대화 상자를 닫고 변경사항을 저장합니다.

또한, 라이브러리:생성 메소드를 사용하여 기본 팔레트 파일을 프로그램적으로 설정할 수 있습니다.

프로젝트에서 아이템 보이기와 숨기기

다음과 같은 방법으로 **프로젝트 탐색기** 윈도우에서 의존성과 빌드 스펙 아이템을 보이게 하거나 숨길 수 있습니다.

- 프로젝트 » 필터 보기를 선택하여 보이거나 숨기려는 아이템을 선택합니다.
- 프로젝트 탐색기 윈도우의 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 보기를 선택하고, 보이게 하거나 숨기려는 아이템을 선택합니다.
- 프로젝트 도구 모음의 프로젝트 보기 필터 버튼을 클릭하여, 보이게 하거나 숨기려는 아이템을 선택합니다.

프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모두 확장 또는 모두 축소를 선택하여, 프로젝트 탐색기 윈도우의 모든 아이템을 확장하거나 축소합니다. 또한, 프로젝트 탐색기 윈도우의 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모두 확장 또는 모두 축소를 선택하여, 아이템과 아이템 내의 모든 폴더를 확장하거나 축소할 수 있습니다.

프로젝트에서 파일 보기

프로젝트 탐색기에는 두 개의 페이지가 있습니다. **아이템** 페이지는 프로젝트의 내용을 디스플레이합니다. **파일** 페이지는 디스크에 상응하는 파일이 있는 프로젝트 아이템을 디스플레이합니다. 타겟 아래의 폴더나 아이템에서 마우스 오른쪽 버튼을 클릭한 후 바

로 가기 메뉴에서 **아이템 보기**에서 **보이기** 또는 **파일 보기**에서 **보이기**를 선택하여 다른 페이지로 전환할 수 있습니다.

LabVIEW 프로젝트 **파일을 추가**할 때, LabVIEW는 디스크에 파일의 참조를 포함합니다. 프로젝트 탐색기 윈도우의 **아이템** 페이지에서, 프로젝트»아이템 경로 보하기를 선택하여 경로 열을 디스플레이하고 파일 경로를 봅니다. 또한 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보기»전체 경로를 선택하여 경로 열을 디스플레이하고 파일 경로를 볼 수 있습니다. 경로 프로퍼티를 사용하여 프로그램적으로 디스크에서의 파일 위치를 반환하십시오.

프로젝트 탐색기 윈도우의 파일 페이지를 사용하여 프로젝트 파일의 디스크 위치를 봅니다. 파일 페이지의 파일 이름과 폴더를 구성하여 이름 변경, 재구성 및 제거와 같은 디스크 작업을 수행할 수 있습니다. 예를 들어, 파일 페이지에서 파일의 이름을 변경하면, LabVIEW가 디스크에서 이 파일의 이름을 변경합니다. 또한 프로젝트 파일 정보 대화 상자를 사용하여 프로젝트 파일의 디스크 위치를 볼 수 있습니다. 프로젝트»파일 정보를 선택하여 프로젝트 파일 정보 대화 상자를 디스플레이합니다. 또한, 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보기»파일 정보를 선택하여 프로젝트 파일 정보 대화 상자를 디스플레이할 수 있습니다.

프로젝트 라이브러리 작업하기

프로젝트 라이브러리 생성하기

LabVIEW 프로젝트 라이브러리는 VI, 타입 정의, 공유 라이브러리, 팔레트 파일, 다른 프로젝트 라이브러리를 포함한 기타 파일의 집합입니다. 새 프로젝트 라이브러리를 생성하고 저장할 때 LabVIEW는 프로젝트 라이브러리 파일 .lvlib를 생성합니다. 이 파일에는 프로젝트 라이브러리의 프로퍼티와 프로젝트 라이브러리가 소유하는 파일에 대한 참조가 포함됩니다.

다음 단계를 따라 프로젝트 라이브러리를 생성합니다.

1. **프로젝트 탐색기** 윈도우에서, 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **새로 만들기**»라이브러리를 선택합니다. LabVIEW는 프로젝트 탐색기 윈도우에 LabVIEW 프로젝트의 일부로 나타나는 프로젝트 라이브러리 파일을 생성합니다. 시작하기 윈도우나 새 VI에서 파일»새로 만들기를 선택하여 새로 만들기 대화 상자를 디스플레이할 수도 있습니다. 새로 만들기 트리의 기타 파일 폴더에서 라이브러리 옵션을 더블 클릭합니다. 새 프로젝트 라이브러리를 위한 독립 프로젝트 라이브러리 윈도우가 나타납니다.
2. 파일을 프로젝트 라이브러리에 추가하려면, 프로젝트 라이브러리 아이콘에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **추가**»파일 또는 **추가**»폴더 (스냅샷)를 선택합니다. 프로젝트에 추가할 수 있는 것과 같은 타입의 파일을 추가할 수 있습니다. 또한, 프로젝트의 아이템을 끌어서 프로젝트 라이브러리 아래에 나타나게 할 수 있습니다. 하나의 프로젝트 라이브러리만이 특정 VI를 소유할 수

있습니다. 하지만, 텍스트 파일이나 HTML 파일처럼 LabVIEW 전용 파일이 아닌 경우에는 여러 프로젝트 라이브러리와 연결할 수 있습니다.



노트

메모리에 없는 VI를 프로젝트 라이브러리에 추가하는 경우, 대화 상자가 나타나 VI의 변경사항을 적용하도록 입력요청합니다. VI의 변경을 저장하여 소유하는 프로젝트 라이브러리에 바르게 링크되도록 합니다.

1. 파일>>모두 저장을 선택하거나 프로젝트 라이브러리 아이콘에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 저장>>다른 이름으로 저장을 선택하여, 프로젝트 라이브러리 파일의 파일 이름을 결정하고 저장합니다.
2. 프로젝트 라이브러리 아이콘에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 프로젝트 라이브러리 프로퍼티 대화 상자를 디스플레이합니다.
3. 일반 셋팅 페이지에서, 버전 번호를 설정하고, **아이콘을 생성**하고, 프로젝트 라이브러리에 **보안 셋팅을 지정**할 수 있습니다.



노트 LabVIEW는 사용자가 생성한 프로젝트 라이브러리 아이콘을 라이브러리에 있는 기존의 모든 객체의 아이콘에 적용합니다. 또한, 사용자가 생성한 아이콘이 프로젝트 라이브러리 내에서 생성한 새 VI의 기본 아이콘이 됩니다. 템플릿 아이콘을 생성하여 프로젝트 라이브러리가 소유하는 모든 VI를 편집하고 사용할 수 있습니다.

1. 문서 페이지에서 프로젝트 라이브러리의 기본 도움말 정보를 입력합니다.
2. 아이템 셋팅 페이지에서 프로젝트 라이브러리의 파일에 대한 **접근 옵션을 설정**합니다.
3. 확인 버튼을 클릭하여 편집된 셋팅으로 프로젝트 라이브러리를 업데이트하고 대화 상자를 닫습니다.

또한, **프로젝트 폴더에서 프로젝트 라이브러리를 생성**할 수 있습니다.

정렬 옵션을 사용하여 잡기지 않은 프로젝트 라이브러리 내의 아이템을 구성할 수 있습니다.

labview\vi.lib\Utility\Variable 디렉토리의 CreateOrAddLibrary polymorphic VI를 사용하여 프로그램적으로 라이브러리를 프로젝트나 타겟, 폴더 또는 다른 라이브러리 같은 상위 아이템에 추가합니다.

프로젝트 폴더에서 프로젝트 라이브러리 생성하기

LabVIEW 프로젝트의 버추얼 폴더에서 **LabVIEW 프로젝트 라이브러리**를 생성할 수 있습니다. 새 프로젝트 라이브러리는 폴더가 포함하는 아이템을 소유합니다.

프로젝트 탐색기 윈도우의 버추얼 폴더에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 라이브러리로 변환을 선택합니다. LabVIEW는 폴더를 프로젝트 라이브러리로

변환하며, 이는 프로젝트 탐색기 윈도우에서 라이브러리 아래에 나열되는 라이브러리 소유 아이템과 함께 나타납니다.



노트

자동 적용 폴더를 프로젝트 라이브러리로 변환할 수는 없습니다.

저장할 때 새 프로젝트 라이브러리 파일의 이름을 정할 수 있습니다. 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 저장을 선택합니다.

프로젝트 라이브러리에서 접근 옵션 설정하기

LabVIEW 프로젝트 라이브러리가 소유한 아이템과 폴더의 접근 설정을 퍼블릭 또는 프라이빗으로 설정할 수 있습니다. 아이템을 프라이빗으로 설정하고 **프로젝트 라이브러리를 잠그는** 경우, 아이템은 프로젝트 라이브러리 또는 팔레트에서 보이지 않습니다. 프로젝트 라이브러리가 잠금해제 상태인 경우더라도 프라이빗 VI를 프로젝트 라이브러리가 소유하지 않은 다른 VI 또는 어플리케이션의 subVI로 사용할 수 없습니다.

프로젝트 라이브러리의 어떤 아이템을 퍼블릭으로 설정하고 어떤 아이템을 프라이빗으로 설정할지 결정합니다. 퍼블릭 아이템에는 사용자가 찾고 사용할 수 있는 팔레트 VI, X 컨트롤, 인스트루먼트 드라이버, 기타 도구가 포함됩니다. 프라이빗 아이템에는 지원 VI, 저작권이 있는 파일, 또는 사용자 코드를 깨지 않고 나중에 편집하려는 아이템이 포함됩니다.

소유하는 프로젝트 라이브러리 내에서 **프로젝트 서브라이브러리** 파일 .lvlib의 접근을 프라이빗으로 설정할 수 있습니다. 그러나 프로젝트 라이브러리 자체를 편집하는 경우, 서브라이브러리가 소유하는 아이템은 퍼블릭 또는 프라이빗 접근 설정을 유지합니다.

다음 단계를 따라 프로젝트 라이브러리에 접근 옵션을 설정합니다.

1. **프로젝트 탐색기** 윈도우 또는 독립 프로젝트 라이브러리 윈도우의 프로젝트 라이브러리 아이콘에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 프로젝트 라이브러리 프로퍼티 대화 상자를 엽니다.
2. 아이템 설정 페이지의 목차 트리에서 선택할 아이템을 클릭합니다. 아이템에 대한 현재 접근 설정이 접근 영역 상자에 나타납니다. 접근 영역 상자의 다음 라디오 버튼 중 하나를 클릭하여 아이템에 적용합니다.
 - 퍼블릭—프로젝트 라이브러리를 사용자가 볼 때 보이는 아이템입니다. 다른 VI와 어플리케이션은 퍼블릭 VI를 호출할 수 있습니다.
 - 공동 영역—프로젝트 라이브러리를 사용자가 볼 때 보이는 아이템입니다. 프로젝트 라이브러리 내의 **friends**와 VI만이 공동 영역 VI를 호출할 수 있습니다. VI가 공동 영역에 있는 라이브러리의 모든 멤버를 호출할 수 있게 하려면, VI를 라이브러리의 friend로 지정합니다.
 - Private—프로젝트 라이브러리를 잠근 경우 프로젝트 라이브러리와 팔레트를 사용자가 볼 때 보이지 않는 아이템. 프로젝트 라이브러리에서 다른 VI와 어플리케이션은 고유한 프라이빗 VI를 호출할 수 없습니다.
 - 미지정—이 옵션은 폴더를 선택할 때에만 나타납니다. 폴더에 지정된 접근 아이템이 없습니다. 접근은 퍼블릭입니다. 기본으로, 프로젝트 라이브러리

의 폴더는 지정된 접근을 가지고 있지 않으므로 폴더는 퍼블릭으로 접근이 가능합니다.



노트

폴더에 대해 접근 옵션을 지정하는 경우, 접근 셋팅은 폴더의 모든 아이টে 적용되고 폴더의 개별 아이টে 대한 접근 옵션을 덮어씁니다.



노트 다형성 VI의 개별 인스턴스를 프라이빗으로 설정하고 기본 다형성 VI를 퍼블릭으로 설정할 수 있습니다. 인스턴스 VI가 프라이빗인 경우에도 다형성 VI는 깨지지 않습니다. 사용자가 다형성 VI 선택기를 통해서만 인스턴스에 접근하도록 하려는 경우 인스턴스 VI를 프라이빗으로 설정하는 것이 유용합니다. 이를 통하여 사용자 문제를 발생시키지 않고 인스턴스의 순서를 편집할 수 있습니다.

1. 확인 버튼을 클릭하여 변경을 프로젝트 라이브러리에 적용하고 대화 상자를 닫습니다.

프라이빗으로 설정된 아이템이 프로젝트 탐색기 윈도우에 프라이빗 아이콘과 함께 나타납니다. 프로젝트 라이브러리를 잠그는 경우, 프라이빗 아이템은 프로젝트 탐색기 윈도우에 나타나지 않습니다.

또한, 소스 영역:설정 메소드를 사용하여 접근 옵션을 프로그램적으로 설정할 수 있습니다.

프로젝트 라이브러리 보호하기

LabVIEW 프로젝트 라이브러리를 잠그거나 암호로 보호하여 편집 권한을 제한할 수 있습니다. 프로젝트 라이브러리를 잠그면, 사용자가 아이템을 추가 또는 제거하지 못하며 **프라이빗으로 설정**한 아이템을 볼 수 없습니다. 프로젝트 라이브러리에 암호를 설정하면, 사용자가 암호 없이 프로젝트 라이브러리 프로퍼티에서 아이템을 추가 또는 제거하거나 편집할 수 없습니다. 사용자는 프로젝트 라이브러리 프로퍼티 대화 상자를 열 수 있으나, 보호 옵션을 제외한 모든 대화 상자 구성요소가 비활성화됩니다. 사용자는 프로젝트 라이브러리를 잠금해제하거나 암호를 입력하여 대화 상자 구성요소를 활성화해야 합니다.



노트

프로젝트 라이브러리에 암호 보호를 추가해도 소유하는 VI에 암호 보호가 추가되지는 않습니다. **개별 VI에 암호 보호를 할당**해야 합니다.

다음 단계를 따라 프로젝트 라이브러리에 보호 레벨을 설정합니다.

1. 프로젝트 탐색기 또는 독립 프로젝트 라이브러리 윈도우의 프로젝트 라이브러리 아이콘에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 프로젝트 라이브러리 프로퍼티 대화 상자를 엽니다.
2. 보호 페이지에서 다음 옵션 중 하나를 선택하여 프로젝트 라이브러리에 적용합니다.

- 잠금해제(암호 없음)—사용자가 프로젝트 라이브러리가 소유하는 퍼블릭 및 프라이빗 아이템을 볼 수 있고 프로젝트 라이브러리 및 그 프로퍼티를 편집할 수 있습니다.
 - 잠금(암호 없음)—사용자는 프로젝트 라이브러리의 아이템 추가 및 제거, 프로젝트 라이브러리 프로퍼티 편집, 또는 프로젝트 라이브러리 고유의 프라이빗 아이템 보기를 할 수 없습니다. 예를 들어, 프로젝트 라이브러리를 개발하는 도중에 다른 사용자가 프라이빗 파일을 보지 못하게 하려는 경우, 프로젝트 라이브러리를 잠금해야 합니다.
 - 암호로 보호됨—사용자는 프로젝트 라이브러리의 아이템 추가 및 제거, 프로젝트 라이브러리 프로퍼티 편집 또는 프로젝트 라이브러리 고유의 프라이빗 아이템 보기를 할 수 없습니다. 사용자는 프로젝트 라이브러리를 편집하기 위해 반드시 암호를 입력해야 합니다. 예를 들어, 프로젝트 라이브러리를 개발하는 도중에 개발팀의 일부 사용자만 편집 권한을 가지도록 하려는 경우, 프로젝트 라이브러리에 암호를 설정하고 암호를 해당 사용자에게 알려줍니다.
3. 암호로 보호됨을 선택하는 경우, 암호 입력 버튼을 클릭하여 암호를 입력합니다.
 4. 인증 대화 상자에서 암호를 입력하고 확인을 클릭합니다.
 5. 확인 버튼을 클릭하여 변경을 프로젝트 라이브러리에 적용하고 대화 상자를 닫습니다.



노트 프로젝트 라이브러리에 암호를 적용하면 자동으로 암호를 LabVIEW 암호 캐시에 저장합니다. 이를 통해 마치 암호를 입력한 것처럼 프로젝트 라이브러리에 접근하여 변경 사항을 적용할 수 있습니다. LabVIEW를 다시 시작하거나 옵션 대화 상자의 환경 페이지에서 암호 캐시를 지워 캐시에서 암호를 제거할 수 있습니다.

또한, 잠금 상태:설정 메소드를 사용하여 보호 옵션을 프로그램적으로 설정할 수 있습니다.

LLB를 프로젝트 라이브러리로 변환하기

LLB를 **LabVIEW 프로젝트 라이브러리**로 변환할지 여부를 결정하기 전에 **LLB**를 사용할 수 있는 방법을 고려하십시오. LLB와 프로젝트 라이브러리는 다른 목적을 가지고 있으며 상호 배타적이지 않습니다.

예를 들어, 어플리케이션 프로그래밍 인터페이스(API)의 역할을 하는 파일의 자체 포함 집합을 변환할 수 있습니다. LLB를 LabVIEW 프로젝트 라이브러리로 변환하는 것은 몇 가지 장점이 있습니다. 프로젝트 라이브러리가 소유하는 개별 아이템에 **소스 컨트롤** 작업을 수행할 수 있습니다. 그러나 LLB에서는 개별 VI에 소스 컨트롤 작업을 수행할 수 없습니다. 또한, 프로젝트 라이브러리에 포함된 아이템에 대해 퍼블릭 또는 프라이빗으로 접근 옵션을 설정할 수 있습니다.

LLB를 프로젝트 라이브러리로 변환하는 경우, LLB를 열었을 때 최상위 VI가 자동으로 실행되도록 설정하는 기능을 잃게 됩니다. 또한, 프로젝트 라이브러리에 VI를 최상위 레벨로 표시할 수 없습니다. 예를 들어, 단일 어플리케이션으로 작동하는 VI로 구성된 LLB는 변환하지 마십시오. 변환하지 않아야 LLB를 열었을 때 자동으로 실행됩니다. 대신 LLB의 이 타입을 LabVIEW 프로젝트에서 구성하는 것을 고려하십시오.

LLB가 하나의 최상위 레벨 VI와 subVI의 집합으로 구성된 경우, LLB의 VI를 프로젝트 라이브러리에 추가할 것을 고려하십시오. LLB내의 해당 프로젝트 라이브러리 파일(.lvlib)을 VI와 함께 저장하십시오. 프로젝트 라이브러리에서, SubVI의 접근을 프라이빗으로 설정하십시오. 이 방법으로 파일을 구성하면 사용자들이 subVI에 접근하지 않고도 VI의 최상위 레벨 파일을 실행할 수 있게 됩니다.

다음 단계를 따라 LLB를 프로젝트 라이브러리로 변환합니다.

1. **LLB를 디렉토리로 변환**합니다.
2. **프로젝트 탐색기** 윈도우에서, 타겟에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 추가 >> 폴더(스냅샷)를 선택하여 파일 대화 상자를 디스플레이합니다. 단계 1에서 생성한 디렉토리를 탐색하고 폴더 선택 버튼을 클릭합니다. LabVIEW는 프로젝트에 **버추얼 폴더**를 생성합니다.
3. 프로젝트 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 라이브러리로 변환을 선택하여 **프로젝트 폴더에서 프로젝트 라이브러리를 생성**합니다.
4. 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 저장 >> 저장을 선택하여 새 프로젝트 라이브러리 파일의 이름을 정하고 저장합니다.

LabVIEW 프로젝트의 VI를 묶음 프로젝트 라이브러리로 변환하기

VI의 그룹에 대해 **묶음 프로젝트 라이브러리를 만들 필요**가 있다고 결정되면, 다음 단계에 따라 규모가 큰 어플리케이션의 여러 VI를 묶음 라이브러리로 변환합니다.

1. 이 어플리케이션의 모든 VI를 LabVIEW 프로젝트에 놓습니다.
2. 각각의 모듈이나 함께 작동하는 VI 그룹에 대해 개별 프로젝트 라이브러리를 생성합니다. 여러 모듈에서 한 개의 SubVI를 사용할 경우, 각 프로젝트 라이브러리에 이 SubVI를 복제하거나, 모든 공유 SubVI를 별도의 프로젝트 라이브러리에 놓습니다.
3. **잘못된 의존성을 방지**하려면 프로젝트 라이브러리에서 SubVI를 참조하는 모든 VI를 업데이트하십시오.
4. 각각의 프로젝트 라이브러리를 LabVIEW 프로젝트에 추가해서, 각 프로젝트 라이브러리에 대해 개별 LabVIEW 프로젝트를 생성합니다.
5. 각 LabVIEW 프로젝트에 대해, 이 프로젝트 라이브러리를 최상위 레벨 파일로 갖는 묶음 라이브러리를 생성합니다.
6. 모든 프로젝트 라이브러리가 포함된 LabVIEW 프로젝트의 각 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택하여, 묶음 라이브러리로 **각 프로젝트 라이브러리를 대체**합니다.
7. 프로젝트에서 원본 프로젝트 라이브러리와 VI를 제거합니다.



노트

묶음 라이브러리에서 파일을 변경해야 할 경우, 이 프로젝트 라이브러리를 가진 LabVIEW 프로젝트에서 해당 VI를 변경해야 합니다. 변경 사항을 저장한 후, 변경 사항을 테스트하고, 묶음 라이브러리를 다시 만들고, 가장 최신의 묶음 라이브러리를 모든 묶음 라이브러리를 가진 LabVIEW 프로젝트에 복사합니다.

묶음 프로젝트 라이브러리 안에서 VI를 호출하는 VI 편집하기

묶음 프로젝트 라이브러리를 호출하는 VI를 편집할 경우에는, **묶음 라이브러리에서 반출된 VI**를 접근해야 할 수도 있습니다. 다음 단계를 따라 반출된 VI를 호출자 VI에 추가합니다.

1. 묶음 라이브러리에서 반출된 VI에는 다음 중 한 가지 방법으로 접근합니다.
 - 묶음 라이브러리의 파일 리스트 중 반출된 VI를 선택합니다.
 - **프로젝트 탐색기** 윈도우에서, 내 컴퓨터 아래에 있는 의존성을 탐색하여 프로젝트의 묶음 라이브러리 폴더를 엽니다.
 - 묶음 라이브러리에서 컨트롤 또는 함수 팔레트에 **새 팔레트를 추가**합니다.
2. 반출된 VI를 호출자 VI의 블록 다이어그램으로 끌어다 놓습니다.

묶음 프로젝트 라이브러리의 유의점과 권장사항

다음 리스트는 **묶음 프로젝트 라이브러리**를 만들 때 고려해야 할 유의점과 권장사항을 설명합니다.

- VI가 한 타겟에 대해 컴파일된 묶음 프로젝트 라이브러리를 호출하고, **이 VI를 다른 OS가 설치된 다른 타겟**에서 여는 경우, 묶음 프로젝트 라이브러리는 로드되지 않습니다.
- 묶음 라이브러리로 **디버그 빌드를 생성**하면, 각 VI의 블록 다이어그램을 볼 수 있습니다. 이렇게 블록 다이어그램을 사용해서 문제를 디버그할 수 있지만, 이 VI를 편집하거나, 변경 사항을 묶음 프로젝트 라이브러리에 저장할 수는 없습니다. 이 프로젝트 라이브러리를 가진 LabVIEW 프로젝트에서 변경 사항을 VI에 저장하십시오. 묶음 라이브러리를 업데이트하려면, 라이브러리를 다시 만들어야 합니다.
- 묶음 라이브러리에는 관련된 VI만 포함시키십시오. 묶음 라이브러리에서 VI를 열면, 이 묶음 라이브러리에 있는 모든 VI가 로드됩니다. 관련이 없는 VI를 포함시킬 경우, VI가 많아져 로드할 시간이 더 필요하므로 로드 시간이 증가합니다.
- 사용자가 생성한 묶음 라이브러리를 팔레트에 링크하기 위해서는 팔레트를 같은 버전, 타겟, OS에 묶음 라이브러리로 저장하십시오.
- LabVIEW는 최상위 레벨 라이브러리의 의존성을 묶음 라이브러리에 포함시키지 않지만 이는 빌드와 함께 반출되지 않으며 묶음 라이브러리에서 보이지도 않습니다. 묶음 라이브러리에서 의존성을 반출하려면 묶음 라이브러리를 빌드하기 전에 최상위 레벨 프로젝트 라이브러리에서 의존성을 배치시킵니다.
- **반출된 VI**는 타입 정의나 클래스의 입력이나 출력의 커넥터 팬을 가지고 있는 경우엔, 타입 정의나 최상위 프로젝트 라이브러리의 클래스를 포함합니다. 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 클릭하면, 묶음 라이브러리의 반출된 아이템만 대체하고 타입 정의와 클래스는 대체하지 않습니다. 그 결과로 묶음 라이브러리의 VI는 프로젝트 라이브러리를 묶음 라이브러리로 대체 경우 연결이 끊어 질 수도 있습니다.
- LabVIEW 프로젝트에서 라이브러리를 생성한 후, 묶음 라이브러리를 만들기 전에 **프로젝트 라이브러리의 LabVIEW 프로젝트를 별도로 생성**합니다. 프로젝트 라이브러리를 생성하고 프로젝트 라이브러리를 묶음 라이브러리로 대체하는, 바로 그 LabVIEW 프로젝트에서 묶음 라이브러리를 만드는 경우, 묶음 라이브러리를 다시 만들 수 없습니다.

추가 프로젝트 옵션

최근에 열어본 파일 접근하기

다음 단계를 따라 최근에 열어본 파일을 찾습니다.

1. 파일»최근 파일을 선택한 후 서브메뉴가 나타나면 파일 이름 .vi, .ctl, .vit, .ctt를 선택합니다. 파일»최근 프로젝트를 선택하여 최근에 열어본 프로젝트 파일 .lvproj를 찾습니다.
2. 파일 이름을 클릭하여 파일을 엽니다.

파일은 가장 최근에 열어본 파일부터 순서대로 나열됩니다. 리스트가 비어있는 경우, 최근 파일과 최근 프로젝트 메뉴 아이템은 비활성화됩니다.

두 개 또는 그 이상의 파일이 같은 파일 이름을 가지면서 다른 디렉토리에 위치하는 경우, 최근 파일 또는 최근 프로젝트 리스트는 전체 경로를 디스플레이합니다.

시작하기 윈도우 비활성화하기

LabVIEW를 시작할 때, 시작하기 윈도우가 나타납니다. 다음 단계를 따라 시작하기 윈도우 없이 LabVIEW를 실행합니다.

1. 도구»옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 환경을 선택합니다.
3. 시작할 때 시작하기 윈도우 건너뛰기 확인란에 확인 표시를 합니다.
4. 확인 버튼을 클릭합니다.

수동으로 백업 파일 복구하기

자동 복구 에러 대화 상자는 LabVIEW에서 에러가 발생하고 자동으로 저장된 파일을 복구할 수 없을 때 나타납니다.

다음 단계에 따라 수동으로 백업 파일 복구를 시도합니다.

1. 백업 파일을 사용자 컴퓨터에서 안전한 위치에 복사합니다.
2. 백업 파일 삭제 버튼을 클릭하여 LabVIEW 기본 데이터 디렉토리에서 백업을 삭제하고 계속해서 LabVIEW를 시작합니다.
3. LabVIEW에서 파일»열기를 선택하고 복사된 파일을 탐색하여 단계 1에서 복사한 파일을 엽니다.

LabVIEW 파일 확장자

다음 테이블은 LabVIEW가 생성하고 데이터를 저장하는데 사용하는 공통 파일 타입을 포함합니다.



노트

.aliases 파일을 삭제해도 프로젝트에 영향을 미치지 않습니다.

파일과 프로젝트 아이템 이름 바꾸기

LabVIEW의 여러가지 타입의 파일이나 프로젝트 아이템의 이름을 변경하려면 다음과 같이 서로 다른 단계를 따라야 합니다. 다음 중 하나의 방법을 사용하여 파일이나 프로젝트 아이템의 이름을 변경합니다:

- **VI—파일**» 다른 이름으로 저장을 선택하여 다른 이름으로 저장 대화 상자를 엽니다. 다른 이름으로 저장 대화 상자에서 이름 바꾸기 옵션을 선택합니다. 또한, VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 이름 바꾸기를 선택하여 VI의 이름을 변경할 수 있습니다. 이 옵션은 원본 VI를 닫고 삭제하며 새 이름으로 VI를 열기 때문에 디스크와 메모리에는 새 VI만 존재하게 됩니다.
- **LabVIEW 프로젝트 라이브러리, X컨트롤, LabVIEW 클래스—프로젝트 탐색기** 윈도우의 프로젝트 라이브러리, X컨트롤, 또는 LabVIEW 클래스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다른 이름으로 저장을 선택하여 다른 이름으로 저장 대화 상자를 엽니다. 다른 이름으로 저장 대화 상자에서 이름 바꾸기 옵션을 선택합니다. 이 옵션은 디스크와 메모리 상에 새 파일만 존재하는 것처럼 원본 파일을 닫고 삭제하며 새 이름으로 파일을 엽니다. 또한 프로젝트 라이브러리, X컨트롤 또는 LabVIEW 클래스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이름 바꾸기를 선택하여 새 이름을 입력합니다.
- **LabVIEW 프로젝트 — 프로젝트 탐색기** 윈도우에서, **파일**» 다른 이름으로 저장을 선택하여 다른 이름으로 저장 대화 상자를 엽니다. 다른 이름으로 저장 대화 상자에서 이름 바꾸기 옵션을 선택합니다.
- **공유 변수—프로젝트 탐색기** 윈도우에서, 공유 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이름 바꾸기를 선택하고, 공유 변수의 새 이름을 입력합니다. 또한, 프로젝트 탐색기 윈도우의 공유 변수를 더블 클릭하여 공유 변수 프로퍼티 대화 상자를 디스플레이하고 이름 텍스트 박스에 공유 변수의 새 이름을 입력할 수 있습니다.
- **버추얼 폴더—프로젝트 탐색기** 윈도우에서, 버추얼 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이름 바꾸기를 선택하고 폴더의 새 이름을 입력합니다.
- **빌드 스펙—프로젝트 탐색기** 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 빌드 스펙의 프로퍼티 대화 상자를 디스플레이합니다. 빌드 스펙 이름 텍스트 박스를 편집합니다. 이 텍스트 박스는 빌드 스펙의 타입에 따라 일반적으로 대화 상자의 기본 페이지에 위치합니다. 예를 들어, 소스 배포 프로퍼티 대화 상자의 정보 페이지에서 소스 배포 빌드 스펙 이름을 바꿀 수 있습니다.



노트

프로젝트 탐색기 윈도우에서 빌드 스펙을 숨긴 경우, 이에 접근하려면 반드시 **해당 아이템을 디스플레이**해야 합니다.

LabVIEW 사용자 정의하기

폰트 스타일 설정하기

프런트패널 또는 블록다이어그램에서 각각의 객체에 대한 폰트 특성을 설정하려면, 객체를 선택하고 도구 모음의 텍스트 셋팅 풀다운 메뉴로부터 폰트 대화 상자를 선택하여 폰트 선택 대화 상자를 디스플레이합니다. 이 대화 상자는 폰트 스타일 대화 상자과 같은 옵션을 포함합니다.

프런트패널 또는 블록다이어그램에서 모든 새로운 객체에 대한 폰트 특성을 설정하려면 선택한 모든 객체의 선택을 해제하고 도구 모음의 텍스트 셋팅 풀다운 메뉴로부터 폰트 대화 상자를 선택하여 패널 기본 폰트 대화 상자를 디스플레이합니다. 이 대화 상자는 폰트 스타일 대화 상자과 같은 옵션을 포함합니다.

다음 단계에 따라 LabVIEW, LabVIEW 대화 상자 또는 LabVIEW 메뉴와 팔레트에서 새 객체 모두에 대한 폰트 특성을 설정합니다:

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 옵션 대화 상자의 항목 리스트에서 환경을 선택합니다.
3. 폰트 섹션의 폰트 선택기 풀다운 메뉴에서 수정하려는 텍스트 타입을 선택합니다.
4. 기본 폰트 사용 확인란으로부터 확인 표시를 제거합니다.
5. 폰트 스타일 버튼을 클릭하여 폰트 스타일 대화 상자를 디스플레이합니다.

사용자 색 선택하기

다음 단계를 따라 사용자 색을 선택합니다.

1. 도구 팔레트에서, 색칠 도구의 전경색과 배경색 상자를 클릭합니다. 색 선택기가 나타납니다. 또한, 색칠 도구로 객체, 프런트패널 팬 또는 블록다이어그램 배경에서 마우스 오른쪽 버튼을 클릭하여 색 선택기에 접근할 수 있습니다.



팁

<Shift> 키를 누르고 마우스 오른쪽 버튼을 클릭하여 커서의 위치에서 도구 팔레트의 임시 버전을 디스플레이할 수 있습니다.

2. 색 선택기의 오른쪽 하단 모서리의 더 많은 색 버튼을 클릭한 후 OS의 색 유틸리티를 사용하여 특정 색을 선택합니다.
3. 빨간색, 녹색, 파란색 필드에 0에서 255사이의 값을 입력합니다. 빨간색, 녹색, 파란색을 다양한 비율로 혼합하여 보이는 스펙트럼에서 임의의 색을 얻을 수 있습니다. 값이 높을수록, 색의 명암이 더 짙습니다. 예를 들어, 빨간색 필드에 255를 입력하고 녹색과 파란색 필드에 0을 입력할 경우, 짙은 빨간색을 생성합니다.

또한, 색 상자 상수에서 사용자 색을 선택할 수 있습니다. 색 상자 상수를 사용할 경우, 수행 도구를 사용하여 색 상자 상수를 클릭하고 색 선택기를 디스플레이하여 단계 2를 진행합니다.

사용자 정의 색 설정하기

어플리케이션에서 VI의 일관된 색을 유지하기 위해서 프런트패널 배경과 객체를 **색 선택기**의 사용자 정의 색으로 칠합니다.

사용자 정의 색 이름과 연계된 색을 변경할 수 있고, 색 선택기에 디스플레이할 새로운 색을 정의할 수 있습니다. 하지만, 사용자 정의 색 이름과 연계된 색을 변경하는 경우, LabVIEW는 컨트롤 팔레트의 객체를 변경하는 데에는 적용하지 않습니다. 예를 들어, LED On 색을 기본적인 밝은 녹색 대신에 밝은 파랑으로 변경하고 프런트패널에 LED 컨트롤을 추가하는 경우, 컨트롤의 On 색은 기본적인 밝은 녹색입니다.

다음 단계를 따라 색 선택기에 나타나는 사용자 정의 색을 변경하거나 색 선택기에 디스플레이할 새로운 색을 정의합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 환경을 선택합니다.
3. 색 섹션에 있는 기본 색 사용 확인란에서 확인 표시를 제거하여, 확인란 아래 열거된 색 이름과 연관된 색을 변경합니다. 색 이름 옆의 색 상자를 클릭하여 색 선택기를 디스플레이하고 새 색을 선택합니다.
4. 사용자 색 버튼을 클릭하여 사용자 색 대화 상자를 디스플레이합니다.
5. 다음 방법 중 하나로 사용자 정의 색을 설정합니다:
 - 새로운 색을 정의하려면, 빈 색 이름 텍스트 박스에 이름을 입력하고 새로운 이름 옆의 색 상자를 클릭하여 그 이름과 연관된 색을 선택합니다. 색 선택기에 색을 디스플레이하려면 반드시 사용자 정의 색에 이름을 붙여야 합니다.
 - 색 이름과 연관된 색을 변경하려면, 색 이름 옆의 색 상자를 클릭하여 색 선택기를 디스플레이한 후 새로운 색을 선택합니다.
 - 색 선택기에서 사용자 정의 색을 제거하려면, 색 이름을 삭제합니다.
 - 기본 사용자 정의 색을 복구하려면, 대화 상자의 모든 텍스트 박스에서 색 이름을 삭제합니다.
6. 변경을 저장하기 위해 확인 버튼을 두 번 클릭합니다.

시스템 컬러 보기

시스템 색은 컴퓨터에 따라 다릅니다. 따라서 시스템 색을 사용하는 모든 VI는 현재 사용 중인 컴퓨터 고유의 시스템 색을 반영합니다. 시스템 색은 패널 & 객체, 윈도우, 하이라이트의 모양뿐만 아니라, 시스템 색 위에 나타나는 텍스트의 색상도 결정합니다.

사용할 수 있는 시스템 색을 보려면:

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 환경을 선택합니다.
3. 색 섹션의 기본 색 사용 확인란에서 확인 표시를 제거합니다.

4. 다음 색 상자 중 하나를 클릭하여 색 선택기를 디스플레이합니다. 색 선택기의 오른쪽 아래에서 사용할 수 있는 시스템 색을 볼 수 있습니다.



팁
색 선택기에 나타나는 시스템 색을 변경하려면 사용자 컴퓨터 OS의 색 유틸리티를 사용하십시오.

시스템 색에 대한 더 자세한 정보는 ni.com을 참조하십시오.

색 상자에서 색 설정하기

색 상자는 지정된 값에 상응하는 색을 디스플레이하는 숫자 컨트롤이거나 인디케이터입니다. 예를 들면, 색 상자를 사용하여 다른 조건, 즉 범위를 벗어난 값 등을 가리킵니다. 색 값은 RRGGBB의 형으로 16진수의 숫자로 표시됩니다. 처음 두 개의 자릿수는 빨간색 값을 제어합니다. 그 다음 두 개의 자릿수는 녹색 값을 제어합니다. 마지막 두 자리는 파란색 값을 제어합니다.

다음 단계를 따라 색 상자의 색을 설정합니다.

1. 수행 도구를 가지고 색 상자를 클릭하여 **색 선택기**를 디스플레이합니다. 또한, 수행 도구를 가진 색 상자에서 마우스 오른쪽 버튼을 클릭하여 색 선택기를 디스플레이합니다.



노트
수행 도구를 가지고 색 상자를 클릭하여 선택한 가장 최근의 색을 적용합니다.

1. 색 선택기에서 색을 선택합니다. 또는 색 선택기의 오른쪽 아래 가장자리의 더 많은 색 버튼을 클릭한 후, OS의 색 유틸리티를 사용하여 **추가 또는 사용자 정의 색에서 선택**할 수 있습니다.

LabVIEW 옵션 설정하기

다음 단계를 따라 LabVIEW를 사용자 정의합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 옵션의 항목을 선택합니다.
3. 변경하려는 옵션을 설정합니다.
4. 확인 버튼을 클릭합니다.



노트
일부 옵션의 변경사항은 다음 LabVIEW 시작부터 적용됩니다.

VI 검색 경로에 디렉토리 추가하기

LabVIEW는 메모리에 있는 VI를 처음으로 검색합니다. VI가 메모리에 있는 경우, LabVIEW는 해당 버전을 엽니다. VI가 메모리에 없는 경우, LabVIEW는 VI를 찾기 위해 VI 검색 경로를 사용합니다. LabVIEW가 VI를 찾지 못하는 경우, 파일 대화 상자에서 어느 이름의 VI도 선택할 수 있습니다. LabVIEW는 찾을 수 없는 VI의 모든 인스턴스를 선택한 VI로 대체합니다. 이미 메모리에 있는 VI의 버전과 다른 버전을 열고자 하는 경우, 버전은 반드시 다른 이름이어야 합니다.

다음 단계를 통해서 VI 검색 경로에 새 디렉토리를 추가합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 경로를 선택합니다.
3. 경로 선택기 풀다운 메뉴에서 VI 검색 경로를 선택합니다.
4. 기본 사용 확인란으로부터 확인 표시를 제거합니다.
5. 파일 대화 상자를 디스플레이하는 탐색 버튼 또는 경로 리스트를 포함하는 탐색 버튼 오른쪽의 풀다운 메뉴를 사용하여 LabVIEW 개발 환경이 검색할 디렉토리를 탐색합니다.
6. 폴더 선택 버튼을 클릭하여 디렉토리를 선택합니다.
7. 디렉토리를 선택한 후, LabVIEW는 경로를 탐색 버튼 오른쪽의 텍스트 박스에 경로를 삽입합니다. 이 텍스트 박스에서 경로를 편집할 수 있습니다. 경로를 선택하면, LabVIEW는 일반적으로 서브디렉토리를 포함하지 않은 디렉토리를 검색합니다. 별표를 새 경로 아이템에 추가하여 검색 계층구조를 만들 수 있습니다. 예를 들어, LabVIEW와 해당 서브디렉토리를 검색하기 위해서는 LabVIEW/*, LabVIEW *, 또는 LabVIEW:*를 입력합니다.
8. 앞에 삽입, 뒤에 삽입 또는 대체 버튼을 사용하여 경로 리스트에서 디렉토리를 추가하거나 이동합니다. 리스트의 경로를 다른 곳으로 다시 삽입하려는 경우, 제거 버튼을 클릭하여 제거된 경로를 텍스트 박스에 놓습니다.
9. 검색 경로에 새 디렉토리를 추가하는 것이 끝나면, 확인 버튼을 클릭합니다. 변경한 VI 검색 경로는 LabVIEW를 다시 시작할 때 적용됩니다.

기본 메뉴 바로 가기 복구하기

메뉴 아이템에서 기본 바로 가기를 변경한 경우, 기본 LabVIEW 메뉴 바로 가기를 복원할 수 있습니다. 다음 단계를 따라 기본 메뉴 바로 가기를 복구합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 메뉴 바로 가기를 선택합니다.
3. 메뉴 아이템 리스트에서, 기본 바로 가기를 복구하려는 메뉴 아이템을 선택합니다.
4. 복구 버튼을 클릭해서 선택한 메뉴 아이템의 기본 LabVIEW 바로 가기를 복구합니다.
5. 모든 메뉴 아이템의 기본 LabVIEW 바로 가기를 복구하려면, 모두 복구 버튼을 클릭합니다.
6. 확인 버튼을 클릭합니다.

메뉴 바로 가기 설정하기

다음 단계를 따라 편집 모드 VI 메뉴 아이템에 대한 키보드 바로 가기 키를 설정합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 메뉴 바로 가기를 선택합니다.
3. 메뉴 아이템 윈도우에서, 바로 가기를 만들려고 하는 메뉴 아이템을 선택합니다.
4. 바로 가기 키 텍스트 박스를 선택하고 바로 가기로 지정하려는 선택한 메뉴 아이템의 키보드의 키 조합을 누릅니다.
5. 이름을 변경하려면 설정 버튼을 클릭합니다.
6. 확인 버튼을 클릭합니다.

선택한 메뉴 아이템을 바로 가기로 지정하려는 키보드의 키 조합을 누르십시오. <문자나 숫자 키> 또는 <문자나 숫자 키-Shift>를 누르면 LabVIEW는 자동으로 <Ctrl>을 조합 키로 추가합니다. (macOS) LabVIEW는 자동으로 <Command>를 조합키로 추가합니다. 기능키 <F1> ~ <F24>의 경우, LabVIEW는 자동으로 <Ctrl> 키를 조합키로 추가하지 않지만 <Ctrl>을 조합키로 이용하여 이러한 기능키를 사용할 수 있습니다. 현재 LabVIEW는 <Alt>와의 키 조합을 바로 가기 키로 지원하지 않습니다.

LabVIEW가 지정한 키 조합을 선택했을 경우, 경고가 나타나며 LabVIEW가 설정 버튼을 비활성화시킵니다. 이미 다른 메뉴 아이템에 지정해 놓은 키 조합을 선택하는 경우, 충돌 경고가 나타내고 해당 키보드 바로 가기 키를 가진 메뉴 아이템이 디스플레이되나 설정 버튼은 계속 활성화되어 있습니다. 또한 기본 LabVIEW 메뉴 바로 가기로 **복구**할 수 있습니다.

LabVIEW는 비활성화된 메뉴 아이템의 키보드 바로 가기 키에 대해 응답하지 않습니다.



노트

(macOS) 기능키를 사용하는 키보드의 바로 가기 키는 Exposé가 기능키를 바로 가기 키로 배정하지 경우에만 작동합니다. LabVIEW는 macOS에서 <VolumeUp>, <VolumeDown> 및 <F15>부터 <F24>까지의 기능 키를 바로 가기 키로 지원하지 않습니다.

경로 옵션 설정하기

다음 단계를 따라 LabVIEW가 탐색하는 디렉토리의 순서를 지정합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 경로를 선택합니다.



노트

경로 이름의 포맷, 즉 콜론, 슬래시, 백슬래시의 사용은 다른 플랫폼에서 약간씩 다릅니다.

1. 풀다운 메뉴에서 보거나 편집하려면 경로 디렉토리를 선택합니다. 또한, **vi 검색 경로**에 디렉토리를 추가할 수 있습니다.
2. 기본 사용 확인란에서 확인 표시를 제거하여 경로 환경 옵션에 접근합니다.
3. 텍스트박스에 새 경로를 입력하거나 탐색 버튼을 클릭하여 디렉토리를 선택합니다.
4. 확인 버튼을 클릭합니다.



노트 임시 디렉토리, 기본 디렉토리, 기본 데이터 디렉토리 옵션의 모든 변경사항은 다음에 LabVIEW를 시작할 때 적용됩니다.

암호 변경하기(Windows)

다음 단계를 따라 현재 로그인 사용자의 암호를 변경합니다.

1. 도구 >> 보안 >> 암호 변경을 선택하여 NI 보안 암호 변경 대화 상자를 디스플레이합니다.
2. 이전 암호 텍스트 박스에서, 현재 로그인 사용자의 암호를 입력합니다.
3. 새 암호 텍스트 박스에서 사용자의 새로운 암호를 입력합니다.
4. 암호 확인 텍스트 박스에서 사용자의 새로운 암호를 입력합니다.
5. 확인 버튼을 클릭하여 암호를 변경합니다.

팔레트 사용자 정의하기

팔레트 항목 추가하기

내장된 팔레트 항목 외에, 다른 항목을 생성하여 팔레트 아이템을 구성할 수 있습니다. 다음 단계에 따라 컨트롤 또는 함수 팔레트의 최상위 레벨에 항목을 추가합니다.

1. 컨트롤 팔레트의 경우 labview\menus\Controls 디렉토리, 함수 팔레트의 경우 labview\menus\Categories 디렉토리의 위치에, 생성하려는 항목에 대한 폴더를 새로 생성합니다.
2. 항목에 포함시키려는 아이템을 새 폴더에 추가합니다. 함수 팔레트 항목의 아이템에는 VI, 함수, LLB, .mnu 파일이 포함될 수 있습니다. 컨트롤 팔레트 항목의 아이템에는 컨트롤과 .mnu 파일이 포함될 수 있습니다.



노트

dir.mnu 파일을 항목 폴더로 복사하지 마십시오. 항목을 추가할 때, LabVIEW는 필요한 경우 dir.mnu 파일을 생성합니다.

1. LabVIEW를 다시 시작합니다. LabVIEW는 팔레트가 추가된 항목을 포함하도록 업데이트합니다.

즐거찾기 항목에 팔레트 아이템 추가하기

즐거찾기 항목을 사용하여 함수 팔레트의 아이템 중 자주 사용하는 아이템을 묶어 그룹화합니다. 항목(표준), 항목(아이콘 및 텍스트), 아이콘, 아이콘과 텍스트 포맷을 사용하여 즐거찾기 항목에 아이템을 추가할 수 있습니다.

다음 단계를 따라 즐거찾기 항목에 팔레트나 객체를 추가합니다.

1. 고정된 함수 팔레트에서 하나의 객체를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 아이템을 즐거찾기에 추가를 선택합니다. 항목(표준)과 항목(아이콘 및 텍스트) 포맷에서는 팔레트를 확장하여 서브팔레트를 디스플레이한 후, 서브팔레트의 제목에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 즐거찾기에 서브팔레트 추가를 선택할 수 있습니다.
2. 즐거찾기 항목을 선택합니다. 선택한 아이템이 즐거찾기 항목에 나타나는 것을 확인합니다.

즐거찾기 항목의 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 즐거찾기로부터 아이템 제거 또는 즐거찾기로부터 서브팔레트 제거를 선택하여 아이템을 제거합니다. 항목(표준)과 항목(아이콘 및 텍스트) 포맷에서는 즐거찾기 구성 대화 상자를 사용하여 즐거찾기 항목에서 객체와 팔레트를 제거할 수도 있습니다. 즐거찾기 항목에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 즐거찾기 구성을 선택하여 즐거찾기 구성 대화 상자를 디스플레이합니다.

사용자와 인스트루먼트 드라이버 서브팔레트에 VI와 컨트롤을 추가하기

사용자 컨트롤과 사용자 라이브러리 팔레트에는 labview\user.lib에 있는 각 디렉토리, LLB, 또는 팔레트 파일 .mnu의 서브팔레트와 labview\userr.lib에 있는 각 파일의 아이콘이 포함되어 있습니다.

수정없이 프로젝트간에 사용 가능한 subVI의 경우에만 **subVI를 user.lib 디렉토리에 저장**합니다. user.lib에 있는 VI의 경로는 labview 디렉토리의 상대적인 위치에 있습니다. 다른 곳에 저장하는 subVI의 경로는 상위 VI를 기준으로 한 상대적인 경로입니다. 그러므로, 특수한 경우에 labview\user.lib에서 VI를 복사해서 수정해도 labview\user.lib에 위치한 SubVI의 경로는 변경되지 않습니다.



주의

사용자 VI 및 다른 파일을 **LabVIEW와 함께 설치되는 LLB 파일**에 저장하지 마십시오. 컨트롤과 VI를 컨트롤과 함수 팔레트에 추가하기 위해서는 사용자 컨트롤과 VI를 user.lib 디렉토리에 저장하십시오.

다음 단계를 따라 새 VI와 컨트롤을 사용자 팔레트에 추가합니다.

1. 컨트롤과 VI를 사용자 컨트롤 및 사용자 라이브러리 팔레트에 각각 추가하려면 이를 labview\user.lib 디렉토리에 저장합니다.
2. LabVIEW를 다시 시작합니다. LabVIEW는 labview\user.lib 디렉토리에 있는 컨트롤과 VI를 포함하도록 팔레트를 업데이트합니다.

인스트루먼트 드라이버 팔레트는 `labview\instr.lib` 디렉토리에 대응합니다. 이 디렉토리에 인스트루먼트 드라이버를 저장하여 함수 팔레트에 추가합니다.



노트

서브디렉토리 또는 LLB를 `labview\user.lib` 또는 `labview\instr.lib` 디렉토리에 추가하고 서브디렉토리 또는 파일 이름이 밑줄로 시작하는 경우, 서브디렉토리 또는 파일의 내용은 사용자 라이브러리 또는 인스트루먼트 드라이버 팔레트에 나타나지 않습니다.

함수 또는 컨트롤 팔레트에서 VI나 객체 추가하기

VI, 사용자 함수 또는 사용자 컨트롤을 함수나 컨트롤 팔레트에 추가한 후, 해당 객체를 새 VI 또는 기존 VI에 추가할 수 있습니다.

다음 단계에 따라 함수 팔레트에서 VI나 함수를 추가합니다.

1. VI나 함수에 추가하고 싶은 VI를 생성하거나 엽니다.
2. 함수 팔레트에서 VI나 함수를 추가한 위치를 탐색합니다.
3. 해당 VI 또는 함수의 아이콘을 클릭하고 VI의 블록다이어그램에 놓습니다.

다음 단계에 따라 컨트롤 팔레트에서 VI나 컨트롤을 추가합니다:

1. 해당 VI나 컨트롤을 추가하고 싶은 VI를 생성하거나 엽니다.
2. 컨트롤 팔레트에서 VI나 컨트롤을 추가한 위치를 탐색합니다.
3. 해당 VI나 컨트롤의 아이콘을 클릭하고 VI의 프런트패널에 놓습니다.



노트

함수 또는 컨트롤 팔레트에서 추가한 VI, 함수 또는 컨트롤이 프런트패널과 블록다이어그램에 대한 내용을 포함하고 있으면, LabVIEW는 그러한 내용을 프런트패널 및 블록다이어그램에 모두 추가합니다.

관련 정보

함수 또는 컨트롤 팔레트에 VI나 객체 추가하기

함수 또는 컨트롤 팔레트에 VI나 객체 추가하기

함수나 컨트롤 팔레트 또는 서브 팔레트에 VI 및 객체를 추가하여 새 VI 또는 기존 VI에 사용합니다. 서브팔레트는 VI, 함수, 컨트롤, 서브팔레트를 포함할 수 있습니다. 서브팔레트는 다른 위치에 있는 VI를 포함할 수도 있습니다.

함수 팔레트에 VI나 함수 추가하기

함수 팔레트에 VI 블록 다이어그램이나 사용자 정의된 함수를 추가할 수 있습니다. 이후 SubVI를 부르지 않고, 함수 팔레트에서 바로 다른 VI의 블록 다이어그램에 있는 VI의 블록 다이어그램 내용을 불러와 놓을 수 있습니다.

다음 단계를 따라 함수 팔레트에 VI나 함수를 추가합니다.

1. 함수 팔레트에 추가할 VI나 함수를 생성합니다. 파일을 저장하고 닫습니다.
2. 도구>>고급>>팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
3. 함수 팔레트에 VI나 함수를 추가하려는 위치를 탐색합니다.
4. 팔레트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 삽입>>VI를 선택합니다.
5. 함수 팔레트에 추가할 VI나 함수를 탐색합니다.
6. 팔레트에 추가하려는 VI나 함수를 선택하고 열기 버튼을 클릭합니다.
7. VI나 함수의 아이콘에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 VI 내용 놓기를 선택합니다.
8. 컨트롤과 함수 팔레트 세트 편집 대화 상자에 있는 변경 저장을 클릭하여 팔레트 변경사항 미리보기 대화 상자를 엽니다.
9. 계속을 클릭하여 변경 내역을 수락하고 컨트롤과 함수 팔레트 세트 편집 대화 상자를 닫습니다.

컨트롤 팔레트에 VI나 컨트롤 추가하기

컨트롤 팔레트에 VI 프런트패널이나 사용자 정의된 컨트롤을 추가할 수 있습니다. 사용자 컨트롤을 추가하는 것과는 달리, VI를 사용하면 여러 컨트롤이나 장식을 한 번에 프런트패널에 추가할 수 있습니다.



노트

컨트롤 팔레트에 사용자 정의된 장식을 추가하려면 VI를 생성한 후 그 VI의 프런트패널에 장식을 추가합니다. 이후 해당 VI를 컨트롤 팔레트에 추가할 수 있습니다.

다음 단계를 따라 컨트롤 팔레트에 VI나 컨트롤을 추가합니다.

1. 컨트롤 팔레트에 추가할 VI나 컨트롤을 생성합니다. 파일을 저장하고 닫습니다.
2. 도구>>고급>>팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
3. 컨트롤 팔레트에 VI나 컨트롤을 추가하려는 위치를 탐색합니다.
4. 팔레트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 삽입>>사용자 컨트롤을 선택합니다.
5. 팔레트에 추가할 VI나 컨트롤을 탐색합니다.

노트 .vi 파일 확장자가 검색에 포함되도록 하려면 파일 타입 풀다운 메뉴에서 모든 파일(**)을 선택해야 합니다.

6. 팔레트에 추가하려는 VI나 컨트롤을 선택하고 열기 버튼을 클릭합니다.
7. 컨트롤과 함수 팔레트 세트 편집 대화 상자에 있는 변경 저장을 클릭하여 팔레트 변경사항 미리보기 대화 상자를 엽니다.
8. 계속을 클릭하여 변경 내역을 수락하고 컨트롤과 함수 팔레트 세트 편집 대화 상자를 닫습니다.

팔레트 세트 자동 업데이트하기

LabVIEW는 LLB에 파일을 추가하거나 제거할 때 LLB에 링크된 팔레트를 자동으로 업데이트합니다. 디렉토리에 새 파일을 더하거나 디렉토리에서 파일을 제거하는 경우 팔레트가 자동으로 업데이트 되도록 설정할 수 있습니다.

다음 단계를 따라 디렉토리의 내용을 기반으로 업데이트하기 위해 .mnu 파일을 설정합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 팔레트의 아이콘에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 디렉토리에 동기화를 선택합니다.
3. 팔레트 세트와 동기화하려는 디렉토리를 탐색하고 선택합니다. National Instruments는 동일 디렉토리에 .mnu 파일을 가질 것을 권장하지만, 임의의 디렉토리를 선택하여 팔레트 세트를 동기화할 수도 있습니다.



노트

서브디렉토리 또는 .llb 파일을 동기화할 디렉토리에 추가하고 서브디렉토리 또는 파일 이름이 밑줄(_)로 시작하는 경우, 서브디렉토리 또는 파일의 내용은 팔레트에 나타나지 않습니다.

아이템이 디렉토리에 동기화된 경우, 아이템에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 동기화 경로 디스플레이를 선택하여 소스 디렉토리의 경로를 디스플레이할 수 있습니다.

아이템이 디렉토리에 동기화된 경우, 아이템을 팔레트에서 삭제할 수 없습니다. 디렉토리에 동기화된 아이템을 숨기려면, 아이템에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 동기화된 아이템 숨기기를 선택합니다. 숨겨진 아이템은 팔레트에 나타나지 않으며 **팔레트를 검색**할 때 포함되지 않습니다.

팔레트 포맷 변경하기

다음 단계를 따라 모든 팔레트를 다른 포맷으로 변경합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이하고 항목 리스트에서 컨트롤/함수 팔레트를 선택하여 컨트롤/함수 팔레트 페이지를 디스플레이합니다. 또한 컨트롤 또는 함수 팔레트 도구 모음에서 사용자 정의 버튼을 클릭한 후 바로 가기 메뉴에서 옵션을 선택하여 이 페이지를 디스플레이할 수 있습니다.

2. 팔레트 풀다운 메뉴에서 항목(표준), 항목(아이콘 및 텍스트), 아이콘, 아이콘과 텍스트, 텍스트, 트리와 같은 팔레트 포맷을 선택합니다.
3. 확인 버튼을 클릭합니다. 컨트롤과 함수 팔레트는 선택한 팔레트 포맷으로 변경됩니다.

컨트롤 팔레트 또는 함수 팔레트만 다른 포맷으로 변경하려면, 해당 팔레트에서 사용자 정의 버튼을 클릭하고 이 팔레트 보기 바로 가기 메뉴에서 포맷을 선택합니다.



노트

업그레이드할 때, LabVIEW는 이전에 설치한 LabVIEW 버전으로부터 팔레트 포맷을 복구하려고 시도합니다.

서브팔레트

.NET 서브팔레트 생성하기

.NET 컨트롤을 사용자 컨트롤로 변환하고 팔레트에 추가할 수 있습니다.



노트

LabVIEW에서 .NET 객체를 생성하고 이 객체와 통신하려면 LabVIEW와 함께 설치되는 **.NET CLR 4.0**이 필요합니다. .NET 2.0 혼합 모드 어셈블리를 로드하려면 **.NET 2.0 설정 파일**을 사용해야 합니다. LabVIEW에서 .NET 제한에 대한 더 자세한 정보는 **LabVIEW에서 .NET를 사용할 때의 요구조건**을 참조하십시오. National Instruments는 **LabVIEW 프로젝트** 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

다음 단계를 따라 팔레트에 .NET 컨트롤을 추가합니다.

1. 도구 >> 반입 >> .NET 컨트롤을 팔레트로 선택하여 .NET 컨트롤을 팔레트에 추가 대화 상자를 디스플레이합니다.
2. 어셈블리 풀다운 메뉴에서 어셈블리를 선택합니다. 또한 탐색 버튼을 클릭하여 프라이빗 어셈블리에 접근할 수 있습니다. .NET 어셈블리는 .dll 파일 타입입니다. 프라이빗 어셈블리를 선택한 후 다시 이 대화 상자를 띄우면 어셈블리 풀다운 메뉴에 선택한 어셈블리가 나타납니다.



노트 프라이빗 어셈블리를 사용하는 VI를 다른 폴더나 컴퓨터로 이동하는 경우, 관련된 프라이빗 .NET 어셈블리 파일을 **LabVIEW가 검색할 수 있는 서브디렉토리**로 이동하거나 파일을 Global Assembly Cache(GAC)에 저장해야 합니다. 프라이빗 어셈블리를 사용하는 VI를 공유 라이브러리 또는 독립 어플리케이션에 만들려는 경우, LabVIEW는 관련된 프라이빗 .NET 어셈블리 파일들을 라이브러리 또는 어플리케이션과 같은 디렉토리의 data 서브디렉토리에 복사합니다.

1. 컨트롤 리스트에서 변환하려는 .NET 컨트롤을 선택합니다.

2. 대상 텍스트 박스에 컨트롤을 저장할 디렉토리 또는 .llb 파일을 입력합니다. 기본 대상은 labview\menus\Controls\DotNet & ActiveX 디렉토리로, 추가한 컨트롤을 .NET & ActiveX 팔레트에 놓습니다. 또한, 탐색 버튼을 클릭하여 컨트롤을 저장하려는 디렉토리를 탐색할 수 있습니다.
3. 확인 버튼을 클릭하여 컨트롤을 팔레트에 추가합니다.



주의 사용자 VI 및 다른 파일을 **LabVIEW와 함께 설치되는 .llb 파일**에 저장하지 마십시오.

ActiveX 서브팔레트 생성하기

ActiveX 컨트롤을 사용자 컨트롤로 변환하고 팔레트에 추가할 수 있습니다.

다음 단계를 따라 팔레트에 ActiveX 컨트롤을 추가합니다.

1. 도구 >> 반입 >> ActiveX 컨트롤을 팔레트로 선택하여 컴퓨터에 등록된 ActiveX 컨트롤을 나열하는 ActiveX 컨트롤을 팔레트에 추가 대화 상자를 디스플레이합니다.
2. 컨트롤 리스트에서 하나 또는 그 이상의 컨트롤을 선택한 후 확인 버튼을 클릭합니다.
3. 나타난 파일 대화 상자에서 컨트롤을 저장하려는 디렉토리 또는 .llb 파일을 선택합니다. 기본 대상은 labview\menus\Controls\DotNet & ActiveX 디렉토리로, 추가한 컨트롤을 .NET & ActiveX 팔레트에 놓습니다.



주의
사용자 VI 및 다른 파일을 **LabVIEW와 함께 설치되는 .llb 파일**에 저장하지 마십시오.

서브팔레트 설명 생성하기와 편집하기

컨트롤과 함수 팔레트의 서브팔레트에 대한 설명을 생성하여 서브팔레트에 대한 내용과 목적을 설명하고, 사용자에게 선택할 수 있는 아이템에 대한 설명을 제공합니다. 커서를 컨트롤과 함수 팔레트의 서브팔레트 아이콘 위로 움직이면, 서브팔레트에 대한 설명이 기본 도움말 윈도우에 나타납니다. 커서를 서브팔레트의 빈 공간 위로 움직이면, 설명이 기본 도움말 윈도우에 나타납니다.

다음 단계를 따라 서브팔레트 설명을 생성하고 편집합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 서브팔레트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 팔레트 문서 편집을 선택합니다.
3. 팔레트 설명 텍스트 박스에서 서브팔레트에 대한 설명을 입력 또는 편집합니다.
4. (옵션) HTML 파일 또는 컴파일된 도움말 파일을 **서브팔레트와 연결**합니다.

5. 확인 버튼을 클릭합니다.
6. 컨트롤과 함수 팔레트 세트 편집 대화 상자에 있는 변경 저장 버튼을 클릭하여 팔레트 변경사항 미리보기 대화 상자를 열어 변경을 저장합니다.

팔레트 세트에서 서브팔레트 생성하기

다음 단계를 따라 컨트롤 또는 함수 팔레트에 새로운 빈 서브팔레트를 삽입합니다. 서브팔레트 생성 후에, **서브팔레트에 객체를 추가**할 수 있습니다.

1. 팔레트 세트에 추가하려는 각각의 새로운 객체에 대해 **아이콘을 생성**합니다.
2. 도구>>고급>>팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
3. 팔레트에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 삽입>>서브팔레트를 선택하여 서브팔레트 삽입 대화 상자를 디스플레이합니다.
4. 새 팔레트 파일 생성을 선택하여 새로운 빈 서브팔레트에 삽입하고 확인 버튼을 클릭합니다.
5. 나타나는 파일 대화 상자에 .mnu 파일에 사용할 이름을 입력합니다. .mnu 확장자를 파일 이름에 추가하여 팔레트로 표시합니다. 새로 생성한 서브팔레트는 서로 다른 .mnu 파일에 따로 저장해야 합니다.
6. 확인 버튼을 클릭합니다. 메뉴 아이템 팔레트 편집 대화 상자가 나타납니다.
7. 새로운 서브팔레트의 이름을 입력한 후 확인 버튼을 클릭합니다. 새롭고 빈 서브팔레트가 팔레트에 나타납니다.

LabVIEW가 찾을 수 없는 아이템만을 포함하는 빈 서브팔레트 또는 서브팔레트는 팔레트를 편집할 때만 보입니다. 팔레트가 아이콘 포맷을 사용할 경우, 팔레트의 물음표는 LabVIEW가 팔레트의 객체를 찾을 수 없음을 나타냅니다. 팔레트가 텍스트 포맷을 사용하는 경우, 고정된 팔레트의 X 표시와 임시 팔레트의 비활성화된 객체는 LabVIEW가 팔레트에서 객체를 찾을 수 없음을 나타냅니다.

팔레트에서 빈 슬롯 삭제하기

다음 단계를 따라 컨트롤 또는 함수 팔레트에서 빈 슬롯을 삭제합니다.

1. 도구>>고급>>팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 컨트롤 또는 함수 팔레트의 빈 공간에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 빈 슬롯 삭제를 선택합니다.
3. 팔레트에서 빈 슬롯을 삭제한 후에, 컨트롤과 함수 팔레트 세트 편집 대화 상자에 있는 변경 저장 버튼을 클릭하여 팔레트 변경사항 미리보기 대화 상자를 열고 변경을 저장합니다. 취소 버튼을 클릭하여 현 편집 세션의 변경을 취소하고 이 대화 상자를 닫습니다.

팔레트 세트 편집하기

다음 단계에 따라 컨트롤과 함수 팔레트를 편집합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 컨트롤 또는 함수 팔레트에서 마우스 오른쪽쪽을 클릭하고 바로 가기 메뉴의 옵션에서 원하는 것들을 선택합니다. 팔레트의 빈 공간에서 마우스 오른쪽 버튼을 클릭하여 객체, **행 또는 열**을 삽입하고, 표준 메뉴 보기를 토글하거나, 팔레트에서 **빈 슬롯을 삭제**합니다. 아이콘에서 마우스 오른쪽 버튼을 클릭하여 **서브팔레트를 이동**하거나, 서브팔레트를 삭제하거나, 아이콘을 복사하거나, 서브팔레트 아이콘을 편집하거나, 서브팔레트 이름을 변경하거나, 파일이 **디렉토리의 내용을 기반으로 업데이트**되도록 설정하거나, **아이콘 짧은 이름**을 편집합니다. 객체를 움직이기 위해, 팔레트의 새 위치로 그것을 끌어옵니다.
3. 팔레트를 편집한 후에, 컨트롤과 함수 팔레트 세트 편집 대화 상자에 있는 변경 저장 버튼을 클릭하여 팔레트 변경사항 미리보기 대화 상자를 열고 변경사항을 저장합니다. 취소 버튼을 클릭하여 현 편집 세션의 변경을 취소하고 이 대화 상자를 닫습니다.

특정한 타겟에 대한 팔레트 세트를 편집하려면, 수정하려는 팔레트 세트 타겟의 VI에서 도구 >> 고급 >> 팔레트 세트 편집을 선택합니다.

팔레트를 편집하는 경우, LabVIEW는 편집한 내용을 labview\menus 디렉토리에 저장합니다. LabVIEW는 편집 사항을 원본 팔레트의 복사본에 저장합니다. 원본 팔레트를 보호하면 원본 팔레트를 손상시키지 않고도 팔레트를 실험해 볼 수 있습니다. 원본 팔레트로 돌아가려면 컨트롤과 함수 팔레트 편집 대화 상자에서 기본으로 복구 버튼을 클릭합니다.

컨트롤이나 함수 팔레트에서 보이는 팔레트 항목을 편집하려면, 팔레트 도구 모음에서 사용자 정의 버튼을 클릭한 후, 바로 가기 메뉴에서 팔레트 보기 변경을 선택하여 팔레트 보기 변경 대화 상자를 디스플레이합니다. 또한 고정된 팔레트의 아래에 있는 팔레트 보기 변경 링크를 클릭하여 팔레트 보기 변경 대화 상자를 디스플레이할 수 있습니다.



노트

묶음 프로젝트 라이브러리 내의 팔레트를 편집하려면, 팔레트를 변경한 후, 묶음 라이브러리를 다시 만들어 변경 사항을 라이브러리에 업데이트합니다.

팔레트 편집 VI를 사용하여 컨트롤 또는 함수 팔레트 세트를 프로그램식으로 편집할 수도 있습니다.

팔레트 세트에서 아이콘 짧은 이름 편집하기

다음 단계를 따라 팔레트 포맷이 아이콘과 텍스트일 때 각 아이콘 아래에 나타나는 텍스트 라벨을 설정합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.

2. 팔레트의 아이콘에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 짧은 이름 편집을 선택하여 팔레트 아이템 이름 편집 대화 상자를 디스플레이합니다.
3. 아이콘 아래에 나타내려는 텍스트 라벨을 입력한 후 확인 버튼을 클릭합니다.

팔레트 세트에서 행이나 열을 삽입하기

다음 단계를 따라 컨트롤과 함수 팔레트에 행 또는 열을 삽입합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 팔레트의 빈 공간에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 삽입 >> 빈 슬롯 또는 삽입 >> 빈 행을 선택합니다.

또한, 객체를 클릭하고 그것을 끌어서 팔레트의 오른쪽이나 아래쪽에 놓으면서 <Shift> 키를 눌러서 행이나 열을 생성할 수 있습니다.

행 또는 열이 팔레트의 끝에 나타나고 숨겨지거나 LabVIEW가 찾을 수 없는 동기화된 아이콘만 포함하는 경우, 행 또는 열은 팔레트를 편집할 때만 나타납니다. 팔레트가 아이콘 포맷을 사용할 경우, 팔레트의 물음표는 LabVIEW가 팔레트의 객체를 찾을 수 없음을 나타냅니다. 팔레트가 텍스트 포맷을 사용하는 경우, 고정된 팔레트의 X 표시와 임시 팔레트의 비활성화된 객체는 LabVIEW가 팔레트에서 객체를 찾을 수 없음을 나타냅니다.

HTML 파일 또는 컴파일된 도움말 파일에 서브팔레트 링크하기

컨트롤 팔레트와 함수 팔레트의 서브팔레트를 HTML 파일 또는 [컴파일된 도움말 파일](#)에 링크할 경우, 커서를 서브팔레트 위로 움직일 때 또는 커서를 서브팔레트의 열린 공간 위로 움직일 때 해당 파일에 대한 링크가 기본 도움말 윈도우에 나타납니다.

다음 단계를 따라 서브팔레트를 HTML 파일 또는 컴파일된 도움말 파일에 링크합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 서브팔레트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 팔레트 문서 편집을 선택하여 팔레트 문서 대화 상자를 디스플레이합니다.
3. 탐색 버튼을 클릭합니다.
4. 링크하려는 .chm, .hlp, .htm, 또는 .html 파일을 탐색한 후 열기 버튼을 클릭합니다. 파일 경로가 도움말 경로 텍스트 박스에 나타납니다.
5. 도움말 경로가 .htm 또는 .html 파일에 대한 경로 또는 [기호형 경로](#)를 포함할 경우, LabVIEW는 도움말 태그 텍스트 박스를 무시합니다. 도움말 경로가 컴파일된 도움말 파일(.chm 또는 .hlp)에 대한 경로를 포함할 경우, 도움말 태그 텍스트 박스를 사용하여 서브팔레트를 해당 도움말 파일의 특정 주제에 링크합니다.
 - .chm 파일의 경우, .htm 또는 .html 파일 이름을 입력하거나 도움말 태그 텍스트 박스에 인덱스 키워드를 입력할 수 있습니다. .htm 또는 .html

파일 이름은 반드시 HTML 도움말 프로젝트의 컴파일되지 않은 개별 HTML 파일의 이름과 일치해야만 합니다. 인덱스 키워드는 반드시 HTML 도움말 프로젝트의 인덱스에 있는 키워드와 일치해야 합니다. HTML 파일 안에서 북마크를 링크하려면, 파일 이름 끝에 #를 추가하고 뒤에 북마크 이름을 넣습니다. Windows의 .chm 파일 및 macOS와 Linux의 컴파일되지 않은 HTML 파일에 링크를 하려면, 도움말 태그 텍스트 박스에 HTML 파일 이름을 입력합니다. (Windows) LabVIEW는 .chm 파일 내에 있는 HTML 파일에 링크를 겁니다. (macOS와 Linux) LabVIEW는 HTML 파일에 링크를 걸며, 이 때 HTML 파일은 반드시 help\html\help.chm 디렉토리에 있어야 합니다. 여기서 help.chm은 4단계에서 도움말 경로 텍스트 박스에 입력한 .chm 파일 이름입니다.

- .hlp 파일에서는 도움말 태그 텍스트 박스에 인덱스 키워드를 입력할 수 있습니다. 인덱스 키워드는 반드시 WinHelp 또는 HyperHelp 프로젝트의 인덱스에 있는 키워드와 일치해야 합니다.
6. 확인 버튼을 클릭합니다.
 7. 컨트롤과 함수 팔레트 세트 편집 대화 상자에 있는 변경 저장 버튼을 클릭하여 팔레트 변경사항 미리보기 대화 상자를 열어 변경을 저장합니다. 커서를 서브팔레트 아이콘 또는 서브팔레트의 열린 공간 위로 움직일 때, 파란색 상세 도움말 링크가 기본 도움말 윈도우에 나타나고, 상세 도움말 버튼이 활성화됩니다.
 8. 파란색 링크 또는 기본 도움말 윈도우의 상세 도움말 버튼을 클릭하여 HTML 또는 컴파일된 도움말 파일에 대한 서브팔레트의 링크를 테스트합니다.

팔레트 세트에서 서브팔레트 이동하기

다음 단계를 따라 컨트롤과 함수 팔레트에서 서브팔레트를 이동합니다.

1. 도구 >> 고급 >> 팔레트 세트 편집을 선택하여 컨트롤과 함수 팔레트 세트 편집 대화 상자를 디스플레이합니다.
2. 서브팔레트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 서브팔레트 이동을 선택합니다.
3. 서브팔레트를 붙여넣으려는 팔레트를 클릭합니다.

또한, 객체를 클릭하여 그것을 끄는 것처럼 <Shift> 키를 눌러서 서브팔레트를 움직일 수 있습니다.

팔레트 항목 구성하기

항목(표준)과 항목(아이콘 및 텍스트) 포맷에서 항목 순서를 변경할 수 있습니다. 고정된 팔레트에서, 다음 방법을 사용하여 항목이 팔레트에서 나타나는 순서를 변경합니다:

- 항목에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 이 항목을 위로 이동 또는 이 항목을 아래로 이동을 선택합니다.
- 항목 왼쪽의 이중 라인을 클릭하고 화살표를 항목의 새 위치로 끕니다.

- 항목에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 위쪽으로 이동(기본으로 확장)을 선택하여 항목을 팔레트의 위로 이동합니다. 기본으로, LabVIEW는 가장 위의 항목을 확장합니다.

팔레트 세트 공유하기

팔레트 세트를 편집한 후, 사용자 정의된 팔레트를 다른 사용자와 공유할 수 있습니다.

1. 기본 데이터 디렉토리로부터 labview\menus 디렉토리를 복사합니다.
2. 사용자가 복사본을 사용자 기본 데이터 디렉토리에 놓도록 합니다.

사용자가 LabVIEW를 다시 시작할 때, 사용자 정의된 팔레트를 이용할 수 있습니다.

팔레트 항목 보이기와 숨기기

컨트롤 또는 함수 팔레트에서 어떤 항목을 보이거나 숨길지 지정할 수 있습니다. 고정된 팔레트에서는 다음 방법을 사용하여 보일 항목과 숨길 항목을 지정합니다.

- 팔레트 도구 모음에서 사용자 정의 버튼을 클릭하고 바로 가기 메뉴에서 팔레트 보기 변경을 선택하여 팔레트 보기 변경 대화 상자를 디스플레이합니다. 이 대화 상자에서 보려는 항목을 선택할 수 있습니다. LabVIEW는 선택되지 않은 항목을 숨깁니다.
- 또한 고정된 컨트롤이나 함수 팔레트의 아래에 있는 팔레트 보기 변경 링크를 클릭하여 팔레트 보기 변경 대화 상자를 디스플레이할 수 있습니다.
- 컨트롤 또는 함수 팔레트의 아래에 있는 이중 화살표를 클릭하면 현재 팔레트에서 숨겨진 항목을 디스플레이할 수 있습니다.



노트

숨겨진 항목은 모든 팔레트 보기 포맷에서 숨겨집니다. 모든 보기 포맷에서 항목을 보이거나 숨길 수 있습니다.

파일 복구에 대한 텍스트 비교 설정하기

다음 단계를 따라 복구된 프로젝트.lvproj, 프로젝트 라이브러리.lvlib, X컨트롤.xctl 및 LabVIEW 클래스.lvclass 파일에 대한 텍스트 비교를 설정합니다.



노트

LabVIEW가 텍스트 비교를 설정하기 전에 앞에서 나열된 파일 타입 중 하나를 자동으로 복구하는 경우, LabVIEW는 복구할 파일 선택 윈도우에서 백업을 원래 것과 비교 버튼을 비활성화합니다.

1. LabVIEW 데이터 디렉토리에서, 새로운 텍스트 파일(LVAutoSave_Text_Compare.txt)을 생성합니다.

2. **포맷 지정자 구문**을 사용하여 복구한 파일 비교에 사용할 타사 텍스트 비교 소프트웨어를 호출하는 문자열을 입력합니다. 따옴표안에 텍스트 비교 실행 파일에 대한 경로를 입력한 후, 문자열 "%s" "%s"를 입력하십시오. 예를 들어, Windows에서 C:\ 드라이브에 있는 프로그램을 호출하려는 경우, 다음 텍스트를 입력하십시오. 여기서 xxx는 텍스트 비교에 사용할 실행 파일의 이름입니다. "C:\\Program Files\\폴더 이름\\xxx.exe" "%s" "%s"
3. 다른 텍스트를 입력하거나 캐리지 리턴하지 마십시오. LVAutoSave_Text_Compare.txt를 닫고 파일을 닫습니다.

LabVIEW를 사용한 소스 컨트롤

소스 컨트롤을 사용하기 위해 LabVIEW 설정하기

LabVIEW와 함께 소스 컨트롤을 사용하기 전에, 사용자가 **선택**한 타사 소스 컨트롤 제공자와 함께 작동하도록 LabVIEW를 설정해야 합니다. LabVIEW와 작동하는 가장 최근의 타사 소스 컨트롤 제공자 리스트는 기술지원 데이터베이스를 참조하십시오.



노트

소스 컨트롤은 Professional Development System에서만 사용 가능합니다.

다음 단계를 따라 LabVIEW를 소스 컨트롤 제공자와 함께 작동하도록 설정합니다.

1. LabVIEW를 실행하는 컴퓨터에 타사 소스 컨트롤 제공자가 설치되어 있으며 소프트웨어 프로젝트의 필수사양을 충족시키기 위해 설정되었는지 확인합니다. 추가적인 도움은 소스 컨트롤 관리자와 상의하십시오.
2. LabVIEW 프로젝트 또는 새 VI에서, 도구 >> 소스 컨트롤 >> 소스 컨트롤 설정을 선택하여 옵션 대화 상자의 소스 컨트롤 페이지를 디스플레이합니다. 시작하기 윈도우에서 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이한 후, 소스 컨트롤 항목을 클릭하여 소스 컨트롤 페이지를 디스플레이합니다.
3. 소스 컨트롤 제공자 이름 풀다운 메뉴에서 LabVIEW와 함께 사용하려는 소스 코드 컨트롤 제공자를 선택합니다. 타사 소스 컨트롤 제공자의 대화 상자가 나타나 소스 컨트롤 아래의 LabVIEW 파일에 대한 연결, 사용자 정보, 기타 셋팅을 설정할 수 있습니다. 예를 들어, Perforce 명령 라인을 선택하는 경우, Perforce 프로젝트 옵션 대화 상자가 나타납니다. (Windows) LabVIEW는 Windows 레지스트리를 스캔하여 어떤 소스 컨트롤 제공자가 설치될 것인지 결정하고 해당 정보를 이용하여 소스 컨트롤 제공자 이름 풀다운 메뉴에 채웁니다. (macOS 및 Linux) LabVIEW는 Perforce가 설치되어 있는지를 확인하기 위해 환경 변수에 대한 쿼리를 실행합니다. LabVIEW는 로그인 스크립트를 실행하기 전에 기본 환경 경로를 검색합니다. 설치되어 있는 경우 소스 컨트롤 제공자 이름 풀다운 메뉴에 Perforce 명령 라인 옵션이 나타납니다.
4. 소스 컨트롤 제공자 대화 상자에서 선택한 소스 컨트롤 프로젝트 셋팅이 소스 컨트롤 프로젝트 텍스트 박스에 나타납니다. 변경 버튼을 클릭하여 셋팅을 다시 설정합니다. 제공자가 이 기능을 지원하지 않는 경우, 변경 버튼은 비활성화됩니다.
5. 고급 버튼을 클릭하여 소스 컨트롤 제공자 특징의 고급 옵션 설정 옵션을 설정합니다. 소스 컨트롤 제공자의 대화 상자가 나타납니다. 예를 들어, Perforce 명령 라인 인터페이스를 선택하는 경우, Perforce 설정 옵션 대화 상자가 나타납니다. 제공자가 이 기능을 지원하지 않는 경우, 고급 버튼은 비활성화됩니다.

- 원하는 소스 컨트롤 디스플레이와 입력요청 옵션을 설정합니다. 제공자 대화 상자를 닫습니다.
- 확인 버튼을 클릭하여 설정 셋팅을 저장하고 옵션 대화 상자를 닫습니다.



팁 소스 컨트롤에 저장하는 VI 및 다른 파일 타입으로부터 **컴파일된 코드를 분리하는 것**을 고려해봅니다. VI를 편집하면, 기본적으로 LabVIEW는 변경된 VI를 호출하는 모든 VI를 자동으로 다시 컴파일하기 때문에, 저장되지 않은 변경사항이 호출 VI에 생깁니다. 호출 VI가 소스 컨트롤에 저장된 경우, 이를 체크아웃하고 나서 다시 체크인해야 합니다. 그러나 **VI 계층구조** 내에 있는 VI에서 컴파일된 코드를 분리하는 경우, VI 하나가 변경되었을 때 VI 계층구조의 다른 곳에 저장되지 않은 변경이 생기지 않습니다. 이러한 동작으로 수정하려는 VI만 체크아웃할 수 있습니다.

LabVIEW가 소스 컨트롤 제공자와 함께 작동하도록 설정한 후, **파일을 소스 컨트롤에 추가**하고, 파일을 **체크아웃**하고, 편집된 파일을 **체크인**할 수 있으며, LabVIEW 내 소스 컨트롤에서 파일의 **이름을 변경**하고 파일의 **최신 버전 얻기**를 할 수 있습니다. 또한 **개별 LabVIEW 프로젝트에서 소스 컨트롤을 설정**할 수도 있습니다. 소스 컨트롤 페이지의 제공자를 변경하는 경우, 변경은 LabVIEW에서 사용하는 모든 VI와 프로젝트에 영향을 미칩니다.

LabVIEW에서 이용할 수 없는 소스 컨트롤 작업을 수행하려면 그 특정한 함수를 위해 소스 컨트롤 제공자로부터 직접 작동시킵니다.

LabVIEW 프로젝트의 소스 컨트롤 설정하기

LabVIEW 프로젝트의 소스 컨트롤 셋팅을 설정하기 전 **소스 컨트롤을 사용할 LabVIEW를 설정**해야 합니다. LabVIEW 프로젝트는 기본적으로 LabVIEW 환경에서 지정한 소스 컨트롤 설정을 사용합니다.



노트

소스 컨트롤은 Professional Development System에서만 사용 가능합니다.

다음 단계를 따라 개별 LabVIEW 프로젝트의 소스 컨트롤 셋팅을 설정합니다.

- 아직 설정하지 않은 경우, **소스 컨트롤을 사용할 LabVIEW를 설정**합니다.
- 프로젝트 탐색기** 윈도우에서, 프로젝트 >> 프로퍼티를 선택하거나, 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 프로젝트 프로퍼티 대화 상자를 디스플레이합니다.
- 항목 리스트에서 소스 컨트롤을 선택하여 소스 컨트롤 페이지를 디스플레이합니다.
- 다음 옵션 중에서 하나를 선택하십시오:
 - 현재 소스 컨트롤 셋팅 사용—LabVIEW 프로젝트에서 LabVIEW 환경에 대해 설정한 소스 컨트롤 셋팅을 사용합니다. 이것은 기본 설정입니다.
 - 프로젝트 특정 소스 컨트롤 셋팅 사용—LabVIEW 프로젝트에서 사용할 LabVIEW 현재 소스 컨트롤 셋팅이 지정한 소스 컨트롤 프로젝트가 아닌 다른 소스 컨트롤 프로젝트를 지정합니다. 변경 버튼을 클릭하여 소스 컨트롤

를 프로젝트를 지정할 수 있는 설정 대화 상자를 디스플레이합니다. 나타나는 대화 상자는 사용하는 소스 컨트롤 제공자에 특정적입니다.

- 이 프로젝트에 소스 컨트롤 사용하지 않음—LabVIEW 환경에서 설정한 소스 컨트롤 셋팅에 관계없이 LabVIEW 프로젝트의 소스 컨트롤을 비활성화합니다.

5. 확인 버튼을 클릭하여 소스 설정을 LabVIEW 프로젝트에 적용합니다.



노트 LabVIEW 프로젝트의 소스 컨트롤 제공자를 변경할 수 없습니다. LabVIEW 프로젝트에서의 소스 컨트롤 셋팅이 LabVIEW 환경에서 지정한 제공자와 일치해야 합니다. LabVIEW 환경에서 소스 컨트롤 제공자를 변경한 경우, 다른 소스 컨트롤 제공자에 특정한 설정 셋팅을 가진 LabVIEW 프로젝트는 소스 컨트롤이 활성화되지 않은 것처럼 동작하며, 경고가 소스 컨트롤 프로퍼티 페이지에 나타납니다. 옵션 대화 상자의 소스 컨트롤 페이지를 사용하여 LabVIEW 환경에 대한 소스 컨트롤 제공자를 변경합니다. LabVIEW에서는 한 번에 하나의 소스 컨트롤 제공자만을 사용할 수 있습니다.

소스 컨트롤에서 파일 체크아웃하기

파일을 변경하려면 소스 컨트롤에서 파일을 체크아웃해야 합니다. **소스 컨트롤을 설정** 하여 입력요청하도록 한 경우, 체크아웃하지 않은 소스 컨트롤의 파일을 편집하려고 하면 LabVIEW는 파일을 체크아웃하도록 입력요청합니다.

소스 컨트롤에 파일을 체크아웃하려면, LabVIEW는 **소스 컨트롤을 사용할 수 있도록** 설정되어 있어야 합니다.



주의

소스 컨트롤에서 파일을 체크아웃하기 전에 로컬 파일을 수정한 경우 로컬 디렉토리의 파일 버전이 소스 컨트롤의 버전과 다르게 됩니다. 소스 컨트롤 제공자에 따라, 소스 컨트롤에서 파일을 체크아웃할 때 로컬 파일에 실시한 변경을 잃을 수도 있습니다. 파일 체크아웃의 효과에 관한 추가적인 정보는 소스 컨트롤 제공자 문서를 참조하십시오.

다음 단계를 따라 LabVIEW를 사용하여 파일을 체크아웃합니다.

- LabVIEW 프로젝트에서 작업하는 경우, 체크아웃하려는 폴더나 파일에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 체크아웃을 선택합니다. 또한 소스 컨트롤 도구 모음의 체크아웃 버튼을 클릭하거나 도구»소스 컨트롤»체크아웃을 선택할 수 있습니다. 프로젝트 외의 개별 VI에서 작업하고 있다면, 체크아웃하려는 VI를 여십시오. 도구»소스 컨트롤»체크아웃을 선택합니다.
- 프로젝트 내에서 작업하는 경우, 파일이 소스 컨트롤에서 체크아웃되었음을 보여주기 위해 파일 아이콘에 붉은색 체크 표시가 나타납니다.

일반적으로 오랜 시간 동안 파일을 체크아웃하는 것을 피해야 합니다. 가능하면 점진적으로 파일을 변경합니다. 파일을 체크인할 때, 완전히 테스트하도록 하십시오. 특정 VI를 체크인하기 전에 다른 VI를 수정해야 하는 경우, 다른 VI를 체크아웃한 다음 변경을 하고, 모든 파일을 체크인하기 전에 테스트합니다.

파일을 여러 번 변경해야 한다면 하나를 수정하고 파일을 체크인한 후 다음 수정을 시작하기 위해 다시 파일을 체크아웃하는 것을 고려하십시오. 이렇게 하면 다른 개발자가 변경에 접근할 수 있고, 나중에 이후의 변경이 맞지 않았다고 생각될 경우 다시 돌아갈 수 있는 체크포인트를 만들게 됩니다.

소스 컨트롤에서 파일 체크아웃 취소하기

파일을 체크아웃한 후 실시한 변경을 실행 취소하려는 경우, 파일 **체크아웃 작업**을 실행 취소할 수 있습니다. 소스 컨트롤 제공자는 체크아웃 작업이나 변경에 대한 레코드를 저장하지 않습니다.

다음 단계를 따라 파일 체크아웃을 실행 취소합니다.

1. LabVIEW 프로젝트에서 작업하는 경우, 체크아웃 작업을 실행 취소하려는 파일에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 체크아웃 취소를 선택합니다. 또한, 소스 컨트롤 도구 모음의 체크아웃 취소 버튼을 클릭하거나 도구 >> 소스 컨트롤 >> 체크아웃 취소를 선택할 수 있습니다. 프로젝트 외의 개별 VI에서 작업하고 있다면, 체크아웃 작업을 취소하려는 VI를 여십시오. 도구 >> 소스 컨트롤 >> 체크아웃 취소를 선택합니다.
2. 프로젝트 내에서 작업하는 경우, 파일이 더 이상 소스 컨트롤에서 체크아웃되지 않았기 때문에 파일 아이콘의 빨간색 체크 표시가 사라집니다.□

소스 컨트롤에 파일 체크인하기

체크아웃한 파일의 변경을 끝냈을 때, 파일을 체크인하여 해당 버전을 소스 컨트롤에 복사함으로써 다른 사용자가 사용할 수 있도록 합니다.

소스 컨트롤에 파일을 체크인하려면, LabVIEW는 **소스 컨트롤을 사용할 수 있도록** 설정되어 있어야 합니다.

다음 단계를 따라 소스 컨트롤에 파일을 체크인합니다.

1. 프로젝트에서 작업하는 경우, 체크인하려는 파일을 선택합니다. 프로젝트 외의 개별 VI에서 작업하는 경우, 체크인하려는 VI를 엽니다.
2. 도구 >> 소스 컨트롤 >> 체크인을 선택하여 소스 컨트롤 작업 대화 상자를 디스플레이합니다. 또한 프로젝트 내에서, 소스 컨트롤 도구 모음의 체크인 버튼을 클릭하거나 파일이나 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 체크인을 선택할 수 있습니다.
3. 파일 트리는 소스 컨트롤에 체크인할 모든 파일의 이름과 경로를 포함합니다. 소스 컨트롤에 체크인하지 않으려는 모든 파일의 이름 옆의 확인란에서 확인 표시를 제거합니다.
4. 주석 텍스트 박스에 변경에 대한 요약을 입력합니다.
5. 파일을 체크인하기 위해서 확인 버튼을 클릭합니다. 소스 컨트롤 작업 내의 파일 중 하나라도 변경 사항을 저장하지 않은 상태에서 확인 버튼을 누르면 저장되지 않은 파일 대화 상자가 나타납니다. 프로젝트 내에서 작업하는 경우, 파일이 더 이

상 소스 컨트롤에서 체크아웃되지 않았음을 보여주기 위해 파일 아이콘의 붉은색 체크 표시가 사라집니다.



노트

소스 컨트롤 제공자에 따라, 소스 컨트롤 작업 대화 상자를 닫으려고 시도하고 주석 텍스트 박스가 비었을 경우, 정보를 입력할 수 있도록 제공자 특성의 대화 상자가 디스플레이될 수도 있습니다.

사용하는 소스 컨트롤 제공자에 따라, 제공자가 파일이 체크인될 때 자동으로 파일을 읽기 전용으로 설정하여 사용자들이 파일을 다시 체크아웃하지 않고 실수로 수정하는 것을 방지하는 경우도 있습니다.

소스 컨트롤에서 최신 버전의 파일 얻기

다른 사용자가 파일을 편집했거나 새 파일을 소스 컨트롤에 추가했을 때, LabVIEW를 사용하여 소스 컨트롤에서 로컬 디렉토리로 최신 버전의 파일을 복사할 수 있습니다.

다음 단계를 따라 소스 컨트롤에서 최신 버전의 파일을 얻습니다.

1. LabVIEW 프로젝트에서 작업하는 경우, 최신 버전을 가져오려는 폴더나 파일을 선택합니다. 프로젝트 외의 개별 VI에서 작업하고 있다면, VI를 여십시오.
2. 도구 » 소스 컨트롤 » 최신 버전 얻기를 선택합니다. 대화 상자가 나타나 파일의 최신 버전이 컴퓨터의 현재 파일을 덮어쓰는 것을 알려줍니다. 예 버튼을 클릭하여 작업을 확인하고 소스 컨트롤 작업 대화 상자를 디스플레이합니다.
3. 파일 트리는 소스 컨트롤에서 최신 버전을 가져오려는 모든 파일의 이름과 경로를 포함합니다. 소스 컨트롤에서 최신 버전을 가져오지 않는 모든 파일의 이름 옆의 확인란에서 확인 표시를 제거합니다.
4. 확인 버튼을 클릭합니다. 파일이 컴퓨터에 있는 경우, 소스 파일의 최신 버전이 로컬 버전을 덮어쓰게 되며 파일의 변경 사항은 모두 잃게 됩니다.
5. 최신 버전을 가져오려는 VI를 선택하고 VI가 작업 도중 열린 경우, LabVIEW는 디스크에서 VI를 다시 로드하도록 입력요청합니다. VI를 다시 로드하면 모든 변경 사항을 잃게 됩니다.

소스 컨트롤 프로퍼티를 보면 파일이 최신 버전인지 확인할 수 있습니다. VI의 프런트패널이나 블록 다이어그램 윈도우의 메뉴에서 도구 » 소스 컨트롤 » 프로퍼티를 선택하여 파일에 대한 소스 컨트롤 프로퍼티를 디스플레이합니다.

소스 컨트롤에서 찾을 수 없거나 이전 버전 파일 불러오기

또한 LabVIEW 프로젝트에서 찾을 수 없거나 이전 버전 파일의 최신 버전을 불러올 수 있습니다. LabVIEW는 소스 컨트롤에서 찾을 수 없는 파일 또는 이전 버전 파일의 최신 버전을 불러온 다음, 로컬 디렉토리에 파일을 복사합니다.

다음 단계를 따라 프로젝트에서 찾을 수 없거나 이전 버전 파일의 위치를 찾습니다.

1. 프로젝트 루트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 파일의 가장 최신 버전 얻기를 선택합니다. 찾을 수 없는 아이템이 있는 경우, 찾을 수 없는 파일 정보 대화 상자가 나타나서 소스 컨트롤에서 찾을 수 없는 의존성이나 파일 등과 같이 LabVIEW가 불러올 수 없는 파일을 나열합니다.
2. 다른 남아있는 파일을 불러오려면 찾을 수 없는 파일 정보 대화 상자에서 계속을 클릭해서 소스 컨트롤 작업 대화 상자의 수정된 버전인 가장 최신 버전 얻기 대화 상자를 디스플레이합니다.
3. 불러오려는 파일을 선택합니다. 기본으로 LabVIEW는 모든 파일을 선택합니다.
4. 확인 버튼을 클릭합니다. LabVIEW는 선택된 파일을 불러온 다음 이를 로컬 디렉토리에 복사합니다.

소스 컨트롤에서 VI 비교하기

LabVIEW를 사용하여 소스 컨트롤 파일의 현재 버전과 소스 컨트롤 제공자의 최신 버전을 비교할 수 있습니다. 파일 비교는 파일이 자주 변경될 수 있는 큰 프로젝트에서 작업할 때 유용합니다. 한 번에 하나의 파일에서만 이 작업을 수행할 수 있습니다.

정렬 옵션을 사용하여 프로젝트 내의 아이템을 구성할 수 있습니다. 선택한 정렬 옵션은 디스크 상에 있는 프로젝트의 구성을 그대로 보존합니다. 따라서 프로젝트를 소스 컨트롤에 제출할 때 통합하고 비교해야 하는 변경 내역이 더 적습니다.

비교할 VI를 선택했을 경우, LabVIEW는 차이 대화 상자에서 결과를 디스플레이합니다.



노트

LabVIEW와 타사 소스 컨트롤 제공자가 호환되지 않는 경우, LabVIEW는 타사 소스 컨트롤 제공자의 기본 비교 도구를 실행합니다. Perforce SCM 및 Rational Clearcase는 LabVIEW가 VI의 그래픽 차이를 비교할 때 사용하는 절차와 호환되지 않습니다.

다음 단계를 따라 소스 컨트롤에서 두 VI를 비교합니다.

1. LabVIEW 프로젝트에서 작업 중인 경우, 비교하려는 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차이 보이기를 선택하여 차이 대화 상자를 디스플레이합니다. 프로젝트 외의 개별 VI에서 작업하고 있다면, 비교하려는 VI를 여십시오. 도구 » 소스 컨트롤 » 차이 보이기를 선택하여 차이 대화 상자를 디스플레이합니다.



노트 VI가 저장되지 않은 변경사항을 포함하는 경우, 저장되지 않은 파일 대화상자가 나타나고 비교 작업을 진행하기 전에 변경사항을 저장하도록 요청합니다.

1. 차이 대화 상자에서 **비교 옵션을 선택**합니다.

다음 단계를 따라 소스 컨트롤에서 VI가 아닌 두 파일을 비교합니다. VI가 아닌 파일은 프로젝트 내에서만 비교할 수 있습니다.

1. 프로젝트에서는, 비교하려는 파일에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 차이 보이기를 선택합니다. 소스 컨트롤 제공자의 기본 비교 도구가 시작됩니다.
2. 비교가 완료되면, 타사 제공자는 두 파일 사이의 차이를 디스플레이합니다.

소스 컨트롤에서 개정 히스토리 보이기

소스 컨트롤에서 파일 상호작용의 간략한 요약은 볼 수 있습니다.

다음 단계를 따라 소스 컨트롤에서 파일의 히스토리를 봅니다.

1. LabVIEW 프로젝트에서 작업하는 경우, 소스 컨트롤 히스토리를 보려는 파일을 선택합니다. 프로젝트 외의 개별 VI에서 작업하고 있다면, VI를 여십시오.
2. 도구 » 소스 컨트롤 » 히스토리 보하기를 선택하여 파일이 소스 컨트롤에 추가된 후 변경된 사항의 리스트를 포함하는 제공자 특성의 대화 상자를 디스플레이합니다. 소스 컨트롤 제공자에 따라, 리스트의 정보는 개정 번호, 변경한 사용자, 변경 날짜, 개정에 첨부된 주석을 포함할 수 있습니다.
3. 대화 상자를 닫습니다.

소스 컨트롤에 파일 추가하기

LabVIEW 프로젝트의 폴더, 개별 VI 또는 모든 지원 파일을 소스 컨트롤에 추가할 수 있습니다. [소스 컨트롤 제공자와 함께 작업하기 위해 LabVIEW를 설정](#)할 경우, 프로젝트에 파일을 추가할 때 소스 컨트롤에 파일을 추가하도록 입력요청하게 LabVIEW를 설정할 수 있습니다.

파일이 소스 컨트롤에 추가된 후, LabVIEW의 파일에서 소스 컨트롤 작업을 수행할 수 있습니다.

다음 단계를 따라 소스 컨트롤에 LabVIEW의 파일을 추가합니다.

1. 파일을 저장하려는 소스 컨트롤 제공자 디렉토리에 대응하는 로컬 디렉토리에 파일을 저장합니다. 새 VI 또는 LabVIEW 프로젝트 라이브러리는 저장하고 이름을 정하기 전에는 소스 컨트롤에 추가할 수 없습니다.
2. 도구 » 소스 컨트롤 » 소스 컨트롤에 추가를 선택하여 소스 컨트롤 작업 대화 상자를 디스플레이합니다. 또한 소스 컨트롤 도구 모음의 소스 컨트롤에 추가 버튼을 클릭할 수 있습니다. 프로젝트 내의 추가하려는 파일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 소스 컨트롤에 추가를 선택합니다.
3. 파일 트리는 소스 컨트롤에 추가할 모든 파일의 이름과 경로를 포함합니다. 소스 컨트롤에 추가하지 않으려는 모든 파일의 이름 옆의 확인란에서 확인 표시를 제거합니다.

4. 확인 버튼을 클릭합니다. LabVIEW는 파일을 소스 컨트롤에 추가합니다. 프로젝트 내에서 작업하는 경우, 파일이 소스 컨트롤에 있다는 것을 보여주기 위해 파일 아이콘에 흰색 사각형 이 나타납니다.



노트

Perforce를 사용하는 경우, LabVIEW는 파일을 기본 변경 리스트에 추가합니다. 새로운 변경 리스트 작성 또는 변경 리스트 간 탐색 등의 고급 변경 리스트 작업을 실행하기 위해서는 Perforce를 사용해야 합니다.

파일이 소스 컨트롤에 추가된 후, 편집한 **파일을 체크아웃**, **체크인**할 수 있고, 소스 컨트롤에서 파일의 **최신 버전 얻기**를 할 수 있습니다.

소스 컨트롤에서 파일 삭제하기

소스 컨트롤에서 파일을 지울 때 주의하십시오. 일부 소스 컨트롤 제공자는 파일의 로컬 디렉토리 복사본과 함께, 제공자가 보관하는 파일의 이전 버전 및 해당 파일에 대한 히스토리 로그를 삭제합니다. 소스 컨트롤에서 파일 삭제의 효과에 관한 추가적인 정보는 소스 컨트롤 제공자 문서를 참조하십시오.

다음 단계를 따라 소스 컨트롤에서 파일을 삭제합니다.

1. LabVIEW 프로젝트에서 작업하는 경우, 소스 컨트롤에서 삭제하려는 폴더나 파일을 선택합니다.
2. 도구 >> 소스 컨트롤 >> 소스 컨트롤에서 제거를 선택합니다.
3. 대화 상자가 나타나 소스 컨트롤에서 파일을 제거하면 파일이 완전히 삭제될 수 있다는 것을 알려줍니다. 예 버튼을 클릭하여 파일 삭제를 확인하고 소스 컨트롤 작업 대화 상자를 디스플레이합니다.
4. 파일 트리는 소스 컨트롤에서 제거할 모든 파일의 이름과 경로를 포함합니다. 소스 컨트롤에서 제거하지 않으려는 모든 파일의 이름 옆의 확인란에서 확인 표시를 제거합니다.
5. 확인 버튼을 클릭합니다.



노트

Perforce를 사용하는 경우, LabVIEW는 파일을 변경 리스트에 추가합니다. Perforce를 사용하여 변경 리스트를 제출해야 합니다.

소스 컨트롤에서 파일 이름 바꾸기

소스 컨트롤은 파일의 이름에 근거해서 개정 히스토리를 저장합니다. 따라서 파일의 이름을 바꿀 때, 개정 히스토리를 그대로 유지하려면 **소스 컨트롤 제공자**의 이름 변경 과정을 따라야 합니다.

파일 이름을 바꾸면 해당 파일에 **의존성을 가진 링크**가 소실될 수도 있습니다. 소스 컨트롤에서 파일 이름을 바꿀 때 다음 단계를 따르면 이러한 문제를 방지할 수 있습니다.

1. 이름을 바꾸려는 파일이 포함되어 있는 LabVIEW 프로젝트를 여십시오. 파일 이름을 바꿀 때 이 프로젝트가 열려 있어야 LabVIEW가 이름 변경을 감지할 수 있습니다.
2. 소스 컨트롤 제공자에서 모든 프로젝트 파일을 **체크아웃**합니다.
3. 소스 코드 컨트롤 제공자의 이름 변경 과정을 따라 파일 이름을 바꿉니다.
4. LabVIEW 프로젝트로 돌아가 <F5>를 눌러 프로젝트를 새로 고칩니다. 이름이 바뀐 파일에는 이름이 바뀌었다는 것을 나타내는 경고가 뜹니다.
5. 파일에서 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택합니다.
6. 새로운 버전의 파일을 탐색한 후, 선택합니다.

소스 컨트롤 VI를 사용하여 독립 어플리케이션 만들기

어플리케이션 빌더의 도구를 사용하여 독립 어플리케이션과 공유 라이브러리를 만들 때 [소스 컨트롤] VI를 포함시킬 수 있습니다.

[소스 컨트롤] VI를 가진 어플리케이션 또는 공유 라이브러리를 만들 때 다음 단계를 포함합니다:

- 설정한 소스 컨트롤 제공자의 LabVIEW 프로젝트 라이브러리와 의존성 폴더를 어플리케이션 또는 공유 라이브러리를 만드는 LabVIEW 프로젝트에 추가합니다. LabVIEW와 함께 Perforce 명령 라인 인터페이스를 사용하는 경우, `labview\vi.lib\SourceControl\Providers\P4CMD` 디렉토리에 위치한 `NI_SCC_P4CMD.lvlib`를 추가합니다. Windows 기반의 소스 컨트롤 제공자를 사용하는 경우, `labview\vi.lib\SourceControl\Providers\WinCI` 디렉토리에 위치한 `NI_SCC_WinCI.lvlib`를 추가합니다.
- `labview\vi.lib\SourceControl\sccconnsrv` 디렉토리에 위치한 `NI_SCC_ConnSrv.lvlib`와 의존성을 어플리케이션 또는 공유 라이브러리를 만드는 LabVIEW 프로젝트에 추가합니다. `NI_SCC_ConnSrv.lvlib`는 소스 컨트롤 서버의 프로젝트 라이브러리입니다.
- 독립 어플리케이션을 만드는 경우, 각 소스 컨트롤 프로젝트 라이브러리 아래에 있는 폴더를 어플리케이션 프로퍼티 대화 상자의 소스 파일 페이지에 있는 항상 포함 리스트박스에 추가합니다. 예를 들어 `NI_SCC_WinCI.lvlib`를 사용하는 경우, WinCI 폴더를 항상 포함 리스트박스에 추가합니다.
- 공유 라이브러리를 만드는 경우, 소스 컨트롤 프로젝트 라이브러리 아래에 있는 폴더를 공유 라이브러리 프로퍼티 대화 상자의 소스 파일 페이지에 있는 항상 포함 리스트박스에 추가합니다. 예를 들어 `NI_SCC_WinCI.lvlib`를 사용하는 경우, WinCI 폴더를 항상 포함 리스트박스에 추가합니다.



노트

`labview\labview.ini` 파일의 소스 컨트롤 설정 정보를 빌드 중인 어플리케이션의 .ini 파일로 복사하는 방법에 대한 추가 정보는 ni.com/kb의 NI 기술지원 데이터베이스를 참조하십시오.

LabVIEW 클래스

LabVIEW 클래스 또는 인터페이스 생성하기

다음 단계에 따라 LabVIEW **클래스** 또는 **인터페이스**를 생성합니다.

1. **새로운 LabVIEW 프로젝트를 생성**하거나 기존의 프로젝트를 엽니다.
2. 프로젝트 탐색기 윈도우의 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 새로 만들기>>클래스 또는 새로 만들기>>인터페이스를 선택합니다.

노트
또한 파일>>새로 만들기를 선택하여 새로 만들기 대화상자를 디스플레이하고, 새로 생성 리스트의 기타 파일>>클래스 또는 기타 파일>>인터페이스를 선택하여 새로운 클래스나 인터페이스를 생성할 수 있습니다.
3. 새 클래스/인터페이스 대화 상자에서 클래스 또는 인터페이스의 목적을 나타내는 이름을 입력합니다.
 - 클래스의 경우, 모든 이름을 명사로 지정합니다. 예를 들어, Reads Waveforms.lvclass 대신 Waveform Reader.lvclass라는 이름을 지정합니다.
 - 인터페이스의 경우, **인터페이스 명명 권장 방법**을 참조하십시오.
4. 직상위 클래스와 직상위 인터페이스 섹션에서 새 클래스 또는 인터페이스가 상속하는 클래스와 인터페이스를 선택합니다.
5. 저장 옵션 섹션에서 다음 옵션 중 하나를 선택합니다.
 - 폴더에 저장을 선택하여 클래스 또는 인터페이스를 바로 저장할 경로를 지정합니다.
 - 클래스 또는 인터페이스를 저장하지 않을 경우 저장 안함을 선택합니다.
6. 확인을 클릭합니다. LabVIEW 클래스 또는 인터페이스 파일이 프로젝트 탐색기 윈도우에 나타납니다. LabVIEW 클래스의 경우, 클래스와 동일한 이름의 프라이빗 데이터 컨트롤 .ctl도 나타납니다.

다음의 작업을 완성할 수 있습니다:

- (클래스의 경우) **클래스의 프라이빗 데이터 정의하기**.

노트 또한 **컨트롤 혹은 타입 정의로부터 LabVIEW 클래스를 생성**할 수 있습니다.

- **멤버 vi 생성하기.**
- 클래스 또는 인터페이스 프로퍼티 편집하기.

또한 필요한 순서에 따라 데이터 정의, 멤버 vi 생성, 또는 클래스나 인터페이스 프로퍼티 편집을 할 수 있습니다. 어떤 클래스는 프라이빗 데이터를 필요로 하지 않지만, 프라이빗 데이터 컨트롤을 정의하지 않도록 멤버 vi를 필요로 할 수 있습니다. 클래스 또는 인터페이스를 개발하기 전에 와이어 모양 및 다른 클래스나 인터페이스의 프로퍼티를 변경할 수 있습니다. 개별 프로젝트와 개발 계획에 가장 적절한 순서대로 클래스나 인터페이스의 구성요소를 개발할 수 있습니다.



노트 LabVIEW 클래스 또는 인터페이스와 동일한 이름의 디렉토리를 디스크에 생성하여 클래스 라이브러리 파일, 클래스 멤버 VI, **클래스 사용자 정의 기본 프로브**를 저장할 수 있습니다. 디렉토리에 클래스 라이브러리가 소유하는 파일을 포함할 수 있습니다. 만일 하나 이상의 클래스 라이브러리를 동일한 디렉토리에 포함한다면, 같은 이름의 VI를 다른 클래스 라이브러리에 포함하려는 경우 충돌이 발생할 수 있습니다. 개발 프로세스에서 **다이나믹 멤버 VI**를 덮어쓰는 경우 이름과 관련된 충돌이 발생할 수 있습니다.

컨트롤의 내용을 LabVIEW 클래스로 변환하기

기존의 컨트롤 및 타입 정의에서 LabVIEW 클래스를 생성할 수 있습니다. 다음 단계를 따라 컨트롤 또는 타입 정의의 내용을 LabVIEW 클래스로 변환합니다.

1. 기존의 사용자 컨트롤 또는 타입 정의를 포함하는 LabVIEW 프로젝트를 엽니다.
2. **프로젝트 탐색기** 윈도우의 컨트롤 또는 타입 정의에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 컨트롤의 내용을 클래스로 변환을 선택합니다. LabVIEW는 프라이빗 데이터 컨트롤이 있는 클래스를 생성합니다. 새 클래스는 자신이 생성되었던 컨트롤의 이름을 유지합니다.



노트

LabVIEW는 새 클래스를 컨트롤의 프런트패널에 추가하여 계속해서 그 컨트롤을 사용합니다.

LabVIEW는 프로젝트 탐색기 윈도우에 클래스를 생성한 컨트롤을 두지만, LabVIEW는 디스크에서 컨트롤을 수정합니다. LabVIEW가 컨트롤을 업데이트하여 LabVIEW 클래스를 사용하기 때문에, 컨트롤에 대한 모든 참조가 깨질 수 있습니다. 사용자는 이러한 VI의 프라이빗 데이터에 대한 접근자(Accessors)를 제공하거나 VI를 새로 생성된 클래스에 옮겨야 합니다.

LabVIEW 클래스의 프라이빗 데이터 정의하기

LabVIEW 클래스의 프라이빗 데이터 컨트롤에 컨트롤과 클러스터를 추가하여 데이터 타입을 생성할 수 있습니다.



노트

인터페이스는 프라이빗 데이터를 포함하지 않으므로 인터페이스에 컨트롤을 추가할 수 없습니다.

다음의 단계를 따라 LabVIEW 프라이빗 데이터 컨트롤을 정의합니다.

1. **새로운 LabVIEW 클래스를 생성**하거나 파일>>열기를 선택하고 열고자 하는 클래스를 검색하여 기존의 클래스를 엽니다. 프라이빗 데이터 컨트롤이 **프로젝트 탐색기** 윈도우에서 클래스 아이콘 아래에 나타납니다.
2. 프라이빗 데이터 컨트롤을 더블 클릭합니다. **컨트롤 편집기 윈도우**가 열립니다.

- 클래스 프라이빗 데이터의 클러스터에 컨트롤과 인디케이터를 추가하여 필요한 경우 데이터 타입을 정의합니다.

노트 프라이빗 데이터 컨트롤은 **x컨트롤**을 포함할 수 없습니다.

- 편집>>현재값을 기본값으로 선택하여 전체 클러스터의 기본값을 설정합니다. LabVIEW는 각 원소의 현재값을 원소의 기본값으로 설정합니다.

노트 프라이빗 데이터 컨트롤의 기본값으로 설정한 값은 클래스 타입 내의 모든 컨트롤, 인디케이터, 상수의 기본값입니다.

- 프라이빗 데이터 컨트롤을 저장합니다.



노트 LabVIEW는 .lvclass 파일 안에 프라이빗 데이터 컨트롤을 저장합니다. 컨트롤은 디스크 내에 별도의 파일을 갖지 않습니다. LabVIEW 클래스 정의에 프라이빗 데이터 컨트롤을 저장함으로써 사용자 에러를 방지할 수 있습니다.

LabVIEW 클래스 또는 인터페이스에서 멤버 VI 생성하기

멤버 VI를 생성하여 **클래스** 또는 **인터페이스**가 수행할 수 있는 작업을 정의합니다. 멤버 VI는 프라이빗 또는 보호된 영역에서 다른 메소드를 호출할 수 있는 권한이 있습니다. 클래스 멤버 VI는 클래스 프라이빗 데이터에 접근할 수 있는 권한이 있습니다. 멤버 VI는 여러 가지 방법으로 생성할 수 있습니다.

새 VI에서 멤버 VI 생성하기

다음 단계를 따라 새 VI에서 멤버 VI를 생성합니다.

- 새로운 LabVIEW 클래스 또는 인터페이스를 생성**하거나 파일>>열기를 선택하고 열고자 하는 클래스나 인터페이스를 검색하여 기존의 클래스나 인터페이스를 엽니다.



노트

파일>>열기를 사용하여 클래스나 인터페이스를 여는 경우, 프로젝트 탐색기 윈도우 대신 클래스 윈도우에서 클래스나 인터페이스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스나 인터페이스를 개발할 수 있습니다.

- (클래스의 경우) **클래스의 프라이빗 데이터 정의하기**.
- 프로젝트 탐색기** 윈도우에서 멤버 VI를 생성하려는 클래스 또는 인터페이스의 프로젝트 아이템을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴에서 새로 만들기>>VI를 선택합니다. 새로운 멤버 VI(🔗)가 프로젝트 탐색기 윈도우에 나타나고, LabVIEW가 VI를 엽니다.
- 이제 클래스 또는 인터페이스가 수행할 수 있는 작업을 정의하는 멤버 VI를 개발할 수 있습니다.

(클래스의 경우) 프로퍼티 정의 폴더에서 멤버 VI 생성하기

프로퍼티 정의 폴더에서 멤버 VI를 생성할 경우, LabVIEW 클래스의 프라이빗 데이터는 프로퍼티 노드를 사용해서 접근할 수 있는 클래스의 프로퍼티가 됩니다.



노트 프로퍼티 정의 폴더에 지정한 이름이 프로퍼티 노드에 나타나는 프로퍼티 이름을 결정합니다. 프로퍼티 이름을 변경하려면, 이에 해당하는 프로퍼티 정의 폴더의 이름을 변경해야 합니다. 프라이빗 데이터의 이름을 변경해도 프로퍼티 노드에 나타나는 프로퍼티 이름은 변경되지 않습니다.

다음 단계를 따라 프로퍼티 정의 폴더에서 멤버 VI를 생성합니다.

1. **새로운 LabVIEW 클래스를 생성**하거나 **파일» 열기**를 선택하고 열고자 하는 클래스를 검색하여 기존의 클래스를 엽니다.

노트 파일» 열기를 사용하여 클래스를 여는 경우, 프로젝트 탐색기 윈도우 대신 클래스 윈도우에서 클래스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스를 개발할 수 있습니다.

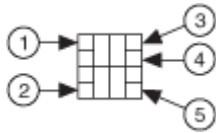
2. **클래스의 프라이빗 데이터 정의하기.**
3. **프로젝트 탐색기** 윈도우에서 멤버 VI를 생성하려는 클래스의 아이콘(📁)을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴에서 새로 만들기» 프로퍼티 정의 폴더를 선택합니다. 새 프로퍼티 정의 폴더(📁)가 프로젝트 탐색기 윈도우에 나타납니다.
4. 프로퍼티 정의 폴더에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기» VI를 선택합니다. 또한 **다이나믹 디스패치 템플릿** 또는 **정적 디스패치 템플릿**을 사용하여 새 VI를 생성할 수 있습니다. 새로운 멤버 VI(📄)가 프로젝트 탐색기 윈도우에 나타나고, LabVIEW가 VI를 엽니다.

노트 또한 프로퍼티 정의 폴더에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 추가» 파일을 선택해서 프로퍼티 정의 폴더에 기존의 멤버 VI를 추가할 수도 있습니다.

5. 이제 멤버 VI를 개발하여 클래스의 프라이빗 데이터와 상호작용할 수 있습니다. 각 프로퍼티 정의 폴더는 이 클래스의 프라이빗 데이터를 읽는 VI 한 개, 쓰는 VI 한 개, 또는 두 VI를 모두 포함할 수 있습니다.

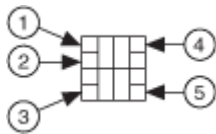
노트 프로퍼티 노드를 사용하여 LabVIEW 클래스에 접근하는 경우, 클래스 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 접근자 VI 열기를 선택하여 LabVIEW 클래스 프로퍼티의 구현 대상을 쉽게 볼 수 있습니다. 사용자가 LabVIEW 클래스 또는 LabVIEW 클래스를 가리키는 데이터 값 참조를 프로퍼티 노드의 참조 입력에 연결하고, 접근자 VI의 블록 다이어그램을 사용할 수 있을 때에만 이 옵션이 나타납니다. 프로퍼티가 하나 이상의 구현 대상에 대해 다이나믹한 경우, 이 옵션을 선택하면 구현 대상 선택 대화 상자가 디스플레이됩니다. 이 대화 상자를 사용하여 프로퍼티의 모든 구현 대상 또는 다이나믹 디스패치 멤버 VI를 보고 **하나 이상의 구현 대상을 엽니다.**

프로퍼티 노드를 사용해서 프라이빗 데이터를 읽거나 쓸 수 있게 해주는 멤버 VI는 $4 \times 2 \times 2 \times 4$ 의 커넥터 팬 패턴을 가져야 합니다. 프라이빗 데이터를 읽는 멤버 VI의 터미널은 다음의 커넥터 팬 터미널을 가져야 합니다.



1 클래스 입력	3 클래스 출력	5 에러 출력
2 에러 입력	4 값 출력	

프라이빗 데이터를 쓰는 멤버 VI의 터미널은 다음 커넥터 팬 터미널을 가져야 합니다.



1 클래스 입력	3 에러 입력	5 에러 출력
2 값 입력	4 클래스 출력	



노트 멤버 VI의 클래스 입력과 클래스 출력은 같은 LabVIEW 클래스여야 합니다. 프로퍼티 정의 폴더가 읽기와 쓰기 멤버 VI를 둘 다 포함할 경우, 읽기 VI의 값 출력과 쓰기 VI의 값 입력은 같은 데이터 타입이어야 합니다.

다이나믹 디스패치 템플릿에서 멤버 VI 생성하기

VI 템플릿을 사용하여 새로 만들기 대화 상자를 사용하는 새 멤버 VI를 생성할 수 있습니다. 그러나 LabVIEW를 사용하면 쉽고 빠르게 많은 다이나믹 메소드가 필요로 하는 여러 가지 기능을 포함하는 다이나믹 디스패치 멤버 VI를 생성할 수 있습니다. LabVIEW는 여러 입력과 에러 출력 클러스터, 에러 핸들링을 위한 케이스 구조, 입력 클래스 또는 인터페이스, 출력 클래스 또는 인터페이스로 새 멤버 VI를 채웁니다. LabVIEW는 입력 및 출력 터미널 모두를 **VI 커넥터 팬에서 다이나믹한 것으로** 설정합니다. 다음 단계를 따라 다이나믹 디스패치 템플릿에서 멤버 VI를 생성합니다.

1. **새로운 LabVIEW 클래스 또는 인터페이스를 생성**하거나 파일» 열기를 선택하고 열고자 하는 클래스나 인터페이스를 검색하여 기존의 클래스나 인터페이스를 엽니다.

노트 파일» 열기를 사용하여 클래스나 인터페이스를 여는 경우, **프로젝트 탐색기** 윈도우 대신 클래스 윈도우에서 클래스나 인터페이스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스나 인터페이스를 개발할 수 있습니다.

2. (클래스의 경우) **클래스의 프라이빗 데이터 정의하기**.
3. 프로젝트 탐색기 윈도우에서 멤버 VI를 생성하려는 클래스 또는 인터페이스의 프로젝트 아이템을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴에서 새로 만들기» 다이나믹 디스패치 템플릿으로부터 VI를 선택합니다. 새로운 VI가 프로젝트 탐색기 윈도우에 나타나고, LabVIEW가 VI를 자동으로 엽니다. VI의 프런트패널

이 클래스 또는 인터페이스의 인디케이터, 클래스 또는 인터페이스의 컨트롤, 그리고 에러 입력 및 에러 출력 클러스터를 디스플레이합니다.

4. <Ctrl-E> 키를 눌러서 블록다이어그램을 디스플레이합니다. (macOS) <Option-E> 키를 누릅니다. (Linux) <Alt-E> 키를 누릅니다. 블록다이어그램은 두 개의 클래스 또는 인터페이스의 터미널, 에러 입력 및 에러 출력 클러스터 터미널 및 에러 핸들링을 위한 케이스 구조를 디스플레이합니다.
5. 이제 멤버 VI를 개발하여 클래스의 프라이빗 데이터와 상호작용할 수 있습니다.

정적 디스패치 템플릿에서 멤버 VI 생성하기

또한 LabVIEW를 사용하면 쉽고 빠르게 정적 디스패치 멤버 VI 생성할 수 있습니다. LabVIEW는 에러 입력과 에러 출력 클러스터, 에러 핸들링을 위한 케이스 구조, 입력 클래스 또는 인터페이스, 출력 클래스 또는 인터페이스로 새 멤버 VI를 채웁니다. 다이내믹 디스패치 VI와 달리, LabVIEW는 정적 디스패치 VI의 커넥터 팬에서 입력과 출력 터미널을 다이내믹으로 설정하지 않습니다. 다음 단계를 따라 정적 디스패치 템플릿에서 멤버 VI를 생성합니다.

1. **새로운 LabVIEW 클래스 또는 인터페이스를 생성**하거나 파일>>열기를 선택하고 열고자 하는 클래스나 인터페이스를 검색하여 기존의 클래스나 인터페이스를 엽니다.

노트 파일>>열기를 사용하여 클래스나 인터페이스를 여는 경우, **프로젝트 탐색기** 윈도우 대신 클래스 윈도우에서 클래스나 인터페이스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스나 인터페이스를 개발할 수 있습니다.

2. (클래스의 경우) **클래스의 프라이빗 데이터 정의하기**.
3. 프로젝트 탐색기 윈도우에서 멤버 VI를 생성하려는 클래스 또는 인터페이스의 프로젝트 아이템을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴의 새로 만들기>>정적 디스패치 템플릿으로부터 VI를 선택합니다. 새로운 VI가 프로젝트 탐색기 윈도우에 나타나고, LabVIEW가 VI를 자동으로 엽니다. VI의 프런트패널이 클래스 또는 인터페이스의 인디케이터, 클래스 또는 인터페이스의 컨트롤, 그리고 에러 입력 및 에러 출력 클러스터를 디스플레이합니다.
4. <Ctrl-E> 키를 눌러서 블록다이어그램을 디스플레이합니다. (macOS) <Option-E> 키를 누릅니다. (Linux) <Alt-E> 키를 누릅니다. 블록다이어그램은 두 개의 클래스 또는 인터페이스의 터미널, 에러 입력 및 에러 출력 클러스터 터미널 및 에러 핸들링을 위한 케이스 구조를 디스플레이합니다.
5. 이제 멤버 VI를 개발하여 클래스의 프라이빗 데이터와 상호작용할 수 있습니다.

(클래스의 경우) 데이터 멤버 접근을 위한 멤버 VI 생성하기

LabVIEW 클래스의 멤버인 읽기 또는 쓰기 접근자 VI를 빠르게 생성할 수 있습니다. 읽기 접근자 VI는 호출 VI의 데이터에 접근할 수 있도록 LabVIEW 클래스 데이터를 팝니다. 쓰기 접근자 VI는 호출 VI에서 전달되는 클래스 데이터의 새 값을 팝니다. 접근자 VI를 사용하여 LabVIEW 클래스의 멤버가 아닌 VI의 클래스 데이터에서 읽거나 클래스 데이터로 쓸 수 있습니다. 다음 단계를 따라 VI를 생성하여 클래스의 데이터에 접근합니다.

1. **새로운 LabVIEW 클래스를 생성**하거나 파일>>열기를 선택하고 열고자 하는 클래스를 검색하여 기존의 클래스를 엽니다.

노트 파일»열기를 사용하여 클래스를 여는 경우, **프로젝트 탐색기** 윈도우 대신 클래스 윈도우에서 클래스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스를 개발할 수 있습니다.

2. 클래스의 프라이빗 데이터 정의하기.

3. 프로젝트 탐색기 윈도우에서 멤버 VI를 생성하려는 클래스의 아이콘(🔵)을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴의 새로 만들기»데이터 멤버 접근을 위한 VI를 선택하여 접근자 생성 대화 상자를 디스플레이합니다. 또한 LabVIEW 클래스의 프라이빗 데이터 컨트롤에 있는 데이터 멤버에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 접근자 생성을 선택할 수도 있습니다.

노트 이 옵션을 사용하기 전에 새 LabVIEW 클래스를 저장해야 합니다. 새 클래스를 저장하지 않으면 LabVIEW는 데이터 멤버 접근을 위한 VI 및 접근자 생성 옵션을 비활성시킵니다.

4. 접근하려는 데이터 멤버, 접근 타입, 새 VI가 **다이내믹일지, 정적일지의 여부**, 또한 프로퍼티 노드를 사용해서 데이터 멤버에 접근할 수 있을지 선택합니다. 프로퍼티 노드를 통해 선택한 데이터 멤버에 접근할 수 있도록 지정할 경우, LabVIEW는 이 멤버 VI를 **프로퍼티 정의 폴더**에 추가합니다.

노트 프로퍼티 노드를 사용하여 LabVIEW 클래스에 접근하고 프로퍼티 노드를 통해 선택한 데이터 멤버에 접근할 수 있도록 지정한 경우, 클래스 프로퍼티에서 쉽게 LabVIEW 클래스 프로퍼티의 구현 대상을 볼 수 있습니다. 클래스 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 접근자 VI 열기를 선택합니다. 사용자가 LabVIEW 클래스 또는 LabVIEW 클래스를 가리키는 데이터 값 참조를 프로퍼티 노드의 참조 입력에 연결하고, 접근자 VI의 블록다이어그램을 사용할 수 있을 때에만 이 옵션이 나타납니다. 프로퍼티가 하나 이상의 구현 대상에 대해 다이나믹한 경우, 이 옵션을 선택하면 구현 대상 선택 대화 상자가 디스플레이됩니다. 이 대화 상자를 사용하여 프로퍼티의 모든 구현 대상 또는 다이나믹 디스패치 멤버 VI를 보고 **하나 이상의 구현 대상을 엽니다**.

<Ctrl> 키를 눌러 하나 이상의 데이터 멤버를 선택하는 경우, LabVIEW는 각 데이터 멤버의 접근자 VI를 생성합니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다. 대화 상자에서 읽기 및 쓰기를 선택하는 경우, LabVIEW는 읽기 권한을 제공하는 접근자 VI와 쓰기 권한을 제공하는 다른 접근자 VI를 생성합니다.

5. 생성 버튼을 클릭하여 한 개 또는 여러 개의 접근자 VI를 생성합니다. 프로젝트 탐색기 윈도우에서 선택한 각 데이터 멤버 및 각 타입의 접근에 대한 새 VI가 나타나고, LabVIEW는 자동으로 각 VI를 엽니다. VI의 이름은 선택한 접근자 타입과 데이터 멤버 이름으로 구성됩니다. 예를 들어, Read Numeric.vi.VI의 프런트패널이 LabVIEW 클래스 인디케이터, LabVIEW 클래스 컨트롤, 선택한 데이터 멤버 그리고 에러 입력 및 에러 출력 클러스터를 디스플레이합니다.
6. <Ctrl-E> 키를 눌러서 블록다이어그램을 디스플레이합니다. (macOS) <Option-E> 키를 누릅니다. (Linux) <Alt-E> 키를 누릅니다. 블록다이어그램은 두 개의 LabVIEW 클래스 터미널, 선택한 데이터 멤버, 에러 입력 및 에러 출력 클러스터 터미널과 에러 핸들링을 위한 케이스 구조를 디스플레이합니다.
7. 호출 VI로부터의 클래스 데이터를 조작할 수 있습니다.

멤버 VI 생성하여 상위 멤버 VI를 덮어쓰기

상위 다이나믹 디스패치 멤버 VI의 기능을 덮어쓰거나 확장시키는 다이나믹 디스패치 멤버 VI를 생성할 수 있습니다. 두 멤버 VI에서 다음의 설정이 같은 경우에만 한 멤버 VI가 상위 클래스나 인터페이스의 다른 멤버 VI를 덮어쓸 수 있습니다.

- 재호출 설정
- 선호하는 실행 설정
- 우선순위 설정
- **접근 영역 설정**
- **커넥터 팬 터미널**
- **커넥터 팬 패턴**

바로 가기 메뉴의 새로 만들기>>덮어쓰기 위한 VI를 선택하면 LabVIEW는 올바른 덮어쓰기의 모든 조건을 충족하는 덮어쓰기 멤버 VI를 생성합니다. 다음의 단계를 따라 새로 만들기>>덮어쓰기 위한 VI 옵션을 사용하는 덮어쓰기 멤버 VI를 생성합니다.

1. **새로운 LabVIEW 클래스 또는 인터페이스를 생성**하거나 파일>>열기를 선택하고 열고자 하는 클래스나 인터페이스를 검색하여 기존의 클래스나 인터페이스를 엽니다. 새로운 클래스 또는 인터페이스를 생성하는 경우, 클래스나 인터페이스의 덮어쓰기 멤버 VI를 생성하기 이전에 반드시 클래스나 인터페이스를 저장해야 합니다.

노트 파일>>열기를 사용하여 클래스를 여는 경우, **프로젝트 탐색기** 윈도우 대신 클래스 윈도우에서 클래스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스를 개발할 수 있습니다.

2. 프로젝트 탐색기 윈도우에서 멤버 VI를 생성하려는 클래스 또는 인터페이스의 프로젝트 아이템을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴의 새로 만들기>>덮어쓰기 위한 VI를 선택합니다.

노트 LabVIEW는 사용자가 선택한 클래스 또는 인터페이스가 덮어쓸 수 있는 **멤버 VI를 상속**하지 않는 경우, 덮어쓰기 위한 VI 옵션을 비활성화합니다. 상위 클래스 또는 인터페이스에 있는 VI가 **다이나믹 디스패치 멤버 VI**이어야만 이를 덮어쓸 수 있습니다.

3. 새로 덮어쓰기 대화 상자가 나타납니다. 이 대화 상자는 상위 클래스 또는 인터페이스가 정의하는 다이나믹 메소드를 디스플레이합니다. 덮어쓰고자 하는 멤버 VI를 선택합니다.
4. 확인 버튼을 클릭합니다.
5. 새로운 VI가 프로젝트 탐색기 윈도우에 나타나고, LabVIEW가 VI를 자동으로 엽니다. LabVIEW는 상위 VI 아이콘을 사용하며 직하위 클래스나 인터페이스의 아이콘 마스크를 덮어써서 새 VI의 아이콘을 생성합니다. 또한, 프론트패널은 상위 멤버 VI로부터 복제된 컨트롤과 인디케이터, LabVIEW 클래스 인디케이터, 에러 입력 및 에러 출력 클러스터를 디스플레이합니다.
6. <Ctrl-E> 키를 눌러서 블록다이어그램을 디스플레이합니다. (macOS) <Option-E> 키를 누릅니다. (Linux) <Alt-E> 키를 누릅니다. 블록다이어그램은 LabVIEW가 상위 멤버 VI로부터 복사한 터미널을 디스플레이합니다. LabVIEW는 또한 자동으로 터미널을 직상위 클래스 메소드 호출 노드에 연결합니다.

이제 필요에 따라 덮어쓰기 멤버 VI의 블록다이어그램 코드를 변경할 수 있습니다.



노트 새로 생성된 덮어쓰기 멤버 VI는 초기에 상위 멤버 VI의 상위 실행과 동일하게 작동합니다. 덮어쓰기 멤버 VI를 사용하여 상위 멤버 VI 형태에서 기능을 추가하거나 제거할 수 있습니다.

하위 클래스 또는 인터페이스에서 덮어쓰기 VI를 필요로 하는 멤버 VI 생성하기

하위 클래스 또는 인터페이스가 멤버 VI의 기능을 덮어쓰거나 확장시키도록 요청하는 다이나믹 디스패치 VI를 생성할 수 있습니다. 하위 항목이 상위 멤버 VI를 덮어쓰도록 요청하는 경우, 이 멤버 VI는 하위 항목에서 이 VI를 구현해야 할 필요가 없습니다. 기본적으로 인터페이스는 하위 항목이 멤버 VI를 덮어쓰도록 요청하지만, 클래스는 하위 항목이 멤버 VI를 덮어쓰도록 요청하지 않습니다.

다음 단계에 따라 하위 항목이 멤버 VI를 덮어쓰도록 요청합니다.

1. **새로운 LabVIEW 클래스 또는 인터페이스를 생성**하거나 파일»열기를 선택하고 열고자 하는 클래스나 인터페이스를 검색하여 기존의 클래스나 인터페이스를 엽니다.

노트 파일»열기를 사용하여 클래스나 인터페이스를 여는 경우, **프로젝트 탐색기** 윈도우 대신 클래스 윈도우에서 클래스나 인터페이스가 열립니다. 프로젝트 탐색기 윈도우에서와 마찬가지로 클래스 윈도우에서 클래스나 인터페이스를 개발할 수 있습니다.

2. 프로젝트 탐색기 윈도우에서 멤버 VI를 생성하려는 클래스 또는 인터페이스의 프로젝트 항목을 마우스 오른쪽 버튼으로 클릭한 후, 바로 가기 메뉴에서 새로 만들기»다이나믹 디스패치 템플릿으로부터 VI를 선택합니다. 새로운 VI가 프로젝트 탐색기 윈도우에 나타나고, LabVIEW가 VI를 자동으로 엽니다. VI의 프런트패널이 클래스 또는 인터페이스의 인디케이터, 클래스 또는 인터페이스의 컨트롤, 그리고 여러 입력 및 여러 출력 클러스터를 디스플레이합니다.
3. 프로젝트 탐색기 윈도우에서, 멤버 VI를 생성한 클래스 또는 인터페이스의 프로젝트 항목에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택하여 클래스/인터페이스 프로퍼티 대화 상자를 디스플레이합니다.
4. 아이템 설정을 선택하여 아이템 설정 페이지를 디스플레이하고, 하위 항목이 덮어쓸 다이나믹 디스패치 VI를 선택합니다.
5. 하위 항목이 반드시 덮어쓰기 확인란에 확인 표시를 합니다. 이제 각 하위 항목이 이 VI의 구현을 덮어써야 하기 때문에, 이 VI에 대한 구현을 제공할 필요가 없습니다.

LabVIEW 클래스 또는 인터페이스에서 상속 계층구조 보기

LabVIEW 클래스 계층구조 윈도우는 LabVIEW 클래스 또는 인터페이스의 상속에 대한 그래픽 표시를 디스플레이합니다. 이 윈도우를 사용하여 현재 메모리에 있는 클래스 또는 인터페이스의 계층구조를 확인할 수 있습니다. 다음 단계를 따라 LabVIEW 클래스 계층구조 윈도우를 사용합니다.

1. 보고 싶은 계층구조의 클래스 또는 인터페이스에서 마우스 오른쪽 버튼을 클릭한 후, 클래스 계층구조 보기를 선택합니다. LabVIEW 클래스 계층구조 윈도우가 나타납니다.

2. 다음 옵션 중 선택합니다.

- LabVIEW 클래스 계층구조 윈도우에서 클래스 또는 인터페이스 위로 커서를 이동시켜 클래스나 인터페이스의 이름을 봅니다.
- 윈도우에서 찾으려는 클래스 또는 인터페이스의 이름을 입력하여 LabVIEW 클래스 계층구조를 검색합니다. 검색 문자열이 나타나고 사용자가 입력하면서 테스트가 디스플레이됩니다. LabVIEW는 한 번에 하나씩 검색 문자열로 시작하는 이름을 가진 아이টে을 하이라이트합니다.
- LabVIEW 클래스 계층구조 윈도우에서 보기>> 모든 클래스 보이기를 선택하여 LabVIEW 프로젝트에 있는 모든 클래스나 인터페이스의 계층구조를 봅니다.
- LabVIEW 클래스 계층구조 윈도우의 도구 모음에 있는 버튼을 사용하여 다양한 LabVIEW 클래스 계층구조 보기 방법을 선택할 수 있습니다.

또한, 클래스 탐색기 윈도우를 사용하여 LabVIEW 클래스 계층구조를 볼 수 있습니다.

LabVIEW 클래스 또는 인터페이스에서 상속 변경하기

기본적으로, 모든 LabVIEW 클래스나 인터페이스는 **LabVIEW 객체로부터 상속**됩니다. 다른 클래스로부터 상속된 클래스를 생성하려면, 클래스의 상속을 변경해야 합니다. 다음 단계를 따라 LabVIEW 클래스의 상속을 변경합니다.

1. **두 개의 새로운 LabVIEW 클래스 또는 인터페이스를 생성**하거나 기존의 클래스나 인터페이스를 사용합니다.
2. 상속을 변경하고 싶은 클래스나 인터페이스의 프로젝트 아이টে에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택합니다. 클래스/인터페이스 프로퍼티 대화 상자가 나타납니다.
3. 항목 리스트에서 상속을 선택합니다. 상속 페이지는 현재 LabVIEW 클래스 또는 인터페이스의 상속을 디스플레이합니다. 인터페이스의 상속을 변경하는 경우 5 단계로 건너뛵니다.
4. 다음 단계를 완료하여 직상위 클래스를 변경합니다.
 1. 직상위 클래스 변경 버튼을 클릭하여 직상위 클래스 변경 대화 상자를 디스플레이합니다. 프로젝트의 모든 클래스 리스트에서의 다이아몬드 문양은 사용자가 변경하고자 하는 현재 클래스를 나타냅니다.



노트

인터페이스는 항상 LabVIEW 객체에서 상속되므로 직상위 클래스 변경 버튼은 인터페이스에 사용할 수 없습니다.

2. 현재 클래스가 상속하도록 하려는 클래스를 선택합니다.

노트

- LabVIEW 클래스는 하위 클래스 또는 자신을 프라이빗 데이터의 일부로 사용하는 클래스로부터 상속받을 수 없습니다. 프로젝트의 모든 클래스 리스트에서 LabVIEW 클래스가 상속받을 수 없는 클래스를 선택한 경우, LabVIEW는 그 클래스 옆에 유효하지 않음을 나타내는 문양을 디스플레이합니다.
 - 하나의 기존 구체적 클래스에서 하나의 구체적 클래스를 상속하는 것을 피하십시오. 대신, 인터페이스나 추상 클래스를 두 구체적 클래스에 대한 공통적인 직상위 항목으로 생성합니다. 구체적 직하위 클래스에 대한 상태 전이를 정의해야 하는 경우, 추상 클래스를 공통 클래스로 생성합니다. 그렇지 않은 경우, 인터페이스를 공통 직상위 항목으로 생성합니다.
3. 선택된 것에서 상속 버튼을 클릭합니다. LabVIEW는 클래스/인터페이스 프로퍼티 대화 상자의 상속 페이지로 돌아와 직상위 클래스 계층구조를 업데이트합니다.
 5. 다음 단계를 완료하여 직상위 인터페이스를 변경합니다.
 1. 직상위 인터페이스 변경 버튼을 클릭하여 직상위 인터페이스 변경 대화 상자를 디스플레이합니다.
 2. 사용 가능한 인터페이스 리스트에서 인터페이스를 추가하거나 또는 다음의 인터페이스에서 상속 리스트에서 인터페이스를 제거하여 직상위 인터페이스를 설정합니다.

노트 LabVIEW 클래스나 인터페이스는 자신의 하위 항목에서 상속할 수 없습니다. 다음의 인터페이스에서 상속 리스트에 클래스나 인터페이스가 상속받을 수 없는 인터페이스를 추가한 경우, LabVIEW는 해당 인터페이스 옆에 유효하지 않음을 나타내는 문양을 디스플레이합니다.

3. 확인을 클릭합니다. LabVIEW는 클래스/인터페이스 프로퍼티 대화 상자의 상속 페이지로 돌아와 직상위 인터페이스 리스트를 업데이트합니다.
6. 확인을 클릭하여 상속 설정을 저장하고 클래스/인터페이스 프로퍼티 대화 상자를 닫습니다.

LabVIEW 클래스 또는 인터페이스에 아이템 추가하기

이전에 생성한 아이템을 LabVIEW 클래스 또는 인터페이스에 추가할 수 있습니다. 다음 단계에 따라 LabVIEW 클래스나 인터페이스에 아이템을 추가합니다.

1. **프로젝트 탐색기** 윈도우의 클래스 또는 인터페이스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 추가»파일을 선택합니다. 삽입할 파일 선택 대화 상자가 나타납니다.
2. 추가하려는 파일 위치를 탐색한 후 확인 버튼을 클릭합니다. 프로젝트 탐색기 윈도우에서 선택한 클래스 또는 인터페이스 아래에 파일이 추가됩니다.



노트

LabVIEW 클래스 또는 인터페이스를 위해 생성한 디렉토리에 LabVIEW 클래스나 인터페이스에 추가한 파일을 저장하는 것을 고려하십시오. 이렇게 하면 클래스 또는 인터페이스 배포가 더욱 쉬워질 수 있습니다.

LabVIEW 클래스 또는 인터페이스의 와이어 모양 변경하기

기본적으로 클래스는 LabVIEW의 기본 클래스 와이어를 사용하거나 상위 클래스의 와이어 모양을 상속합니다. 인터페이스는 LabVIEW의 기본 인터페이스 와이어를 사용합니다. LabVIEW 클래스 또는 인터페이스의 와이어 모양을 변경하여 블록 다이어그램을 읽기 쉽게 만들 수 있습니다. 지나치게 많은 와이어 색과 패턴이 블록 다이어그램을 읽는 것을 어렵게 할 수 있다는 것을 염두에 두면서, 와이어 모양을 간간히 변경할 수 있습니다. 다음 단계를 따라 LabVIEW 클래스 또는 인터페이스의 와이어 모양을 변경합니다.

1. **프로젝트 탐색기** 윈도우에서, 연결을 변경하려는 클래스 또는 인터페이스의 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 **프로퍼티**를 선택하여 클래스/인터페이스 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 와이어 모양을 선택합니다.
3. 와이어 모양을 사용자 정의하려면, 사용자 디자인 사용을 선택합니다.
4. 셋팅 섹션에서 원하는 모양으로 와이어를 변경합니다. 와이어 모양, 와이어 색, 그리고 와이어 폭을 변경할 수 있습니다. 와이어 미리보기에서 와이어를 미리 볼 수 있습니다.
5. 변경을 마쳤으면 확인 버튼을 클릭합니다.



노트

LabVIEW 클래스 또는 인터페이스 와이어의 경계 전경색을 변경하면 LabVIEW는 이와 일치하도록 클래스나 인터페이스의 블록 다이어그램 터미널의 경계색을 변경합니다.

블록 다이어그램을 손쉽게 알아보고 관리하기 용이하게 만들기 위해 클래스 와이어 모양을 변경하는 것에 대한 추가적인 팁은 [LabVIEW Style Checklist](#)를 참조하십시오.

다이나믹 디스패치 터미널 생성하기

다이나믹 디스패치 멤버 VI를 생성하려면, 커넥터 팬의 입력 터미널 중 하나를 다이나믹 디스패치 입력(필수)으로 설정해야 합니다. 다음 단계를 따라 커넥터 팬의 입력 터미널을 다이나믹 디스패치 입력(필수)으로 설정합니다.

1. 이미 지정되지 않은 경우 원하는 커넥터 팬 터미널에 **프론트패널 컨트롤 또는 인디케이터를 지정**합니다.
2. 다이나믹 디스패치 입력으로 설정하려는 커넥터 팬의 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 연결은 >> **다이나믹 디스패치 입력(필수)**을 선택합니다.



노트

터미널이 멤버 VI를 소유하는 클래스 또는 인터페이스와 동일한 클래스나 인터페이스 타입인 경우에만 터미널을 다이나믹 디스패치 입력(필수)로 설정할 수 있습니다.

커넥터 팬의 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 연결은 >> 다이나믹 디스패치 출력(권장)을 선택하여 다이나믹 디스패치 출력 터미널을 생성할 수 있습니다.



노트 다이나믹 디스패치 멤버 VI는 하나 이상의 다이나믹 입력을 가질 수 없지만 여러 개의 다이나믹 출력을 포함할 수 있습니다.

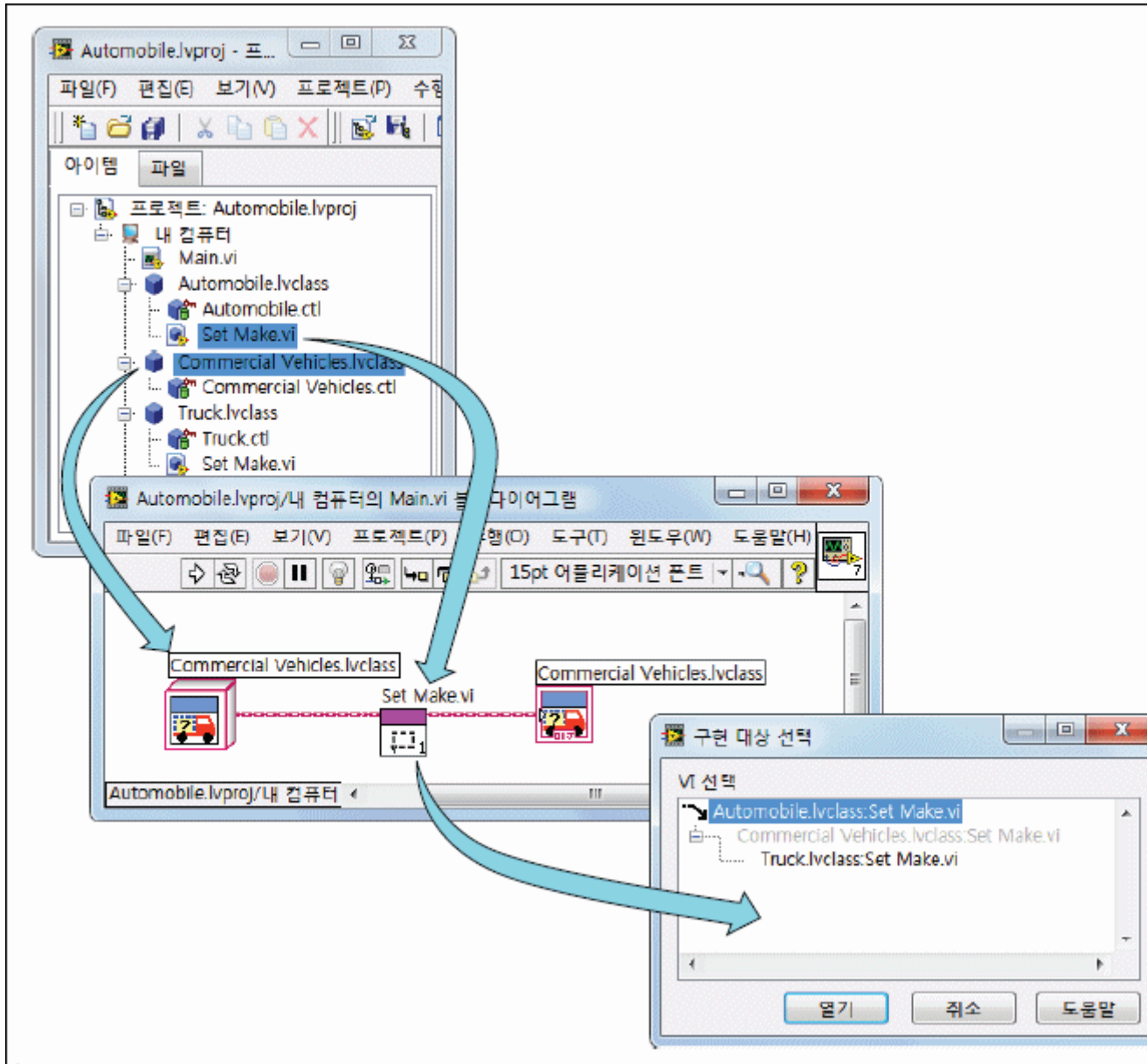
다이나믹 디스패치 멤버 VI의 구현 대상 보기 및 열기

다이나믹 디스패치 SubVI는 LabVIEW 클래스 계층구조의 VI 세트 중 하나를 호출할 수 있습니다. LabVIEW는 다이나믹 디스패치 터미널로 유입되는 클래스 또는 인터페이스의 데이터 타입에 따라 런타임시 호출하는 SubVI의 구현 대상을 결정합니다.

다음 단계를 따라 구현 대상 선택 대화 상자를 사용하여 하나 이상의 다이나믹 디스패치 SubVI 구현 대상을 엽니다.

1. 블록 다이어그램에서 다이나믹 디스패치 VI를 더블 클릭합니다. VI에서 하나 이상의 구현이 있으면, 구현 대상 선택 대화 상자가 나타납니다. 화살표 문양(↗)은 와이어의 런타임 데이터 타입이 와이어의 편집 모드 데이터 타입과 같을 때 실행되는 도구를 나타냅니다. LabVIEW 클래스 또는 인터페이스의 와이어는 자체 타입의 데이터나 직하위 타입의 데이터를 전달할 수 있으므로, 이 문양은 LabVIEW가 런타임시 실행하는 구현이 클래스 계층구조에서 문양 레벨이나 문양 레벨 아래에 있음을 나타냅니다. LabVIEW는 LabVIEW 클래스 계층구조의 모두 클래스와 인터페이스를 나열합니다. 클래스가 상위 클래스의 다이나믹 디스패치 멤버 VI를 상속하지만 자체의 구현이 없는 경우, LabVIEW는 VI 선택 리스트의 클래스나 인터페이스를 비활성화합니다. 또한 블록 다이어그램의 VI에 연결된 클래스나 인터페이스가 VI의 자체 구현을 가지지 않는 경우, VI 선택 리스트에서 VI를 상속하는 상위 클래스의 왼쪽에 화살표 문양이 나타납니다. 아래 예제에서, Commercial Vehicles 클래스는 Automobile 클래스의 Set Make VI를 상속합니다. Commercial Vehicles 클래스가 블록 다이어그램의 Set Make VI의 상위 도구에 연결되어 있으므로, 화살표 문양

이 상위 도구 옆에 나타납니다.



2. VI 선택 리스트에서 열려는 VI를 선택합니다. 또한, <Ctrl> 키를 누르고 여러 VI를 선택하여 여러 VI를 열 수 있습니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.
3. 열기 버튼을 클릭하여 대화 상자를 닫고 선택한 VI를 엽니다.

멤버 VI의 접근 영역 설정하기

멤버 VI에 대한 사용자 접근을 통제하기 위해서 멤버 VI의 접근 영역을 설정할 수 있습니다. 다음의 단계를 따라 멤버 VI의 접근 영역을 설정합니다.

1. 프로젝트 탐색기 윈도우에서, 멤버 VI 접근을 변경하려는 클래스 또는 인터페이스의 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택합니다. 클래스/인터페이스 프로퍼티 대화 상자가 나타납니다.

2. 항목 리스트에서 아이템 셋팅을 선택합니다.
3. LabVIEW 클래스 또는 인터페이스의 아이템에 대한 접근 영역을 사용자 정의하려면, 아이템 셋팅 페이지에 있는 콘텐츠 리스트에서 아이템을 선택합니다.
4. 접근 영역 섹션에서 다음 옵션 중 하나를 선택합니다:
 - 퍼블릭—모든 VI가 멤버 VI를 SubVI로 호출할 수 있습니다.
 - 공동 영역—클래스 또는 인터페이스의 친구, 클래스 또는 인터페이스의 멤버 또는 친구 라이브러리의 멤버인 VI만이 멤버 VI를 호출할 수 있습니다. 공동 영역 멤버 VI는 프로젝트 탐색기 윈도우에서 짙은 파란색 열쇠 문양으로 표시됩니다.
 - 보호됨—오직 동일한 혹은 하위 클래스나 인터페이스에 속한 VI만 멤버 VI를 호출할 수 있습니다. 보호된 멤버 VI는 짙은 노란색 키 문양을 프로젝트 탐색기 윈도우에 디스플레이합니다.
 - 프라이빗—동일한 클래스나 인터페이스에 속한 VI만 멤버 VI를 호출할 수 있습니다. 프라이빗 멤버 VI는 붉은색 키 문양을 프로젝트 탐색기 윈도우에 디스플레이합니다.
 - 미지정—이 옵션은 폴더를 선택할 때에만 나타납니다. 폴더에 지정된 접근 아이템이 없습니다. 접근은 퍼블릭입니다.
5. 접근 영역을 변경하고 싶은 모든 아이템에 대하여 3, 4단계를 반복합니다.
6. 확인 버튼을 클릭하여 클래스/인터페이스 프로퍼티 대화 상자를 닫습니다.

LabVIEW 클래스 또는 인터페이스 잠그기 및 잠금 해제하기

LabVIEW 클래스 또는 인터페이스 잠그기를 통해 프라이빗 멤버 VI에 대한 접근을 막음으로써 어플리케이션에 에러가 발생하는 것을 방지할 수 있습니다. [프로젝트 라이브러리](#)와 관련해서, LabVIEW 클래스나 인터페이스에 암호 보호를 추가한다고 멤버 VI에 암호 보호가 추가되는 것은 아닙니다. 멤버 VI는 개별적으로 암호 보호를 해야 합니다.

LabVIEW 클래스 또는 인터페이스 잠그기

다음 단계를 따라 LabVIEW 클래스 또는 인터페이스를 잠급니다.

1. LabVIEW 클래스 또는 인터페이스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택합니다. 클래스/인터페이스 프로퍼티 대화 상자가 나타납니다.
2. 보호 페이지에서, 잠금(암호 없음) 또는 암호로 보호됨을 선택하여 LabVIEW 클래스나 인터페이스를 잠급니다.
 1. 암호로 보호됨을 선택한 경우, 암호 입력 대화 상자에 암호를 입력하고 확인합니다.
 2. 확인 버튼을 클릭하여 암호 입력 대화 상자를 닫고 클래스/인터페이스 프로퍼티 대화 상자로 돌아갑니다.
3. 확인 버튼을 클릭합니다.

LabVIEW 클래스 또는 인터페이스 잠금 해제하기

잠긴 클래스 또는 인터페이스의 LabVIEW 프로브나 멤버 VI에서 잠긴 클래스의 프라이빗 데이터를 보려면, 해당 클래스 또는 인터페이스의 잠금을 해제해야 합니다. 다음 단계를 따라 클래스 또는 인터페이스의 잠금을 해제합니다.

1. 클래스 또는 인터페이스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택합니다. 클래스/인터페이스 프로퍼티 대화 상자가 나타납니다.
2. 보호 페이지에서 잠금해제 (암호 없음)을 선택합니다. 클래스 또는 인터페이스가 암호로 잠겨있는 경우, 인증 대화 상자에 암호를 입력하고 확인 버튼을 클릭하여 클래스/인터페이스 프로퍼티 대화 상자로 돌아갑니다.
3. 확인 버튼을 클릭합니다.



노트

클래스 또는 인터페이스를 잠그면 **기본 도움말**과 **일반 프로브**의 클래스나 인터페이스의 모양에 영향을 미칩니다. 또한 잠금 상태인 클래스나 인터페이스의 프라이빗 멤버 VI가 **팔레트**에서 숨겨집니다. 다음에 LabVIEW를 실행할 때 팔레트의 변화가 반영됩니다.



노트 사용자가 VI의 프런트패널에 클래스 프라이빗 데이터 컨트롤을 추가하면 LabVIEW는 그 클래스를 잠급니다. VI에서 컨트롤을 삭제한다고 클래스의 잠금 상태가 해제되지 않습니다. 대신에 컨트롤을 제거하고 VI를 닫거나 저장해야, 메모리에서 클래스 컨트롤을 제거할 수 있습니다.

LabVIEW 클래스 또는 인터페이스에 기본 프로브 설정하기

새로운 컴퓨터에 클래스 또는 인터페이스를 옮기거나 기존의 컴퓨터에 LabVIEW를 다시 재설치하는 경우, 클래스나 인터페이스의 기본 프로브 사용자는 일반 프로브를 사용할 수 있습니다. 그러나, LabVIEW 클래스 또는 인터페이스의 사용자 정의 멤버 프로브를 생성하고 이를 클래스나 인터페이스의 기본 프로브로 설정하는 경우, 사용자는 새로운 컴퓨터에 이 클래스나 인터페이스를 배포한 후 기본 프로브로 설정한 사용자 프로브를 이용할 수 있습니다.

사용자가 생성한 **사용자 프로브**를 LabVIEW 클래스 또는 인터페이스 데이터 타입의 기본 프로브로 설정할 수 있습니다. 다음 단계를 따라 사용자 프로브를 LabVIEW 클래스나 인터페이스의 기본 프로브로 설정합니다.

1. 사용자 프로브가 아직 멤버가 아닌 경우, **사용자 프로브를 LabVIEW 클래스 또는 인터페이스에 추가**합니다.
2. LabVIEW 클래스 또는 인터페이스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택합니다. 클래스/인터페이스 프로퍼티 대화 상자가 나타납니다.
3. 항목 리스트에서 프로브를 선택합니다.
4. 기본 프로브 풀다운 메뉴에서 생성한 사용자 프로브의 이름을 선택합니다.

5. 확인 버튼을 클릭하여 클래스/인터페이스 프로퍼티 대화 상자를 닫습니다.



노트

이때 프로브는 LabVIEW 클래스 또는 인터페이스의 멤버여야 합니다.

블록다이어그램

2진 파일로부터 읽기

다음 단계를 따라 **2진 파일**로부터 배정도 부동소수를 읽습니다.

1. **[2진 파일에 쓰기]**는 사용자에게 필요한 .bin 파일을 생성하여 다음 단계를 완성합니다.
2. 블록다이어그램에 [2진 파일로부터 읽기] 함수를 추가합니다.
3. 로그된 데이터의 데이터 타입을 [2진 파일로부터 읽기] 함수에 연결합니다.
 1. 블록다이어그램에 숫자형 상수를 추가합니다.
 2. 숫자형 상수를 [2진 파일로부터 읽기] 함수의 바이트 타입 입력에 연결합니다.
 3. 읽으려는 데이터 타입이 배정도 부동소수라는 것을 LabVIEW에 알려주기 위해 숫자형 상수에 0.0을 입력합니다.

또한 배열을 [2진 파일로부터 읽기] 함수에 연결할 수 있습니다.

4. [2진 파일로부터 읽기] 함수의 데이터 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다.
5. VI를 실행합니다. 나타나는 대화 상자에서 **부동소수 값**을 가진 2진 파일을 선택합니다.

2진 데이터를 읽은 후, 이 데이터를 복사하지 않고 미리 할당된 배열에 넣는 것에 대한 정보는 [미리 할당된 2진 파일로부터 읽기] 함수를 참조하십시오.

2진 형식의 배정도 부동소수 배열로 저장된 기존 파일을 열거나 읽는 방법에 대한 예제는 labview\examples\File IO\Binary\Simple Binary 디렉토리에 있는 Simple Binary File 프로젝트를 참조하십시오.

2진 파일에 쓰기

다음 단계를 따라 배정도 부동소수를 **2진 파일**에 씁니다.

1. 블록다이어그램에 [2진 파일에 쓰기] 함수를 추가합니다.
2. 쓰려는 데이터를 2진 파일에 연결합니다.
 1. 블록다이어그램에 숫자형 상수를 추가합니다.

2. 숫자형 상수를 [2진 파일에 쓰기] 함수의 데이터 입력에 연결합니다.
3. 1.23을 숫자형 상수에 입력합니다. LabVIEW는 숫자형 상수의 형을 배정도, 부동소수로 변경합니다.

또한 배열을 2진 파일에 연결할 수 있습니다.

3. VI를 실행합니다.
4. 나타나는 대화 상자에 number.bin과 같은 파일의 이름을 생성합니다. LabVIEW는 파일 입력으로 지정된 디렉토리에 파일을 생성하고 배정도 부동소수를 파일에 저장합니다.

2진 파일에 쓴 후, 2진 파일 **읽기**를 할 수 있습니다.

[2진 파일에 쓰기] 함수가 배정도 부동소수 값의 배열을 2진 파일에 쓰는 방법에 대한 예제는 labview\examples\File IO\Binary\Simple Binary 디렉토리에 있는 Simple Binary File 프로젝트를 참조하십시오.

설정 파일 VI 사용하기

설정 파일 VI를 사용하여 Windows 설정 셋팅 .ini 파일을 읽고 생성하며 경로와 같은 플랫폼 특성의 데이터를 플랫폼에 독립적인 포맷으로 써서 이 VI가 생성하는 파일을 여러 플랫폼에서 사용할 수 있도록 합니다.



노트

ANSI 포맷인 경우에만 Windows 설정 셋팅 파일에 설정 파일 VI를 사용할 수 있습니다.

다음 단계를 따라 설정 파일 VI를 사용합니다.

1. 블록다이어그램에 [설정 데이터 열기] VI를 추가하여 설정 데이터 객체의 참조를 엽니다. 설정 데이터 객체는 OS의 설정에 대한 선호 또는 셋팅입니다.
2. 설정 데이터 객체의 참조를 연 후, 각 데이터 타입에 [키 읽기] VI와 [키 쓰기] VI를 사용하여 사용자가 입력한 값을 읽고 씁니다.

노트 [키 쓰기] VI는 메모리의 데이터만을 변경합니다.

3. 블록다이어그램에 [설정 데이터 닫기] VI를 추가하여 설정 데이터 객체의 참조를 닫습니다. VI를 실행하여 설정 데이터 객체의 현재 내용을 설정 셋팅 파일에 쓸 수 있습니다.



노트 LabVIEW는 설정 파일 쓰기? 입력을 참으로 설정하고 [설정 데이터 닫기] VI를 호출할 때까지 파일을 쓰지 않습니다.

데이터 로깅

데이터로그 파일로부터 읽기

다음 단계를 따라 데이터로그 파일로부터 데이터를 읽습니다.

1. 블록다이어그램에 [데이터로그 열기/생성/대체] 함수를 추가하여, 데이터로그 파일을 열어서 읽습니다.
2. **데이터 레코드**와 일치하는 **클러스터를 생성**하고, 이 클러스터를 [데이터로그 열기/생성/대체] 함수의 레코드 타입 입력에 연결합니다. LabVIEW는 클러스터를 사용하여 지정된 파일이 레코드 타입과 일치하는 레코드 타입을 포함하는 데이터로그 파일인지 확인합니다. 일치하지 않는 경우, LabVIEW는 VI를 실행할 때 에러를 보고합니다.
3. 블록다이어그램에 [데이터로그 읽기] 함수를 추가합니다.
4. [데이터로그 열기/생성/대체] 함수의 참조 번호 출력 출력을 [데이터로그 읽기] 함수의 참조 번호 입력에 연결합니다.
5. 데이터로그 파일을 닫습니다.
 1. 블록다이어그램에 [파일 닫기] 함수를 추가합니다.
 2. [데이터로그 읽기] 함수의 참조 번호 출력 출력을 [파일 닫기] 함수의 참조 번호 입력에 연결합니다.
6. [데이터로그 읽기] 함수의 레코드 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다. 프런트패널에 인디케이터가 추가됩니다.
7. VI를 실행합니다. LabVIEW는 프런트패널 인디케이터에 데이터를 디스플레이합니다.

데이터로그 파일에 쓰기

다음 단계를 따라 각 레코드를 클러스터 형태의 파일에 씁니다.

1. 블록다이어그램에 [데이터로그 열기/생성/대체] 함수를 추가합니다.
2. [데이터로그 열기/생성/대체] 함수의 동작 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 나타나는 열거형 상수에서 create를 선택합니다.
3. **데이터 레코드**와 일치하는 **클러스터를 생성**하고, 이 클러스터를 [데이터로그 열기/생성/대체] 함수의 레코드 타입 입력에 연결합니다. 클러스터는 LabVIEW가 데이터를 파일에 쓰는 방법을 지정합니다.
4. 블록다이어그램에 [데이터로그 쓰기] 함수를 추가합니다.
5. [데이터로그 열기/생성/대체] 함수의 참조 번호 출력 출력을 [데이터로그 쓰기] 함수의 참조 번호 입력에 연결합니다.
6. 데이터 레코드를 나타내는 컨트롤 또는 상수를 생성하고 [데이터로그 쓰기] 함수의 레코드 입력에 연결합니다.
7. 데이터로그 파일을 닫습니다.

1. 블록다이어그램에 [파일 닫기] 함수를 추가합니다.
2. [데이터로그 쓰기] 함수의 참조 번호 출력 출력을 [파일 닫기] 함수의 참조 번호 입력에 연결합니다.
8. 데이터로그 파일에 쓴 후, **파일 읽기**를 합니다.

LabVIEW 데이터로그 파일에 시뮬레이트된 온도 읽기와 타임스탬프를 저장하는 예제는 labview\examples\File I/O\Datalog 디렉토리에 있는 Datalog File 프로젝트를 참조하십시오.

 예제 열기  관련 예제 찾기

프론트패널 데이터 로깅하기

VI를 실행할 때마다 **프론트패널 데이터 로깅**은 프론트패널 데이터를 별도의 2진 포맷 데이터로그 파일에 저장합니다. 다음과 같이 데이터를 가져올 수 있습니다:

- 데이터를 대화식으로 가져오기 위해 데이터를 저장했던 같은 VI를 사용합니다.
- VI를 **subVI로** 사용하여 프로그램적으로 데이터를 가져옵니다.
- 데이터를 불러오기 위해 [파일 I/O] VI와 함수를 사용합니다.

각 VI는 LabVIEW가 프론트패널의 데이터를 저장한 데이터로그 파일의 위치를 기록하는 로그 파일 연결을 유지합니다. 로그 파일 연결은 VI와 VI의 데이터를 저장한 데이터로그 파일을 연결합니다.



노트

LabVIEW는 리모트 패널을 통해 프론트패널 데이터 로깅을 지원하지 않습니다.

로그 파일 연결 변경

다음 단계를 따라 **로그 파일 연결**을 변경하여 프론트패널 데이터를 로그하거나 다른 데이터로그 파일에서 프론트패널 데이터를 가져옵니다.

1. 수행>>데이터 로깅>>로그 파일 연결 변경을 선택하여 파일 대화 상자를 디스플레이합니다.
2. 다른 데이터로그 파일을 선택하거나 새 데이터로그 파일을 생성한 후 확인 버튼을 클릭합니다.
3. VI를 실행하고 새 로그 파일 연결에 데이터를 **로그**합니다.

로그 파일 연결 제거하기

로그 파일 연결을 사용하여 프론트패널 데이터를 로깅하거나 가져올 때 사용하기 위해 데이터로그 파일을 갖는 VI와 연계합니다. 하나의 VI와 연동된 두 개 또는 더 많은 데이터

로그 파일을 가질 수 있습니다. 이것은 VI 데이터를 테스트하거나 비교할 때 도움이 됩니다. 예를 들어, VI를 처음 실행할 때 로그된 데이터를 VI를 두 번째 실행할 때 로그된 데이터와 비교할 수 있습니다.

수행>>데이터 로깅>>로그 파일 연결 제거를 선택하여 로그 파일 연결을 제거합니다. LabVIEW는 다음에 VI를 실행할 때 데이터로그 파일을 지정하기 위해서 자동으로 로깅을 활성화하거나 또는 대화식으로 로그 데이터를 선택할지를 물어봅니다.

데이터로그 타입

데이터로그 파일에서 각 레코드의 데이터 타입은 모든 타입이 될 수 있습니다. 그러나, 데이터 타입은 데이터로그 파일에 데이터를 로그하는 방법에 따라 변합니다.

[데이터로그 쓰기] 함수를 이용하여 생성한 데이터로그 파일에서, 데이터 타입은 데이터로그 파일에 LabVIEW가 쓴 데이터와 일치합니다.

프론트패널 데이터 로깅을 이용하여 생성한 데이터로그 파일에서, 데이터 타입은 두 개의 클러스터를 가지는 클러스터입니다. 첫번째 클러스터는 타임스탬프를 가지고 있고, 두번째 클러스터는 프론트패널의 데이터 값을 가지고 있습니다. 타임스탬프는 초를 나타내는 32비트의 부호없는 정수형과 LabVIEW 시스템 시간에서 경과된 밀리초를 나타내는 16비트의 부호없는 정수형을 가지고 있습니다. 프론트패널 데이터 클러스터는 각각의 탭 순서를 가진 모든 프론트패널 컨트롤과 인디케이터에 대응하는 데이터 타입을 가지고 있습니다.

데이터로그 레코드 삭제하기

다음 단계를 따라 지정한 데이터로그 레코드를 삭제합니다.

1. 수행>>데이터 로깅>>가져오기를 선택하고 삭제하려는 **레코드를 확인**합니다.
2. 데이터 레코드 삭제하기 버튼을 클릭하여 삭제하려고 하는 레코드에 확인 표시를 합니다. 버튼을 다시 한번 클릭하면, 이 레코드는 삭제 대상에서 제외됩니다.
3. 수행>>데이터 로깅>>데이터 제거를 선택하여 제거하려고 확인 표시한 모든 레코드를 삭제합니다.
4. 검색 시작 & 종료 버튼을 클릭하여 검색 모드를 나갑니다. 검색 시작 & 종료 버튼을 클릭하기 전에 확인 표시한 레코드를 삭제하지 않으면, LabVIEW는 확인 표시된 레코드의 삭제를 요청합니다.

데이터 로깅 활성화하기

데이터 로깅을 자동적으로 수행하거나 대화식으로 수행할 수 있습니다.

다음 단계에 따라 자동 데이터 로깅을 활성화합니다.

1. 자동 데이터 로깅을 활성화하기 위해서 수행>>완료 시 로그를 선택합니다. 자동 데이터 로깅이 활성화되면, 완료 시 로그 메뉴 아이템 옆에 확인 표시가 나타납니다. 다시 수행>>완료 시 로그를 선택하여 자동 데이터 로깅을 비활성화합니다.

2. VI를 실행합니다.

또한, 종료 시 로그 프로퍼티를 사용하여 프로그램적으로 자동적인 데이터로그를 활성화할 수 있습니다.

처음 VI의 프런트패널 데이터를 로그하면, LabVIEW는 데이터로그 파일의 이름을 입력요청합니다. 데이터로그 파일에 대해 .dat나 .txt처럼 어떤 파일 확장자이든지 사용할 수 있습니다. LabVIEW는 VI를 실행할 때마다 데이터를 로그하며 VI를 실행하는 추가적인 시간마다 데이터로그 파일에 새로운 레코드를 추가합니다. LabVIEW가 데이터로그 파일에 레코드를 로그한 후에는 레코드를 덮어쓸 수 없습니다.

다음 단계를 따라 대화식 데이터 로깅을 수행합니다.

1. 데이터를 로그하려는 VI를 실행합니다.
2. VI가 실행을 정지한 후에 수행>>데이터 로깅>>로그를 선택합니다.

이 VI의 프런트패널 데이터를 최초로 로그하는 경우, LabVIEW는 데이터로그 파일의 이름을 기입하라는 입력 요청을 합니다. .dat 파일 확장자를 사용하여 파일에 이름을 붙입니다.



노트

프런트패널 데이터 로깅에서 웨이브폼 차트는 한 번에 한 개의 데이터 포인트만을 로그합니다. 차트 인디케이터에 배열을 연결할 경우, 데이터로그 파일은 차트가 디스플레이하는 배열의 부분을 포함합니다.

SubVI를 사용하여 로그된 프런트패널 데이터를 가져오기

데이터 접근을 활성화할 때, subVI의 입력과 출력은 실제로 출력으로 작용하여 현재 VI의 프런트패널에 로그된 데이터를 반환합니다.



노트

SubVI 또는 익스프레스 VI를 확장가능으로 디스플레이하는 경우, 해당 노드에 대한 데이터베이스 접근을 활성화할 수 없습니다.

다음 단계를 따라 subVI를 사용하여 로그된 프런트패널 데이터를 가져옵니다.

1. 함수 팔레트상의 VI 선택 아이콘이나 텍스트를 클릭합니다.
2. **프런트패널 데이터를 로그한 VI**를 탐색하여 블록다이어그램에 놓습니다.
3. VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터베이스 접근 활성화를 선택합니다. 파일 저장소처럼 생긴 노란색 상자가 VI 주변에 나타납니다.
4. 레코드 # 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택합니다. LabVIEW는 숫자형 컨트롤을 레코드 수를 입력하는데 사용하는 프런트패널에 추가합니다.
5. 유효하지 않은 레코드 # 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다. LabVIEW는 원형 LED를 생성하여, 이를 통해 레코드가 유효한지의 여부를 표시합니다.

6. 프런트패널 데이터 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다. LabVIEW는 프런트패널 데이터를 포함하는 클러스터를 생성합니다.
7. 타임스탬프 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다. LabVIEW는 타임스탬프 값을 포함하는 클러스터를 생성합니다.
8. VI를 실행합니다. 프런트패널 클러스터와 타임스탬프 클러스터는 입력에 연결된 레코드 수에 따라 현재 VI의 프런트패널에 데이터를 반환합니다. 또한, 지정된 레코드 수가 유효하지 않은지 여부를 나타내는 불리언 값을 반환합니다.

음수 레코드 수를 사용하여 최종적으로 로그된 레코드와 상대적인 레코드에 접근할 수 있습니다. -1은 마지막 레코드를 나타내며 -2는 마지막에서 두 번째를 나타내는 식으로 진행되어 -n은 첫 번째 레코드를 나타냅니다. -n에서 n - 1 범위 밖의 숫자를 레코드 # 터미널에 연결하는 경우, 유효하지 않은 레코드 # 출력은 참이고, SubVI는 데이터를 가져오지 않습니다.

다음 단계를 따라 특정한 입력과 출력에 대한 값을 가져옵니다.

1. 함수 팔레트상의 VI 선택 아이콘이나 텍스트를 클릭합니다.
2. 프런트패널 데이터를 로그한 VI를 탐색하여 블록다이어그램에 놓습니다.
3. VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터베이스 접근 활성화를 선택합니다. 파일 저장소처럼 생긴 노란색 상자가 VI 주변에 나타납니다.
4. 레코드 # 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택합니다. LabVIEW는 숫자형 컨트롤을 레코드 수를 입력하는데 사용하는 프런트패널에 추가합니다.
5. 인디케이터를 생성하여 프런트패널 데이터와 타임스탬프 데이터를 가져오는 대신, 직접 subVI의 터미널에 연결하고 인디케이터를 생성합니다.
6. VI를 실행합니다. VI는 연결한 터미널에 따라 VI 프런트패널에 데이터를 반환합니다.

데이터베이스 접근을 비활성화하려면, SubVI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터베이스 접근 비활성화를 선택합니다.

기본 데이터 디렉토리를 선택하기

기본 데이터 디렉토리를 선택하여 LabVIEW가 생성하는 모든 데이터 파일에 대한 기본 위치를 지정합니다.



노트

기본 데이터 디렉토리를 선택하면 측정 파일로부터 읽기 익스프레스 VI, 측정 파일에 쓰기 익스프레스 VI, 기본 데이터 디렉토리 상수, 기본 데이터 디렉토리 프로퍼티에 대한 디렉토리가 설정됩니다.

다음 단계를 따라 기본 데이터 디렉토리를 선택합니다.

1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.

2. 항목 리스트에서 경로를 선택합니다.
3. 폴다운 메뉴로부터 기본 데이터 디렉토리를 선택합니다.
4. 기본 사용 확인란에서 확인 표시를 제거하여 경로 환경 옵션에 접근합니다.
5. 텍스트 박스에 새 경로를 입력하거나 탐색 버튼을 클릭하여 디렉토리를 선택하고 대체 버튼을 클릭합니다. 또한, ! 버튼을 클릭하여, 자동으로 **<osdatadir>** 태그를 경로 필드에 입력할 수 있습니다. **<osdatadir>**는 OS의 기본 파일 디렉토리로의 경로를 나타냅니다.
6. 확인 버튼을 클릭합니다.



노트 기본 데이터 디렉토리 옵션의 모든 변경사항은 LabVIEW를 다시 시작할 때 적용됩니다.

로그된 데이터 대화식으로 보기

데이터를 로그한 후, 대화식으로 볼 수 있습니다.

다음 단계를 따라 데이터를 로그한 VI가 실행을 멈춘 후 로그된 프런트패널 데이터를 봅니다.

1. 수행>>데이터 로깅>>가져오기를 선택합니다. 프런트패널 윈도우의 도구 모음이 데이터 검색 도구 모음이 됩니다. 하이라이트된 숫자는 보고 있는 데이터 레코드를 나타냅니다. 대괄호 안의 숫자는 현재 VI에 로그한 레코드의 범위를 나타냅니다. 날짜와 시간은 언제 선택된 레코드를 로그했는지 나타냅니다.
2. 증가 또는 감소 버튼을 클릭하여 다음 또는 이전 레코드를 봅니다. 또한, 위 아래 방향 화살표 키를 누를 수 있습니다.
3. 검색 시작 & 종료 버튼을 클릭하여 데이터 검색 모드를 나가고 보고 있는 데이터 로그를 가진 VI로 돌아갑니다.

LabVIEW 측정 파일

LabVIEW 측정 .1vm 포맷은 1차원 데이터에 대한 텍스트 기반 파일 포맷입니다. 이 텍스트 기반 측정 파일은 스프레드시트 또는 텍스트 편집 어플리케이션에서 열 수 있도록 탭으로 자리를 구분한 텍스트 파일입니다. .1vm 파일에는 데이터가 생성된 날짜 및 시간과 같은 데이터에 관한 헤더 정보가 포함됩니다. LabVIEW는 데이터를 소수점 여섯 자리까지 .1vm 파일에 저장합니다.

.1vm 포맷은 Microsoft Excel과 같은 스프레드시트 어플리케이션 또는 메모장과 같은 텍스트 편집기로 반입했을 때 분석 및 읽기가 쉽습니다. .1vm 포맷은 여러 데이터 세트, 데이터 세트를 그룹으로 묶기 및 기존 파일에 데이터 세트 추가하기를 지원합니다.

.1vm 파일은 쉼표를 숫자간의 구분문자로 사용합니다. .1vm 파일의 데이터를 문자열에서 숫자로 변환하기 위해서는 현지화 코드 포맷 지정자를 사용하여 소수점을 마침표로 지정합니다.

모든 텍스트 기반 포맷의 경우와 같이 이 파일 포맷은 고성능 또는 매우 큰 대용량의 데이터 세트 용으로 설계되지 않았습니다.



노트

매우 큰 데이터 세트의 경우, 2진 파일 포맷을 사용하십시오.

LabVIEW 측정 파일로부터 데이터 읽기

다음 단계에 따라 [측정 파일로부터 읽기] 익스프레스 VI를 사용하여 LabVIEW 측정 파일로부터 숫자형 또는 웨이브폼 데이터를 읽습니다. [측정 파일에 쓰기] 익스프레스 VI를 사용하여 측정 파일에 **데이터를 씁니다**.

1. 블록 다이어그램에 [측정 파일로부터 읽기] 익스프레스 VI를 추가합니다.
2. 측정 파일로부터 읽기 설정 대화 상자의 파일 이름 텍스트 박스는 입력 파일의 전체 경로를 디스플레이합니다. 다른 파일을 선택하고자 하는 경우 탐색 버튼을 클릭합니다.
3. 파일 포맷 섹션에서, 데이터를 읽으려고 하는 파일의 포맷을 선택합니다.
4. 측정 파일로부터 읽기 설정 대화 상자의 나머지 섹션을 설정하고 확인 버튼을 클릭하여 설정을 저장하고 대화 상자를 닫습니다.
5. [측정 파일로부터 읽기] 익스프레스 VI의 신호 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 숫자형 인디케이터 또는 생성>> 그래프 인디케이터를 선택합니다.
6. 블록 다이어그램의 **모든 에러 입력과 에러 출력 터미널을 연결**합니다.
7. VI를 실행합니다. LabVIEW는 프런트패널 인디케이터의 측정 파일로부터 읽기 설정 대화 상자의 파일 이름 섹션에 지정된 파일로부터 숫자 또는 웨이브폼 데이터를 디스플레이합니다.

LabVIEW 측정 파일에 데이터 쓰기

다음 단계에 따라 [측정 파일에 쓰기] 익스프레스 VI를 사용하여 LabVIEW 측정 파일에 숫자형 또는 웨이브폼 데이터를 씁니다. [측정 파일로부터 읽기] 익스프레스 VI를 사용하여 측정 파일로부터 **데이터를 읽습니다**.

1. 블록 다이어그램에 [측정 파일에 쓰기] 익스프레스 VI를 추가합니다. 측정 파일로 쓰기 설정 대화 상자의 파일 이름 텍스트 박스는 출력 파일의 전체 경로를 디스플레이합니다.
2. 파일 포맷 섹션에서, 데이터를 쓰고자 하는 파일의 포맷을 선택합니다.
3. 부분 헤더 섹션에서, 파일의 헤더 옵션을 선택합니다.
4. 파일 포맷 섹션의 텍스트(LVM)를 선택하였고 데이터를 스프레드시트 또는 텍스트 편집기에서 보려는 경우, X 값 열 섹션을 설정합니다.

5. 측정 파일에 쓰기 설정 대화 상자의 나머지 섹션을 설정하고 확인 버튼을 클릭하여 설정을 저장하고 대화 상자를 닫습니다.



노트

LabVIEW는 이 VI에 연결된 데이터 폼으로부터 자동으로 열 제목을 생성합니다. 열 제목을 바꾸려면 [다이나믹 데이터 속성 설정] 익스프레스 VI를 사용하십시오.

6. 블록 다이어그램에 [신호 시뮬레이션] 익스프레스 VI를 추가하고 VI를 설정합니다.
7. [신호 시뮬레이션] 익스프레스 VI의 출력을 [측정 파일에 쓰기] 익스프레스 VI의 신호 입력에 연결합니다.
8. 블록 다이어그램의 **모든 에러 입력과 에러 출력 터미널을 연결**합니다.
9. VI를 실행합니다. VI는 [신호 시뮬레이션] 익스프레스 VI의 데이터를 측정 파일에 쓰기 설정 대화 상자의 파일 이름 섹션에 지정된 파일에 씁니다.
10. 저장한 데이터를 확인하려면, .lvn 파일의 경우 출력 파일을 스프레드시트 또는 워드 프로세싱 애플리케이션에서 엽니다. 측정 파일에 쓰기 설정 대화 상자의 파일 이름 섹션에 디스플레이된 파일 경로를 사용하여 파일에 접근합니다.

스프레드시트 파일로부터 읽기

다음 단계를 따라 스프레드시트 파일로부터 읽습니다.

1. 블록 다이어그램에 [구분된 스프레드시트 읽기] VI를 추가합니다.
2. 스프레드시트 파일에서 읽으려는 데이터 타입에 대응하는 [구분된 스프레드시트 읽기] VI의 **다형성 인스턴스를 선택**합니다.
3. 데이터를 읽으려는 파일의 **상대 경로**를 생성합니다.
4. 추가된 경로 출력을 [구분된 스프레드시트 읽기] VI의 파일 경로 입력에 연결합니다.
5. 모든 행 출력에서 마우스 오른쪽 버튼을 클릭한 후 생성» 인디케이터를 선택합니다.
6. VI를 실행합니다. LabVIEW는 모든 행 인디케이터에 스프레드시트 파일에서의 데이터를 반환합니다.

스프레드시트 파일에 쓰기

텍스트 기반 스프레드시트 파일에 데이터를 쓰려면, LabVIEW를 사용하여 데이터를 스프레드시트 문자열로 포맷하고 문자열을 파일에 씁니다. 다음은 파일에 쓸 데이터 타입과 사용자에게 필요한 성능이 무엇인지에 따라 LabVIEW를 통해 데이터를 스프레드시트 파일에 쓰는 몇 가지 예입니다:

- 1D 또는 2D 배열 데이터를 한 번 스프레드시트 파일에 쓰려면, [구분된 스프레드시트 쓰기] VI를 사용합니다.
- 여러 다른 데이터 타입의, 임의 차원의 데이터를 쓰려면, [배열을 스프레드시트 문자열로] 함수를 사용합니다.

- 여러 쓰기 작업이 필요한 경우, 디스크 스트리밍 프로그래밍 패턴을 사용하여 데이터를 스프레드시트 파일에 스트리밍합니다.



관련 정보

[스프레드시트 파일에 데이터 스트리밍하기](#) (839 페이지 참조)

스프레드시트 파일에 데이터 스트리밍하기

스프레드시트 파일에 여러 쓰기 작업이 필요한 경우 **디스크 스트리밍**을 사용합니다. 디스크 스트리밍을 사용하면 LabVIEW가 파일을 열고 닫기 위해 OS와 상호작용하는 횟수가 줄어듭니다. 데이터를 스프레드시트 문자열로 포맷하고 이 문자열을 파일에 쓸 때 디스크 스트리밍을 사용할 수 있습니다.

사용 객체

다음 블록 다이어그램 객체를 사용하여 데이터를 포맷하고 파일에 데이터를 스트리밍합니다.

[파일 열기/생성/대체] 함수	While 루프	[문자열로 포맷] 함수	[텍스트 파일에 쓰기] 함수	[파일 닫기] 함수

다음 단계

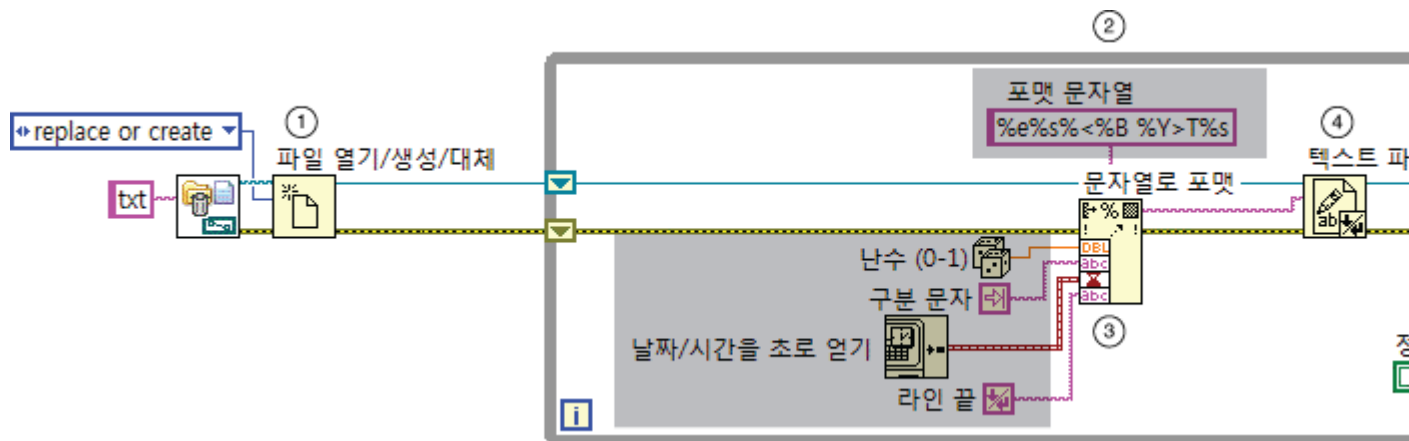
다음 블록 다이어그램을 생성하여 다른 데이터 타입의 데이터를 스프레드시트 문자열로 포맷하고 포맷한 문자열을 파일에 스트리밍합니다.



노트

한 가지 데이터 타입의 데이터를 스프레드시트 문자열로 포맷하려면, [배열을 스프레드시트 문자열로] 함수를 사용합니다.

회색 섹션을 고유한 프로그래밍 목적에 맞도록 사용자 정의합니다.



다음은 위의 다이어그램에서 중요한 사항을 상세히 설명한 것입니다.

① ⑤	일반적인 디스크 스트리밍 작업을 생성하려면, While 루프의 바깥쪽에 [파일 열기/생성/대체] 함수와 [파일 닫기] 함수를 놓습니다.
②	While 루프 때문에 이 VI는 각 반복마다 파일을 열고 닫는 것과 관련된 오버헤드 없이 연속적으로 데이터를 포맷하여 파일에 쓸 수 있습니다. 이전 반복의 값을 다음 반복으로 전달하고자 할 때 시프트 레지스터를 사용합니다.
③	While 루프 바깥쪽의 입력 파라미터를 [문자열로 포맷] 함수의 입력 1..n 입력에 연결합니다. 라인의 끝 상수, 탭 상수 등과 같은 구분 문자를 입력 파라미터로 연결하여 입력 데이터를 구분할 수 있습니다. 포맷 지정자 를 사용하여 포맷 문자열 입력을 지정하여, 입력 파라미터를 스프레드시트 문자열로 포맷합니다.
	노트 또한 [문자열로 포맷] 함수 및 [텍스트 파일 쓰기] 함수 대신에 [파일로 포맷] 함수를 사용하여 데이터를 텍스트로 포맷하고 이 텍스트를 파일에 쓸 수 있습니다.
④	[텍스트 파일에 쓰기] 함수는 스프레드시트 문자열을 파일에 씁니다.

시프트 레지스터를 사용하여 데이터를 버퍼링하면 디스크 스트리밍의 성능을 향상시킬 수 있습니다. 우선, 데이터 버퍼가 있는 시프트 레지스터를 초기화합니다. 그런 다음, 버퍼에 데이터를 씁니다. 버퍼가 꽉 차면, 버퍼를 디스크에 저장합니다.

예제

데이터를 스프레드시트 파일에 스트리밍하는 예제는 labview\examples\File IO\Spreadsheet\Tab-Delimited Data 디렉토리에 있는 Tab-Delimited Data 프로젝트를 참조하십시오. 이 예제는 [배열을 스프레드시트 문자열로] 함수를 사용하여 한 가지 데이터 타입의 데이터를 스프레드시트 문자열로 변환하고 변환된 문자열을 파일에 씁니다.

데이터를 스프레드시트 파일에 스트리밍할 때 데이터를 버퍼링하는 예제는 labview\examples\File IO\Spreadsheet\Tab-Delimited Data 디렉토리에 있는 Buffered Stream to Tab-Delimited Text File VI를 참조하십시오.

TDM/TDMS 파일

사용자가 직접 데이터 파일 포맷을 설계하고 관리하는 수고를 줄이기 위해, National Instruments는 TDM(Technical Data Management)이라는 융통성 있는 데이터 모델을 정의했습니다. TDM은 LabVIEW, LabWindows/CVI, Measurement Studio, SignalExpress 및 DIAdem에서 사용할 수 있습니다. 또한 TDM 데이터 모델을 사용하면, 사용하는 타사 제품에 관계없이 잘 정돈된 데이터를 쓸 수 있습니다.

TDM 데이터 모델은 다음 2개의 파일 포맷을 지원합니다:

- (Windows) TDM — TDM 파일 포맷은 측정 하드웨어가 수집한 원시 데이터와 메타 데이터를 별도의 파일에 저장합니다. 메타 데이터는 측정 채널의 이름 및 단위, 테스트 엔지니어 및 그 외 다른 정보가 될 수 있습니다. 메타 데이터는 XML 포맷에 .tdm 확장자를 갖습니다. 원시 데이터는 컴팩트 2진 포맷에 .tdx 확장자를 갖습니다.
- TDM 스트리밍 (TDMS)—TDMS 파일 포맷은 원시 데이터와 메타 데이터를 모두 2진 포맷에 .tdms 확장자를 갖는 하나의 파일로 저장합니다.



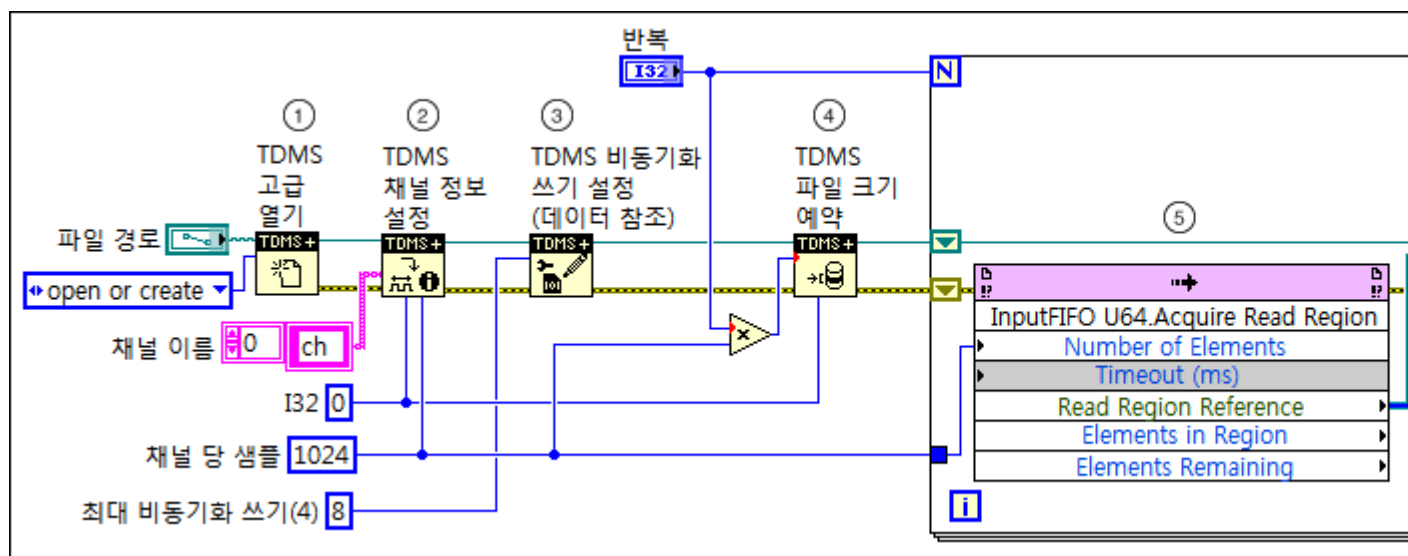
노트

.tdms 파일을 생성하거나 열 때 LabVIEW가 자동으로 .tdms_index 파일을 생성할 수 있습니다. LabVIEW는 .tdms_index 파일을 사용하여 .tdms 파일에 대한 임의의 접근 속도를 높입니다. TDMS 어플리케이션에서 .tdms_index 파일은 옵션입니다. 디스크 공간의 제약이 있는 경우, [TDMS 열기] 함수를 사용하여 이 파일의 생성을 중지합니다.

TDM 데이터 모델은 메타 데이터를 그룹과 채널로 구성하고, 사용자 정의 프로퍼티를 추가합니다. 파일의 원소가 그룹인지, 채널인지 또는 프로퍼티인지 식별이 가능합니다.

외부 데이터를 TDMS 파일에 스트리밍하기 (Windows)

다음 그림은 TDMS 고급 데이터 참조 I/O 함수를 사용하여 .tdms 파일에 외부 데이터를 스트리밍할 때의 가이드라인을 보여줍니다.



다음과 같은 이벤트가 위의 그림에서 발생합니다.



노트

메모리 직접 참조(DMA: direct memory access)를 사용해서 데이터를 전송하려면 NI-RIO 디바이스 드라이버와 LabVIEW FPGA Module이 설치되어 있어야 합니다.



관련 정보

[TDM/TDMS 파일](#) (841 페이지 참조)

텍스트 파일에서 읽기

다음 단계를 따라 텍스트 파일에서 문자나 문자열을 읽습니다.

1. 문자열 인디케이터를 프런트패널에 추가합니다.
2. 블록다이어그램에 [텍스트 파일로부터 읽기] 함수를 추가합니다.
3. [텍스트 파일로부터 읽기] 함수의 파일 입력에서 마우스 오른쪽 버튼을 클릭하고 **생성>>상수**를 선택합니다. 경로 상수가 나타납니다. 읽고자 하는 파일에 대한 절대 경로를 상수에 입력합니다.



팁

경로를 입력하는 대신, 데스크탑에서 파일을 끌어 상수에 놓을 수 있습니다.

4. [텍스트 파일에서 읽기] 함수의 텍스트 출력을 문자열 인디케이터에 연결합니다.
5. VI를 실행합니다. VI가 상수에서 지정한 파일의 텍스트를 읽고, 읽은 텍스트를 프런트패널 문자열 인디케이터에 디스플레이합니다.

텍스트 파일에서 데이터를 읽고 플롯하는 것에 대한 예제는 `labview\examples\File IO\Spreadsheet\Tab-Delimited Data` 디렉토리에 있는 Tab-Delimited Data 프로젝트를 참조하십시오.

텍스트 파일에 쓰기

다음 단계를 따라 텍스트 파일에 씁니다.

1. 문자열 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 텍스트 파일에 쓰려는 문자열 컨트롤 텍스트를 입력합니다.
3. 블록다이어그램에 [텍스트 파일에 쓰기] 함수를 추가합니다.
4. 절대 경로를 [텍스트 파일에 쓰기] 함수의 파일 입력에 연결합니다.
5. 문자열 컨트롤을 [텍스트 파일에 쓰기] 함수의 텍스트 입력에 연결합니다.
6. VI를 실행합니다. VI는 프런트패널 윈도우 문자열 컨트롤의 텍스트를 파일 입력에 연결된 파일에 씁니다.

텍스트 파일에 쓴 후, **파일 읽기**를 합니다.

시뮬레이트된 수집 데이터의 텍스트 스프레드시트를 파일에 쓰는 예제는 labview\examples\File IO\Spreadsheet\Tab-Delimited Data 디렉토리에 있는 Tab-Delimited Data 프로젝트를 참조하십시오.

LabVIEW 데이터를 XML로 변환하기

LabVIEW 데이터를 XML로 변환하여 데이터를 포맷하면 데이터를 파일로 저장할 때, 데이터를 설명하는 태그로부터 데이터의 값, 이름, 타입을 쉽게 알 수 있습니다.

다음 단계를 따라 LabVIEW 데이터를 XML 포맷으로 변환합니다.

1. 블록다이어그램에 [패턴화된 XML로] 함수를 추가합니다.
2. [패턴화된 XML로] 함수의 임의의 데이터 입력으로 변환하려는 데이터를 연결합니다.
3. 함수의 xml 문자열 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다. 이제 블록다이어그램은 문자열 인디케이터 터미널을, 프런트 패널은 문자열 인디케이터를 가집니다.
4. VI를 실행합니다. LabVIEW는 데이터 타입을 **LabVIEW XML 스키마**에 따라 XML 포맷으로 변환합니다. 프런트패널 xml 문자열 인디케이터는 XML 포맷의 데이터를 디스플레이합니다.

LabVIEW 데이터를 XML 포맷으로 변환하는 예제는 labview\examples\File IO\XML\Flatten and Unflatten XML\Flatten and Unflatten XML.lvproj를 참조하십시오.

 예제 열기  관련 예제 찾기

XML을 LabVIEW 데이터로 변환하기

LabVIEW 데이터로 변환하려는 XML 문자열은 반드시 **LabVIEW XML 스키마**를 따라야만 합니다. XML 문자열로부터 하나 이상의 데이터 타입을 변환할 수 없습니다.

다음 단계를 따라 XML 문자열을 LabVIEW 데이터로 변환합니다.

1. 블록다이어그램에 [XML로부터 패턴화 해제] 함수를 추가합니다.
2. 변환하려는 데이터를 포함하는 XML 문자열을 [XML로부터 패턴화 해제] 함수의 xml 문자열 입력에 연결합니다.
3. 함수의 타입 입력에 상수 또는 컨트롤을 연결하여 XML 문자열에 데이터의 데이터 타입을 지정합니다. 예를 들어, XML 문자열이 배정도 부동소수의 배열인 경우, 데이터 타입을 타입 입력에 연결합니다.
4. 함수의 값 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 같은 데이터 타입의 인디케이터를 타입 입력으로 생성합니다.

니다. 블록다이어그램 인디케이터 터미널이 나타나고 프런트패널 인디케이터가 나타납니다.

5. VI를 실행합니다. LabVIEW는 xml 문자열을 LabVIEW XML 형식을 따라서 타입의 LabVIEW 데이터 타입으로 변환합니다. 값 인디케이터가 LabVIEW 데이터를 디스플레이합니다.

XML을 LabVIEW 데이터로 변환하는 예제는 labview\examples\File IO\XML\Flatten and Unflatten XML\Flatten and Unflatten XML.lvproj를 참조하십시오.



예제 열기



관련 예제 찾기

자동으로 블록다이어그램 정리하기

LabVIEW는 블록다이어그램에서 기존의 와이어의 경로를 모두 다시 설정하고, 객체를 재정렬할 수 있습니다.

다음 단계에 따라 블록다이어그램에서 기존 와이어의 경로를 모두 다시 설정하고, 기존 객체를 모두 재정렬합니다.

1. 정리 옵션을 설정합니다. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이하고 항목 리스트에서 블록다이어그램을 선택합니다. 블록다이어그램 페이지에서 LabVIEW가 자동으로 컨트롤을 다이어그램 왼쪽으로 옮기고, 인디케이터를 다이어그램 오른쪽으로 옮기고, 블록다이어그램 객체와 와이어 사이에 지정된 수의 픽셀을 놓고, 블록다이어그램 레이아웃을 간결하게 만들도록 설정할 수 있습니다.
2. (선택 옵션) 구조에서 마우스 오른쪽 버튼을 클릭하고 다이어그램 정리에서 제외를 선택하여, LabVIEW가 그 구조의 내용을 재구성하지 않으면서 구조를 옮길 수 있도록 허용합니다.
3. 편집>>다이어그램 정리를 선택합니다. 또한, 블록다이어그램 도구 모음에서 다이어그램 정리 버튼 을 클릭하거나 <Ctrl-U> 키를 눌러도 됩니다. (macOS) <Command-U> 키를 누릅니다. (Linux) <Alt-U> 키를 누릅니다.



노트

와이어나 개별 노드같이 특정 객체를 다시 정리할 수도 있습니다. 여러 객체를 선택하려면 <Shift>를 누른 후, 클릭합니다. 정리할 객체를 선택하면, 다이어그램 정리 버튼이 다이어그램 부분 정리로 변경됩니다. LabVIEW는 전체 블록다이어그램이 아니라 선택한 객체만을 정리합니다. 특정 와이어를 선택해 정리하면, LabVIEW는 와이어만 정리하며, 와이어에 연결된 객체는 정리하지 않습니다.

또한, 블록다이어그램에서 선택하는 와이어와 객체를 재정렬할 수 있습니다.

다음 단계에 따라, 블록다이어그램에서 선택한 기존의 와이어 및 객체를 자동으로 경로 재설정하고 재정렬합니다.

1. (선택 옵션) 구조에서 마우스 오른쪽 버튼을 클릭하고 다이어그램 정리에서 제외를 선택하여, LabVIEW가 그 구조의 내용을 재구성하지 않으면서 구조를 옮길 수 있도록 허용합니다.
2. 정리하려는 아이템을 둘러싸도록 마우스를 클릭하고 끌어서 선택 사각형을 만듭니다.



노트 사용자가 구조를 선택하지 않고 선택한 객체가 서로 겹치지 않는 경우, 또는 대응하는 입력이나 출력이 없이 와이어만 선택한 경우에 LabVIEW는 블록다이어그램을 정리하지 않습니다.

1. (선택 옵션) <Shift> 키를 누른 채 추가로 객체 주변에 선택 사각형을 클릭하고 끌어서 만들면, 여러 선택을 생성할 수 있습니다. 또한, <Shift> 키를 누른 채로 선택 해제하려는 객체를 클릭할 수 있습니다.
2. 편집>>선택 영역 정리를 선택합니다.

블록다이어그램에 컨트롤과 인디케이터 생성하기

블록다이어그램에 컨트롤과 인디케이터를 생성할 수 있습니다. 블록다이어그램의 VI, 함수, 또는 익스프레스 VI의 왼쪽 측면에 입력, 또는 컨트롤을 배치합니다. 오른쪽 측면에 출력, 또는 인디케이터를 배치합니다.

SubVI나 익스프레스 VI를 확장 가능 노드로 디스플레이하는 경우, 터미널을 볼 수 있게 노드의 크기를 조정하면 와이어 연결이 보다 용이합니다. **SubVI 또는 익스프레스 VI를 아이콘으로 디스플레이**하는 경우, 와이어 연결이 보다 쉽도록 **터미널을 디스플레이**합니다.

다음 단계를 따라 대화식으로 블록다이어그램에 컨트롤을 생성합니다.

1. 블록다이어그램의 VI, 함수, 또는 익스프레스 VI의 입력 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 정확한 데이터 타입의 컨트롤을 생성합니다. LabVIEW는 컨트롤을 생성하고 마우스 오른쪽 버튼을 클릭한 입력 터미널에 그 컨트롤을 연결합니다.

다음 단계를 따라 대화식으로 블록다이어그램에 인디케이터를 생성합니다.

1. 블록다이어그램의 VI, 함수, 또는 익스프레스 VI의 출력 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 정확한 데이터 타입의 인디케이터를 생성합니다. LabVIEW는 인디케이터를 생성하고 출력 터미널에서 마우스 오른쪽 버튼을 클릭하여 인디케이터를 연결합니다.

블록다이어그램의 익스프레스 VI, VI, 또는 함수의 다이나믹 데이터 타입 출력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>그래프 인디케이터를 선택하여 데이터를 그래프에 출력하거나 바로 가기 메뉴에서 생성>>숫자형 인디케이터를 선택하여 데이터를 숫자형 인디케이터에 디스플레이합니다.

파일»VI 프로퍼티를 선택한 후 항목 풀다운 메뉴에서 편집자 옵션을 선택하여, 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»컨트롤 또는 생성»인디케이터를 선택할 때 LabVIEW가 생성하는 컨트롤 또는 인디케이터의 스타일을 변경합니다. 도구»옵션을 선택하고 항목 리스트에서 프런트패널을 선택하여, 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»컨트롤 또는 생성»인디케이터를 선택할 때 LabVIEW가 새 VI에 생성하는 컨트롤 또는 인디케이터의 스타일을 변경합니다.

사용자 정의 상수 생성 및 편집하기

상수는 블록다이어그램에 고정 데이터 값을 제공하는 블록다이어그램의 터미널입니다. 범용 상수는 원주율(π)과 무한대(∞) 등 고정된 값을 갖는 상수입니다. 사용자 정의 상수는 VI를 실행하기 전에 정의하고 편집하는 상수입니다.

다음 단계를 따라 사용자 정의 상수를 생성합니다.

1. VI 또는 함수의 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»상수를 선택합니다. LabVIEW는 입력 터미널의 기본 데이터 타입의 상수를 생성하고 값을 하이라이트합니다.
2. 상수에 지정하려는 값을 입력하고 <Enter> 키를 누릅니다. 또한, 위와 아래 방향 화살표 키를 눌러서 값을 상수에 지정할 수 있습니다. LabVIEW는 상수를 크기 조정하여 내용에 맞춥니다. **라벨의 크기를 조정**하는 것과 같은 방법으로 문자열 상수의 크기를 조정할 수 있습니다.
3. 수행 도구 또는 라벨링 도구를 사용하여 상수를 클릭하고 그 값을 편집합니다. **자동 도구 선택이 활성화**된 경우, 상수를 더블 클릭하여 라벨링 도구로 바꾼 후 값을 편집합니다.



노트

LabVIEW는 **상수 계산 단순화**를 사용하여 VI의 성능을 최적화합니다. 도구»옵션을 선택하고, 항목 리스트에서 블록다이어그램을 선택한 후, 상수 계산 단순화 및 루프 불변값 와이어에 해쉬 마크 보이기 확인란에 확인 표시를 하면, 블록다이어그램에서 상수 계산 단순화를 디스플레이할 수 있습니다.

또한, 다음 방법으로 상수를 생성할 수 있습니다:

- 상수를 **검색**하거나 함수 팔레트에서 선택한 후, 블록다이어그램에 추가합니다. 대부분의 상수는 팔레트의 맨 위 또는 맨 아래에 위치합니다.
- LabVIEW 도움말에서 상수를 검색한 후 블록다이어그램에 추가하기 버튼을 클릭하면 커서에 상수가 놓여서 이를 블록다이어그램에 추가할 수 있습니다. 팔레트에서 찾기 버튼을 클릭하여 함수 팔레트에서 상수를 하이라이트합니다.
- 프런트패널 컨트롤을 블록다이어그램으로 끌어옵니다. 프런트패널 컨트롤을 블록다이어그램으로 끌어 이동하면 LabVIEW는 프런트패널 컨트롤의 값으로 상수를 생성합니다. 참조 번호 상수의 값은 참조 번호 아님입니다. 컨트롤은 프런트패널에 남습니다. 프런트패널 컨트롤의 값을 변경해도 블록다이어그램의 상수값에 영향을 주지 않으며 그 반대의 경우도 마찬가지입니다.

아이콘 또는 데이터 타입에서 터미널 디스플레이하기

블록다이어그램에서 프런트패널 컨트롤 또는 인디케이터가 아이콘으로 나타나거나 데이터 타입 터미널로 나타나도록 설정할 수 있습니다. 초기설정에 의해, 프런트패널 객체는 블록다이어그램에서 아이콘 터미널로 나타납니다. 아이콘 터미널을 사용하면 블록다이어그램에서 프런트패널 객체의 타입을 디스플레이할 수 있고, 동시에 프런트패널 객체의 데이터 타입도 나타낼 수 있습니다. 데이터 타입 터미널을 사용하여 블록다이어그램의 공간을 절약합니다.

다음 단계를 따라 블록다이어그램에 데이터 타입으로서 터미널을 디스플레이합니다.

1. 블록다이어그램의 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴의 아이콘으로 보기를 선택하여 확인표시를 제거합니다. 터미널을 아이콘으로 디스플레이하려면, 다시 바로 가기 메뉴에서 아이콘으로 보기를 선택합니다.



노트

아이콘 터미널은 데이터 타입 터미널보다 크기 때문에, 데이터 타입 터미널을 아이콘 터미널로 변환할 때 의도하지 않게 다른 블록다이어그램 객체를 가릴 수 있습니다.

옵션 대화 상자의 블록다이어그램 페이지에서 프런트패널 터미널을 아이콘으로 놓기 옵션을 설정하여 기본적으로 생성한 새 프런트패널 객체의 터미널을 아이콘 또는 데이터 타입으로 디스플레이하도록 LabVIEW를 설정할 수 있습니다.

블록다이어그램 객체 제거하고 다시 연결하기

블록다이어그램 객체를 제거하고 다시 연결할 때, LabVIEW는 선택한 객체를 제거하고, 이 객체와 연결된 모든 와이어와 상수를 제거한 후, 제거한 객체의 입력과 출력에 연결되어 있던 데이터 타입과 동일한 데이터 타입의 와이어를 연결합니다.



노트

<Delete> 또는 <Backspace>를 사용하여 블록다이어그램 객체를 삭제하는 경우, 이 객체만 제거됩니다. LabVIEW는 삭제된 객체의 입력과 출력을 다시 연결하지 않습니다.

하나 이상의 블록다이어그램 객체를 제거하고 다시 연결하려면, 제거할 객체 주위에 사각형 선택 영역을 생성합니다. 다음 방법 중 하나를 선택하여 사각형 선택 영역에서 객체를 제거하고 다시 연결할 수 있습니다.

- 사각형 선택 영역에서 마우스 오른쪽 버튼으로 클릭하고 제거 및 다시 연결을 선택합니다.
- 빠른 탐색 대화 상자를 열고 **빠른 탐색 키보드 바로가기 키** <Ctrl-R>을 누릅니다.

LabVIEW는 또한 사각형 선택 영역에 있는 독립 라벨을 포함한 장식을 제거합니다.

블록다이어그램 객체 삽입하기

다음 단계를 따라 와이어로 연결되어 있는 블록다이어그램에 객체를 삽입합니다.

1. 객체를 삽입하려는 위치에 있는 와이어에서 마우스 오른쪽 버튼을 클릭합니다.



팁

객체 하단에 있는 가장 적절한 터미널에 와이어를 연결하려면 와이어에서 또는 와이어 바로 위에서 마우스 오른쪽 버튼을 클릭합니다. 객체 상단에 있는 가장 적절한 터미널에 와이어를 연결하려면, 와이어 바로 아래에서 마우스 오른쪽 버튼을 클릭합니다.

1. 바로 가기 메뉴에서 삽입을 선택하여 함수 팔레트를 디스플레이합니다.
2. 함수 팔레트의 VI 또는 함수를 선택하여 와이어에 새로운 객체를 삽입합니다. 객체의 터미널 수 또는 데이터 타입이 다른 경우, **깨진 와이어**가 생길 수 있습니다.

블록다이어그램 객체 대체하기

해당 포인트까지 객체에 대해 선택한 모든 편집 옵션이 바뀌지 않고 블록다이어그램 객체를 다른 객체로 대체할 수 있습니다.

다음 단계를 따라 블록다이어그램 객체를 대체합니다.

1. 블록다이어그램 객체에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 대체를 선택하여, 함수 팔레트를 디스플레이합니다.



노트

선택한 블록다이어그램 객체에 따라, 다른 대체 옵션을 볼 수 있습니다. 예를 들어, While 루프에서 마우스 오른쪽 버튼을 클릭하면, While 루프를 For 루프 또는 (Real-Time, Windows) Timed 루프로만 대체하도록 선택할 수 있습니다.

1. 함수 팔레트의 VI 또는 함수를 선택하여 블록다이어그램의 객체를 대체합니다. 객체의 터미널 수 또는 데이터 타입이 다른 경우, **깨진 와이어**가 생길 수 있습니다.



노트 메모리의 VI와 같은 이름을 가진 VI를 선택할 때, 대화 상자가 나타나 VI를 대체하거나 작업을 취소하도록 입력요청합니다. 대체를 선택한 경우, LabVIEW는 메모리의 모든 VI를 선택한 VI로 대체합니다.

노드 크기 조정하기

SubVI, 익스프레스 VI, 프로퍼티 노드, 배열 함수와 같은 몇몇 함수와 같이 블록다이어그램의 대부분의 노드는 그 크기를 조정할 수 있습니다.



노트

SubVI와 익스프레스 VI를 확장 가능 노드로 볼 때에만 크기를 조정할 수 있습니다. [배열 인덱스], [배열 부분 대체], [배열에 삽입], [배열 잘라내기] 함수는 자동으로 크기를 조정하여 사용자가 연결하는 입력 배열의 차원에 맞춥니다.

다음 단계를 따라 블록다이어그램의 노드의 크기를 조정합니다.

1. 위치 도구를 노드 위로 이동합니다. 노드의 위와 아래에 나타나는 핸들의 크기를 조정합니다.
2. 커서를 크기 조정 핸들 위로 이동하여 커서를 크기 조정 커서로 변경합니다.
3. 크기 조정 커서를 사용하여 노드의 경계를 아래로 끌어 추가적인 터미널을 디스플레이합니다.
4. 마우스 버튼을 놓습니다.

크기 조정 작업을 취소하려면, 마우스 버튼을 놓기 전에 노드 경계를 블록다이어그램 윈도우를 지나도록 끕니다.

또한, subVI 또는 익스프레스 VI에 하나의 입력 또는 출력을 삽입할 수 있습니다.



관련 정보

[SubVI와 익스프레스 VI를 아이콘 또는 확장 가능 노드로 디스플레이하기](#) (1337 페이지 참조)

[입력과 출력을 삽입하고 제거하기](#) (1337 페이지 참조)

코드의 일부 재사용하기

블록다이어그램에서 코드의 일부 또는 VI 스니펫을 저장하여, 향후에 다시 사용하거나 다른 LabVIEW 사용자와 공유할 수 있습니다. LabVIEW는 코드를 .png 이미지 파일에 내장시킵니다. 이 이미지는 사용자 코드를 그림으로 보여주고 사용자가 지정하는 실제 코드를 포함하고 있습니다.

다음 단계를 완료하여 블록다이어그램에서 코드의 일부를 재사용합니다.

1. 블록다이어그램에서 코드 섹션을 선택합니다.



노트

저장하지 않은 SubVI를 포함하는 코드 섹션을 선택하지 마십시오. 코드 섹션에 SubVI가 포함되어 있으면, 먼저 반드시 이 SubVI를 디스크에 저장해야 합니다.

1. 편집 » 선택사항에서 VI 스니펫 생성을 선택합니다.
2. 대화 상자가 나타나면, 선택한 VI를 저장하려는 디렉토리를 선택하거나 검색합니다.
3. 확인 버튼을 클릭합니다.
4. (Windows와 macOS)(옵션) .png 파일을 저장했던 디렉토리에서 끌어와서 블록다이어그램 위에 놓습니다. LabVIEW는 파일 안에 있는 실제 코드를 놓고, 코드에서 연결되지 않은 와이어 부분에 자동으로 컨트롤과 인디케이터를 연결합니다.
5. (선택 옵션) 다음 중 한 방법으로 다른 LabVIEW 사용자와 .png 파일을 공유합니다.
 - 이메일과 같은 웹 문서에 파일을 첨부합니다. (Windows와 macOS) 다른 사용자가 이 파일을 다운로드하고 블록다이어그램에 파일을 끌어놓아서 코드를 사용할 수 있습니다.
 - 메시지 포럼과 같은 HTML 페이지에 이미지를 내장시킵니다. (Windows와 macOS) 다른 사용자가 이 이미지를 블록다이어그램에 끌어놓아서 코드를 사용할 수 있습니다.



노트 (Linux) VI의 일부분을 저장하고 공유하여 코드의 이미지를 나타낼 수 있습니다. 그러나 VI의 일부분을 블록다이어그램에 끌어놓아서 코드를 사용할 수는 없습니다.

LabVIEW는 .png 파일에 SubVI를 내장시키지 않습니다. VI의 일부에 대한 .png 파일을 블록다이어그램에 놓을 때, 해당 VI 부분에서 호출하는 SubVI는 모두 사용자의 컴퓨터에 있어야 합니다. 그렇지 않은 경우, 그 VI 스니펫에는 깨진 코드가 포함됩니다.

이미지 편집 소프트웨어에서 VI 스니펫 .png 파일을 열면, LabVIEW는 VI 스니펫이 저장되어 있는 LabVIEW의 버전을 이미지의 오른쪽 위에 디스플레이합니다. VI 스니펫의 코드를 사용하려면 사용하고 있는 LabVIEW의 버전이 VI 스니펫이 저장된 LabVIEW의 버전과 같거나 그 이후 버전이어야 합니다.



노트 이미지 편집 소프트웨어에 VI 스니펫 .png 파일을 저장할 때 소프트웨어가 .png 파일의 코드를 제거할 수도 있습니다. 그 후 .png 파일을 블록다이어그램에 추가하면, 블록다이어그램에는 코드는 나타나지 않고, 이미지만 나타나게 됩니다. 코드를 잃는 것을 막기 위해 .png 파일은 이미지 편집 소프트웨어에 저장하지 마십시오.

다형성 VI의 인스턴스를 수동으로 선택하기

다형성 VI를 블록다이어그램에 놓을 때, 기본 다형성 VI는 다형성 VI 윈도우의 인스턴스 VI 리스트에서 첫번째 인스턴스입니다. 다형성 VI 터미널에서 컨트롤, 인디케이터, 상수를 생성할 때, 인스턴스는 각 객체의 데이터 타입을 결정합니다. 기본 인스턴스를 변경하려면, 다형성 VI 윈도우에서 화살표 버튼을 사용하여 인스턴스 VI 리스트의 선택된 인스턴스를 리스트의 맨 위쪽으로 움직입니다.

대부분의 다형성 VI에서, 다형성 VI의 입력에 연결된 데이터 타입은 사용할 인스턴스를 결정합니다. 예를 들면, [키 읽기] VI의 기본값 입력 터미널에 문자열 상수를 연결하면,

LabVIEW는 다형성 VI의 문자열 인스턴스를 선택하고 값 출력 터미널은 문자열 데이터 타입으로 변환합니다.

다형성 VI 입력에 연결한 데이터 타입이 사용할 인스턴스를 결정하지 않는다면, 그 인스턴스를 수동으로 선택해야 합니다. 다형성 VI의 인스턴스를 수동으로 선택하면, 그 VI는 선택한 인스턴스의 데이터 타입만을 받고 반환하기 때문에 이 VI는 더이상 다형성 VI로서 작동하지 않습니다.

다음의 방법을 사용하여 다형성 VI의 인스턴스를 수동으로 선택합니다. 내장된 다형성 VI에 대해, 커서를 바로 가기 메뉴를 통해 움직일 때 기본 도움말 윈도우는 각 인스턴스의 설명을 디스플레이합니다.

- 블록 다이어그램에서 다형성 VI에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 타입 선택을 선택하고 사용할 인스턴스를 선택합니다.
- 수행 도구를 사용하여 다형성 VI 선택자를 클릭하고 바로 가기 메뉴에서 인스턴스를 선택하십시오. 블록 다이어그램의 다형성 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 다형성 VI 선택자를 선택하여 선택자를 디스플레이합니다. 다음 그림은 다형성 VI 선택자의 보기를 보여줍니다.

읽기 전용 ▼

처리되는 모든 데이터 타입을 다시 받을 수 있도록 다형성 VI를 변경하려면 다형성 VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 타입 선택 >> 자동을 선택하거나, 수행 도구를 사용하여 다형성 VI 선택자를 클릭하고 바로 가기 메뉴에서 자동을 선택합니다.



팁

다형성 VI 선택자와 연관된 다형성 VI 노드를 찾으려면 선택자에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 다형성 VI 찾기를 선택합니다.

와이어 사용하기

자동으로 객체 연결하기

선택한 객체를 블록 다이어그램의 다른 객체 가까이로 움직일 때, LabVIEW는 임시 와이어를 그려서 유효한 연결을 보여줍니다. 객체를 블록 다이어그램에 추가하기 위해 마우스 버튼을 놓으면, LabVIEW는 자동으로 와이어를 연결합니다. 또한 이미 블록 다이어그램에 있는 객체를 연결할 수도 있습니다. LabVIEW는 가장 잘 일치하는 터미널을 연결하고 일치하지 않는 터미널은 연결하지 않습니다.

위치 도구를 사용하여 객체를 움직이는 동안 스페이스 바를 눌러서 자동 와이어링을 활성화 또는 비활성화할 수 있습니다. 기본 설정상, 함수 팔레트에서 객체를 선택할 때 또는 블록 다이어그램에 이미 있는 객체를 복제할 때 자동 와이어링이 활성화됩니다. 위치 도구를 사용하여 이미 블록 다이어그램에 위치한 객체를 움직일 때 기본적으로 자동 와이어링은 비활성화됩니다. 자동 와이어링이 활성화되었을 때, 선택된 객체는 객체를 끌 때 모양을 유지합니다. 자동 와이어링이 비활성화되었을 때, 선택된 객체는 객체를 끌 때 점선 표시로 나타납니다.

다음 단계를 통해 블록다이어그램에 객체를 추가할 때 자동으로 객체를 연결하는 방법을 연습합니다.

1. 필요한 경우, **자동 와이어링을 활성화**하고 자동으로 연결할 객체에 대한 최소와 최소 거리를 설정합니다.
2. 숫자형 컨트롤을 프런트패널 윈도우에 추가합니다.
3. <Ctrl-E> 키를 눌러서 블록다이어그램을 디스플레이합니다. (macOS) <Command-E> 키를 누릅니다. (Linux) <Alt-E> 키를 누르십시오.
4. 함수 팔레트에서 [더하기] 함수를 선택합니다.
5. 블록다이어그램에서 함수의 왼쪽 측면을 숫자형 컨트롤 터미널의 오른쪽 측면에 가깝게 이동합니다. 함수를 터미널에 가깝게 이동할 때, LabVIEW는 유효한 연결을 보여주는 임시 와이어를 그립니다. 단계 1에서 설정한 최대 및 최소 거리내에 함수가 있는지를 확인합니다.
6. 숫자형 컨트롤 터미널과 [더하기] 함수의 왼쪽 위 터미널 사이에 임시 와이어가 나타날 때, 마우스 버튼을 클릭하여 함수를 추가하고 객체를 연결합니다.
7. 모든 필수 터미널이 연결되었는지를 확인합니다. 그렇지 않은 경우, vi는 깨지고 실행되지 않을 것입니다. 기본 도움말 윈도우를 사용하여 블록다이어그램 노드에 필요한 터미널이 무엇인지 확인합니다. **필수 터미널**의 라벨은 기본 도움말 윈도우에 굵은체로 나타납니다.

자동 와이어링 설정하기

다음 단계를 따라 자동 와이어링을 활성화하거나 설정합니다.

1. 도구>> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 블록다이어그램을 선택합니다.
3. 자동 와이어링을 활성화하기 위해 자동 와이어링 활성화 확인란에 확인 표시를 합니다.
4. 최소 거리와 최대 거리를 편집하여, 객체를 자동으로 연결하는데 필요한 객체 간 최소 거리를 픽셀 단위로 설정하고, 자동으로 와이어를 연결할 수 있는 객체 간 최대 거리를 픽셀 단위로 설정합니다.

와이어 삭제하기

다음 단계를 따라 모든 와이어 또는 부분을 삭제합니다.

1. 위치 도구를 사용하여 와이어를 한번 클릭하여 와이어의 한 부분을 선택하고 두 번 클릭하여 가지를 선택하고 세 번 클릭하여 전체 와이어를 선택합니다.
2. <Delete> 키 또는 <Backspace> 키를 누릅니다.

와이어 결가지를 제거하려면 와이어 결가지를 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 와이어 결가지 삭제를 선택합니다.



노트

먼저 와이어를 삭제하지 않고 기존의 와이어 경로를 변경할 수 있습니다. 같은 터미널 사이에 새로운 와이어 경로를 생성하고, LabVIEW가 이전 와이어 경로를 삭제합니다.

VI, 함수, 익스프레스 VI의 터미널 디스플레이하기

와이어를 연결하기 쉽도록 SubVI, 함수, 익스프레스 VI의 커넥터 팬 터미널을 디스플레이합니다.



노트

SubVI 또는 익스프레스 VI를 확장 가능으로 디스플레이하는 경우, 해당 노드의 터미널을 디스플레이할 수 없습니다.

다음 단계를 따라 SubVI, 함수, 익스프레스 VI의 터미널을 디스플레이합니다.

1. VI 또는 함수에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 보이는 아이템 >> 터미널을 선택합니다.

아이콘으로 되돌아가려면, VI 또는 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 터미널을 선택하여 터미널 바로 가기 메뉴 아이템 옆에 있는 확인 표시를 제거합니다.

와이어 움직이기

다음 경로를 따라 일부 와이어나 모든 와이어를 다시 위치시킵니다.

1. 위치 도구를 사용하여 와이어를 한번 클릭하여 와이어의 한 부분을 선택하고 두 번 클릭하여 가지를 선택하고 세 번 클릭하여 전체 와이어를 선택합니다.
2. 새 위치로 와이어를 끕니다. 또한, 이동 방향과 진행을 제한할 수 있습니다.

인접하고 선택되지 않은 부분이 늘어나서 새 위치에 맞게 조절됩니다.

다음 단계를 따라 하나 이상의 와이어 부분을 다시 위치시킵니다.

1. 위치 도구를 사용하여 와이어를 한번 클릭해서 와이어의 한 부분을 선택합니다.
2. (옵션) <Shift> 키를 누르고 위치 도구를 사용하여 와이어를 한번 클릭해서 와이어의 다른 부분을 선택합니다.
3. 새 위치로 그 부분을 끕니다. 또한, 이동 방향과 진행을 제한할 수 있습니다.

연속적이지 않은 부분을 포함해서 동시에 여러 와이어 부분을 선택하고 끌어올 수 있습니다.

터널을 **움직이면** 때때로 여분의 와이어 부분이 생성되는데, 이것은 **숨겨진 와이어 부분**이라고 불리고 구조 경계 아래에 놓입니다. 이 부분은 숨겨져 있기 때문에 이 부분을 선택하고 끌어올 수 없습니다. 터널에 어느 와이어가 연결되어 있는지 확실하지 않으면, 와이어를 세 번 클릭하여 전체 와이어 경로를 봅니다.



노트

와이어를 선택하고 움직이지 않고 기존의 와이어의 경로를 변경할 수 있습니다. 같은 터미널 사이에 새로운 와이어 경로를 생성하고, LabVIEW가 이전 와이어 경로를 삭제합니다.

와이어 경로 설정하기

LabVIEW는 와이어링할 때, 와이어 경로를 자동으로 지정합니다. LabVIEW는 루프 및 구조와 같은 블록 다이어그램에 있는 기존 객체의 주변으로 와이어의 경로를 정합니다. 또한, LabVIEW는 와이어의 굽어진 곳의 수를 줄이도록 경로를 정합니다. 가능하다면, 컨트롤 터미널에서 자동적으로 경로가 정해진 와이어는 터미널의 오른쪽 편으로 나오고 인디케이터 터미널에서 자동적으로 경로가 정해진 와이어는 터미널의 왼쪽으로 들어갑니다.

기존의 와이어의 경로를 자동으로 설정하려면, 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 와이어 정리를 선택합니다.



노트

또한, 블록 다이어그램에서 **자동으로 기존의 와이어를 모두 재정리하고 기존의 객체를 모두 재정렬**할 수 있습니다.

다음 단계를 따라 일시적으로 자동 와이어 경로 설정을 비활성화해서 수동으로 와이어의 경로를 설정할 수 있습니다.

1. 와이어링 도구를 사용하여 터미널을 클릭하고 마우스를 놓습니다.
2. <A> 키를 누릅니다. LabVIEW는 일시적으로 현재 와이어의 자동 와이어 경로 설정을 비활성화합니다.
3. (옵션) 마우스 버튼을 클릭하여 와이어를 설정한 후 커서를 새 방향으로 이동합니다. 스페이스바를 눌러 와이어 방향을 수평과 수직 사이에서 변환합니다.
4. 다른 터미널을 클릭하여 객체를 연결하고 와이어를 끝냅니다. LabVIEW는 자동 와이어 경로 설정을 다시 활성화합니다.

또한, 와이어를 시작하거나 고정시킨 후 다른 터미널에 연결하거나 포인트를 고정하는 동안 마우스 버튼을 누른 후 마우스 버튼을 놓아서 자동 경로 설정을 임시로 비활성화할 수 있습니다. 마우스 버튼을 놓은 후, LabVIEW는 자동 경로 설정을 다시 활성화합니다.

적절한 블록 다이어그램 옵션을 설정하여 모든 새 와이어의 자동 경로 설정을 비활성화할 수 있습니다.

와이어 선택하기

위치 도구를 사용하여 와이어를 한번 클릭하여 와이어의 한 부분을 선택하고 두번 클릭하여 가지를 선택하고 세번 클릭하여 전체 와이어를 선택합니다.

선택된 와이어 그룹에 와이어를 추가하기 위해, 새 와이어를 선택하는 동안 <Shift> 키를 누릅니다.

위치 도구를 사용하여 선택하려는 와이어의 부분을 둘러싸는 선택 사각형을 끕니다.

와이어를 선택한 후에, 이를 **삭제**하거나 **이동**시킬 수 있습니다.

블록다이어그램 객체 수동으로 연결하기

와이어를 통하여 블록다이어그램 객체 사이에 데이터를 전송합니다. 와이어링 도구를 사용하여 VI 또는 함수의 터미널을 다른 VI 또는 함수의 터미널에 연결합니다. **자동 와이어링**을 사용하여 블록다이어그램에 객체를 놓을 때 객체를 연결합니다.

다음 단계를 따라 블록다이어그램에서 와이어링을 연습합니다.

1. 블록다이어그램에 두 VI 또는 함수를 놓습니다.
2. 기본 도움말 윈도우를 디스플레이하고 커서를 VI 또는 함수 위로 이동합니다. 기본 도움말 윈도우는 VI 또는 함수의 각 터미널 옆에 라벨을 나열합니다.
3. 기본 도움말 윈도우의 각 터미널 옆의 라벨을 사용하여 어떤 터미널에 연결하려는 지 결정합니다.
4. 와이어링 도구를 사용하여 커서를 VI 또는 함수의 출력 터미널 위로 이동합니다. 와이어링 도구를 터미널 위로 이동할 때, 터미널이 깜박거리며 팁 상자가 나타납니다.
5. 와이어링 도구를 사용하여 터미널을 클릭하고 마우스를 놓습니다. 커서를 블록다이어그램을 넘어서 이동하면, LabVIEW는 와이어가 실패에서 풀리는 것처럼 터미널과 와이어링 도구 사이에 와이어를 그립니다.



팁

시작한 와이어를 취소하려면 <Esc> 키를 누르거나 마우스 오른쪽 버튼을 클릭합니다.

1. 마우스 버튼을 누르지 않고, 커서를 다른 VI 또는 함수의 입력 터미널로 이동합니다. 두번째 터미널이 깜빡이고 팁 상자가 나타납니다.
2. 와이어링 도구를 사용하여 두번째 터미널을 클릭합니다. 터미널에 와이어를 연결하면 깨진 와이어가 생성될 수 있습니다. VI를 실행하기 전에 **깨진 와이어를 수정**해야 합니다.
3. 모든 필수 터미널이 연결되었는지를 확인합니다. 기본 도움말 윈도우를 사용하여 블록다이어그램 노드에 필요한 터미널이 무엇인지 확인합니다. **필수 터미널**의 라벨은 기본 도움말 윈도우에 굵은체로 나타납니다. 그렇지 않은 경우, VI는 깨지고 실행되지 않을 것입니다.



팁 연결하는 동안 와이어를 굽히려면 클릭하여 와이어를 고정한 후 수직 방향으로 커서를 움직입니다. 와이어 연결없이 와이어를 고정한 후 와이어를 만드는 것을 멈추기 위해서는 더블 클릭을 합니다.

기존의 와이어를 연결하려면, 기존의 와이어에서 작업을 시작하거나 끝내고 위의 단계를 마치십시오. 와이어링 도구가 올바르게 위치되면 와이어가 깜빡이게 되어 새 와이어를 기존 와이어에 연결할 수 있습니다. **자동 도구 선택이 활성화**될 때 기존 가지에서 연결하려면, 커서가 와이어링 도구로 바뀔 때까지 와이어쪽으로 커서를 이동합니다. 또한, 기존 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 와이어 결가지 생성을 선택할 수 있습니다.

와이어를 삭제하고, **와이어를 이동**하고, **와이어를 선택**하고, **구조를 와이어로 연결**할 수 있습니다.

피해야 할 연결 기술

효율적으로 연결하여 블록 다이어그램의 모양을 향상시킵니다. 복잡한 연결 구조는 에러를 일으키지는 않으나, 블록 다이어그램을 읽고 디버그하는 것을 어렵게 하거나 VI의 작동을 잘못 표현할 수 있습니다.



노트

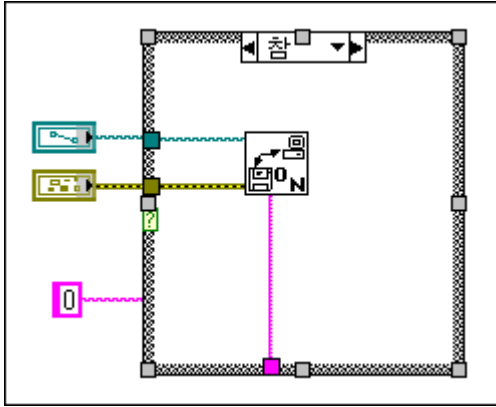
와이어에 무엇을 연결해야할지 모르는 경우, 더블 클릭하여 가지를 선택하거나 트리플 클릭하여 전체 와이어를 선택합니다.

객체 아래에 연결

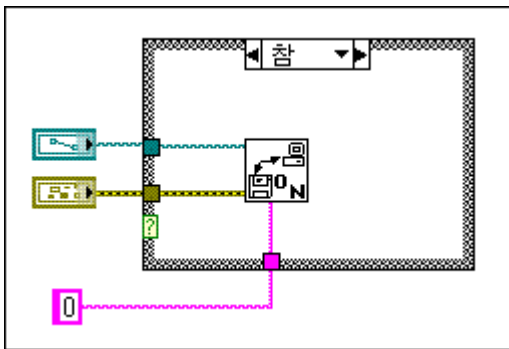
와이어 위에 객체를 놓지 않습니다. 터미널 또는 아이콘을 와이어 위에 놓으면 연결이 없는 경우에도 연결이 존재하는 것처럼 보입니다. 와이어를 아이콘 또는 터미널을 통과하여 끄는 경우 또한 연결된 것처럼 보이나, 실제 와이어는 아이콘 뒤에 위치합니다.

숨겨진 와이어 부분

LabVIEW가 최종 와이어의 일부분을 숨길 수 있으므로 구조 경계의 아래 또는 겹쳐진 객체 사이에는 연결하지 마십시오. 다음 그림과 같이, 터널을 이동시키거나 구조를 확대함으로써 의도하지 않게 숨겨진 와이어 부분이 생길 수 있습니다.



숨겨진 와이어 부분이 다시 나타나도록 다음 그림에서와 같이 상수에 연결된 와이어 부분을 끌거나, 상수를 이동하거나, 구조를 크기 조정할 수 있습니다. 또한 와이어에서 마우스 오른쪽 버튼을 클릭하여 바로 가기 메뉴에서 와이어 정리를 선택하여 숨겨진 와이어 부분을 볼 수 있습니다. 자동으로 블록다이어그램 상에 있는 모든 와이어를 재정리하고 객체를 재정렬하려면, 편집>>다이어그램 정리를 선택합니다. 또한 블록다이어그램의 도구 모음에서 다이어그램 정리 버튼  을 클릭하거나, <Ctrl-U> 키를 눌러서 블록다이어그램을 정리할 수 있습니다. (macOS) <Command-U> 키를 누릅니다. (Linux) <Alt-U> 키를 누릅니다.



구조 연결하기

구조밖에 있는 터미널에서 구조 안에 있는 터미널로 직접 연결할 수 있습니다. LabVIEW는 와이어가 구조 경계를 지나는 터널을 생성합니다. 터널을 통하여 데이터가 루프와 구조의 안과 밖으로 흐를 수 있습니다.

피해야 할 연결

구조 연결에서 다음의 경우는 피합니다:

- 터널 겹치기.
- 다층 시퀀스 구조의 여러 프레임에서 연결하기.
- 케이스 구조의 일부 케이스에서 터널을 연결하지 않음
- 시퀀스 로컬 터미널에 하나 이상의 값 지정하기.

상수에만 연결된 구조의 경우, LabVIEW는 상수 계산 단순화를 사용하여 성능을 최적화합니다. 도구 >> 옵션 >> 블록 다이어그램을 선택한 후 블록 다이어그램 페이지에서 상수 계산 단순화 구조에 해쉬 마크 보이기 확인란에 확인 표시를 하면, 상수 계산 단순화 구조 내에서 해쉬 마크를 디스플레이하도록 LabVIEW를 설정할 수 있습니다.



관련 정보

[터널 겹치기](#) (859 페이지 참조)

[다중 시퀀스 구조의 여러 프레임에서 연결](#) (859 페이지 참조)

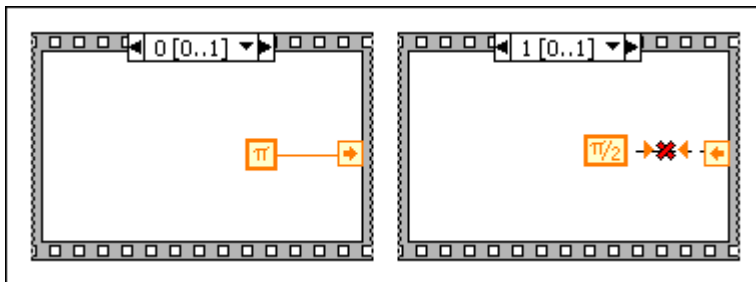
[구조에서 모든 케이스의 터널 연결 안됨](#) (858 페이지 참조)

[하나 이상의 값을 시퀀스 로컬 터미널에 할당하기](#) (858 페이지 참조)

[블록 다이어그램 객체](#) (52 페이지 참조)

하나 이상의 값을 시퀀스 로컬 터미널에 할당하기

하나의 프레임에서만 다중 시퀀스 구조의 시퀀스 로컬 터미널에 값을 할당할 수 있으나, 이 값을 모든 뒤따르는 프레임에서 사용할 수 있습니다. 다음 예의 왼쪽에서, π 의 값은 0번 프레임의 시퀀스 로컬 터미널에 할당됩니다. 다른 값을 1번 프레임의 같은 시퀀스 로컬 터미널에 할당하려고 하면, 아래 예에서 오른쪽과 같이 **깨진 와이어**가 생성됩니다.



구조에서 모든 케이스의 터널 연결 안됨

여러 케이스가 있는 구조에서 발생하는 일반적인 에러는 일부 케이스를 연결하지 않는 것입니다. 구조의 바깥에 있는 객체와 연결할 때 모든 케이스에 대해 소스와 연결되지 않으면 그 결과로 에러가 발생합니다.

와이어로 연결되지 않은 케이스에서 자동으로 입력 및 출력 터널이 연결되도록 터널을 설정하는 방법도 있습니다.



관련 정보

[구조를 사용하여 하나 이상의 코드 섹션 실행하기](#) (373 페이지 참조)

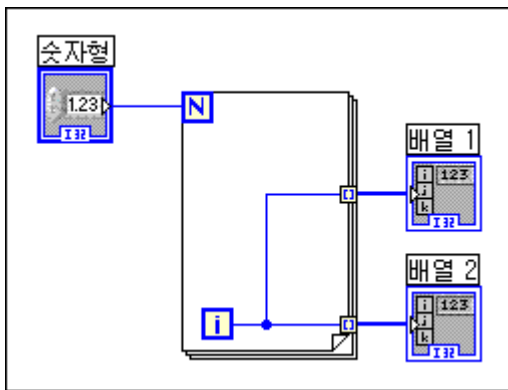
[여러 케이스가 있는 구조에서 자동으로 터널 연결하기](#) (860 페이지 참조)

터널 겹치기

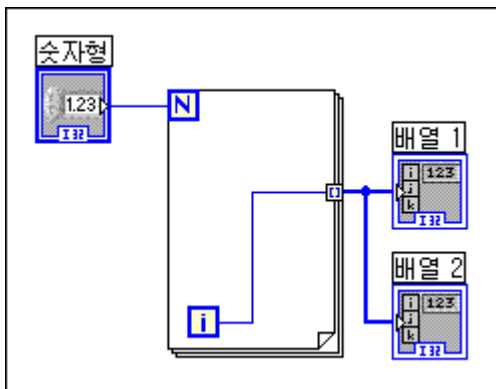
LabVIEW는 와이어를 연결할 때 터널을 생성하기 때문에, 터널이 종종 겹치는 경우가 있습니다. 터널 겹치기는 블록 다이어그램의 실행에는 영향을 미치지 않으나, 어떤 터널이 어떤 와이어 경로에 속하는지 구별하기 어렵기 때문에 편집하기 어렵습니다.

터널이 겹친 경우, 한 터널을 이동시켜 다른 터널이 보이도록 합니다. 또한 터널에서 모든 와이어를 제거하여 터널을 삭제한 후 구조를 올바르게 다시 연결할 수 있습니다.

구조 내부의 객체를 구조 외부의 객체에 연결해야 하고, 구조 내부의 객체가 이미 구조 외부로 데이터를 전달하는 와이어를 가지고 있는 경우, 다음 블록 다이어그램에서처럼 다시 구조를 통과하여 연결하지 마십시오.



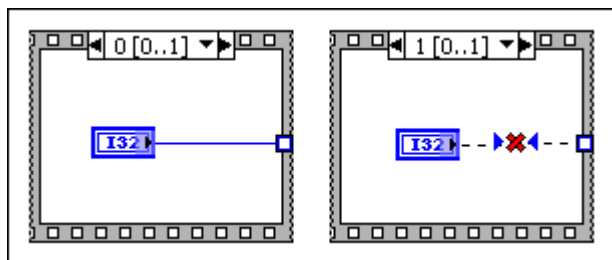
그 대신, 다음에서와 같이 터널에서 두번째 와이어를 만드십시오.



케이스 구조에서, 겹치는 터널은 각 케이스에 연결된 것으로 나타날 수 있습니다.

다층 시퀀스 구조의 여러 프레임에서 연결

이 에러는 두 다층 시퀀스 구조 프레임이 같은 터널에 값을 지정하려할 때 발생합니다. 다음 블록 다이어그램에서와 같이 터널이 하얀색으로 변하여 이 에러를 표시합니다.



같은 터널에 두 소스를 연결하지 마십시오.

여러 케이스가 있는 구조에서 자동으로 터널 연결하기

다음 단계를 완료하여 다중 시퀀스 구조를 제외한 다른 **여러 개의 케이스를 포함하는 구조**에서 자동으로 입력 및 출력 터널을 연결합니다.

1. 구조에 있는 한 케이스의 입력과 출력 터널을 연결한 후, 출력 터널에서 마우스 오른쪽 버튼을 클릭하여 링크된 입력 터널 바로 가기 메뉴를 디스플레이합니다.
2. 링크된 입력 터널 바로 가기 메뉴에서 다음 와이어링 옵션 중에서 하나를 선택합니다.
 - 와이어 연결 없는 케이스 생성 & 연결—LabVIEW는 와이어로 연결하지 않은 기존 케이스 모두에 터널을 연결합니다. LabVIEW는 또한 사용자가 생성하는 새 케이스에도 모두 터널을 연결합니다.
 - 생성—LabVIEW는 사용자가 생성하는 모든 새 케이스에 자동으로 터널을 연결합니다. 기존 케이스는 연결이 안된 채로 놔둡니다.

터널 선택기 커서가 나타납니다.



노트

사용자가 링크할 수 있는 터널이 없는 경우, LabVIEW는 이러한 옵션을 비활성화합니다.

1. 출력 터널과 연결 또는 링크하려는 입력 터널을 클릭합니다. 입력 및 출력 터널에 흰색 삼각 문양이 나타나서, 서로 링크되어 있음을 표시합니다.

와이어 연결 없는 케이스 생성 & 연결을 선택한 경우, LabVIEW가 모든 케이스에 터널을 연결하는 것을 확인할 수 있습니다. 구조에 케이스 또는 **서브 다이어그램을 추가**하는 경우, LabVIEW는 새 케이스에 터널을 연결합니다. 생성을 선택한 경우, LabVIEW는 사용자가 새로 추가하는 케이스에만 터널을 연결합니다. 기존 케이스는 연결이 안된 채로 남아 있습니다.

링크된 출력 터널에서 마우스 오른쪽 버튼을 클릭하고 링크된 입력 터널 >> 찾기를 선택하여, 대응하는 입력 터널을 찾습니다. 링크된 출력 터널에서 마우스 오른쪽 버튼을 클릭하고 링크된 입력 터널 >> 삭제를 선택하여, 입력 터널과 출력 터널 사이의 연결을 제거합니다. 사용자가 선택한 터널에 링크된 입력 터널이 없는 경우, LabVIEW는 이 옵션을 비활성화합니다.



노트 링크된 출력 터널에서 마우스 오른쪽 버튼을 클릭하고 연결되지 않으면 기본값 사용을 선택한 경우, LabVIEW는 입력 터널과 출력 터널 사이의 연결을 제거합니다.

LabVIEW의 검색 옵션

프런트패널 객체에서 블록다이어그램 터미널 찾기

다음 단계를 따라 프런트패널 객체에서 블록다이어그램 터미널을 찾습니다.

1. 프런트패널 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 터미널 찾기를 선택합니다. LabVIEW는 블록다이어그램 터미널을 하이라이트합니다.

블록다이어그램 터미널에서 프런트패널 객체 찾기

다음 단계를 따라 블록다이어그램 터미널에서 프런트패널 객체를 찾습니다.

1. 블록다이어그램의 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 컨트롤 찾기 또는 인디케이터 찾기를 선택합니다. LabVIEW는 프런트패널 컨트롤 또는 인디케이터를 하이라이트합니다.



팁

또한 블록다이어그램을 더블 클릭하여 프런트패널 객체를 찾을 수 있습니다.

VI의 모든 인스턴스 또는 타입 정의 찾기

다음 단계를 따라 VI의 모든 인스턴스 또는 타입 정의를 찾습니다.

1. 블록다이어그램 또는 VI 계층구조 윈도우 안의 VI 또는 타입 정의에서 마우스 오른쪽 버튼을 클릭합니다. 또한, 프런트패널 또는 블록다이어그램의 오른쪽 위 코너의 VI 아이콘 또는 커넥터 팬을 마우스 오른쪽 버튼으로 클릭할 수 있습니다. (Statechart) Statechart 편집기 윈도우의 오른쪽 상단 코너에 있는 아이콘에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 모든 인스턴스 찾기를 선택합니다. LabVIEW는 검색 결과 윈도우에 결과를 디스플레이합니다.

VI나 타입 정의의 일부 또는 모든 인스턴스를 대체할 수 있습니다. 또한, 블록다이어그램 객체 또는 텍스트를 찾고 대체할 수 있습니다.



관련 정보

[사용자 컨트롤, 인디케이터, 타입 정의 생성하기](#) (92 페이지 참조)

[블록다이어그램 객체 찾고 대체하기](#) (865 페이지 참조)

[블록다이어그램 객체 찾고 대체하기](#) (865 페이지 참조)

[VI에서 텍스트 찾고 대체하기](#) (864 페이지 참조)

팔레트에서 객체 찾기

객체 이름으로 검색하기

검색하는 팔레트 객체의 이름을 아는 경우, 빠른 탐색 대화 상자를 사용하여 해당 객체를 블록다이어그램이나 프런트패널 윈도우에 바로 추가할 수 있습니다. 다음 키보드 바로 가기 키를 사용하여 빠른 탐색 대화 상자를 디스플레이합니다:

- (Windows) <Ctrl-Space> 키를 누릅니다.
- (macOS) <Command-Shift-Space> 키를 누릅니다.
- (Linux) <Alt-Space> 키를 누릅니다.
- 중국어 키보드에서는, <Ctrl-Shift-Space> 키를 누릅니다.



팁

옵션 대화 상자의 컨트롤/함수 팔레트 페이지에 있는 시작하는 동안 팔레트 로드 라디오 버튼을 활성화시키면 보다 빠르게 빠른 탐색 대화 상자를 열 수 있습니다. LabVIEW를 시작하는 동안 팔레트 로드를 선택하면 LabVIEW가 열리는 속도가 조금 느려질 수 있습니다.

기능으로 검색하기

검색하는 팔레트 객체의 이름을 알지 못하는 경우, 다음과 같은 방법을 사용할 수 있습니다:

- [어플리케이션 전체 검색](#) 기능을 사용하여 키워드 검색
- 컨트롤 또는 함수 팔레트를 탐색하여 객체 검색

객체 또는 로컬 변수 터미널 찾기

다음 단계를 따라 객체 또는 로컬 변수 터미널을 찾습니다.

1. 컨트롤, 인디케이터 또는 터미널을 마우스 오른쪽 버튼으로 클릭합니다.

2. 바로 가기 메뉴에서 찾기»로컬 변수를 선택합니다. 이는 오직 여러 로컬 변수가 있는 경우에만 나타납니다.
3. 검색 결과 윈도우에서 로컬 변수를 선택하고 이동하기 버튼을 클릭합니다. LabVIEW는 로컬 변수를 하이라이트합니다.

객체 또는 터미널에 링크된 프로퍼티 노드 찾기

다음 단계를 따라 객체 또는 터미널에 링크된 프로퍼티 노드를 찾습니다.

1. 컨트롤, 인디케이터 또는 터미널을 마우스 오른쪽 버튼으로 클릭합니다.
2. 바로 가기 메뉴에서 찾기»프로퍼티 노드를 선택합니다. 이는 오직 여러 프로퍼티 노드가 있는 경우에만 나타납니다.
3. 검색 결과 윈도우에서 프로퍼티 노드를 선택하고 이동하기 버튼을 클릭합니다. LabVIEW는 프로퍼티 노드를 하이라이트합니다.

디렉토리에서 VI 찾기

도구»디스크에서 VI 찾기를 선택하여 디렉토리에서 파일 이름으로 VI를 찾습니다. 정확한 파일 이름을 검색하거나 와일드카드를 사용하여 파일 이름의 일부를 검색할 수 있습니다. 또한, LLB 파일 안의 VI를 검색할 수 있습니다.

다음 단계에 따라 디렉토리와 그 서브디렉토리에서 VI를 찾습니다.

1. 도구»디스크에서 VI 찾기를 선택하여 디스크에서 VI 찾기 윈도우를 디스플레이합니다.
2. 검색할 디렉토리 필드 옆의 탐색 버튼을 클릭하고, VI를 검색할 디렉토리를 탐색한 후, 현재 폴더 버튼을 클릭합니다.
3. 검색할 파일 이름 텍스트 박스에 검색하려는 텍스트를 입력합니다. 임의의 여러 문자와 일치시키기 위해 별표 와일드카드 *를 사용합니다.
4. 찾기 버튼을 클릭합니다. 검색 결과 리스트박스는 LabVIEW가 그 결과를 찾을 때 디스플레이합니다. 진행 중인 검색을 취소하려면 정지 버튼을 클릭합니다.
5. 하나 이상의 검색 결과를 선택하고 선택한 VI 열기 버튼을 클릭하여 VI를 엽니다. 또한 검색 결과를 더블 클릭하여 VI를 열 수 있습니다. <Ctrl> 키를 누른 상태에서 검색 결과를 추가로 클릭하면 하나 이상의 결과를 선택할 수 있습니다. (macOS) <Command>-<Shift>를 클릭합니다. (Linux) <Alt>-<Shift>를 누른 채로 클릭합니다. <Shift> 키를 누르고 현재 결과 위와 아래에 있는 검색 결과를 클릭하여 현재 결과와 선택한 두번째 결과 사이의 모든 결과를 선택합니다.
6. 디스크에서 VI 찾기 윈도우에서 닫기 버튼을 클릭하여 윈도우를 닫습니다.

또한 텍스트의 특정한 객체나 문자열을 위해 **메모리에서 VI를 검색**할 수 있습니다.

VI, 객체, 텍스트 찾고 대체하기

어플리케이션 인스턴스의 한 VI 또는 여러 VI에서 특정한 객체나 텍스트 문자열을 검색하고 해당 항목을 다른 항목으로 대체할 수 있습니다. 편집>>찾기와 대체를 선택하여 찾기 대화 상자를 디스플레이합니다. 찾기 대화 상자는 객체 또는 텍스트를 검색하느냐에 따라서 다른 검색 옵션을 디스플레이합니다. 찾기 대화 상자를 사용하여 VI, 익스프레스 VI, 함수, 컨트롤, 인디케이터, 타입 정의, 글로벌 변수, 프로퍼티 노드, 브레이크포인트, 또는 텍스트를 위한 메모리 안에 있는 VI를 검색합니다. 현재 VI나, 어플리케이션 인스턴스에 있는 선택된 VI, 또는 어플리케이션 인스턴스의 모든 VI를 검색할 수 있습니다.

(Statechart) Statechart의 guard와 action 코드를 포함하여 statechart 다이어그램에서 특정한 객체나 텍스트의 문자열을 검색할 수 있습니다.



노트

또한, 디렉토리에서 파일 이름으로 VI를 검색할 수 있습니다.

LabVIEW가 찾은 모든 항목을 대체하거나, 대체하려는 항목을 선택할 수 있습니다.



노트 실행 중인 VI의 객체나 텍스트는 대체할 수 없습니다.



관련 정보

[어플리케이션 인스턴스로 작업하기](#) (308 페이지 참조)

[블록다이어그램 객체 찾고 대체하기](#) (865 페이지 참조)

[VI에서 텍스트 찾고 대체하기](#) (864 페이지 참조)

[디렉토리에서 VI 찾기](#) (863 페이지 참조)

VI에서 텍스트 찾고 대체하기

프론트패널 윈도우, 블록다이어그램 윈도우, VI 개정 히스토리, VI와 객체 설명의 텍스트를 [VI에서 검색](#)하여 기존 텍스트를 다른 텍스트로 대체할 수 있습니다. 또한, [라이브러리 함수 호출 노드]의 설정에서 텍스트를 찾고 대체할 수 있습니다.

(Statechart) Statechart의 guard 및 action 코드에서 텍스트를 찾을 수 있습니다. Statechart 편집기 윈도우에서 텍스트를 찾아 대체할 수 있습니다.

다음 단계를 따라 VI 또는 어플리케이션 인스턴스 내 VI의 텍스트를 찾고 대체합니다.

1. 편집 >> 찾기와 대체를 선택하여 +찾기 대화 상자를 디스플레이합니다. 또한, 현재 VI의 텍스트의 일부를 선택하고 편집>>찾기와 대체를 선택하면, 검색을 위해 미리 선택한 텍스트가 포함된 찾기 대화 상자를 디스플레이할 수 있습니다.
2. 텍스트 버튼을 클릭합니다.

3. 어플리케이션 인스턴스 선택 메뉴에서, 텍스트를 검색하려는 **어플리케이션 인스턴스**를 선택합니다. 기본 어플리케이션 인스턴스는 찾기 대화 상자를 연 어플리케이션 인스턴스입니다.
4. 검색할 단어 입력 텍스트 박스에서 찾으려는 텍스트를 입력합니다.
5. 더 많은 옵션 버튼을 클릭하여 텍스트 검색 옵션 대화 상자를 디스플레이하고 LabVIEW가 텍스트를 검색할 위치를 설정합니다.
6. 검색 영역 풀다운 메뉴에서 검색할 영역을 설정합니다.
7. 찾기 버튼을 클릭합니다. 검색 결과 윈도우는 LabVIEW가 찾은 텍스트의 모든 인스턴스를 나열합니다. 단 하나의 인스턴스가 발견되면 LabVIEW는 프런트패널 윈도우나 블록다이어그램에 결과를 하이라이트하고 검색 결과 윈도우는 디스플레이되지 않습니다.
8. 검색 결과 리스트에서 엔트리를 선택하고 이동하기 버튼을 클릭하여 VI의 텍스트를 찾습니다. 또한, 검색 결과 리스트에서 엔트리를 더블 클릭하여 VI에서 텍스트를 찾습니다.
9. 대체 리스트에서, 원래 텍스트를 대체하려는 텍스트를 입력하거나 이전 텍스트 입력을 선택합니다.
10. 원래 텍스트의 모든 인스턴스를 새 텍스트로 대체하려면, 모두 대체 버튼을 클릭합니다. 원래 텍스트의 선택된 인스턴스만을 새 텍스트로 대체하려면, 검색 결과 리스트에서 대체하려는 인스턴스를 선택한 후 선택 대체 버튼을 클릭합니다. 텍스트 부분을 대체한 후, 검색 결과 리스트에서 대응하는 엔트리를 선택하여 VI의 새 텍스트를 찾습니다.



노트

LabVIEW가 해당 인스턴스를 대체할 수 없을 경우 검색 결과 리스트의 엔트리 옆에 취소선 아이콘 이 나타납니다. 예를 들어, 프로퍼티 노드 또는 인보크 노드의 텍스트를 대체할 수 없습니다. 또한 실행 중인 VI의 텍스트를 대체할 수 없습니다. 모두 대체 또는 선택 대체 버튼을 클릭할 경우, LabVIEW는 취소선 아이콘으로 표시된 것들을 제외하고 모든 인스턴스를 대체합니다.

블록다이어그램 객체 찾고 대체하기

VI, 익스프레스 VI, 함수, 터미널, 타입 정의, 글로벌 변수, 컨트롤 참조, 또는 브레이크포인트를 **어플리케이션 인스턴스 안의 VI에서 검색**하고 다른 블록다이어그램 객체로 대체할 수 있습니다.

(Statechart) 또한 Statechart의 guard 및 action 코드에서 블록다이어그램 객체를 찾을 수 있습니다.

다음 단계를 따라 VI 또는 어플리케이션 인스턴스 내의 VI의 다른 블록다이어그램 객체를 찾고 대체합니다.

1. 편집 >> 찾기와 대체를 선택하여 +찾기 대화 상자를 디스플레이합니다. 또한, 현재 VI의 객체를 선택하고 편집 >> 찾기와 대체를 선택하면, 검색을 위해 미리 선택한 객체가 포함된 찾기 대화 상자를 디스플레이할 수 있습니다.
2. 객체 버튼을 클릭합니다.

3. 어플리케이션 인스턴스 선택 메뉴에서, 객체를 검색하려는 **어플리케이션 인스턴스**를 선택합니다. 기본 어플리케이션 인스턴스는 찾기 대화 상자를 연 어플리케이션 인스턴스입니다.
4. 객체 선택 버튼을 클릭합니다. 나타나는 메뉴에서 찾으려는 객체를 선택합니다.
5. 검색 영역 풀다운 메뉴에서 검색할 영역을 설정합니다.
6. 찾기 버튼을 클릭합니다. 검색 결과 윈도우는 LabVIEW가 찾은 객체의 모든 인스턴스를 나열합니다. 단 하나의 인스턴스가 발견되면 LabVIEW는 프런트패널 윈도우나 블록다이어그램에 결과를 하이라이트하고 검색 결과 윈도우는 디스플레이되지 않습니다.



노트

로컬 변수나 글로벌 변수를 검색할 경우, 검색 결과 리스트는 변수가 읽거나 쓰도록 설정되는지를 나타냅니다.

1. 검색 결과 리스트의 엔트리를 선택하고 이동하기 버튼을 클릭하여 블록다이어그램의 객체를 찾습니다. 또한, 검색 결과 리스트에서 엔트리를 더블 클릭하여 블록다이어그램의 객체를 찾습니다.
2. 대체 버튼을 클릭합니다. 나타나는 메뉴에서, 찾은 객체를 대체하려는 객체를 선택합니다.
3. 원래 객체의 모든 인스턴스를 새 객체로 대체하려면, 모두 대체 버튼을 클릭합니다. 원래 객체의 선택된 인스턴스만을 새 객체로 대체하려면, 검색 결과 리스트에서 대체하려는 인스턴스를 선택한 후 선택 대체 버튼을 클릭합니다. 새 객체가 호환될 수 없는 대체 대상인 경우 LabVIEW는 객체를 대체하지 않습니다.

LabVIEW 기타

도구 선택하기

자동 도구 선택이 활성화되어 있고 커서를 프런트패널이나 블록다이어그램 객체에서 움직이면, LabVIEW는 도구 팔레트에서 상응하는 도구를 자동적으로 선택합니다. 자동 도구 선택을 비활성화할 수 있고 도구 팔레트에서 원하는 도구를 클릭하여 수동으로 도구를 선택할 수 있습니다.

<Tab> 키를 사용하여 가장 자주 사용하는 도구 사이를 팔레트에 나타나는 순서대로 돌아가며 이동하려는 경우, 도구 팔레트에서 자동 도구 선택 버튼을 클릭하여 자동 도구 선택을 비활성화합니다. 자동 도구 선택을 다시 비활성화하기 위해서 <Shift-Tab> 키를 누르고 자동 도구 선택 버튼을 클릭합니다. 또한, 자동 도구 선택을 비활성화하려면 도구 팔레트에서 도구를 수동으로 선택할 수 있습니다. 자동 도구 선택이 비활성화되어 있다면, 다음으로 가장 많이 사용되는 도구로 전환하기 위해 스페이스바를 누를 수 있습니다.



팁

자동 도구 선택을 비활성화하여 <Tab> 키를 눌러서 가장 자주 사용하는 도구를 선택하고, <Shift-Tab> 키를 누르거나 자동 도구 선택 버튼을 클릭할 때에만 자동 도구 선택을 토글할 수 있도록 LabVIEW를 설정할 수 있습니다. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이하고, 항목 리스트에서 환경을 선택한 후, 자동 도구 선택 잠그기 확인란에서 확인 표시를 제거합니다.

자동 도구 선택이 활성화되어 있으면, 다음의 방법에서 특정한 도구를 선택할 수 있습니다:

- <Shift> 키를 눌러서 일시적으로 위치 도구로 전환합니다.
- 프런트패널이나 블록다이어그램의 열린 공간에서 커서를 움직이고 <Shift-Ctrl> 키를 눌러서 일시적으로 스크롤링 도구로 전환합니다. (macOS) <Shift-Option> 키를 누릅니다. (Linux) <Shift-Alt> 키를 누르십시오.

큰 프런트패널과 블록다이어그램 탐색하기

탐색 윈도우는 편집 모드에서 활성화된 프런트패널 또는 활성화된 블록다이어그램의 개요를 보여줍니다. 탐색 윈도우를 사용하여 큰 프런트패널 또는 블록다이어그램을 탐색합니다.



노트

LabVIEW에는 줌 기능이 없습니다. 탐색 윈도우를 사용하여 큰 프런트패널 또는 블록다이어그램을 보다 쉽게 탐색합니다.

다음 단계를 따라 탐색 윈도우 사용을 연습합니다.

1. labview\examples\Application Control\VI Server\Exporting VIs 디렉토리에 있는 Exporting VIs 프로젝트를 엽니다.  예제 열기
2. 보기>>탐색 윈도우를 선택하여 탐색 윈도우를 디스플레이하거나 <Ctrl-Shift-N> 키를 누릅니다. (macOS) <Command-Shift-N> 키를 누릅니다. (Linux) <Alt-Shift-N> 키를 누릅니다.
3. 다음 단계를 따라 탐색 윈도우를 사용하여 큰 프런트패널 또는 블록다이어그램을 보고 탐색하는 것을 연습합니다.
 1. 탐색 윈도우에서 이미지의 영역을 클릭하여 해당 영역을 프런트패널 또는 블록다이어그램에 디스플레이합니다.
 2. 탐색 윈도우에서 이미지를 클릭하고 끌어 프런트패널 또는 블록다이어그램을 스크롤할 수 있습니다. 프런트패널이나 블록다이어그램에서 보이지 않는 영역은 탐색 윈도우에서 희미하게 나타납니다.
 3. 탐색 윈도우를 크기 조정하여 디스플레이하는 이미지의 크기를 조정합니다.



노트 프런트패널이 여러 구획을 포함할 경우, 탐색 윈도우는 프런트패널의 이미지를 디스플레이합니다. 하지만, 프런트패널을 탐색하기 위해서 탐색 윈도우의 이미지를 클릭하거나 끌 수 없습니다.

컨트롤을 인디케이터로 또는 인디케이터를 컨트롤로 바꾸기

다음 단계를 따라 컨트롤을 인디케이터로, 또는 인디케이터를 컨트롤로 변경합니다.

1. 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 인디케이터로 변경 또는 컨트롤로 변경을 선택합니다.

프런트패널 인디케이터 지우기

다음 단계를 따라 최상위 VI를 실행하거나 SubVI를 호출할 때마다 자동으로 모든 프런트 패널 인디케이터를 비웁니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. VI 프로퍼티 대화 상자의 상단에 위치한 항목 풀다운 메뉴에서 실행을 선택합니다.
3. 호출 시 인디케이터 지우기 확인란에 확인 표시를 합니다.
4. 확인 버튼을 클릭합니다.

바로 가기 메뉴 플러그인

바로 가기 메뉴 플러그인 이해하기

바로 가기 메뉴 플러그인을 생성하여 프런트패널 및 블록다이어그램 객체의 바로 가기 메뉴에 사용자 항목을 추가할 수 있습니다. 바로 가기 메뉴 플러그인은 VI 스크립팅을 사용하여 사용자 정의하는 VI를 호출합니다. 다음과 같은 객체에서 마우스 오른쪽 버튼을 클릭하면 사용자 항목이 나타나도록 생성할 수 있습니다.

- 편집 시 프런트패널 객체
- 편집 시 블록다이어그램 객체
- 실행 시 블록다이어그램 객체

바로 가기 메뉴 플러그인에 대한 더 자세한 정보와 NI 커뮤니티에서 이에 대한 코드 예제를 보려면 ni.com/lvmenus를 참조하십시오.

바로 가기 메뉴 플러그인 구조

바로 가기 메뉴 플러그인은 다음과 같은 파일을 포함해야 합니다. 이때 x는 플러그인의 이름입니다.

- `x.ct1`—플러그인이 작용할 객체를 지정하는 타입 정의.
- `x.vi`—바로 가기 메뉴를 만드는 빌드 VI.
- `Execute x.vi`—바로 가기 메뉴에서 사용자 항목을 선택하면 실행되는 플러그인 실행 VI.

파일 이름이 플러그인 이름과 같은 LLB(x.llb)에 이 파일을 저장합니다. LabVIEW는 LLB의 다음 중 한 디렉토리에서 플러그인을 로드 및 검색합니다. 플러그인을 검색할 때 LabVIEW는 서브디렉토리의 내용을 무시합니다.

- <LabVIEW Data>\PopupMenu\edit time panel and diagram—모든 버전의 LabVIEW에서 사용되는 편집 시 프론트패널 및 블록다이어그램 객체에 작용하는 플러그인의 LLB를 포함합니다.
- <LabVIEW Data>\PopupMenu\run time diagram—모든 버전의 LabVIEW에서 사용되는 실행 시 블록다이어그램 객체에 작용하는 플러그인의 LLB를 포함합니다.
- labview\resource\plugins\PopupMenu\edit time panel and diagram—현재 LabVIEW 버전에서만 사용되는 편집 시 프론트패널 및 블록다이어그램 객체에 작용하는 플러그인의 LLB를 포함합니다.
- labview\resource\plugins\PopupMenu\run time diagram—현재 LabVIEW 버전에서만 사용되는 실행 시 블록다이어그램 객체에 작용하는 플러그인의 LLB를 포함합니다.

두 개 LLB의 이름이 같은 경우, labview\resource\plugins\PopupMenu 디렉토리의 LLB가 우선합니다.

바로 가기 메뉴 플러그인 생성하기

바로 가기 메뉴 플러그인을 생성하려면 다음 세 단계를 거쳐야 합니다.

- **파트 1:** 플러그인이 영향을 줄 객체 지정하기.
- **파트 2:** 사용자 아이템이 바로 가기 메뉴에 나타나는 방식 결정하기.
- **파트 3:** 사용자 아이템 선택 시 플러그인이 실행되는 방식 편집하기.

다음 유틸리티 VI를 실행하여 바로 가기 메뉴 플러그인을 생성합니다: labview\resource\plugins\PopupMenu\Create Shortcut Menu Plug-In From Template.vi

이 유틸리티 VI는 플러그인에 필요한 파일을 생성합니다. 이 파일을 사용자 정의하여 바로 가기 메뉴를 만들고 플러그인을 실행하게 됩니다.

➡ 파트 1: 바로 가기 메뉴 플러그인이 영향을 줄 객체를 지정하기

관련 정보

메뉴와 도구 모음

파트 1: 바로 가기 메뉴 플러그인이 영향을 줄 객체를 지정하기

바로 가기 메뉴 플러그인을 어떤 프론트패널 또는 블록다이어그램 객체에 적용할 것인지 지정해야 합니다. 다음 단계에 따라 바로 가기 메뉴 플러그인이 작용할 객체를 지정합니다.



노트

다음 유틸리티 VI를 실행하여 바로 가기 메뉴 생성에 필요한 파일을 생성합니다: `labview\resource\plugins\PopupMenus\Create Shortcut Menu Plug-In From Template.vi`

1. 유틸리티 VI가 생성하는 타입 정의 `x.ct1`을 엽니다(여기서 `x`는 플러그인의 이름).
2. 영향받는 아이템을 편집합니다. 여기에는 바로 가기 메뉴 플러그인이 영향을 줄 객체 또는 참조 번호 타입이 포함되어 있습니다. 우선, 영향받는 아이템 컨트롤은 G객체 참조 번호 타입의 배열을 포함하며, 이는 플러그인이 G객체 참조 번호의 바로 가기 메뉴에 나타남을 의미합니다. 다음과 같은 방법으로 영향받는 아이템 컨트롤을 편집할 수 있습니다.
 - 참조 번호 타입의 배열을 추가하여 객체를 추가합니다.
 - 배열을 삭제하여 객체를 제거합니다.
 - 배열에서 마우스 오른쪽 버튼을 클릭한 후 VI 서버 클래스 선택을 선택하고 일반 계층구조 아래에서 참조 번호 타입을 선택하여 다른 객체를 선택합니다.

노트 객체의 **참조 번호 타입을 확인**하려는 경우, LabVIEW가 제공하는 모든 바로 가기 메뉴 태그 및 타입 검색 플러그인을 사용하면 됩니다.

3. 파일 >> 변경된 사항 적용을 선택하여 변경사항을 적용하고 이 변경사항을 같은 타입 정의를 사용하는 플러그인 VI에 전달합니다.
4. 파일 >> 저장을 선택하여 타입 정의에 변경사항을 저장합니다.

◀ 바로 가기 메뉴 플러그인 이해하기

➡ 파트 2: 플러그인을 사용하여 바로 가기 메뉴 만들기

파트 2: 플러그인을 사용하여 바로 가기 메뉴 만들기

프론트패널 또는 블록다이어그램 객체의 바로 가기 메뉴에서 사용자 아이템이 언제 그리고 어떻게 나타날 것인지 사용자가 지정할 수 있습니다. 또한 다음과 같은 방법으로 바로 가기 메뉴를 만드는 플러그인을 생성할 수 있습니다.

- 조건적으로 사용자 아이템을 객체 타입에 추가합니다.
- 플러그인의 트랜잭션 동작을 토글합니다.
- 여러 사용자 아이템을 추가합니다.
- 아이템을 제거합니다.

다음 그림은 템플릿으로부터 바로 가기 메뉴 플러그인 생성 VI가 생성하는 빌드 VI(x.vi, 여기서 x는 플러그인의 이름)의 블록다이어그램을 보여줍니다. 블록다이어그램 코드를 수정하여 바로 가기 메뉴를 만듭니다. 블록다이어그램 수정 방법에 대한 팁은 블록다이어그램 주석을 참조하십시오.

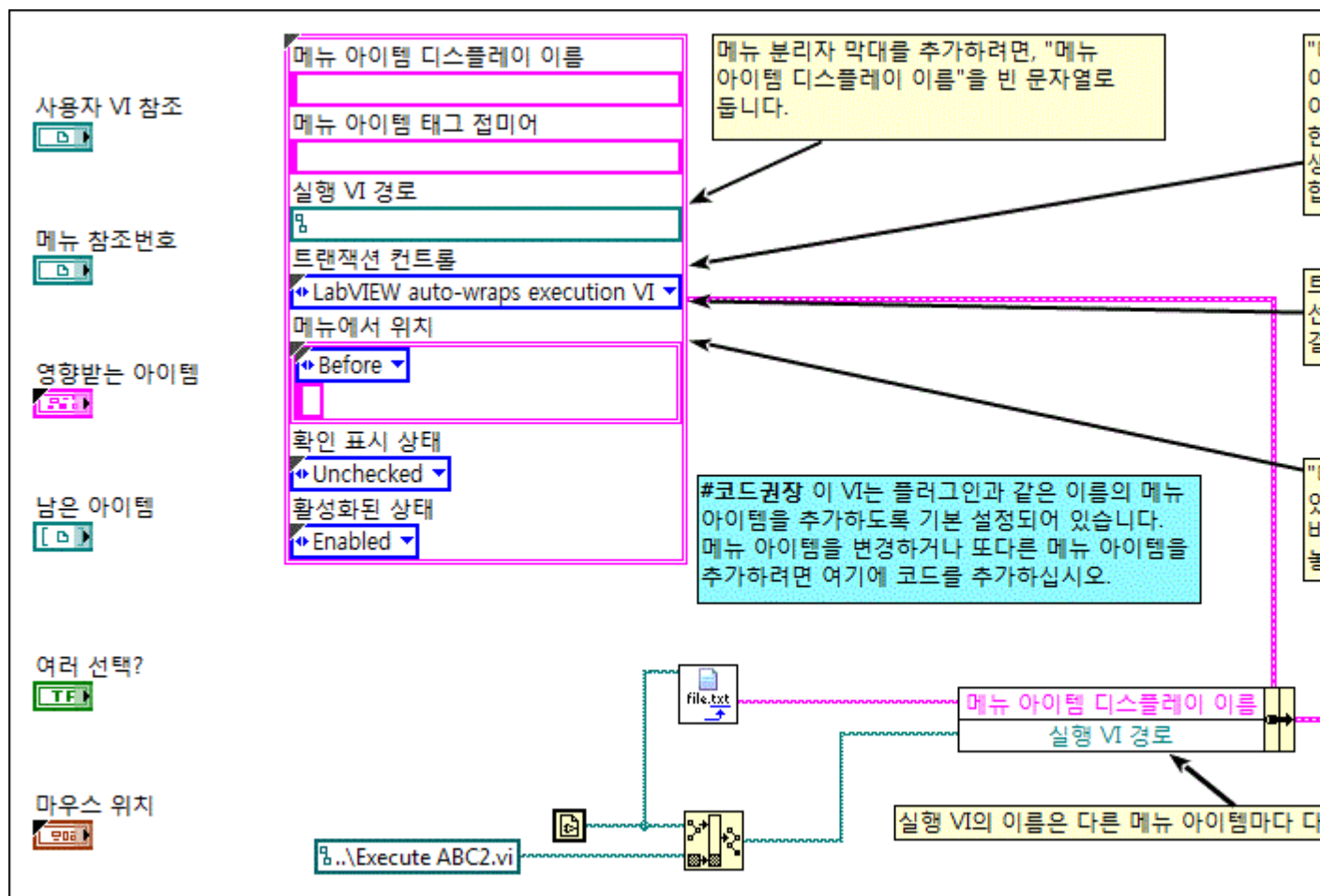


NOTE

다음 유틸리티 VI를 실행하여 바로 가기 메뉴 생성에 필요한 파일을 생성합니다:
labview\resource\plugins\PopupMenu\Create Shortcut Menu Plug-In From Template.vi



주의 커넥터 팬 터미널에 할당된 컨트롤은 수정하지 마십시오. 플러그인의 로드 및 실행을 위해, LabVIEW는 이와 같이 미리 정의된 컨트롤 이름, 타입, 터미널 위치가 필요합니다.



이 VI의 기본 설정은 프로퍼티 아이템 전에 사용자 아이템을 추가하여 바로 가기 메뉴를 만드는 것입니다. 사용자 아이템의 디스플레이 이름은 파일 확장자를 제외한 이 VI의 이름입니다.

← 파트 1: 바로 가기 메뉴 플러그인이 영향을 줄 객체를 지정하기

➡ 파트 3: 바로 가기 메뉴 플러그인 실행하기

파트 3: 바로 가기 메뉴 플러그인 실행하기

VI 스크립팅을 사용하여, 사용자가 바로 가기 메뉴에서 사용자 정의 아이템을 선택할 때 LabVIEW가 수행하는 동작을 정의합니다.



노트

다음 유틸리티 VI를 실행하여 바로 가기 메뉴 생성에 필요한 파일을 생성합니다: `labview\resource\plugins\PopupMenus\Create Shortcut Menu Plug-In From Template.vi`

1. 유틸리티 VI가 생성하는 플러그인 실행 VI `Execute x.vi`(여기서 x는 플러그인의 이름)를 엽니다.
2. 블록 다이어그램에서, VI 스크립팅을 사용하여 플러그인이 작용할 객체의 동작을 수정합니다. VI 서버 참조를 사용하여 객체의 프로퍼티 및 메소드에 접근하고 이를 수정하려면 [프로퍼티 노드] 또는 [인보크 노드]를 추가합니다.
3. (옵션) 객체의 동작을 변경하려면, 바로 가기 메뉴 플러그인의 LLB에 필요한 만큼 VI를 더 추가합니다.

노트 이름 충돌을 피하기 위해 National Instruments는 추가된 VI 및 SubVI의 파일 이름에 플러그인 이름으로 접두어를 붙이거나 프로젝트 라이브러리를 사용하도록 권장합니다. 예를 들어, SubVI의 이름을 `x_subVI.vi`로 붙이거나 추가된 VI 및 SubVI를 프로젝트 라이브러리 `x.lvlib`에 저장합니다. 이때 x는 플러그인의 이름입니다.

4. 파일 >> 저장을 선택하여 변경사항을 저장합니다. 변경사항이 적용될 수 있도록 LabVIEW를 다시 시작합니다.

팁 LabVIEW를 다시 시작하지 않고 플러그인을 다시 로드하려면 빈 VI를 생성한 후, 블록 다이어그램에 [인보크 노드]를 추가하고 [인보크 노드]에서 메뉴: 새로 고침 메소드를 설정하고 이 VI를 실행합니다. 이 메소드는 사용자 플러그인의 설치 프로그램을 작성할 때에도 유용합니다. 설치 과정의 일부로 메소드를 실행할 수 있습니다. JKI VI Package Manager (VIPM) 소프트웨어를 사용하여 사용자 플러그인을 배포하는 경우, 메뉴: 새로 고침 메소드를 설치 후 VI에 추가합니다.

LabVIEW가 바로 가기 메뉴 플러그인을 실행하는 동안 트랜잭션 보호하기

플러그인 실행 VI가 실행될 때 VI 스크립팅 작업의 보호를 위해, LabVIEW는 [사용 중 설정] VI를 호출하여 마우스를 비활성화합니다. 플러그인 실행 VI가 실행을 마친 후, LabVIEW는 [사용 중 설정 해제] VI를 호출하여 마우스를 활성화합니다.

빌드 VI `x.vi`(여기서 x는 플러그인의 이름)의 트랜잭션 컨트롤?을 사용하여 LabVIEW가 취소 가능한 트랜잭션에서 스크립팅 작업을 자동으로 래핑하도록 지시할 수 있습니다. 또는 LabVIEW가 빌드 VI에서 트랜잭션을 컨트롤하지 못하게 지시하여 사용자가 플러그인 실행 VI에서 자신의 트랜잭션을 관리하도록 할 수 있습니다. LabVIEW가 자동으로 트랜잭션을 생성하는 경우, 플러그인 실행 VI의 트랜잭션 실패? 출력을 설정하여 이 VI가 성공적으로 실행하지 못하는 경우 거짓을 반환하도록 합니다.

◀ 파트 2: 플러그인을 사용하여 바로 가기 메뉴 만들기

▶ 바로 가기 메뉴 플러그인 이해하기

객체의 참조 번호 타입과 객체의 메뉴 아이템 태그 식별하기

바로 가기 메뉴 플러그인을 어떤 프런트패널 또는 블록 다이어그램 객체에 적용할 것인지 지정해야 합니다. 플러그인의 타입 정의에서, 객체의 참조 번호 타입을 지정합니다. 객체에서 마우스 오른쪽 버튼을 한 번 클릭하면 한 개 이상의 참조 번호 타입에 대해 참조를 생성할 수 있습니다. 모든 바로 가기 메뉴 태그 및 타입 검색 플러그인을 사용하면 영향 받는 참조 번호 타입을 식별하는데 도움이 됩니다. 객체에서 마우스 오른쪽 버튼을 클릭하고 모든 바로 가기 메뉴 태그 및 타입 검색 플러그인을 선택하면, LabVIEW가 마우스 오른쪽 버튼 클릭으로 영향을 받는 참조 번호 타입을 리스트로 보여줍니다. 참조 번호 타입을 사용하여 플러그인이 적용할 객체를 지정합니다.

또한 모든 바로 가기 메뉴 태그 및 타입 검색 플러그인은 모든 바로 가기 메뉴의 메뉴 아이템 태그를 열거해서 보여줍니다. 메뉴 아이템 태그를 사용하여, 바로 가기 메뉴에서 플러그인 아이템을 디스플레이할 위치를 사용자 정의합니다.

모든 바로 가기 메뉴 태그 및 타입 검색 플러그인을 사용하려면 다음 LLB를 복사합니다.

```
labview\resource\plugins\PopupMenu\support\Lookup All  
Shortcut Menu Tags And Types.llb
```

이 LLB를 다음 디렉토리에 붙여넣습니다:

```
labview\resource\plugins\PopupMenu\edit time panel and  
diagram
```

바로 가기 메뉴에서 모든 바로 가기 메뉴 태그 및 타입 검색 플러그인을 제거하려면 Lookup All Shortcut Menu Tags And Types.llb를 삭제하고 LabVIEW를 다시 시작합니다.

관련 정보

바로 가기 메뉴 플러그인 이해하기

바로 가기 메뉴 플러그인이 영향을 줄 객체 지정하기

플러그인을 사용하여 바로 가기 메뉴 만들기

바로 가기 메뉴 플러그인 문제 해결하기

다음 가이드라인은 LabVIEW 바로 가기 메뉴 플러그인의 유효성을 확인하는데 도움이 됩니다. 잘못 생성된 플러그인의 경우, 객체에서 마우스 오른쪽 버튼을 클릭할 때 플러그인이 나타나지 않거나 플러그인을 선택할 경우 플러그인이 실행되지 않을 수 있습니다. 플러그인이 유효하지 않은 경우 LabVIEW는 에러나 경고를 반환하지 않습니다.

- VI가 깨져 있지 않음—LabVIEW에서 VI를 로드하고 VI에 에러 보고가 없는지 확인합니다.
- 커넥터 팬이 필수 패턴 준수 — [템플릿으로부터 바로 가기 메뉴 플러그인 생성] VI는 특정 커넥터 팬 패턴 및 터미널 이름 구문을 준수하는 빌드 Vix.vi와 플러그인

실행 VI `Execute x.vi`를 생성합니다. 이 패턴 또는 구문을 변경하는 경우 LabVIEW는 이 VI를 플러그인으로 사용할 수 없습니다. 오직 타입 정의의 `x.cti`만 변경할 수 있습니다(여기서 `x`는 플러그인의 이름).

- 플러그인 LLB 이름과 빌드 VI 이름 일치 — LabVIEW는 플러그인 파일과 타입 정의를 플러그인 이름이 같은 LLB 파일 즉, `x.llb`에 저장합니다. 빌드 VI는 파일 확장자만 제외하고 LLB와 이름이 같아야 합니다. 이 명명 규칙을 통해 LabVIEW는 메모리에서 여러 플러그인을 동시에 관리할 수 있습니다.
- 디버깅 도구 비활성화 — 빌드 VI에 대해 디버깅 도구를 비활성화해야 합니다. LabVIEW는 바로 가기 메뉴를 만들 때 브레이크포인트 또는 프로브에서 정지할 수 없습니다.
- 메뉴 아이템 태그의 길이가 255자를 초과하지 않음 — 메뉴 아이템 태그의 최대 길이는 255자입니다. LabVIEW는 빌드 VI로부터 반환되는 플러그인 이름과 메뉴 아이템 태그 접미어를 사용하여 태그 이름을 만듭니다. 전체 태그 이름의 구문은 다음과 같습니다. `APP_SC_PLUGIN_<x.vi>/<menu_item_tag_suffix>`(여기서 `x`는 플러그인의 이름)
- 참조 번호 타입이 플러그인이 작용하는 객체와 일치 — 타입 정의의 참조 번호 타입이 플러그인이 영향을 주는 객체 타입과 일치하는지 확인합니다. 객체의 **참조 번호 타입을 확인**하려는 경우, LabVIEW가 제공하는 모든 바로 가기 메뉴 태그 및 타입 검색 플러그인을 사용하면 됩니다.

관련 정보

바로 가기 메뉴 플러그인 생성하기

객체의 참조 번호 타입과 객체의 메뉴 아이템 태그 식별하기

바로 가기 메뉴 플러그인 생성 요령

바로 가기 메뉴 플러그인 생성 요령

다음 리스트는 바로 가기 메뉴를 플러그인을 생성하고 사용하는 방법에 대한 추가적인 정보를 제공합니다.

- 여러 프런트패널 또는 블록 다이어그램 객체를 선택한 후 마우스 오른쪽 버튼을 클릭하면, 플러그인이 선택한 모든 객체의 참조 번호를 가져옵니다. 참조 번호가 플러그인의 영향받는 아이템 배열과 일치하면 이 영향받는 아이템 배열로 전달됩니다. 다른 모든 객체의 참조 번호는 플러그인의 남은 아이템 배열에 전달됩니다. 이로 인해, 여러 선택한 객체가 영향받는 아이템과 일치하거나 일부 선택한 객체만 영향받는 아이템과 일치하는 경우 플러그인을 활성화할 수 있습니다.
- 여러 객체를 선택하지 않고 마우스 오른쪽 버튼을 클릭하더라도, 여러 참조 번호를 얻을 수 있습니다. 예를 들어, 터미널에서 마우스 오른쪽 버튼을 클릭하면 터미널, 터미널의 노드, 노드의 다이어그램에 대한 참조 번호가 생성됩니다. 여러 개를 선택하는 경우와 같이, 사용자는 각 플러그인이 이런 경우 메뉴 아이템을 어떻게 적용할지 결정할 수 있습니다.
- 여러 플러그인이 비슷한 기능을 수행하기도 합니다. LabVIEW가 제공하는 플러그인을 탐색하여 사용자 플러그인에 유용한 SubVI를 찾을 수도 있습니다. 이러한 플

러그인은 labview\resource\plugins\PopupMenu 디렉토리에 있습니다.

- 여러 플러그인 사이에서 SubVI를 공유하려면, LLB와 같은 폴더에 있는 서브디렉토리에 SubVI를 저장하면 됩니다.
- 바로 가기 메뉴에 메뉴 분리자 막대를 추가하려면, 빌드 VI에서 메뉴 아이템 디스플레이 이름을 빈 문자열로 둡니다.
- LabVIEW를 다시 시작하지 않고 플러그인을 다시 로드하려면, 빈 VI를 생성한 후, 블록다이어그램에 [인보크 노드]를 추가하고, [인보크 노드]에서 메뉴:새로 고침 메소드를 설정하고, 이 VI를 실행합니다. 이 메소드는 사용자 플러그인의 설치 프로그램을 작성할 때에도 유용합니다. 설치 과정의 일부로 메소드를 실행할 수 있습니다. JKI VI Package Manager (VIPM) 소프트웨어를 사용하여 사용자 플러그인을 배포하는 경우, 메뉴:새로 고침 메소드를 설치 후 VI에 추가합니다.

관련 정보

바로 가기 메뉴 플러그인 이해하기

객체의 참조 번호 타입과 객체의 메뉴 아이템 태그 식별하기

바로 가기 메뉴 플러그인 문제 해결하기

VI Package Manager (VIPM) 소프트웨어를 사용하여 LabVIEW 애드온에 액세스하기

메뉴:새로 고침 메소드

프런트패널 또는 블록다이어그램에서 공간 추가 또는 축소하기

다음 단계를 따라 윈도우 크기를 조정하지 않고 프런트패널이나 블록다이어그램에 공간을 추가합니다.

1. <Ctrl> 키를 누른 상태에서 위치 도구를 사용하여 프런트패널 또는 블록다이어그램을 클릭합니다. (macOS) <Option>을 누릅니다. 위치 도구 커서가 화살표에서 더하기 부호로 바뀝니다.
2. 키를 누른 상태에서 공간을 늘리려는 방향으로 마우스를 끌어 이동합니다. 마우스를 끄는 동안 그와 동시에 객체가 이동합니다. 마우스를 끄는 방향이 주로 수직 또는 수평이면, 자동으로 주가 되는 방향으로 맞추어집니다.
3. 마우스 버튼을 놓아 객체의 위치를 결정합니다.

흩어져 있는 객체 사이의 공간을 줄이려면, <Ctrl-Alt>를 누른 상태에서 공간을 축소하려는 방향으로 마우스를 끌어 이동합니다. (macOS) <Option-Ctrl>을 누릅니다.

한 프런트패널 객체에서 읽고 두 블록다이어그램 위치에 쓰기

로컬 변수를 사용해서 블록다이어그램에 있는 여러 객체를 컨트롤 할 수 있습니다. 두 개의 While 루프를 가진 VI의 경우, 버튼과 같은 단일 프런트패널 컨트롤을 사용하여, 두 개의 병렬 While 루프를 컨트롤할 수 있습니다. 이 루프는 동시에 실행됩니다.

프런트패널 컨트롤 터미널을 첫번째 While 루프의 블록다이어그램에 놓습니다. 그 후 로컬 변수를 두번째 While 루프 안에 놓습니다. 이 과정은 블록다이어그램의 두 위치에서 한 개의 프런트 패널 객체에 쓸 때 사용하는 과정과 비슷합니다.



관련 정보

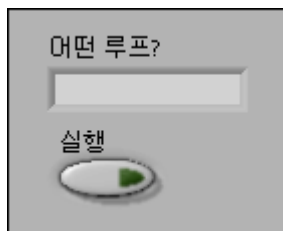
[로컬 변수](#) (560 페이지 참조)

[While 루프: 조건이 발생할 때까지 작업 반복하기](#) (368 페이지 참조)

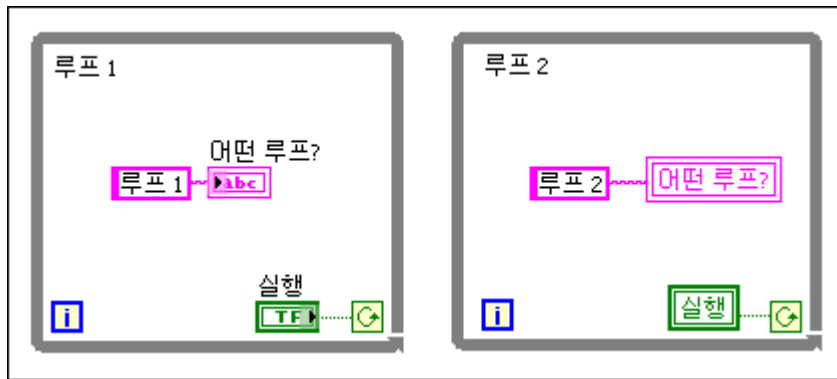
[두 블록다이어그램 위치에서 한 프런트패널 객체에 쓰기](#) (876 페이지 참조)

두 블록다이어그램 위치에서 한 프런트패널 객체에 쓰기

로컬 변수를 사용하여 블록다이어그램의 하나 이상의 위치로부터 하나의 프런트패널 인디케이터를 업데이트할 수 있습니다. 예를 들어, 두 While 루프를 가진 VI를 가진 경우, 프런트패널 인디케이터를 업데이트하여 현재 어떤 루프가 실행 중인지 디스플레이할 수 있습니다. 다음 그림은 그런 VI의 프런트패널을 보여줍니다.



블록다이어그램에서, 다음 그림과 같이 루프 1에 인디케이터 터미널을, 루프 2에 인디케이터의 로컬 변수 인스턴스를 놓습니다. 이 예제는 또한 실행 컨트롤의 로컬 변수 인스턴스를 사용하여 두 While 루프의 실행을 컨트롤합니다. 또한, 로컬 변수를 사용하여 한 VI에서 하나의 프런트패널 버튼이 두 병렬 While 루프를 컨트롤하도록 할 수 있습니다. 와이어는 루프를 연결하지 않으며 동시에 실행됩니다.



노트

로컬 변수를 **기계적 동작**이 누를때 또는 놓을때 래치로 설정된 불리언 컨트롤 또는 인디케이터와 연관시킬 수 없습니다.

VI

VI와 그림, 텍스트 끌어서 놓기

함수 팔레트에서 VI 선택 아이콘이나 텍스트를 클릭한 후, VI를 탐색한 다음, VI를 더블 클릭하거나 블록다이어그램으로 끌어 놓아 이 VI를 호출하는 SubVI를 생성합니다. **사용자 컨트롤**, 타입 정의, **글로벌 변수**에도 같은 방법을 사용합니다. 또한, 텍스트와 그림을 프런트패널과 블록다이어그램에 복사하기 위하여 Windows 탐색기 등의 다른 어플리케이션에서 끌어다 놓을 수 있습니다.



노트

LabVIEW가 대상을 하이라이트할 때까지 객체를 놓을 수 없습니다. 마우스 버튼을 놓기 전에 <Esc> 키를 눌러 끌기와 놓기 작업을 취소할 수 있습니다.

끌어서 놓기 기능을 사용하여 LabVIEW에서 다음과 같은 태스크를 수행할 수 있습니다:

- 프런트패널 컨트롤과 인디케이터를 블록다이어그램에 끌고 와서 **블록다이어그램 상수를 생성**할 수 있으며 그 반대의 경우로도 할 수 있습니다.
- 컨트롤 또는 상수에 subVI의 모든 경로 이름을 놓기 위하여 subVI를 경로 컨트롤 또는 상수로 끌어옵니다.
- VI를 subVI로 놓기 위해 프런트패널이나 블록다이어그램 윈도우의 오른쪽 상단 코너에서 블록다이어그램으로 VI 아이콘을 끌고 옵니다. SubVI에서 상수로서 컨트롤의 기본이 아닌 값을 자동적으로 연결하기 위해 끄는 동안 <Shift> 키를 누릅니다. 블록다이어그램에 이미 subVI가 있다면, 연결된 상수를 업데이트하기 위해 기존의 subVI를 끄는 동안 <Shift> 키를 누릅니다. 만약 컨트롤이 기본값을 가지고 있다면, LabVIEW는 subVI에 상수 연결을 포기합니다. 어떠한 입력도 상수가 연결되지 않는 것에 대해 영향을 받지 않습니다.
- 글로벌 변수와 사용자 컨트롤을 끌어다 놓습니다.
- **VI 서버** 기능의 부분으로 VI를 메모리에 다이나믹하게 로드하기 위해 프런트패널 컨트롤에서 VI 참조 번호에 VI 아이콘을 끌어다 놓습니다.

- 윈도우를 스크롤하기 위해 프런트패널 또는 블록다이어그램의 경계 가까이에 객체를 끕니다. 윈도우가 스크롤하는 동안 속도를 증가시키려면 <Shift> 키를 누릅니다. 윈도우가 스크롤되는 동안 윈도우의 경계 밖에서 마우스 버튼을 놓을 경우, LabVIEW는 마우스 버튼을 놓은 위치까지 계속 윈도우를 스크롤한 뒤 객체를 해당 위치에 놓습니다.

끌어서 놓기 기능을 사용하여 LabVIEW와 다른 어플리케이션 사이에서 다음과 같은 태스크를 수행할 수 있습니다:

- 컨트롤 또는 상수에서 파일의 모든 경로 이름을 놓기 위해 경로 컨트롤 또는 상수로 파일을 끌어옵니다.
- 데이터로그 파일에 데이터 구조를 포함한 클러스터를 생성하기 위해 데이터로그 파일을 데이터로그 파일 참조 번호로 끕니다.
- VI 아이콘, 그림 링 컨트롤, 프런트패널, 또는 블록다이어그램에 **그래픽 파일 끌기**를 해서 컨트롤 안, 프런트패널이나 블록다이어그램, 또는 VI 아이콘에 그림을 놓을 수 있습니다. LabVIEW는 컴퓨터가 지원하는 대부분의 그래픽 파일을 반입할 수 있습니다.
- 프런트패널에 사용자 컨트롤 파일을 끌어와서 프런트패널에 사용자 컨트롤을 놓을 수 있습니다.
- (Windows) 어플리케이션이 ActiveX 기술을 지원한다면, 그 어플리케이션에서 프런트패널 또는 블록다이어그램으로 그림과 텍스트를 끌어올 수 있습니다. 예를 들면, Microsoft Word 문서에서 텍스트의 부분 또는 비트맵을 끌어 프런트패널 또는 블록다이어그램에 추가할 수 있습니다.
- (macOS) 모든 어플리케이션에서 잘라낸 텍스트 또는 선택된 텍스트를 문자열 컨트롤 또는 상수, 프런트패널, 블록다이어그램, 또는 라벨에 끌어다 놓을 수 있습니다.
- (macOS) 다른 어플리케이션에서 잘라낸 이미지 또는 선택된 이미지를 그림 링 컨트롤, 프런트패널, 또는 블록다이어그램에 끌어다 놓을 수 있습니다.



노트(macOS) 다른 어플리케이션에서 텍스트와 그림을 끌어다 놓으려면 시스템에 반드시 Drag Manager가 있어야 합니다. System 7.5와 그 이후 버전에는 Drag Manager가 포함되어 있습니다. System 7.0은 Apple Computer에서 System 7.5 확장을 받아야 가능합니다.

끌어서 놓기 기능에 관한 추가적인 정보는 OS 문서를 참고하십시오.

현재 VI의 스타일 사용자 정의하기

현재 VI의 컨트롤과 인디케이터 스타일 설정하기

VI 프로퍼티 대화 상자의 편집기 옵션 페이지와 옵션 대화 상자의 프런트패널 페이지는 비슷한 옵션을 가지고 있습니다. VI 프로퍼티 대화 상자를 사용하여 현재 VI에서 컨트롤과 인디케이터의 스타일을 설정합니다. 옵션 대화 상자의 프런트패널 페이지를 사용하여 모든 새 VI의 컨트롤과 인디케이터의 스타일을 설정합니다.

다음 단계를 따라 현재 VI의 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>컨트롤이나 생성>>인디케이터를 선택하여 LabVIEW가 생성하는 컨트롤이나 인디케이터의 스타일을 변경합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 편집기 옵션을 선택합니다.
3. 컨트롤/인디케이터 생성 시 컨트롤 스타일 옵션으로부터 스타일을 선택합니다.
4. 확인 버튼을 클릭합니다.



노트

SubVI 터미널에서 마우스의 오른쪽 버튼을 클릭하고 생성>>컨트롤이나 생성>>인디케이터를 바로 가기 메뉴에서 선택하면, LabVIEW는 SubVI 터미널에 상응하는 SubVI 프론트패널 객체의 스타일로 컨트롤이나 인디케이터를 생성합니다. 컨트롤이나 인디케이터에서 오른쪽 마우스 버튼을 클릭하고 바로 가기 메뉴에서 생성>>컨트롤이나 생성>>인디케이터를 선택하면, LabVIEW는 클릭한 컨트롤이나 인디케이터의 스타일로 컨트롤이나 인디케이터를 생성합니다.

현재 VI의 정렬 눈금 크기 설정하기

VI 프로퍼티 대화 상자를 사용하여 현재 VI에서 정렬 눈금의 크기를 설정합니다. 옵션 대화 상자의 프론트패널 페이지와 블록 다이어그램 페이지에 있는 눈금 옵션을 사용하여, 모든 새 VI의 눈금 크기를 설정합니다.

다음 단계를 따라 현재 VI의 눈금의 크기를 설정합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 편집기 옵션을 선택합니다.
3. 프론트패널과 블록 다이어그램 각각의 필드에서 프론트패널과 블록 다이어그램의 눈금을 픽셀 크기로 입력합니다.

현재 VI의 인쇄 옵션 설정하기

인쇄 대화 상자, VI 프로퍼티 대화 상자의 인쇄 옵션 페이지, 옵션 대화 상자의 인쇄 페이지에 비슷한 옵션이 있습니다. 인쇄 대화 상자과 VI 프로퍼티 대화 상자를 사용하여 현재 VI의 인쇄 옵션을 설정합니다. 옵션 대화 상자의 인쇄 페이지를 사용하여 모든 VI에 대한 인쇄 옵션을 설정합니다.

다음 단계를 따라 현재 VI에 인쇄 옵션을 설정합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 인쇄 옵션을 선택합니다.
3. 적절한 인쇄 옵션을 선택합니다.
4. 확인 버튼을 클릭합니다.

인쇄 대화 상자를 사용하여 VI 문서를 인쇄할 때, LabVIEW는 인쇄 대화 상자의 옵션을 사용하여 현재 VI의 인쇄 옵션을 설정합니다. 예를 들면, 인쇄 대화 상자의 프린터 페이지에서 패널 경계 표시 확인란에서 확인 표시를 놓고 VI 문서를 인쇄하면, 확인 표시가 VI 프로퍼티 대화 상자의 인쇄 옵션 페이지에 있는 프린트 패널 경계 표시 확인란에 나타납니다.

OS가 인쇄 대화 상자를 가지고 있으면, 그 대화 상자에서 설정한 어떤 옵션이 현재 VI의 인쇄 옵션으로 설정됩니다. 예를 들면, OS 인쇄 대화 상자에서 페이지 방향(세로 또는 가로)을 변경하면, LabVIEW도 VI를 위한 페이지 방향을 설정합니다.

현재 VI의 개정 히스토리 옵션 설정하기

VI 프로퍼티 대화 상자의 개정 히스토리 페이지와 옵션 대화 상자의 개정 히스토리 페이지는 비슷한 옵션을 가지고 있습니다. VI 프로퍼티 대화 상자를 사용하여 현재 VI의 옵션을 설정합니다. 옵션 대화 상자의 개정 히스토리 페이지를 사용하여 모든 새 VI의 옵션을 설정합니다.

다음 단계를 따라 현재 VI의 히스토리 윈도우에 나타나는 정보를 변경합니다.

1. 파일»VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 개정 히스토리를 선택합니다.
3. 옵션 대화 상자로부터 기본 히스토리 셋팅 사용 확인란에서 확인 표시를 제거하여 옵션 대화 상자의 개정 히스토리 페이지에 있는 옵션을 바꿉니다.
4. 적절한 히스토리 옵션을 선택합니다.

또한, 히스토리 프로퍼티를 사용하여 프로그램적으로 VI 히스토리를 사용자 정의할 수 있습니다.

재귀 VI 생성하기

VI가 **재귀적**인 VI가 되도록 설정할 수 있는데, 이는 VI가 자체 블록 다이어그램 또는 SubVI의 블록 다이어그램에서 스스로를 호출할 수 있는 VI라는 뜻입니다. 같은 프로세스의 출력 작업을 여러 번 실행하려는 경우, 재귀 VI가 유용합니다.

다음 단계를 따라 VI를 재귀 VI로 만듭니다.

1. 재귀 VI로 만들려는 VI를 엽니다.
2. VI 프로퍼티 대화 상자의 실행 페이지에서 공유 복제 재호출 실행을 선택합니다.
3. VI 자신의 블록 다이어그램 또는 SubVI의 블록 다이어그램에서 재귀 VI를 호출합니다.



노트

LabVIEW는 32비트 플랫폼에서 최대 15,000개의 재귀 호출을 허용하며, 64비트 플랫폼에서는 최대 35,000개의 재귀 호출을 허용합니다. 지나치게 많은 재귀 호출을 방지하려면, 케이스 구조에서 재귀 VI를 호출하는 방법을 고려하십시오.

VI 계층구조에서 검색하기

다음 단계를 따라 VI 계층구조 윈도우에서 이름으로 계층구조 노드를 찾습니다.

1. VI 또는 프로젝트를 열고 보기»VI 계층구조를 선택하여 VI 계층구조 윈도우를 디스플레이합니다.
2. 윈도우의 모든 곳에서 찾으려는 아이템의 이름을 타입하여 검색을 시작합니다. 텍스트를 타입하면, 타입한대로 텍스트를 디스플레이하여 검색한 문자열이 나타납니다. LabVIEW는 한 번에 하나씩 검색 문자열로 시작하는 이름을 가진 아이템을 하이라이트합니다.
3. 검색 문자열로 시작하는 이름을 가진 하나 이상의 아이템이 있으면, <Enter>키를 눌러서 검색 문자열과 일치하는 다음 아이템을 검색합니다. <Shift-Enter> 키를 눌러서 검색 문자열과 일치하는 이전 아이템을 찾습니다.

또한 **편집»찾기**를 선택하여 찾기 대화 상자를 디스플레이하고, 이를 통해 VI 계층구조 윈도우에서 아이템을 찾을 수 있습니다. VI 계층구조 윈도우만을 검색하려면, 검색 영역 풀다운 메뉴에서 VI 계층구조를 선택합니다.



노트

LabVIEW 클래스 계층구조 윈도우에서도 이와 같이 검색할 수 있습니다. 보기»LabVIEW 클래스 계층구조를 선택하여 이 윈도우를 디스플레이합니다.

VI의 계층구조 보기

VI 계층구조 윈도우는 메모리 상에 있는 모든 VI에 대한 호출 계층구조뿐만 아니라, 열려 있는 LabVIEW 프로젝트 및 타겟을 모두 그래픽으로 디스플레이합니다. 이 윈도우를 사용하여 메모리 상에서 VI를 구성하는 다른 노드와 SubVI를 보고, VI 계층구조 검색을 합니다.

VI 계층구조 윈도우에 디스플레이되는 아이템은 다음과 같습니다:

- 프로젝트 및 타겟
- 정적 및 다이나믹 SubVI 호출
- 타입 정의
- 글로벌 변수
- 공유 변수
- LabVIEW 클래스
- X컨트롤
- Statechart
- 프로젝트 라이브러리 .lvlib
- 익스프레스 VI
- 정적 VI 참조

VI 계층구조 윈도우는 최상위 아이콘을 사용하여 주요 LabVIEW **어플리케이션 인스턴스**를 나타내며, 그 아래에 프로젝트에 속하지 않거나 프로젝트의 어플리케이션 인스턴스에 속하지 않는 모든 객체가 나타납니다. 프로젝트를 추가하면 VI 계층구조 윈도우는 다른 상위 레벨 아이콘을 디스플레이하여 프로젝트를 나타냅니다. 추가한 각 타겟은 프로젝트 아래에 나타납니다.

다음 단계를 따라 VI 계층구조 윈도우를 사용합니다.

1. 보기>>VI 계층구조를 선택하여 VI 계층구조 윈도우를 디스플레이합니다.
2. 다음 옵션 중 선택합니다.
 - 커서를 VI 계층구조 윈도우에 있는 노드 위로 움직여서 톱 상자에 각 노드의 이름을 디스플레이합니다.
 - 위치 도구를 사용하여 VI 계층구조 윈도우에서 VI 또는 subVI 노드를 선택하고 블록 다이어그램으로 끕니다. 또한 한 노드 또는 여러 노드를 선택하고 클립보드에 복사하여 블록 다이어그램에 붙여넣을 수 있습니다.
 - VI 또는 SubVI 노드를 더블 클릭하여 해당 VI의 프런트패널을 디스플레이합니다.
 - <Ctrl> 키를 누른 채로 VI 또는 SubVI 노드를 더블 클릭하여 해당 VI의 프런트패널과 블록 다이어그램을 디스플레이합니다.
 - 이름으로 **노드를 검색**합니다.
 - 윈도우의 빈 영역에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 모든 VI 보이기를 선택하여 전체 계층구조를 보입니다.
 - 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 하이라이트 연결을 클릭하여, 현재의 노드와 연결된 계층구조의 아이템을 하이라이트합니다.
 - 프로젝트 또는 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 SubVI 숨기기 또는 모든 SubVI 보이기를 선택하여 계층구조에서 SubVI를 숨기거나 보이게 합니다.
 - 프로젝트 또는 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 직접적인 SubVI 보이기를 선택하여 단일 노드와 직접적으로 연결된 SubVI를 보여줍니다. 모든 SubVI가 숨은 상태일 경우에만 이 아이템을 사용할 수 있습니다.
 - 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 VI 계층구조 보이기를 선택하여 단일 노드에 대한 계층구조를 디스플레이합니다.
 - 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 호출자 보이기를 선택하여 지정한 노드에 대한 모든 호출자를 하이라이트합니다.
 - 일시 정지 상태인 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 호출 체인 보이기를 선택하여 VI의 호출 체인을 하이라이트합니다. 호출 체인은 현재 VI에서부터 최상위 VI까지의 호출자의 체인입니다. 실행 중인 VI를 디버깅하고 있는 동안에만 이 옵션을 사용할 수 있습니다.
 - VI 계층구조 윈도우의 도구 모음에 있는 버튼을 사용하여 다양한 VI 계층구조 보기 방법을 선택할 수 있습니다.



노트

VI 계층구조 윈도우는 메모리에 있는 내용을 동적으로 보여줍니다. 계층구조 안에 있는 아이템을 재배치하는 경우, LabVIEW는 아이템의 새 위치를 저장하지 않습니다. VI 계층구조 윈도우를 닫고 다시 열 때, 위치가 재생됩니다.

VI 계층구조에서 다형성 VI의 인스턴스를 디스플레이하기 위해, 각 다형성 VI 아이콘의 아래에 있는 빨간색 화살표를 클릭합니다. 최상위 레벨 VI는 다형성 VI와 다형성 VI의 인스턴스에 링크되어 있다는 것을 가리킵니다.

자동 도구 선택이 활성화되어 있을 때 스크롤링 도구로 스위치하려면, 커서가 윈도우의 아이콘 위에 있지 않을 때 <Ctrl-Shift> 키를 누릅니다. (macOS) <Option-Shift> 키를 누릅니다. (Linux) <Alt-Shift> 키를 누르십시오.

또한, 어플리케이션:메모리상의 모든 VI 프로퍼티를 사용하여 프로그램적으로 메모리에서 VI의 리스트를 가져올 수 있습니다.

다형성 VI

다형성 VI 만들기

다형성 VI는 서로 다른 데이터 타입에 적용됩니다. 다형성 VI는 같은 커넥터 팬 패턴을 가지는 VI의 모음입니다. 각 VI는 다형성 VI의 인스턴스로서, 각각의 인스턴스는 다른 인스턴스에 있는 다른 데이터 타입을 받는 입력 또는 출력 터미널을 1 개 이상 갖고 있습니다.

예를 들어, [키 읽기] VI는 다형성입니다. 이 VI의 기본값 터미널은 불리언; 배정도 부동소수; 32 비트 부호있는 정수; 경로; 문자열; 또는 32 비트 부호없는 정수 데이터를 받습니다.

여러 다른 데이터 타입에서 같은 연산을 하고, 각 데이터 타입에서 연산이 다르게 구현되어야 하는 경우 다형성 VI를 만듭니다. 특정 데이터 타입을 위해 작업을 크게 바꾸지 않고도 여러 데이터 타입에서 작업이 비슷하게 구현될 수 있는 경우, 이 VI 대신 **가변 VI**를 생성합니다.

다음 단계를 따라 다형성 VI를 만듭니다.

1. 같은 **커넥터 팬 패턴**을 가지고 있는 둘 이상의 VI를 생성합니다. 커넥터 팬이 대응하는 입력과 출력 터미널을 가지도록 합니다—한 VI의 커넥터 팬의 터미널이 입력인 경우, 다른 VI의 커넥터 팬의 대응하는 터미널 또한 입력이거나 사용되지 않아야 합니다. 출력 터미널에도 마찬가지입니다. VI는 비슷한 데이터 타입 또는 비슷한 subVI와 함수를 가지고 있지 않습니다.
2. 파일 » 새로 만들기를 선택하여 새로 만들기 대화 상자를 디스플레이합니다.
3. 새로 생성 리스트에서 다형성 VI를 선택합니다.
4. 확인 버튼을 클릭하여 다형성 VI 윈도우를 디스플레이합니다.
5. 추가 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.
6. 생성한 VI 중 하나를 선택하여 다형성 VI에 인스턴스로 포함시키고, 확인 버튼을 클릭합니다. VI는 인스턴스 VI 리스트의 맨 위쪽에 나타납니다. 리스트 상단에 있는 인스턴스는 다형성 VI의 기본 인스턴스이며 다형성 VI의 기본 커넥터 팬을 결정합니다.
7. 5 단계와 6 단계를 반복하여 다른 VI를 포함시킵니다. 기본 커넥터 팬 패턴에 부합하지 않는 VI는 깨진 아이콘으로 (🔍) 나타납니다. 이 경우에도 다형성 VI를 저장할 수 있습니다. 다형성 VI에서 인스턴스를 삭제하려면, 인스턴스를 선택하고 삭제 버튼을 클릭합니다.

8. (옵션) 인스턴스의 순서를 변경하려면, 리스트에서 인스턴스를 선택하고 위로 이동 화살표 버튼 또는 아래로 이동 화살표 버튼을 클릭합니다. 또한, 리스트에서 인스턴스를 선택하고 리스트의 새 위치로 끌 수 있습니다.



노트

인스턴스 VI 리스트의 VI를 더블 클릭하면 VI가 열립니다.

1. (옵션) 다형성 VI가 어떻게 블록 다이어그램에 나타나는지 지정하려면, 블록 다이어그램 모양 옵션을 설정합니다.
2. 아이콘 편집 버튼을 클릭하여 다형성 VI의 **아이콘을 생성**합니다.
3. 파일>>VI 프로퍼티를 선택하고 항목 풀다운 메뉴에서 문서를 선택하여 다형성 VI의 **설명을 생성**합니다. 다형성 VI 설명은 다형성 VI를 구성하는 인스턴스 VI의 설명과 관련되지 않습니다. 다형성 VI 윈도우에서 인스턴스 VI 아이콘 그리기를 선택하는 경우, 각 인스턴스 VI에 대해 설명을 생성하십시오. 그 이유는 커서를 다형성 VI 위로 이동할 때 LabVIEW가 기본 도움말 윈도우에 인스턴스 VI의 커넥터 팬과 설명을 디스플레이하기 때문입니다. 또한, 기본 도움말 윈도우는 커서를 다형성 VI 또는 해당 선택자의 바로 가기 메뉴 위로 이동할 때 각 인스턴스의 설명을 디스플레이합니다.
4. 파일>>저장을 선택하여 VI를 저장합니다.
5. 파일>>닫기를 선택하여 다형성 VI 윈도우를 닫습니다.

또한, 기존의 다형성 VI에 **인스턴스를 추가 또는 제거**하고, 다형성 VI와 해당 선택자의 **바로 가기 메뉴를 편집**하고, 다형성 VI의 **인스턴스를 수동으로 선택**할 수 있습니다.

관련 정보

다형성 VI

가변 VI와 다형성 VI 중 선택하기

다형성 VI의 바로 가기 메뉴 편집하기

다음 단계를 따라 생성한 다형성 VI의 바로 가기 메뉴와 선택자에 디스플레이할 인스턴스 이름을 편집합니다.



노트

LabVIEW에 포함되는 VI 중에서 다형성 VI를 편집할 때에는 주의하십시오.

1. 블록 다이어그램의 다형성 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다형성 VI 열기를 선택하여 다형성 VI 윈도우를 디스플레이합니다. 또한, 파일>>열기를 선택하고 다형성 VI를 열어 윈도우를 디스플레이할 수 있습니다.
2. 인스턴스 VI 리스트에서 편집하려는 이름을 가진 인스턴스를 선택합니다.
3. 이름 편집 버튼을 클릭하여 인스턴스 이름 편집 대화 상자를 디스플레이합니다.

4. 메뉴 이름 텍스트 박스에, 다형성 VI 바로 가기 메뉴에 사용할 이름을 입력합니다. 계층적인 바로 가기 메뉴를 생성하려면 콜론 : 을 사용하여 메뉴 항목을 구분합니다.
5. 다형성 VI 선택자 이름 텍스트 박스에서, **다형성 VI 선택자** 바로 가기 메뉴에 사용할 이름을 입력합니다. 여러 라인의 선택자 이름을 생성하기 위해 키보드의 <Enter> 키를 눌러 새로운 라인을 시작합니다.
6. 확인 버튼을 클릭하여 다형성 VI 윈도우로 돌아갑니다. 바로 가기 메뉴 이름이 인스턴스 VI 리스트의 인스턴스 VI 이름 옆에 나타납니다.
7. 파일>>저장을 선택하여 VI를 저장합니다.
8. 파일>>닫기를 선택하여 다형성 VI 윈도우를 닫습니다.

다형성 VI에서 인스턴스 추가 또는 제거하기

다음 단계를 따라 생성한 다형성 VI에 인스턴스를 추가합니다.



노트

LabVIEW에 포함되는 VI 중에서 다형성 VI를 편집할 때에는 주의하십시오.

1. 블록 다이어그램의 다형성 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다형성 VI 열기를 선택하여 다형성 VI 윈도우를 디스플레이합니다. 또한, 파일>>열기를 선택하고 다형성 VI를 열어 윈도우를 디스플레이할 수 있습니다.
2. 추가 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.
3. VI를 선택하여 다형성 VI에 인스턴스로 포함시키고 확인 버튼을 클릭합니다. 추가한 VI는 다형성 VI의 모든 다른 인스턴스와 같은 **커넥터 팬 패턴**을 가지고 있어야 합니다.
4. (옵션) 인스턴스의 순서를 변경하려면, 리스트에서 인스턴스를 선택하고 위 방향 또는 아래 방향 화살표 버튼을 클릭합니다. 또한, 리스트에서 인스턴스를 선택하고 리스트의 새 위치로 끌 수 있습니다.
5. 파일>>저장을 선택하여 VI를 저장합니다.
6. 파일>>닫기를 선택하여 다형성 VI 윈도우를 닫습니다.

다음 단계를 따라 생성한 다형성 VI에서 인스턴스를 제거합니다.

1. 블록 다이어그램의 다형성 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다형성 VI 열기를 선택하여 다형성 VI 윈도우를 디스플레이합니다. 또한, 파일>>열기를 선택하고 다형성 VI를 열어 윈도우를 디스플레이할 수 있습니다.
2. 인스턴스 VI 리스트에서 제거하려고 하는 인스턴스를 선택합니다.
3. 제거 버튼을 클릭합니다.
4. 파일>>저장을 선택하여 VI를 저장합니다.
5. 파일>>닫기를 선택하여 다형성 VI 윈도우를 닫습니다.

가변 VI 만들기

각 데이터 타입에 대해 별도의 VI를 저장하지 않고, 사용 가능한 모든 데이터 타입에 대해 같은 작업을 수행하려면, **가변 VI**를 생성합니다. 가변 VI는 다른 데이터 타입을 위해 작업을 크게 바꾸지 않고도 여러 데이터 타입에서 작업이 비슷하게 구현될 수 있는 경우에 사용하면 유용합니다. 여러 데이터 타입에서 작업이 다르게 구현되어야 하는 경우, 또는 여러 데이터 타입의 터미널 세트가 다른 경우, 대신 **다형성 VI**를 만듭니다.

다음과 같은 방법으로 가변 VI를 만들 수 있습니다.

- 새 가변 VI를 만들려면 파일>>새로 만들기를 선택하고 새로 만들기 대화 상자에서 가변 VI를 선택합니다.
- 기존의 VI를 가변 VI로 변환하려면, 파일 확장자를 .vim으로 설정합니다. 기존 VI가 호출 VI로 **인라인**되도록 설정되어 있지 않으면, 가변 VI는 깨져 보입니다. 파일>>VI 프로퍼티>>실행을 선택하고, 호출하는 VI로 SubVI 인라인과 미리 할당된 복제 재호출 실행 옵션을 활성화한 후, 디버깅 허용과 자동 에러 핸들링 허용 옵션을 비활성화하여, 가변 VI를 인라인되도록 설정해야 합니다.



노트

표준 VI만 가변 VI로 변환할 수 있습니다. 다형성 VI, 글로벌 VI 또는 X컨트롤 기능은 가변 VI로 변환할 수 없습니다.

블록 다이어그램에 가변 VI를 놓으면, 입력과 출력은 기본 데이터 타입을 보여줍니다. 컨트롤, 인디케이터 또는 상수를 입력이나 출력에 연결하면, 입력과 출력은 연결된 데이터 타입으로 변경됩니다.

예를 들어, [2D 배열 정렬] 가변 VI에서 2D 정렬 입력과 정렬된 2D 배열 출력은 참조 번호를 제외한 모든 데이터 타입의 2D 배열을 받고, 이때 기본 데이터 타입은 배열의 2D 배열입니다. 문자열 2D 배열을 2D 배열에 연결하면, 2D 배열과 정렬된 2D 배열의 데이터 타입은 모두 문자열 2D 배열로 변경됩니다. 그러나, 2D 배열에 1D 배열을 연결하는 경우, 이 가변 VI는 1D 배열을 받도록 구현되지 않았기 때문에 VI가 깨져보입니다.

타입 특수화 구조와 타입 지정 VI 및 함수를 사용하여 다음과 같은 목적으로 특수 가변 VI를 생성할 수 있습니다.

- 특정 데이터 타입에 대해 가변 VI의 코드 섹션을 사용자 정의하기
- 허용되는 데이터 타입 중 일부만 가변 VI가 허용하도록 하기
- 가변 VI가 특정 데이터 타입을 거부하도록 하기

예를 들어, [2D 배열 정렬] 가변 VI는 타입 특수화 구조를 사용하여 인덱스가 정수일 때와 정수 배열일 때 각각 다르게 동작합니다. 이렇게하면 가변 VI는 두 데이터 타입에 대해 모두 예상대로 동작하게 됩니다. 인덱스 입력은 부호있는 32 비트 정수가 아닌 정수 타입을 부호있는 32 비트 정수로 강제 변환합니다. [정수 타입 지정] VI만 사용하여 인덱스가 부호있는 32 비트 정수를 받도록 강제 설정할 수 있습니다.

가변 VI는 본질적으로 다형성이기 때문에 구문 에러를 발생시키지 않는 모든 데이터 유형을 허용합니다. NI는 코드를 다시 사용하기 더 용이하도록 가변 VI에서 강제 변환을 허용할 것을 권장합니다. 애플리케이션이 다음 중 하나에 해당하는 경우에만 가변 VI가 허용할 수 있는 데이터 타입을 제한하십시오.

- 강제 변환을 하는 경우 VI의 속도가 현저하게 감소됩니다.
- 작업 중인 어플리케이션에서 특정 데이터 타입은 아무런 의미가 없습니다.

관련 정보

가변 VI 디버깅하기

가변 VI와 다형성 VI 중 선택하기

가변 VI 디버깅하기

블록 다이어그램에 **가변 VI**를 놓고 특정 데이터 타입을 입력 및 출력에 연결하면 가변 VI가 이 데이터 타입에 맞도록 변경됩니다. 데이터 타입이 연결된 이 인스턴스 VI는 인라인되어있고 디버깅이 비활성화되었기 때문에 직접 디버깅할 수 없습니다. 그러나 이 인스턴스 VI를 표준 VI로 변환하면 디버깅할 수 있습니다.

인스턴스 VI에서 마우스 오른쪽 버튼을 클릭하고 인스턴스 VI를 표준 VI로 변환을 선택하여 인스턴스 VI를 표준 VI로 변환합니다. 이렇게하면 블록 다이어그램에서 표준 VI가 인스턴스 VI를 대체하고 표준 VI의 프런트패널이 열립니다. 표준 VI는 처음에는 디버깅이 비활성화되어 있지만, 파일>>VI 프로퍼티>>실행을 선택하고, 호출하는 VI로 SubVI 인라인 옵션을 비활성화하고, 디버깅 허용 옵션 및 재호출 실행 옵션 중 하나를 활성화하여 디버깅을 활성화할 수 있습니다.

VI 저장하기

VI를 개별 파일로 저장하거나 몇 개의 VI를 그룹화하여 LLB에 저장할 수 있습니다. LLB 파일은 .llb의 확장자로 끝납니다. National Instruments는 특히 많은 개발자들이 같은 프로젝트를 작업하는 경우, VI를 디렉토리 안에 개별적인 파일로 저장하는 것을 권장합니다.



노트

LabVIEW의 새로운 버전으로 업그레이드하기 전에 또는 VI에서 주요한 개정이 있을 때 VI의 백업 복사본을 만듭니다.

또한, 저장:인스트루먼트 메소드를 사용하여 VI를 프로그램적으로 저장할 수 있습니다.



관련 정보

[개별적인 VI 파일 저장하기](#) (897 페이지 참조)

[기존의 LLB에 VI 저장하기](#) (897 페이지 참조)

암호로 보호된 VI 생성하기

VI를 배포할 때, 블록다이어그램을 보호하기 위해 암호를 지정할 수 있습니다. 블록다이어그램은 여전히 유효하지만, 블록다이어그램을 보거나 편집하기 위해서는 사용자가 암호를 입력해야 합니다.



주의

VI 암호 보호 기능은 블록다이어그램을 암호화하지 않습니다. 보안을 최대화하려면, VI에서 **블록다이어그램을 제거**합니다. 암호 보호와 블록다이어그램 제거의 보안상 차이에 대한 더 자세한 정보는 ni.com에서 기술지원 데이터베이스를 참조하십시오.

또한, **VI를 잠그고** 사용자가 블록다이어그램을 볼 수는 있지만 편집할 수 없도록 만들 수 있습니다. 그러나, 사용자는 암호로 보호되지 않은 잠금 상태의 VI를 잠금 해제시킬 수 있습니다.

다음 단계를 따라 VI를 보호하기 위해 암호를 설정합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. VI 프로퍼티 대화 상자에 위치한 항목 풀다운 메뉴에서 보호를 선택합니다.
3. 암호로 보호됨 옵션을 클릭하여 암호 입력 대화 상자를 디스플레이합니다.
4. 새 암호 입력 필드에 암호를 입력합니다. 확인하기 위해 새 암호 다시 입력 필드에 같은 암호를 다시 입력합니다.
5. 확인 버튼을 클릭합니다.



노트

VI에 암호를 적용하면 자동으로 암호를 LabVIEW 패스워드 캐쉬에 저장합니다. 이를 통해 마치 암호를 입력한 것처럼 VI에 접근하여 변경 사항을 적용할 수 있습니다. LabVIEW를 다시 시작하거나 옵션 대화 상자의 환경 페이지에서 암호 캐시를 지워 캐시에서 암호를 제거할 수 있습니다.

또한, 소스 배포 프로퍼티 대화 상자를 사용하여 암호를 추가할 수 있습니다.

또한, 잠금 상태:설정 메소드를 사용하여 VI의 암호를 프로그램적으로 설정할 수 있습니다.

VI 잠그기와 잠금해제하기

VI를 잠그어 사용자가 볼 수는 있지만 암호 없이 블록다이어그램을 편집할 수는 없도록 할 수 있습니다. 또한, **암호 보호 기능을 VI에 지정**해서 사용자가 암호 없이 블록다이어그램을 보거나 편집할 수 없습니다. 잠금 상태:설정 메소드를 사용하여 VI를 프로그램적으로 잠그거나 잠금해제할 수 있습니다.

VI 잠그기

다음 단계를 따라 VI를 잠금니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 보호를 선택합니다.
3. 잠금(암호 없음)을 선택합니다.
4. 확인 버튼을 클릭합니다.

VI 잠금해제하기

다음 단계를 따라 VI를 잠금해제합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 보호를 선택합니다.
3. VI가 암호 없이 잠겨 있을 경우, 잠금해제(암호 없음)을 선택하여 잠금해제합니다. VI가 암호로 잠겨 있을 경우, 잠금해제(암호 없음)을 선택한 후 인증 대화 상자에 적절한 암호를 입력합니다. 입력 요청의 예 버튼을 클릭하여 보호를 해제합니다.

또는 수행>>편집 모드로 변경을 선택하여 VI를 잠금해제합니다. 나타나는 대화 상자에서 잠금해제를 선택합니다.

일반적인 매스 컴파일 상태 메시지

매스 컴파일 대화 상자의 상태 탭 아래에서 다음 열거된 일반적인 매스 컴파일 메시지 중 하나 이상을 받을 수 있습니다.



노트

다음 테이블에 없는 메시지를 받는 경우, 메시지가 가리키는 VI를 엽니다. VI에 에러가 있는 경우, 다시 매스 컴파일을 시도하기 전에 VI의 에러를 수정합니다.

메시지	설명
###Bad VI: "VI의 파일 이름" Path="VI의 경로"	지정된 VI는 LabVIEW가 매스 컴파일할 수 없는 비실행 VI입니다. VI를 열고 에러를 확인합니다.
VI의 경로를 로드하는데 실패	VI가 메모리에 없기 때문에 LabVIEW가 지정된 디렉토리 또는 검색 디렉토리에서 VI를 찾을 수 없습니다. 지정된 VI를 찾고 VI가 손상되지 않았는지 또는 에러는 없는지 확인합니다.
"VI의 경로"에서 이전에 "VI의 파일 이름"을 찾지 못함	LabVIEW가 지정된 디렉토리 또는 검색 디렉토리의 지정된 위치에서 VI를 찾을 수 없습니다. 지정된 아이টে를 찾아 검색 디렉토리나 예상 위치로 옮깁니다.

메시지	설명
메모리에 있는 VI의 경로 때문에 VI의 경로를 로드할 수 없음	LabVIEW가 지정된 디렉토리에서 VI를 찾았지만 같은 이름의 VI가 이미 메모리에 로드되어 있기 때문에 해당 VI를 로드할 수 없습니다. 두 아이템을 비교하고, 올바른 버전이 호출 VI에 있는지 확인합니다.
"VI의 경로"에 있을 것으로 예상되는 VI의 파일 이름이 "VI의 경로"에서 로드되었음	VI가 호출 VI의 지정된 경로에서 발견되지 않은 관계로, LabVIEW는 동일한 이름의 아이템을 검색 디렉토리에서 로드했습니다. VI가 호출자와 디렉토리 경로를 올바르게 지정하고 있는지 그리고 로드된 VI가 호출자 지정 VI와 같은지 확인합니다. VI가 같지 않은 경우, 호출 VI에 올바른 subVI를 추가합니다.
프로젝트 경로에 있는 프로젝트는 이미 열려 있으므로 컴파일할 수 없습니다. 프로젝트를 닫고 다시 컴파일하십시오.	프로젝트가 이미 열려 있기 때문에 LabVIEW가 프로젝트를 컴파일할 수 없습니다. 컴파일하기 전에 프로젝트를 닫습니다.
이미 메모리에 있는 프로젝트 건너뛰기: 프로젝트 경로	LabVIEW가 메모리에 이미 로드된 같은 경로의 VI를 찾았습니다. 그 결과, LabVIEW는 해당 VI를 매스 컴파일하지 않습니다.
CompileFile: error 번호 at VI의 경로	이 에러는 지정된 에러 코드를 반환하며 이를 검색하면 더 많은 정보를 얻을 수 있습니다.
CompileFile: skipping VI의 경로	LabVIEW가 몇 가지 이유 중 하나로 VI를 컴파일할 수 없습니다. VI가 템플릿 VI이거나 VI에 템플릿 컨트롤이 있을 수 있습니다. 해당 경로에 있는 파일의 파일 타입이 경로에서 제시하는 확장자와 다릅니다. 또한 LabVIEW가 VI를 찾거나 열 수 없기 때문에 또는 VI가 손상되었기 때문에, VI를 컴파일하지 못할 수도 있습니다.
CompileFile: VI 경로에서 사용자가 취소	사용자가 VI 로드하기를 취소했기 때문에 VI가 컴파일되지 않았습니다.

LLB

LLB 생성하기

다음 단계를 따라 새 **LLB**를 생성합니다.

1. VI의 블록다이어그램 또는 프런트패널에서 파일>>저장 또는 다른 이름으로 저장을 선택하여 파일 대화 상자를 디스플레이합니다. (macOS) 기본 파일 대화 상자를 사용하는 경우, 파일 대화 상자에서 LLB 사용 버튼을 클릭합니다.



노트

블록다이어그램 또는 프런트패널에서 도구>>LLB 관리자를 선택하여 LLB 관리자 윈도우를 디스플레이해서 이미 저장된 VI가 있는 새 LLB를 생성할 수 있습니다.

1. 새 LLB 버튼을 클릭하여 새 LLB 대화 상자를 디스플레이합니다.
2. 새 LLB의 이름을 입력한 후 생성 버튼을 클릭합니다.
3. 파일 대화 상자가 나타나면, VI의 이름을 입력하고 새 LLB에 저장합니다. LabVIEW는 자동적으로 LLB 이름의 확장자로 .llb를 추가합니다.
4. 확인 버튼을 클릭합니다.

또한 LLB에 출력하도록 소스 배포 프로퍼티 대화 상자의 대상 페이지에서 소스 배포의 대상을 설정할 수 있습니다.

또한, **기존의 LLB에 VI를 저장**할 수 있습니다.

디렉토리를 LLB로 변환하기

LLB의 디렉토리에 있는 **모든 VI를 그룹화**하려면, 디렉토리를 **LLB**로 변환합니다.

다음 단계를 따라 디렉토리를 LLB로 변환합니다.

1. 도구>>LLB 관리자를 선택하여 LLB 관리자 윈도우를 디스플레이합니다.
2. 풀다운 메뉴의 아래 방향 화살표를 클릭하여 VI가 기존의 디렉토리를 탐색합니다. LLB, 다른 파일, 서브디렉토리가 풀다운 메뉴 아래의 리스트에 나타납니다.
3. VI가 기존의 디렉토리를 선택합니다.
4. 디렉토리와 LLB 변환 버튼을 클릭하여 변환할 디렉토리 대화 상자를 디스플레이합니다.
5. 변환 버튼을 클릭합니다.
6. 파일>>닫기를 선택하여 LLB 관리자 윈도우를 나갑니다.

또한, **LLB를 디렉토리로 변환**할 수도 있습니다.

LLB를 디렉토리로 변환하기

LLB에서 VI를 제거하고, 개별적인 VI로 나타내려면, 다음 단계를 완료하여 **LLB**를 디렉토리로 변환합니다.

1. 도구>>LLB 관리자를 선택하여 LLB 관리자 윈도우를 디스플레이합니다.
2. 풀다운 메뉴의 아래 방향 화살표를 클릭하여 LLB가 상주하는 디렉토리를 탐색합니다. LLB, 다른 파일, 서브디렉토리가 풀다운 메뉴 아래의 리스트에 나타납니다.
3. 변환할 LLB를 선택합니다.
4. 디렉토리와 LLB 변환 버튼을 클릭하여 변환 옵션 대화 상자를 디스플레이합니다.
5. (옵션) 디렉토리 이름과 원본 LLB 옵션을 설정합니다.
6. 변환 버튼을 클릭합니다.
7. 파일>>닫기를 선택하여 LLB 관리자 윈도우를 나갑니다.

또한, **디렉토리를 LLB로 변환**할 수도 있습니다.

VI 및 LLB 병합하기

소스 컨트롤을 사용하여 여러 개발자와 소프트웨어 프로젝트를 공유할 수 있습니다. 파일을 체크아웃하고, 변경하고, 업데이트된 파일을 체크인하는 경우, 업데이트된 파일이 마스터 복사본을 대체하고 이를 다른 개발자가 사용할 수 있습니다. 현재 다른 개발자가 체크아웃한 상태의 파일을 체크아웃하는 경우, 파일에 적용하는 변경 사항은 업데이트된 마스터 복사본을 저장하기 전에 다른 개발자가 적용하는 변경 사항과 병합되어야 합니다.

LabVIEW Professional Development System의 병합할 LLB 선택 대화 상자를 사용하여, 자동으로 또는 대화식으로 원본 LLB와 두 개정본 사이의 차이를 해결합니다.

또한, LabVIEW Professional Development System의 병합할 VI 선택 대화 상자를 사용하여 개별 VI를 병합할 수 있습니다.



관련 정보

[LabVIEW에서 소스 컨트롤 사용하기](#) (336 페이지 참조)

[LLB 병합하기](#) (925 페이지 참조)

[VI 병합하기](#) (924 페이지 참조)

LLB에서 VI를 최상위 레벨 VI로 표시하기

LLB에서 VI를 최상위 레벨 VI로 표시하여, **명령 라인에 지정된** LLB 이름으로 LabVIEW를 시작할 때, LabVIEW가 해당 LLB에 있는 모든 최상위 레벨 VI를 열도록 합니다. (macOS와 Linux) VI를 상위 레벨로 표시하여 OS로부터 LLB를 열 때, LabVIEW는 그 LLB의 모든 최상위 레벨 VI를 엽니다.

파일 대화 상자에서 LLB 내용을 볼 때 최상위 레벨 VI는 LLB에 있는 다른 모든 VI로부터 나뉘선에 의해 분리됩니다. 최상위 레벨 VI는 나뉘선 위에 나타납니다.

다음 단계를 따라 VI를 최상위 레벨 VI로서 표시합니다.

1. 도구 » LLB 관리자를 선택하여 LLB 관리자 윈도우를 디스플레이합니다.
2. LLB를 더블 클릭하여 LLB의 VI를 디스플레이합니다.
3. VI를 선택하고 편집 » 최상위 레벨을 선택합니다. 또한, VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 최상위 레벨을 선택할 수 있습니다.
4. 파일 » 닫기를 선택하여 LLB 관리자 윈도우를 나갑니다.

(Windows) 또한, Windows 탐색기의 .llb 파일을 더블 클릭하여 LLB 관리자 윈도우를 디스플레이할 수 있습니다. LLB의 VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 최상위 레벨을 선택하여 VI를 최상위 레벨로 표시합니다.

VI 매스 컴파일하기

VI를 매스 컴파일할 때, LabVIEW는 SubVI가 있는지 확인하고 SubVI를 Main VI에 다시 연결합니다. SubVI를 Main VI에 다시 연결하면 LabVIEW가 해당 VI를 찾을 필요가 없게 되어 로딩 시간을 줄일 수 있습니다. 또한 LabVIEW는 VI를 사용 중인 LabVIEW의 버전으로 업데이트합니다. LabVIEW는 손상된 VI를 감지하고 리포트하며, LabVIEW는 손상된 VI를 로드할 수 없습니다. LabVIEW가 손상된 VI를 발견한 경우, 백업 파일에서 복원할 수 있습니다.

VI를 새로운 버전의 LabVIEW로 업그레이드하는 것과 같은 특정한 경우, VI가 처음 로드될 때 LabVIEW는 VI를 다시 컴파일해야 합니다. VI를 저장할 때까지 LabVIEW는 VI가 로드될 때마다 계속해서 VI를 다시 컴파일합니다. 로드 시 LabVIEW가 많은 수의 VI를 다시 컴파일할 것으로 예상되면, 이후 이와 같은 복수의 작업을 피하기 위해 동시에 모든 VI를 다시 컴파일하고 저장하는 것이 좋습니다. 동시에 VI의 집합을 컴파일하고 저장하는 것은 VI를 매스 컴파일하는 것입니다.

다음 단계를 따라 VI 또는 LLB의 디렉토리를 매스 컴파일합니다.

1. 도구 » 고급 » 매스 컴파일을 선택하여 이 매스 컴파일 대화 상자를 디스플레이합니다.
2. 컴파일하려는 VI나 LLB의 디렉토리를 탐색합니다.
 - 폴더를 탐색하고 현재 폴더 버튼을 클릭하여 폴더에 있는 모든 VI와 LLB를 업데이트합니다.
 - .11b 파일을 선택하여 그 LLB에 있는 모든 VI를 업데이트합니다. 오른쪽에 하나의 마침표를 가진 열린 폴더 아이콘으로 라벨된 VI를 선택하고 선택 버튼을 클릭합니다.
3. (옵션) 로그 결과 확인란에 확인 표시를 하고, 매스 컴파일 작업의 로그를 저장할 위치를 선택합니다.
4. (옵션) 캐시 VI 확인란에 확인 표시를 하고 LabVIEW가 메모리에 저장하기를 원하는 VI의 개수를 입력합니다.



노트

VI를 캐시하는 경우, 사용자가 최상위 VI를 컴파일할 때 LabVIEW가 캐시된 VI를 로드할 필요가 없으므로 매스 컴파일의 실행 시간을 상당히 줄일 수 있습니다. 하지만, VI 캐시는 더 많은 시스템 메모리를 소비합니다. 다른 하위 폴더에 있는 VI와 이름이 같은 VI가 포함된 폴더를 매스 컴파일하고, 이 VI가 라이브러리에 속하지 않는 경우 VI를 캐시하면 [상호 링크](#)가 발생할 수 있습니다.

5. 매스 컴파일 버튼을 클릭하여 VI를 매스 컴파일합니다.

노트 상태 탭 아래에서 [메시지를 받는](#) 경우, 리스트된 VI를 엽니다. VI에 에러가 있는 경우, 다시 매스 컴파일을 시도하기 전에 VI의 에러를 수정합니다.

6. 완료 버튼을 클릭하여 LabVIEW로 돌아갑니다.

VI를 매스 컴파일할 때, 매스 컴파일 대화 상자의 상태는 매스 컴파일 작업에 대한 [경고 메시지](#) 로그를 디스플레이합니다. 예를 들어, 이 로그는 손상된 VI 및 호출 VI에서 지정된 경로 대신 검색 디렉토리로부터 로드된 VI를 보고할 수 있습니다. 깨진 VI 또는 예상하지 못한 위치에서 로드된 VI 등, 컴파일하는 VI에서 예상하지 못한 동작을 감지하도록 매스 컴파일할 수 있습니다.

또한, 매스 컴파일 메소드 또는 **MassCompile** 명령 라인 작업을 사용하여 디렉토리에서 VI를 프로그램적으로 로드하고 컴파일할 수 있습니다.

매스 컴파일 작업 순서

LabVIEW는 마지막으로 이전 버전에서 저장된 VI 및 저장하지 않은 변경 사항이 있어서 사용자가 열 때 다시 컴파일될 VI만을 매스 컴파일합니다. LabVIEW가 VI로 이루어진 디렉토리를 매스 컴파일 할 때, 다음 순서에 따라 디렉토리 계층구조의 각 레벨별로 파일을 처리합니다:

- LabVIEW 프로젝트 파일—LabVIEW가 프로젝트를 발견하면, 프로젝트 파일을 처리한 후에 그 프로젝트가 소유하는 모든 파일을 프로젝트 탐색기 윈도우의 프로젝트 트리에 나타나는 순서대로 처리합니다. **조건적 비활성화 기호**와 같이 프로젝트가 정의하는 프로젝트 및 타겟에 특정한 셋팅은 LabVIEW가 해당 VI를 실행할 때 사용할 플랫폼 또는 프로세서 지정 등과 같이 파일을 컴파일하는 방법을 지정합니다.
- 서브디렉토리 내에 있는 파일—LabVIEW는 글자순으로 서브디렉토리를 하나씩 처리합니다. 예를 들어, 디렉토리가 B와 F라는 폴더를 포함하는 경우에 LabVIEW는 B에 있는 파일을 글자순으로 처리한 다음에 F에 있는 파일을 글자순으로 처리합니다.
- 해당 디렉토리 레벨에 있는 나머지 파일은 모두 글자순으로 처리됨

VI에서 블록다이어그램 제거하기

VI를 다른 컴퓨터 또는 다른 사용자에게 배포하는 경우, VI에서 블록다이어그램 소스 코드를 제거하여 다른 사용자가 VI를 편집하지 못하도록 할 수 있습니다. 또한, **블록다이어그램에 암호 보호를 설정**하면 사용자가 암호를 입력해야 블록다이어그램에 접근할 수 있지만, 이 방법은 블록다이어그램을 제거하는 방법보다는 안전하지 않습니다.



노트

블록다이어그램 없이 VI를 저장하는 경우, 사용자는 VI를 편집할 수 없고, **해당 VI를 다른 플랫폼으로 이동**시킬 수 없고, 해당 VI를 다음 버전의 LabVIEW로 업그레йд할 수 없습니다.

다음 단계를 따라 블록다이어그램 없이 VI를 저장하여 파일 크기를 줄이고 사용자가 소스 코드를 변경하지 못하게 할 수 있습니다.

1. **소스 배포를 생성**합니다.
2. 소스 배포 프로퍼티 대화 상자의 소스 파일 셋팅 페이지에서, 기본 저장 셋팅 사용 확인란에서 확인 표시를 제거하고 블록다이어그램 제거 확인란에 확인 표시를 하여 LabVIEW가 블록다이어그램을 제거하는지 확인하십시오.

노트 **컴파일된 코드가 없는 VI**에서는 블록다이어그램을 제거할 수 없습니다. 추가적인 예외 페이지에서 컴파일된 코드 제거 확인란에 확인 표시를 하면, LabVIEW는 소스 파일 셋팅 페이지에서 기본 저장 셋팅 사용 확인란을 비활성화합니다.

3. 소스 배포를 만들어 블록다이어그램 없이 VI의 복사본을 생성합니다.



주의 블록다이어그램 없이 VI를 저장하는 경우, VI의 원본에 덮어쓰지 마십시오. VI를 다른 디렉토리에 저장하거나 다른 이름을 사용하십시오.

LLB에서 파일 제거하기

다음 단계를 통해서 LLB에서 파일을 삭제합니다.



노트

윈도우 탐색기의 .llb 파일에서 파일을 지우면 파일을 영원히 지우게 되고 휴지통에 파일을 저장하지 않습니다.

1. (옵션) 파일의 복사본을 다른 위치에 저장합니다.
2. 도구 >> LLB 관리자를 선택하여 LLB 관리자 윈도우를 디스플레이합니다.
3. 수정하려는 LLB를 탐색한 후 LLB를 더블 클릭하여 LLB의 파일을 봅니다.
4. LLB에서 삭제하려는 파일을 선택합니다.
5. 삭제 버튼을 클릭하거나 파일 >> 삭제를 선택하여 LLB에서 파일을 삭제합니다. 또한, 파일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 삭제를 선택하여 파일을 삭제할 수 있습니다.



노트 LLB 관리자 윈도우에서 작업한 변경 내용은 취소할 수 없습니다.

1. 파일 >> 닫기를 선택하여 LLB 관리자 윈도우를 나갑니다.

(Windows) 또한 Windows 탐색기의 .llb 파일에서 더블 클릭하여 LLB 관리자 윈도우를 디스플레이하고 LLB에 있는 파일을 삭제할 수 있습니다. LLB의 파일에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 삭제를 선택하여 파일을 삭제합니다.

VI의 마지막 저장된 버전으로 되돌리기

파일 >> 되돌리기를 선택하여 편집하고 있는 VI의 마지막 저장된 버전으로 되돌아갑니다. 대화 상자는 VI에 실시한 모든 변경을 취소할지 여부를 확인합니다.

VI를 저장하고 나서 다시 변경하면, 제목 표시줄에 별표가 표시되고, 열려 있는 VI의 리스트가 윈도우 메뉴 및 모든 윈도우 대화 상자에 디스플레이됩니다. VI를 저장하면, 새로 변경할 때까지 별표는 사라집니다.

또한 VI 되돌리기 메소드를 사용하여 프로그램적으로 VI의 마지막 저장된 버전으로 돌아갈 수 있습니다.

LabVIEW에 포함되어 출시되는 VI의 변화 저장하기

다음의 리스트는 LabVIEW에 포함되어 출시되는 VI와 다른 파일을 변경할 때 일부 유의점과 권장사항을 설명합니다.

- 출시하는 파일을 복사해 놓고 변경하기 전에 다른 디렉토리에 그 파일을 저장합니다.
- 설치를 제거하거나 다시 설치할 때 LabVIEW가 .lib 파일에 저장된 모든 VI 및 기타 파일을 포함하여 .lib 파일을 제거하기 때문에, LabVIEW와 함께 labview\vi.lib 디렉토리에 설치되는 .lib 파일에 사용자가 만든 VI나 기타 파일을 저장하지 마십시오.

이전 버전으로 저장하기

LabVIEW의 이전 버전으로 VI, LabVIEW 프로젝트 및 프로젝트 라이브러리를 저장하여 LabVIEW를 편리하게 업그레이드하고 필요한 경우 하나 이상의 LabVIEW 버전으로 파일을 관리할 수 있습니다.



노트

이전 LabVIEW 버전으로 **묶음 프로젝트 라이브러리**를 저장하려면, 묶음 라이브러리에서 이전 버전으로 파일을 저장한 후, 원하는 LabVIEW 버전으로 **묶음 라이브러리를 다시 만듭니다..**

다음 단계를 따라 LabVIEW의 이전 버전으로 VI, 프로젝트 또는 프로젝트 라이브러리의 계층구조를 저장합니다.

- VI의 계층구조에서 상위 레벨 VI의 경우, 파일>이전 버전으로 저장을 선택하여 이전 버전으로 저장 대화 상자를 디스플레이합니다. 프로젝트의 경우, **프로젝트 탐색기** 윈도우에서 파일>이전 버전으로 저장을 선택합니다. 프로젝트 라이브러리의 경우, 프로젝트 탐색기 윈도우의 라이브러리 파일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이전 버전으로 저장을 선택하거나 프로젝트 라이브러리를 열고 파일>이전 버전으로 저장을 선택합니다.
- LabVIEW 버전 풀다운 메뉴에서 VI, 프로젝트 또는 프로젝트 라이브러리를 저장할 LabVIEW의 버전을 선택합니다.

노트 LabVIEW에 버전 정보 리스트를 제공하는 LabVIEW 툴킷이 사용되는 VI 계층구조, 프로젝트 또는 프로젝트 라이브러리를 저장하는 경우, 툴킷 버전 필드에서 어떤 버전의 툴킷으로 저장할 것인지 선택할 수 있습니다.

- 저장 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.
- VI 계층구조, 프로젝트 또는 프로젝트 라이브러리를 저장하려는 디렉토리를 선택합니다.
- 저장 버튼을 클릭합니다.

노트 이전 LabVIEW 버전에서 암호로 보호된 VI를 저장하려면 암호를 입력해야 합니다.

VI, 프로젝트 또는 프로젝트 라이브러리가 이전 버전에서 이용할 수 없는 기능을 사용한다면, LabVIEW는 가능한 많은 것을 저장하고 변경할 수 없는 것에 대한 리포트를 생성합니다. 리포트는 이전 버전으로 저장 경고 요약 대화 상자에 즉시 나타납니다.

이전 버전으로 저장 경고 요약 대화 상자에 있는 세부사항 보이기 버튼을 클릭하여, 로드 및 저장 경고 리스트 대화 상자를 열고 각 경고에 대한 세부사항을 확인합니다. 또한, 무시 버튼을 클릭하여 경고를 확인하고 요약 대화 상자를 닫을 수 있습니다. 로드 및 저장 경고 리스트 대화 상자에 있는 파일에 저장 버튼을 클릭하여 나중에 볼 수 있도록 텍스트 파일에 경고를 저장합니다.

또한, 저장:이전 버전으로 메소드를 사용하여 프로그램적으로 LabVIEW의 이전 버전으로 VI를 저장할 수 있으며, 이전 버전으로 저장 메소드(프로젝트)를 사용하여 프로그램적으로 LabVIEW 이전 버전으로 LabVIEW 프로젝트를 저장할 수 있고, 저장:이전 버전으로 메소드(라이브러리)를 사용하여 프로그램적으로 LabVIEW 이전 버전으로 프로젝트 라이브러리를 저장할 수 있습니다.

개별적인 VI 파일 저장하기

다음의 방법 중 하나를 사용하여 개별적인 VI를 저장합니다:

- 파일>>저장을 선택하여 기존 VI의 변화를 그것의 현재 디스크 위치에 저장하거나 새 VI를 저장합니다. 새 VI이면, 파일 대화 상자에서 VI 이름과 그 위치를 지정합니다.
- 이전에 저장된 VI에서 파일>>다른 이름으로 저장을 선택하여 다른 이름으로 저장 대화 상자를 디스플레이합니다. 이 대화 상자에서 메모리상의 VI 복사본을 선택한 새 이름으로 디스크에 저장할 수 있으며, VI의 이름을 변경하거나, VI 계층구조를 새 위치에 복사할 수 있습니다.
- 파일>>이전 버전으로 저장을 선택하여 이전 버전으로 저장 대화 상자를 디스플레이합니다. 이 대화 상자에서 사용할 VI를 LabVIEW의 이전 버전으로 저장할 수 있습니다.

기존의 LLB에 VI 저장하기

새 VI를 기존의 LLB에 마치 디렉토리인 것처럼 저장할 수 있습니다.

다음 단계를 따라 VI를 기존의 LLB에 저장합니다.

1. 파일>>저장을 선택하여 파일 대화 상자를 디스플레이합니다. (macOS) 기본 파일 대화 상자를 사용하는 경우, LLB를 선택하기 전에 LLB 사용 버튼을 클릭합니다.
2. VI를 저장하려는 LLB를 탐색합니다. .llb 확장자를 가진 LLB의 이름이 파일 대화 상자에 나타납니다.

**주의**

사용자 VI 및 다른 파일을 LabVIEW와 함께 설치되는 .llb 파일에 저장하지 마십시오. 컨트롤과 함수 팔레트에 추가하려면 **user.lib** 디렉토리에 컨트롤과 VI를 저장하십시오.

1. LLB 파일의 이름을 더블 클릭하거나 확인 버튼을 클릭하여 VI 이름 입력 대화 상자를 디스플레이합니다.
2. VI 이름을 입력하고 확인 버튼을 클릭합니다.

VI 사용자 정의하기

LabVIEW는 VI의 사용자 정의에 사용할 수 있는 여러가지 옵션을 제공합니다. 파일»VI 프로퍼티를 선택하거나, 다음 단계에 따라 VI 사용자 정의 옵션에 접근합니다.

1. 프런트패널 또는 블록 다이어그램의 오른쪽 상단 코너에 위치한 아이콘을 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 VI 프로퍼티를 선택합니다.

지정된 subVI를 요청할 때 subVI의 설정을 변경하려면, SubVI 노드 설정 대화 상자를 사용합니다.

또한, VI 프로퍼티를 이용하여 프로그램적으로 설정할 수 있습니다.

VI 윈도우 제목 변경하기

VI 윈도우 제목을 사용자 정의하여 VI의 파일 이름보다 더욱 설명적으로 만들 수 있습니다. VI 윈도우 제목을 현지 언어로 번역할 수 있으므로 **현지화된 VI**에 유용합니다.

다음 단계를 따라 VI 윈도우 제목을 변경합니다.

1. 파일»VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. VI 프로퍼티 대화 상자의 맨 위 항목 풀다운 메뉴에서 윈도우 모양을 선택합니다.
3. VI 이름과 동일 확인란에서 확인 표시를 제거합니다.
4. 윈도우 제목 텍스트 박스에 VI의 제목을 입력합니다.
5. 확인 버튼을 클릭하여 대화 상자를 종료합니다.

또한 프런트패널 윈도우:제목 프로퍼티를 사용하여 프로그램에서 VI 윈도우 제목을 변경합니다.

탐색 경로와 키워드 복사하기

다음 단계를 따라 한 예제 VI에서 다른 VI로 태스크 기반 탐색 경로와 키워드를 복사합니다.

1. 도구 » NI 예제 탐색기를 위해 예제 VI 준비를 선택하여 NI 예제 탐색기를 위해 예제 VI 준비 대화 상자를 디스플레이합니다.
2. 예제 VI에 태스크 기반 탐색 경로와 키워드를 **추가**합니다.
3. NI 예제 탐색기를 위한 예제 VI 준비 대화 상자의 복사 버튼을 클릭하여 문서 테이블로부터 예제 VI에 대한 태스크 기반 탐색 경로와 키워드를 복사합니다. 오직 태스크 기반의 탐색 경로만 또는 오직 키워드만 복사할 수는 없습니다.
4. 더블 클릭 시 예제를 문서로 리스트에서, 태스크 기반의 탐색 경로와 키워드를 복사하려는 예제 VI를 탐색하고 더블 클릭합니다.
5. (옵션) 예제 VI에 붙여넣기 전에 태스크 기반의 탐색 경로와 키워드를 보기 위해 보기 버튼을 클릭합니다. 확인 버튼을 클릭하여 예제 탐색 경로와 키워드 복사 원도를 닫습니다.
6. 붙여넣기 버튼을 클릭하여 문서를 복사하려는 예제 VI에 태스크 기반의 탐색 경로와 키워드를 추가합니다.
7. 문서 저장 버튼을 클릭하여 문서를 복사한 예제 VI에 태스크 기반의 탐색 경로와 키워드를 저장합니다.

NI 예제 탐색기에 나타내기 위한 예제 VI 준비하기

다음 단계에 따라 사용자가 생성한 예제 VI가 NI 예제 탐색기에 나타나도록 준비하고 태스크 기반 탐색 경로, 키워드, 예제 VI에 대한 기타 정보를 저장하는 데이터 .bin3 파일을 만듭니다. LabVIEW는 .bin3 파일을 사용하여 NI 예제 탐색기에 예제 VI를 디스플레이합니다.



노트

문서화하는 예제 VI가 쓰기 보호되어 있지 않도록 하십시오.

1. 도구 » NI 예제 탐색기를 위해 예제 VI 준비를 선택하여 NI 예제 탐색기를 위해 예제 VI 준비 대화 상자를 디스플레이합니다.
2. 예제를 위한 시작 디렉토리 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고, NI 예제 탐색기에 나타내려는 예제 VI나 프로젝트가 포함된 디렉토리를 탐색하고, 폴더 선택 버튼을 클릭합니다.



노트 생성한 예제 VI와 .bin3 파일은 다음 labview 디렉토리의 서브디렉토리에 있어야 합니다.

- 예제 VI — examples, instr.lib 또는 user.lib. 예를 들어, labview\user.lib\my_examples가 예제 VI를 포함하기에 적당한 디렉토리입니다. 또한 예제 VI가 있는 디렉토리 구조는 사용자 컴퓨터에 저장될 예제 VI의 디렉토리 구조와 동일해야 합니다.
- .bin3 파일 — examples\exbins, instr.lib 또는 user.lib

1. 더블 클릭하여 예제를 문서로 리스트에서 문서화하려는 예제 VI를 찾고 더블 클릭합니다. VI 이름이 문서 테이블에 나타납니다. 위로 스크롤하면 예제 VI 또는 프로젝트를 생성한 ADE (어플리케이션 개발 환경) 및 ADE 버전이 문서 테이블에 나타납니다. ADE 및 버전 값을 바꿀 수 없습니다.
2. 사용자를 위한 예제 VI에 대해 작업 단위의 탐색 경로를 추가하여 NI 예제 탐색기의 탐색 페이지에 있는 작업에 의해 예제 VI를 탐색합니다.
 1. 더블 클릭 시 탐색 경로 추가 리스트에서, 예제 VI를 위한 적절한 작업 단위의 탐색 경로를 선택합니다.
 2. 항목의 가장 낮은 단계를 탐색한 후에, 항목을 추가하기 위해 회색 아이콘을 더블 클릭합니다.

작업 단위의 탐색 경로가 문서 테이블에 나타납니다. 또한, 하나의 예제 VI로부터 다른 것으로 **탐색 경로를 복사**할 수 있습니다. 문서 테이블에서 탐색 경로를 더블 클릭하여 예제 VI에서 탐색 경로를 제거합니다.

3. (옵션) 4 단계를 반복하여 더 많은 작업 단위의 탐색 경로를 예제 VI에 추가합니다.
4. 키워드를 예제 VI에 추가하여 사용자가 NI 예제 탐색기의 검색 페이지에 입력할 수 있습니다.
 1. 더블 클릭 시 단어 추가 리스트박스에서, 리스트를 스크롤하여 키워드를 찾고 단어를 더블 클릭하여 그것을 예제 VI에 추가합니다. 키워드가 문서 테이블에 나타납니다.
 2. 단어를 새 키워드 텍스트 박스에 입력하고 새 키워드 추가 버튼을 클릭하여 새 키워드를 더블 클릭 시 단어 추가 리스트박스에 추가합니다.
 3. 더블 클릭 시 단어 추가 리스트박스에 새 키워드를 더블 클릭하여 키워드를 예제 VI에 추가합니다. 키워드가 문서 테이블에 나타납니다.

또한, 하나의 예제 VI로부터 다른 것으로 **키워드를 복사**할 수 있습니다. 문서 테이블에서 키워드를 더블 클릭하여 예제 VI에서 키워드를 제거합니다. 더블 클릭 시 단어 추가 리스트박스에서 키워드를 선택하고 키워드 제거 버튼을 클릭하여 키워드 리스트에서 키워드를 제거합니다.

5. (옵션) 6 단계를 반복하여 더 많은 키워드를 예제 VI에 추가합니다.

팁 문서 저장 버튼을 클릭하여 추가된 예제 VI 문서를 저장합니다. 예제 VI 문서를 저장한 후에, 13 단계까지 진행하거나 이 대화 상자를 닫고 나중에 예제 VI의 문서 작업을 끝마칠 수 있습니다. NI 예제 탐색기를 위한 예제 준비하기 대화 상자를 다시 시작하고 더블 클릭하여 예제를 문서로 리스트에서 예제 VI를 탐색하여 저장한 예제 VI 문서를 편집합니다.

6. 예제 VI에 대한 **설명을 입력**하지 않았다면, 예제 설명 텍스트 박스에 설명을 입력합니다. 사용자가 VI를 선택할 때 설명이 NI 예제 탐색기의 설명 텍스트 박스에 나타납니다. VI 프로퍼티 대화 상자의 문서 페이지에서 예제 VI에 대한 설명을 이미 **입력**한 경우, 이 설명은 예제 설명 텍스트 박스에 나타납니다.
7. 문서 저장 버튼을 클릭하여 문서를 예제 VI에 저장합니다. 이 버튼을 클릭하면 예제 설명 텍스트 박스나 VI 프로퍼티 대화 상자의 문서 페이지의 예제 VI에서 입력한 기존의 태스크 단위 탐색 경로, 키워드, 설명을 덮어씁니다.
8. (옵션) 또한 하나의 예제 VI로부터 다른 것으로 **문서를 복사**할 수 있습니다.
9. NI 예제 탐색기에 나타내려는 각 VI에서 1 단계부터 10 단계까지를 반복합니다.

노트 예제 VI가 프로젝트 기반 예제의 일부이고 그 예제 VI가 LabVIEW 프로젝트와 독립적으로 실행할 수 있는 경우, 프로젝트와 예제 VI 모두를 문서화할 수 있습니다. 프로젝트를 문서화하려면 더블 클릭하여 예제를 문서로 리스트에서 프로젝트 파일 .lvproj를 탐색하고 선택합니다. 위에 나열된 단계를 마칩니다.

10. 문서로 작성하는 예제 VI에 대한 태스크 단위 검색 경로, 키워드, 기타 정보를 저장하는 .bin3 파일을 만듭니다.

노트 생성한 예제 VI가 NI 예제 탐색기에 나타나려면 태스크 단위 검색 경로나 키워드를 추가해야 합니다. 작업 단위의 검색 경로만 추가하는 경우, 생성한 예제 VI는 NI 예제 탐색기의 탐색 페이지에만 나타납니다. 키워드만 추가하는 경우에는, 생성한 예제 VI는 NI 예제 탐색기의 검색 페이지에만 나타납니다.

1. 데이터 파일 만들기 탭을 클릭합니다.
2. 생성할 데이터 파일 (bin3) 텍스트 박스 옆의 탐색 버튼을 클릭하고, .bin3 파일을 저장할 디렉토리를 탐색합니다. LabVIEW가 .bin3 파일의 위치를 찾을 수 있게 하려면, labview 디렉토리 내의 다음 위치 중 한 곳에 파일을 저장해야 합니다.
 - examples\exbins
 - instr.lib
 - user.lib
3. .bin3 파일을 위해 의미있는 이름을 입력하며 파일 확장자를 포함합니다. 예를 들면, myexamples.bin3을 입력하는데, 이때 myexamples는 생성된 예제 VI를 가진 .lib의 이름입니다. 이름을 입력한 후 확인 버튼을 클릭합니다.

노트 LabVIEW는 다음 접두어를 예약하므로 .bin3 파일의 이름에 ms, test, lv, daq, cvi, tree, fr, de, ja, vi, im, dsc, rt, _, ko, zh, it, es 등의 접두어를 사용할 수 없습니다.

4. 예제를 포함한 디렉토리 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고, NI 예제 탐색기에 나타내려는 예제 VI가 포함된 디렉토리를 탐색한 후, 폴더 선택 버튼을 클릭합니다.
5. 데이터 파일을 만들기 위해 파일 만들기 버튼을 클릭합니다.
11. 완료 버튼을 클릭하여 NI 예제 탐색기를 위해 예제 VI 준비 대화를 종료합니다.

실행 프로퍼티 설정하기

LabVIEW는 VI를 사용자 정의하기 위해 사용할 수 있는 일부 실행 프로퍼티를 포함합니다.

다음의 단계를 따라 VI의 실행 프로퍼티를 결정합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. VI 프로퍼티 대화 상자의 상단에 위치한 항목 풀다운 메뉴에서 실행을 선택합니다.
3. 적절한 실행 프로퍼티를 선택합니다.

또한, 실행 프로퍼티를 사용하여 실행 프로퍼티를 프로그램적으로 설정할 수 있습니다.

윈도우 모양과 크기 설정하기

LabVIEW는 VI의 윈도우 모양, 크기, 위치를 사용자 정의할 수 있는 여러가지 옵션을 포함합니다. 이 옵션은 독립 어플리케이션으로 VI를 배포할 때 유용합니다.

다음 단계를 따라 윈도우의 모양과 크기를 사용자 정의합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 윈도우 모양, 윈도우 크기 또는 윈도우 런타임 위치를 선택합니다.
3. 적절한 윈도우 옵션을 선택합니다.
4. 윈도우 모양 페이지에서, 사용자 정의 버튼을 클릭하여 윈도우 모양 사용자 정의 대화 상자를 디스플레이합니다. 이 대화 상자에서, 제목 표시줄 또는 메뉴 모음 보이기, 윈도우의 최소화 또는 닫기, 런타임에 나타나는 바로 가기 메뉴 등 그 외 여러가지 옵션을 사용자 정의할 수 있습니다.



노트

실행될 때 도구 모음 보이기, 강제 종료 버튼 보이기, 사용자의 윈도우 닫기 허용 확인란에서 확인 표시를 제거하려는 경우, 사용자가 VI를 정지할 수 있도록 정지 버튼을 프론트패널에 추가하는 것을 고려하십시오.

또한, 프론트패널 윈도우와 도구 모음 프로퍼티, 프론트패널 메소드를 사용하여 프로그램적으로 윈도우 모양과 윈도우 크기를 사용자 정의할 수 있습니다. 또한 프론트패널:런타임 위치 메소드를 사용하여 런타임 프론트패널 윈도우 위치와 크기를 프로그램적으로 사용자 정의할 수 있습니다.

VI 실행하기

VI를 실행하면 VI를 디자인한 대로 작업을 수행합니다. 도구 모음의 실행 버튼이 하얀 화살표로 나타나면 VI를 실행할 수 있습니다. 또한, 하얀 화살표는 이 VI에 **커넥터 팬을 생성**하는 경우 이 VI를 subVI로 사용할 수 있다는 것을 나타냅니다.

파일>>VI 프로퍼티를 선택하고 항목 풀다운 메뉴에서 실행을 선택하여 VI의 실행 방법을 설정할 수 있습니다.

다음 단계를 따라 블록다이어그램에서 VI를 실행합니다.

1. 모든 subVI, 함수, 구조를 터미널이 올바른 데이터 타입과 연결합니다.
2. 실행 또는 연속 실행 버튼을 클릭하거나 블록다이어그램 도구 모음의 단계별 실행 버튼을 클릭합니다. 실행 버튼을 클릭하면 VI를 한 번 실행합니다. VI가 데이터 흐름을 완료하면 VI는 정지합니다. 연속 실행 버튼을 클릭하면 수동으로 정지할 때까지 VI를 연속적으로 실행합니다. **단계별 실행** 버튼을 클릭하면 단계별로 VI를 실행합니다.



노트

실행 강제 종료 버튼은 VI가 현재 반복을 끝내기 전에 VI를 즉시 정지시킵니다. 외부 하드웨어와 같은 외부 리소스를 사용하는 VI를 강제 종료하는 경우, 해당 외부 리소스를 적절히 리셋 혹은 해제하지 못하여 리소스가 알지 못하는 상태가 될 수 있습니다. VI에 정지 버튼을 만들면 이러한 문제를 피할 수 있습니다.

다음과 같은 방법으로 블록다이어그램 밖에서 VI를 실행할 수도 있습니다.

- **프로젝트 탐색기** 윈도우에서 VI를 마우스 오른쪽 버튼으로 클릭하고 바로 가기 메뉴에서 실행을 선택하여 **LabVIEW 프로젝트**에서 VI를 실행합니다. VI 프런트패널이 실행 모드에서 열립니다.
- [VI 실행] 메소드를 사용하여 VI를 프로그램적으로 실행합니다.
- **RunVI** 명령 라인 작업을 사용하여 VI를 프로그램식으로 실행합니다.

명령 라인에서 VI 실행하기

명령 라인(Windows에서 시작>> 실행을 선택하는 것과 같이)에서 VI를 실행하기 위해서는 다음 구문을 사용합니다:

c:\labview\labview.exe <labview 디렉토리에 상대적인 VI에 대한 경로>



노트

명령 라인에서 VI를 시작하기 전에 반드시 LabVIEW를 닫아야 합니다.

예를 들어, c:\labview\examples에 있는 readme.vi를 실행하기 위해서는 다음 명령을 사용합니다:

```
c:\labview\labview.exe examples\readme.vi
```

VI가 labview 디렉토리가 아닌 디렉토리에 위치한 경우, 다음 예제에서 보이는 것처럼 사용자가 VI에 대한 전체 경로를 지정해야 합니다:

```
c:\labview\labview.exe c:\coolapp\mycool.vi
```

경로의 디렉토리에 스페이스가 포함된 경우, 다음 예제와 같이 경로를 따옴표로 에워싸야만 합니다.

```
c:\labview\labview.exe "c:\cool application\mycool.vi"
```

LLB에 VI가 있을 경우, 다음 방법 중 하나를 사용합니다:

- 다음 예제와 같이 따옴표에 전체 경로를 넣습니다:

```
c:\labview\labview.exe "c:\coolapp\eagle.llb\mycool.vi"
```
- 도구 » LLB 관리자를 선택하고 **VI를 최상위 VI로 표시**합니다. 이 경우, LLB에 대한 경로만을 지정하면 됩니다. LabVIEW는 다음 예제에서처럼 LLB 관리자 윈도우를 엽니다:

```
c:\labview\labview.exe "c:\coolapp\eagle.llb"
```

또한, 명령 라인에서 LabVIEW를 실행할 때 **LabVIEW 정의** 인수 또는 사용자 정의 인수를 전달할 수 있습니다. 명령 라인에서 사용자 정의 인수를 전달하기 위해서는 사용자 정의 인수의 세트 이전에 빈 스페이스 사이에 두 개의 하이픈 --을 입력합니다. LabVIEW는 labview.exe를 실행할 때 두 하이픈 이후의 어떠한 인수도 사용하지 않습니다. LabVIEW는 두 하이픈 이후의 인수를 실행하는 VI의 블록다이어그램에 전달합니다. 어플리케이션:명령 라인 인수 프로퍼티를 사용하여 LabVIEW가 실행할 때 전달되는 사용자 정의 명령 라인 인수를 읽습니다.

예를 들어, mycool.vi를 실행하고 사용자 정의 인수를 전달하기 위해서는 다음 명령을 사용합니다:

```
c:\labview\labview.exe c:\coolapp\mycool.vi -- 1000 sine
```

mycool.vi의 블록다이어그램에서, 어플리케이션:명령 라인 인수 프로퍼티를 사용하여 전달한 1000과 sine 인수를 읽고 값을 처리합니다. 예를 들어, 인수 값을 **케이스 구조**의 케이스 선택자에 연결할 수 있습니다.

LabVIEW에서 정의된 명령 라인 인수

VI를 명령 라인에서 시작할 때, 다음과 같이 LabVIEW에서 정의된 인수를 사용할 수 있습니다. LabVIEW에서 정의된 인수는 labview.exe 또는 독립 어플리케이션인 실행 프로그램에 적용됩니다. 명령 라인이 **사용자 정의된 인수**의 시작을 나타내는 두 하이픈 -- 앞에 다른 문자열을 포함하는 경우, 실행 프로그램은 그 문자열을 열려고 하는 VI의 파일 이름으로 해석합니다.

플랫폼	인수	설명
모두	-pref 경로	환경 파일 이 시작하려는 실행 프로그램과 같은 디렉토리에 있지 않은 경우, 이 개요를 사용하여 환경 파일의 전체 경로를 지정합니다.
Windows	/Automation 또는 -Automation	실행 프로그램의 객체 인스턴스를 요청하는 ActiveX 클라이언트에 응답하여 실행 프로그램을 시작합니다.
Windows	/RegServer 또는 -RegServer	OS에 실행 프로그램의 ActiveX 서버를 등록합니다.
Windows	/UnregServer 또는 -UnregServer	OS로부터 실행 프로그램의 ActiveX 서버를 등록 해제합니다.
Linux	-display 디스플레이	LabVIEW를 주어진 X Windows 디스플레이에서 실행합니다. 예를 들어, <code>labview -display myhost:0</code> 입니다.
Linux	-launch	LabVIEW가 시작되고 같은 디스플레이에서 이미 실행되고 있는 기존의 어플리케이션 인스턴스 가 있는지 확인합니다. 기존 인스턴스가 실행 중인 경우, 기존의 인스턴스가 명령 라인에 전달된 VI를 열고 새 어플리케이션 인스턴스는 종료됩니다. 기존 인스턴스가 실행 중이 아닌 경우, LabVIEW는 명령 라인에 전달된 VI를 새 인스턴스에서 엽니다.

개정 히스토리에 주석 추가하기

다음 단계를 따라 히스토리 윈도우에 주석을 추가합니다.

1. 편집»VI 개정 히스토리를 선택하여 히스토리 윈도우를 디스플레이합니다.
2. 윈도우 상단 가까이에 있는 주석 텍스트 박스에 VI 변경에 대한 설명을 입력합니다.
3. VI의 히스토리에 직접 주석을 추가하기 위해서 추가 버튼을 클릭합니다. VI를 저장할 때 주석 텍스트 박스가 주석을 포함하는 경우, LabVIEW는 자동으로 주석을 히스토리에 추가합니다. 주석의 헤더는 다음 개정 번호를 포함합니다. 주석을 추가한 이후에 주석은 히스토리의 영구적인 부분이 됩니다. VI 히스토리에 저장된 주석은 편집할 수 없습니다.

VI를 변경할 때마다 주석 텍스트 박스에 정보를 입력할 수 있도록 VI를 편집하는 동안 히스토리 윈도우를 열어두십시오. 주석을 입력하지 않더라도 VI를 저장할 때마다 개정 히

스토리에 엔트리를 포함시키려면, 히스토리 윈도우에서 히스토리를 선택한 다음 빈 엔트리 보이기에 확인 표시가 되어 있는지 확인하십시오.



노트

기본적으로, LabVIEW는 자동적으로 히스토리 주석을 생성하지 않습니다. 히스토리를 최신으로 유지하기 위해서는 VI를 변경할 때 히스토리 윈도우에 변경에 대한 설명을 반드시 입력해야 합니다. VI 프로퍼티 대화 상자의 개정 히스토리 페이지에서 LabVIEW가 생성한 주석 기록 확인란에 확인 표시를 하면, LabVIEW가 생성한 주석을 **현재 VI의 히스토리**에 기록하도록 설정할 수 있습니다. 또한 옵션 대화 상자의 개정 히스토리 페이지에서 LabVIEW가 생성한 주석 기록 확인란에 확인 표시를 하면, LabVIEW가 생성한 주석을 모든 새 VI의 히스토리에 기록할 수 있도록 설정할 수 있습니다.

VI 개정 히스토리와 개정 번호 리셋하기

다음 단계를 따라 VI 개정 히스토리와 개정 번호를 리셋합니다.

1. 편집>>VI 개정 히스토리를 선택하여 히스토리 윈도우를 디스플레이합니다.
2. 리셋 버튼을 클릭합니다. VI에 개정 히스토리가 없는 경우, 리셋 버튼이 비활성화됩니다.
3. 대화 상자가 나타나면 확인 버튼을 클릭하여, VI 개정 히스토리를 삭제하고 개정 번호를 리셋합니다.



노트

개정 번호를 리셋하지 않고 VI 개정 히스토리를 삭제하려면 개정 번호 리셋 포함 확인란에서 확인 표시를 제거합니다.

또한 히스토리 삭제 메소드를 사용하여 프로그램적으로 VI의 개정 히스토리를 삭제할 수 있습니다.

VI 개정 히스토리 보기

편집>>VI 개정 히스토리를 선택하여 히스토리 윈도우를 디스플레이합니다. 히스토리 윈도우는 히스토리의 각 주석의 헤더를 보여줍니다. 헤더에는 변경의 개정 번호, 개정 날짜와 시간, 사용자 이름이 포함됩니다.

히스토리 윈도우의 크기를 줄이려면, 히스토리 화살표(작은 검정 삼각형)을 클릭합니다. 히스토리 윈도우의 크기를 확장하려면, 같은 화살표를 클릭합니다.

히스토리 윈도우의 크기를 조정하여 히스토리와 주석 텍스트 박스의 크기를 변경합니다.

VI 디버깅하기

디버깅 도구 비활성화하기

메모리 사용량을 줄이고 시스템 성능을 조금 더 향상시키기 위하여 디버깅 도구를 비활성화할 수 있습니다.

다음 단계를 따라 VI에서 디버깅을 비활성화합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 실행을 선택합니다.
3. 디버깅 허용 확인란에서 확인 표시를 제거합니다.
4. 자동 에러 핸들링 허용 확인란에서 확인 표시를 제거합니다.

또한, 디버깅 허용 프로퍼티를 사용하여 프로그램적으로 디버깅 도구를 비활성화하고 자동 에러 핸들링 프로퍼티를 사용하여 프로그램적으로 자동 에러 핸들링을 비활성화할 수 있습니다.

다음 단계를 따라 새로 생성하거나 새 VI에서 자동 에러 핸들링을 비활성화합니다.

1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 블록다이어그램을 선택합니다.
3. 새 VI에서 자동 에러 핸들링 활성화 확인란에서 확인 표시를 제거합니다.

블록다이어그램의 섹션 비활성화하기

텍스트 기반 프로그래밍 언어에서 코드의 섹션에 주석을 다는 것과 비슷하게 블록다이어그램의 섹션을 비활성화한 VI를 실행할 수 있습니다.

다음 단계를 따라 블록다이어그램의 섹션이 비활성화된 VI를 실행합니다.

1. 비활성화하려는 노드 둘레에 다이어그램 비활성화 구조를 놓습니다. 기본 서브다이어그램은 비활성화입니다.
2. 활성화 서브다이어그램에 비활성화 서브다이어그램의 코드를 대체하여 활성화하려는 모든 코드를 포함시킵니다.
3. VI를 실행합니다. 활성화 서브다이어그램에 있는 유일한 코드는 실행(run)입니다.

비활성화한 블록다이어그램의 섹션을 실행하려면, 비활성화 서브다이어그램을 선택하고 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다이어그램 비활성화 구조 제거를 선택합니다.



노트

다이어그램 비활성화 구조 제거를 선택하면, LabVIEW는 보이는 서브다이어그램을 제외한 모든 서브다이어그램의 객체를 삭제합니다.

다이어그램 비활성화 구조 사용에 대한 예제는 labview\examples\Structures\Disable Structures 디렉토리에 있는 Diagram Disable Structure VI를 참조하십시오.



예제 열기



관련 예제 찾기

경고 디스플레이하기

경고는 VI의 실행을 막지는 않습니다. VI의 잠재적인 문제점을 피하는데 도움을 주도록 디자인되었습니다.

다음 단계를 따라 에러 리스트 윈도우에 경고를 디스플레이합니다.

1. 보기>>에러 리스트를 선택하여 에러 리스트 윈도우를 디스플레이합니다.
2. 경고 보이기 확인란에 확인 표시를 하여 에러가 있는 아이템 섹션에 각 VI의 모든 경고를 디스플레이합니다.
3. 도구 모음에서 경고 버튼을 클릭하여 에러 리스트 윈도우에 접근합니다. 경고 버튼은 VI가 경고를 가지고 있을 때에만 나타납니다. 에러 리스트 윈도우를 설정하여 경고를 나타낸 후, 도구 모음에 경고 버튼이 나타나면 언제나 에러 리스트 윈도우에 접근할 수 있습니다.

다음 단계를 따라 LabVIEW가 언제나 에러 리스트 윈도우에 경고를 디스플레이하도록 설정합니다.

1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 환경을 선택합니다.
3. 이 페이지의 디버깅 섹션에 있는 기본적으로 에러 리스트 대화 상자에 경고 보이기 확인란에 확인 표시를 하고 확인 버튼을 클릭합니다. 에러 리스트 윈도우가 열린 상태에서 변경할 수 있으며 변경을 즉시 확인할 수 있습니다.

실행 하이라이트

실행 하이라이트를 사용하여 블록다이어그램의 실행 애니메이션을 봅니다. 실행 하이라이트는 블록다이어그램의 한 노드에서 다른 노드까지의 데이터 이동을 와이어를 따

라 움직이는 방울을 사용하여 보여줍니다. 실행 하이라이트를 **단계별 실행**과 함께 사용하여 VI 내의 한 노드에서 다른 노드로 데이터가 어떻게 움직이는지 확인합니다.



노트

실행 하이라이트는 VI의 실행 속도를 현저히 저하시킵니다.

다음 단계를 따라 실행 하이라이트를 사용합니다.

1. VI의 블록다이어그램을 엽니다.
2. 블록다이어그램 도구 모음의 실행 하이라이트 버튼을 클릭하여 실행 하이라이트를 활성화합니다.
3. VI를 실행한 후 VI가 실행될 때 블록다이어그램을 확인합니다. 실행 하이라이트에서, 와이어를 따라 움직이는 방울은 한 노드에서 다른 노드로의 데이터의 이동을 나타내며 각 터미널에서의 값이 디스플레이됩니다.
4. VI를 정지합니다.
5. 실행 하이라이트 버튼을 다시 클릭하여 언제나 실행 하이라이트를 비활성화할 수 있습니다.

VI의 단계별 실행

VI를 단계별로 실행하여 블록다이어그램에서 VI의 각 동작을 확인할 수 있습니다. 단계별 실행 버튼은 단계별 실행 모드에서 VI나 subVI에서의 실행에만 영향을 줍니다. SubVI를 단계별로 실행하거나 또는 일반적인 방식으로 실행할 수 있습니다.

다음 단계를 따라 VI에서 단계별로 실행합니다.

1. 블록다이어그램 도구 모음에서 단계별 실행 시작하기 버튼을 클릭합니다.
2. 다음 버튼 중에 하나를 클릭합니다:



단계별 실행 들어가기—노드를 열고 일시 정지합니다. 단계별 실행 들어가기 버튼을 다시 클릭하면 첫 번째 작업을 실행하고 SubVI나 구조의 다음 작업에서 일시 정지합니다. 또한, <Ctrl> 키와 아래 방향 화살표 키를 누를 수 있습니다. (macOS) <Command> 키와 아래 방향 화살표 키를 누릅니다. (Linux) <Alt> 키와 아래 방향 화살표 키를 누릅니다.



단계별 실행 건너뛰기—노드를 실행하고 다음 노드에서 일시 정지합니다. 또한, <Ctrl> 키와 오른쪽 방향 화살표 키를 누를 수 있습니다. (macOS) <Command> 키와 오른쪽 방향 화살표 키를 누릅니다. (Linux) <Alt> 키와 오른쪽 방향 화살표 키를 누릅니다.



단계별 실행 나가기—현재 노드의 실행을 끝내고 일시 정지합니다. VI가 실행을 끝내면, 단계별 실행 나가기 버튼은 비활성화됩니다. 또한, <Ctrl> 키와 위 방향 화살표 키를 누를 수 있습니다. (macOS) <Command> 키와 위쪽 방향 화살표 키를 누릅니다. (Linux) <Alt> 키와 위 방향 화살표 키를 누릅니다.

VI를 단계별로 실행하면, 노드가 깜박이면서 실행할 준비가 되었음을 나타냅니다.

단계별 실행 건너뛰기, 단계별 실행 들어가기 또는 단계별 실행 나가기 버튼 위로 커서를 이동하여, 해당 버튼을 클릭하면 실행되는 다음 단계에 대한 설명을 팁 상자에서 확인합니다. **실행 하이라이트**를 활성화한 상태에서 VI에서 단계별 실행을 하면, 실행 문양이 현재 실행 중인 SubVI의 아이콘에 나타납니다.

3. 깜빡거리는 경계선이 전체 블록다이어그램을 에워쌀 때 단계별 실행 나가기 버튼을 클릭하여 단계별 실행을 끝냅니다. 단계별 실행이 더 이상 동작하지 않으면 단계별 실행 버튼은 비활성화됩니다.

실행 일시 정지하기

컨트롤과 인디케이터의 값을 바꾸거나, 호출자로 돌아가기 전에 subVI의 실행 횟수를 조절하거나, subVI 실행을 처음으로 돌리기 위해 subVI의 실행을 일시 정지합니다. SubVI의 모든 호출이 실행이 일시 정지된 상태로 시작하게 하거나 SubVI의 특정한 호출을 일시 정지할 수 있습니다.

다음 단계를 따라 SubVI에 대한 모든 호출을 일시 정지합니다.

1. SubVI를 엽니다.
2. 수행>>호출 시 정지를 선택합니다.

다른 VI가 그 SubVI를 호출하면 SubVI는 자동적으로 일시 정지합니다. **단계별 실행**에서 이 메뉴 항목을 선택할 경우, subVI는 바로 일시 정지하지 않습니다. SubVI는 호출될 때 일시 정지합니다.

보기>>VI 계층구조를 선택하면, 호출될 때 정지하도록 설정된 VI가 녹색 느낌표 !와 함께 나타납니다. 일시 정지된 VI가 빨간 느낌표와 함께 나타납니다.

다음 단계를 따라 특정한 SubVI의 호출에서 실행을 일시 정지합니다.

1. SubVI 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 SubVI 노드 설정을 선택합니다. SubVI 노드 설정 대화 상자가 나타납니다.
2. 호출 시 정지 확인란에 확인 표시를 하고 확인을 클릭합니다.

호출 시 정지 프로퍼티를 사용하여 VI를 프로그램적으로 정지할 수 있습니다.

SubVI의 실행을 일시 정지한 후에, 다음 태스크를 수행할 수 있습니다:

- SubVI 프런트패널의 왼쪽 상단의 호출자로 돌아가기 버튼을 클릭하여 호출자 VI로 돌아갑니다. 호출자로 돌아가기 버튼은 일시 정지된 SubVI가 실행되지 않을 때 나타납니다. 현재 VI를 실행하지 않고 호출자 VI로 돌아갈 수 있습니다. 현재 VI를 실행하려면, 호출자로 되돌아가기 전에 실행 버튼을 클릭합니다.
- 호출자 VI로 되돌아가기 전에 원하는 횟수만큼 SubVI를 실행합니다. 실행 버튼을 클릭하거나 수행>>실행을 선택합니다. SubVI가 실행 중일 때 SubVI의 실행 시작 부분으로 되돌아가려면, SubVI 블록다이어그램의 왼쪽 위에 있는 시작으로 건너뛰기 버튼을 클릭합니다.
- 호출자 VI에 다른 값을 반환하려면 컨트롤이나 인디케이터 이름을 변경합니다. 일시 정지된 VI가 실행되지 않을 때만 인디케이터의 값을 설정할 수 있습니다. SubVI

프론트패널의 왼쪽 위의 호출자로 돌아가기 버튼을 클릭하여, SubVI의 인디케이터 값을 호출자 VI에 보냅니다.

메모리 사용

VI 실행 시간과 메모리 사용 프로파일

성능과 메모리 프로파일 윈도우는 어플리케이션이 실행 시간과 메모리를 어떻게 사용하는지 **분석**할 때 사용할 수 있는 강력한 도구입니다. 이 정보와 함께, 최적화하는데 필요한 특정한 VI나 VI의 일부를 나타낼 수 있습니다. 예를 들면, 특정한 SubVI가 실행하는데 오랜 시간이 걸린다면, **해당 VI의 성능을 향상**시키는데 초점을 맞출 수 있습니다.

다음 단계를 따라 VI를 프로파일합니다.

1. 실행 중인 VI를 모두 정지합니다. 실행 중인 VI를 정지하지 않으면 성능과 메모리 프로파일 윈도우가 디스플레이한 결과가 잘못되거나 일관되지 않을 수 있습니다.
2. 도구 >> 프로파일 >> 성능과 메모리를 선택하여 성능과 메모리 프로파일 윈도우를 디스플레이합니다.
3. 메모리 사용 정보를 모으려면, 메모리 사용 프로파일 확인란에 확인 표시를 합니다. 프로파일 과정을 시작한 후에는 이 확인란에 확인 표시를 할 수 없습니다. VI 메모리 사용에 대한 정보를 수집하면 VI 실행에 상당한 양의 오버헤드가 추가됩니다. 그리고 이것은 프로파일 과정 중에 수집하는 모든 타이밍 통계의 정확성에 영향을 줍니다. 그러므로, 메모리 프로파일 작업을 시간 프로파일 작업과 분리해서 수행해야 합니다.
4. 성능과 메모리 프로파일 윈도우에서 시작 버튼을 클릭하여 성능 데이터의 수집을 시작합니다.
5. 프로파일하려는 VI를 실행합니다.
6. VI를 완료할 때까지 실행하거나, 프론트패널 정지 버튼을 클릭하여 계속 실행 중인 VI를 정지시킵니다. 정지 버튼이 없는 경우, 실행 강제 종료 버튼을 클릭합니다.
7. 성능과 메모리 프로파일 윈도우에서 정지 버튼을 클릭하여 프로파일 과정을 끝냅니다. 테이블 디스플레이는 상위 레벨 VI가 실행하는데 걸리는 시간, 모든 subVI가 실행하는데 걸리는 시간, 이 프로파일 과정 동안 실행하는 VI와 모든 subVI가 걸리는 전체 시간을 포함합니다.



노트

성능과 메모리 프로파일 윈도우는 CPU 사용 시간만을 측정합니다. 기다림 함수가 실행하는 동안, CPU는 OS에서 자유롭게 다른 일을 처리하지만, 그러나 성능과 메모리 프로파일 윈도우는 CPU가 기다림으로부터 시작해서 반환하는데 걸리는 작은 양의 시간을 여전히 측정합니다. 성능과 메모리 프로파일 윈도우는 다른 노드를 위한 CPU 사용 시간과 분리해서 참조에 의한 호출이나 프로퍼티 노드에서 컨트롤 참조를 사용하는 CPU 사용 시간은 측정하지 않습니다.

1. 타이밍 통계 확인란에 확인 표시를 놓으면, 테이블 디스플레이는 VI 실행 시간에 대해 더 많은 통계를 포함합니다. 타이밍 세부사항 확인란에 확인 표시를 놓으면, 테이블 디스플레이는 어떤 작동이 가장 많은 시간이 걸릴 것인지 결정하도록 도울 수 있는 몇가지 타이밍 항목 포함합니다. 프로파일 과정을 시작하기 전에 메모

리 사용 확인란에 확인 표시를 하면(이 확인란은 메모리 사용 프로파일 확인란에 확인 표시를 한 경우에만 사용 가능), VI가 어떻게 메모리를 사용하는지에 대한 정보를 확인할 수 있습니다.

성능과 메모리 프로파일 윈도우 외에, LabVIEW VI Analyzer Toolkit을 사용해서도 VI를 대화식으로 또는 프로그램적으로 테스트 및 확인하여 성능, 효율성, 기타 LabVIEW 프로그래밍 측면에 대한 정보를 얻을 수 있습니다.

또한 LabVIEW Desktop Execution Trace Toolkit 을 이용하여 코드 분석과 메모리 손실, 기타 LabVIEW 프로그래밍의 요소를 모니터할 수 있습니다.

LabVIEW 어플리케이션에서 버퍼 할당 확인하기

버퍼 할당 프로파일 윈도우를 사용하여 LabVIEW 어플리케이션에서 버퍼 할당을 확인 및 분석합니다.

큰 버퍼 식별하기

어플리케이션이 할당하는 큰 버퍼를 식별하려면, 시작을 클릭하여 프로파일 작업을 시작하고, 프로파일하려는 VI를 한 개 또는 여러 개 실행한 후, 정지를 클릭하여 프로파일 과정을 끝냅니다. 프로파일 데이터 테이블은 가장 큰 버퍼 할당을 디스플레이합니다. 테이블에 있는 버퍼 할당의 개수는 표시할 버퍼의 최대 개수보다 크지 않습니다.

버퍼 할당 분석하기

버퍼 할당의 원인을 분석하려면, 프로파일 데이터 테이블에서 행을 더블 클릭하여 버퍼 할당을 발생시키는 VI의 블록 다이어그램을 엽니다. LabVIEW가 버퍼 할당을 발생시키는 노드, 와이어 또는 터널을 하이라이트합니다.

프로파일 저장 및 로드하기

분석을 위해 프로파일을 저장하려면, 파일>>프로파일 저장을 선택하여 프로파일 데이터를 2진 파일로 디스크에 저장합니다. 저장한 프로파일을 이 윈도우로 로드하려면, 파일>>프로파일 로드를 선택합니다. 또한 파일>>테이블 반출을 선택하여 프로파일 데이터를 CSV 파일로 반출할 수 있습니다. 이 CSV 파일은 Microsoft Excel 또는 메모장에 로드할 수 있습니다.

어플리케이션의 실행 속도 최적화하기

VI 또는 어플리케이션의 개발 및 테스트를 완료한 후, LabVIEW 환경과 빌드 스펙에서 어플리케이션의 실행 속도를 최대화하도록 특정 셋팅을 설정할 수 있습니다. 이러한 설정은 편집과 디버깅을 지연하거나, 막을 수 있기 때문에 개발이 완료된 후에 설정해야 합니다.

다음 과정을 따라 LabVIEW가 어플리케이션을 만드는 과정에서 전체 컴파일러 최적화를 사용하도록 하여 실행 속도를 향상합니다.

1. 빌드 스펙을 엽니다. 빌드 스펙에서 마우스 오른쪽 버튼을 클릭하여 프로퍼티 대화 상자를 디스플레이합니다.
2. 빌드 스펙의 디버깅 활성화 옵션에 확인 표시가 되어 있지 않아야 합니다. 대화 상자의 고급 페이지에서, 디버깅 활성화 확인란의 확인 표시를 제거합니다. 이 글로벌 디버깅 설정을 사용하여 LabVIEW가 빌드 스펙의 각 VI에서 디버깅 설정을 평가할 수 있습니다. 그러므로, 빌드 스펙의 각 VI에서 나중에 디버깅을 비활성화하려면, 글로벌 설정에서 디버깅을 활성화해서는 안 됩니다.
3. 빌드 스펙에서 각 VI의 디버깅을 비활성화합니다. 각 VI의 디버깅을 비활성화하면 어플리케이션의 실행 속도가 향상됩니다. 또한, 디버깅 설정으로 인해 LabVIEW가 VI를 다시 컴파일해야 하는 경우도 종종 발생합니다. VI를 다시 컴파일해야만 실행 속도에 영향을 미치는 업데이트된 컴파일러 설정이 적용됩니다. 이 과정의 나머지 부분에서는 이러한 설정을 확인하고 다시 컴파일하도록 하는 방법을 다룹니다.
 1. 대화 상자의 소스 파일 설정 페이지에서, 각 VI를 선택하고 VI 프로퍼티 사용자 정의 버튼을 클릭합니다.
 2. 디버깅 허용 옆의 VI 프로퍼티 사용자 확인란에서 확인 표시를 제거합니다.
 3. 디버깅 허용 확인란에서 확인 표시를 제거합니다.
4. 빌드 스펙에서 의존성의 디버깅을 비활성화합니다. 의존성의 디버깅을 비활성화하면 의존성 VI가 빌드 스펙의 다른 VI와 같은 실행 속도 및 재컴파일 효과의 혜택을 얻을 수 있습니다.
 1. 대화 상자의 소스 파일 설정 페이지에서, 의존성을 선택하고 VI 프로퍼티 사용자 정의 버튼을 클릭합니다. VI 프로퍼티 사용자 정의 버튼이 비활성화된 경우, 포함되는 모든 아이템의 VI 프로퍼티 설정 확인란에 확인 표시를 하여 활성화합니다.
 2. 디버깅 허용 옆의 VI 프로퍼티 사용자 확인란에서 확인 표시를 제거합니다.
 3. 디버깅 허용 확인란에서 확인 표시를 제거합니다.
5. 모든 소스 VI가 디버깅을 허용하도록 설정되어 있는지 확인하십시오. 소스 VI에서 디버깅을 할 수 있도록 허용하여, 단계 3과 단계 4의 VI 설정과 빌드 스펙 설정을 의도적으로 다르게 설정할 수 있습니다. 이러한 차이점으로 인해 LabVIEW는 사용자가 어플리케이션을 만들 때 VI를 다시 컴파일하여 이후 단계에서 변경사항을 복사도 임계점에 적용합니다.



노트

개별 VI에서 디버깅 설정을 변경한 경우에만 다음 단계를 완료합니다. LabVIEW에서는 기본적으로 VI의 디버깅이 활성화되어 있습니다.

1. 빌드 스펙과 의존성 리스트에서 각 VI를 엽니다.
2. VI 프로퍼티 대화 상자의 실행 페이지에 있는 디버깅 활성화 확인란에 확인 표시를 합니다.
6. 모든 추가적인 예외를 사용하도록 빌드 스펙을 설정합니다. 추가적인 예외를 사용하여 어플리케이션의 로드 시간 및 실행 속도를 향상시킬 수 있습니다. 빌드 스펙의 프로퍼티 대화 상자에 있는 추가적인 예외 페이지에서, 모든 확인란에 확인 표시를 합니다.
7. 빌드 이전 VI를 사용하여 어플리케이션을 만들기 전에 컴파일러 최적화 임계점을 설정합니다. 빌드 이전 VI에서 프로그램적으로 컴파일러 최적화 임계점을 설정하

면, LabVIEW가 빌드 과정에서 VI를 다시 컴파일할 때 변경사항이 그대로 남아있습니다.

1. 빌드 스펙의 프로퍼티 대화 상자에 있는 빌드 이전/이후 작업 페이지에서, VI 생성 버튼을 클릭하여 적절한 커넥터 팬이 있는 새 빌드 이전 VI를 생성합니다.
2. 빌드 스펙의 프로퍼티 대화 상자를 닫고, 빌드 이전 VI를 엽니다.
3. 빌드 이전 VI에서 어플리케이션:컴파일러 임계점 프로퍼티를 프로그램적으로 10으로 설정하십시오. 어플리케이션:컴파일러 임계점 프로퍼티를 사용하려면 **VI 스크립팅을 활성화**해야 합니다.
8. 빌드 이후 VI를 사용하여 어플리케이션을 만든 후, 컴파일러 최적화 임계점을 5로 복구합니다. 빌드 이후 VI에서 프로그램적으로 컴파일러 최적화 임계점을 복구하면, 개발하는 모든 VI는 편집기 응답 및 VI 실행 속도 중 어디에 우선 순위를 둘지 올바르게 결정합니다.
 1. 빌드 스펙의 프로퍼티 대화 상자에 있는 빌드 이전/이후 작업 페이지에서, VI 생성 버튼을 클릭하여 적절한 커넥터 팬이 있는 새 빌드 이후 VI를 생성합니다.
 2. 빌드 스펙의 프로퍼티 대화 상자를 닫고, 빌드 이후 VI를 엽니다.
 3. 빌드 이후 VI에서 어플리케이션:컴파일러 임계점 프로퍼티를 프로그램적으로 5로 설정하십시오.
9. 어플리케이션을 만듭니다. 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 빌드를 선택합니다.

노트 어플리케이션을 만드는 동안 LabVIEW가 메모리를 소진하게 되면 빌드 이전 VI가 컴파일러 최적화 임계점 10 대신 9를 사용하도록 설정합니다. LabVIEW의 메모리가 부족하지 않을 때까지 계속하여 임계점을 낮춥니다.

편집기 응답과 VI 실행 속도 중 선택하기

VI의 복잡도가 임계점을 넘어서는 경우, LabVIEW는 자동으로 **컴파일러 최적화를 제한**하여 상당한 지연 없이 큰 규모의 VI를 편집할 수 있도록 합니다. 최적화를 제한하면, 크기가 큰 컴파일된 VI가 최대 실행 속도로 실행되지 못하는 단점이 발생할 수 있습니다. LabVIEW가 최적화를 제한하는 지점인 복잡도 임계점을 조절하여, LabVIEW가 언제 편집기 응답을 실행 속도보다 우선할지 사용자가 결정할 수 있습니다.



팁

큰 VI를 여러 SubVI로 나누면 이러한 단점이 발생하는 것을 방지할 수 있습니다. 이렇게하면 VI의 복잡도가 감소되어 편집기 응답, 실행 속도 및 코드 가독성이 최적화됩니다.

큰 VI를 여러 SubVI로 나눌 수 없는 경우, 다음과 같은 상황에서 복잡도 임계점의 조절을 고려해 볼 수 있습니다:

- **LabVIEW가 VI를 컴파일할 때** VI에 상당한 지연이 발생합니다.
- VI를 컴파일 할 때 LabVIEW의 메모리가 소진됩니다.
- 크거나 복잡한 VI의 개발을 완료한 후, 실행 속도를 최적화하려고 합니다.

실행 단계

다음 단계를 완료하여 편집기 응답과 VI 실행 속도 중 하나를 선택합니다.

1. 편집기 응답 또는 실행 속도를 변경하려는 VI를 찾습니다. 복잡도 임계점은 모든 VI에 적용되는 환경 셋팅이지만 개별 VI의 복잡도를 사용하여 현재의 복잡도가 이 VI의 목적을 달성하는지 결정할 수 있습니다.
2. VI의 복잡도를 알아냅니다. VI 프로퍼티 대화 상자의 메모리 사용 페이지에서 컴파일된 코드 복잡도 값을 참조하십시오.
3. VI의 복잡도를 컴파일러 최적화에서의 임계점과 비교합니다. 옵션 대화 상자의 환경 페이지에서 컴파일러 최적화 슬라이더를 참조하십시오. 컴파일러는 VI 복잡도가 슬라이더에 나타난 값 위로 올라가면 컴파일러가 최적화를 제한하여 편집기 응답을 향상시킵니다. VI의 복잡도가 슬라이더의 값보다 낮은 경우, 컴파일러는 VI에 최적화를 적용합니다.
4. 목적에 맞도록 복잡도 임계점을 조절합니다.
 - 편집기 응답에 우선순위를 두거나 메모리 부족 에러를 방지하려면, 슬라이더를 VI 복잡도 아래로 낮춥니다.
 - 실행 속도에 우선순위를 두려면 슬라이더를 VI 복잡도 위로 올립니다.
5. VI를 다시 컴파일하여 변경 사항을 적용합니다. (Windows) 실행 버튼을 클릭하는 동안 <Ctrl>을 누릅니다. (macOS) 실행 버튼을 클릭하는 동안 <Command>를 누릅니다. (Linux) 실행 버튼을 클릭하는 동안 <Alt>를 누릅니다.
6. VI의 개발 또는 배포를 완료한 후, 복잡도 임계점을 5로 변경합니다. 복잡도 임계점은 모든 VI에 적용되기 때문에 National Instruments는 임계점의 기본값인 5를 사용할 것을 권장합니다. 이 값을 사용하면 LabVIEW는 대부분의 VI에서 편집기 응답과 실행 속도 사이에서 올바르게 우선순위를 선택합니다.

팁 및 문제 해결 정보

- 편집기 응답과 실행 속도 중 하나를 선택하여 다른 하나를 희생하게되는 결점을 방지하려면, 코드의 섹션을 SubVI로 옮겨 큰 VI의 복잡도를 감소합니다. 이렇게하면 편집기 응답, 실행 속도와 코드 가독성이 동시에 최적화됩니다.
- 프로젝트에서 어떤 VI가 긴 빌드 시간 또는 메모리 부족 에러를 발생하는지 알 수 없는 경우, 문제가 향상될 때까지 컴파일러 최적화 슬라이더를 한단계씩 낮춥니다.
- 복잡도 임계점은 모든 VI에 적용됩니다. 그러므로, 임계점을 기본값이 아닌 값으로 설정한 채로 두는 경우, 다른 VI의 편집기 응답 또는 실행 속도에 예상하지 못한 변경 사항이 생길 수 있습니다.
- PowerPC 또는 ARM 구조를 실행하는 타겟에 대해 어플리케이션을 만들 때, 컴파일러는 복잡도 임계점을 무시합니다. (각각 VxWorks 및 NI Linux Real-Time targets). 이러한 타겟에서는 VI가 특정 컴파일러 최적화를 사용해야 합니다. 그러므로 VI 복잡도 아래로 복잡도 임계점을 낮추면 LabVIEW가 어플리케이션을 더 빨리 만들 수 없고, 임계점을 VI 복잡도 위로 올리게되면 실행 속도가 향상되지 못할 수도 있습니다.
- 컴파일:코드 복잡도 프로퍼티, 컴파일:최종 컴파일 범위 프로퍼티, 어플리케이션:컴파일러 임계점 프로퍼티를 사용하여 VI 복잡도 및 복잡도 임계점과 프로그램적으로 상호작용할 수 있습니다.

VI를 컴파일할 때 발생하는 메모리 부족 에러 해결하기

LabVIEW 컴파일러는 크기가 크거나, 복잡한 VI를 컴파일할 때 메모리가 소진될 수 있습니다. 다음 방법 중 하나를 사용하여 이 문제를 해결합니다.

예상되는 원인	솔루션
VI 복잡도가 메모리를 보존하고 편집기 응답을 향상하도록 컴파일러가 컴파일러 최적화를 제한하기 시작하는 임계점 아래로 내려갑니다.	복잡도 임계점을 낮추어 관련 VI에서 편집기 응답이 실행 속도보다 우선 하도록 합니다.
컴파일러가 최적화를 제한하지만 VI가 너무 복잡하여 컴파일할 수 없습니다.	(권장) VI를 여러 SubVI로 나누어 VI의 복잡도를 감소합니다.

VI 문자열

VI 문자열 반출 및 반입하기

다음 단계를 따라 VI를 **현지화**할 문자열을 반출하고 반입합니다.

1. 도구 >> 고급 >> 문자열 반출을 선택하여 VI 문자열을 텍스트 파일로 반출합니다.
2. 나타나는 파일 대화에서 텍스트 파일의 위치를 탐색합니다. 텍스트 파일의 기본 이름은 VI 이름과 같습니다. 확인 버튼을 클릭합니다.
3. 캡션 이름을 현지화하기 위해서 나타나는 메시지 박스에서 예를 클릭합니다.
4. 블록 다이어그램 문자열을 현지화하기 위해서 나타나는 메시지 박스에서 예를 클릭합니다.
5. VI 문자열을 현지화하기 위해서 텍스트 파일을 열고 변경을 저장합니다.
6. 도구 >> 고급 >> 문자열 반입을 선택하여 VI 문자열을 반입합니다.
7. 입을 VI 문자열 파일 선택 대화 상자에서 텍스트 파일을 탐색합니다. 확인 버튼을 클릭합니다.
8. LabVIEW는 수행 도중 발생하는 에러 리스트를 포함하는 로그 파일을 생성하며 저장할지 여부를 입력요청합니다. 확인 버튼을 클릭합니다.

LabVIEW는 파일에 알려져 있지 않거나 분실된 태그가 포함된 경우 VI 문자열을 반입하지 않습니다. VI 문자열을 정확하게 현지화하지 않은 경우, 알려져 있지 않거나 분실된 태그가 포함될 수 있습니다. 문자열을 반출하고 번역된 문자열을 반입하는 시간 사이에 VI를 수정한 경우, VI 문자열 파일은 알려져 있지 않거나 분실된 문자열을 포함할 수 있습니다. 반출된 VI 문자열을 포함하는 텍스트 파일의 예는 **반출된 VI 문자열의 예** 항목을 참조하십시오.

반출된 VI 문자열의 예

다음 예의 텍스트는 반출된 VI 문자열을 포함하는 텍스트 파일의 내용을 보여줍니다. 반출된 VI 문자열에 대한 추가적인 정보는 [VI 문자열 반출하기](#)를 참조하십시오.

```
<VI syntaxVersion=11 LVversion=8508002 revision=13 name="Demo Recursion.vi">
<TITLE><NO_TITLE name="Demo Recursion.vi"></TITLE> <HELP_PATH></HELP_PATH>
<HELP_TAG></HELP_TAG> <RTM_PATH type="default"></RTM_PATH> <DESC>이 예는
LabVIEW에서 재귀 VI를 생성하는 방법을 보여줍니다. 재귀란 VI가 블록 다이어그램에서
자신을 다시 사용하는 것을 뜻합니다. 이러한 기능은 VI를 사용하여 함수가 복잡하고 좀
더 직접적인 반복 메소드를 구현하기 힘든  $F(n) = F(n-1) + G()$ 와 같은 문제를 해결할 때 유
용합니다.</DESC> <CONTENT> <GROUPER> <PARTS> </PARTS></GROUPER> <CONTROL
ID=80 type="Numeric" name="How many times to recurse?"> <DESC></DESC> <TIP></
TIP> <PARTS> <PART ID=82 order=0 type="Caption"><LABEL><STEXT>재귀 실행 횟수?</
STEXT></LABEL></PART> </PARTS> </CONTROL> <CONTROL ID=81 type="String"
name="Resulting String"> <DESC></DESC> <TIP></TIP> <PARTS> <PART ID=11 order=0
type="Text"><LABEL><STEXT></STEXT></LABEL></PART> <PART ID=82 order=0
type="Caption"><LABEL><STEXT>결과 문자열</STEXT></LABEL></PART> </PARTS>
<DEFAULT><SAME_AS_LABEL></DEFAULT> </CONTROL> <LABEL><STEXT>이 예에서는 VI
가 블록 다이어그램에서 스스로를 호출하는 재귀 방법을 사용하여 숫자로 이루어진 문
자열을 만듭니다. 이 태스크 자체는 중요하지 않고 재귀를 사용하지 않고도 간단히 수행
할 수 있지만, 이 예에서는 더 복잡한 문제를 해결하는데 유용할 수 있는 재귀에 대해 자
세히 설명합니다.<LF> <LF> <FONT style='B'>Instructions:<FONT predef=APPFONT><LF> 1.
원하는 재귀 실행 횟수를 선택합니다.<LF> 2. VI를 실행합니다.<LF> 3. 출력은 숫자로 이
루어진 문자열로 0에서 시작하여 입력 숫자까지 증가합니다.<LF> <LF> 재귀 동작에 대한
더 자세한 설명을 이 VI 및 이 VI의 SubVI의 블록 다이어그램에서 찾을 수 있습니다.</
STEXT></LABEL> </CONTENT> <BDCONTENT> <LABEL><STEXT>Look in here.</STEXT></
LABEL> <NODE ID=49 type="SubVI" subVIName="Read String.vi"> <DESC></DESC> </
NODE> <DESC></DESC> <TIP></TIP> <PARTS> </PARTS> <NODE ID=260 type="Object
Method SubVI" subVIName="Recurse.vi"> <DESC></DESC> </NODE> </BDCONTENT> </VI>
```

VI 비교 및 병합하기

VI와 VI 계층구조 비교하기

VI 비교 대화 상자는 그래픽 차이를 사용하여 여러 버전의 VI를 비교함으로써 어플리케이션의 변경사항을 추적할 수 있도록 도와줍니다. 프로젝트가 커지고 많은 개발자가 관여함에 따라 이 도구는 더욱 중요해지고 있습니다.



노트

VI 비교 대화 상자는 LabVIEW Professional Development System에서만 사용 가능합니다.

사용자가 찾아내고 확인할 차이점의 타입을 지정할 수 있습니다. 예를 들어 객체의 이동이나 크기 조정과 같은 외관의 변경사항을 무시할 수 있습니다. VI를 비교하면, 요약된 차이점이 윈도우에 표시됩니다. 요약에서 아이템을 선택하면 LabVIEW는 두 VI 사이의 차이를 디스플레이하고 하이라이트합니다.

다음 단계를 따라 두 VI를 비교합니다.

1. LabVIEW는 같은 이름으로 두 개의 VI를 불러올 수 없으므로, 두 VI를 비교하기 위해서 하나의 VI 이름을 반드시 바꿔야 합니다. VI의 이름을 바꿔도 subVI의 이름에 영향을 미치지 않습니다. **두 VI의 계층구조를 비교**한다면 LabVIEW는 임시로 최상위 VI의 이름을 바꿉니다.
2. VI의 프런트패널이나 블록 다이어그램 메뉴에서 도구 » 비교 » VI 비교를 선택하여 VI 비교 대화 상자를 디스플레이합니다.
3. 선택을 클릭하여 VI 선택 대화 상자를 열고 이름으로 VI를 선택합니다. 이미 메모리상에 로드되어 있는 VI만 선택할 수 있습니다.
4. 비교를 클릭합니다.
5. 차이 대화 상자에서 **비교 옵션을 선택**합니다.



노트 LabVIEW 클래스의 프라이빗 데이터 컨트롤을 비교할 때에도 똑같은 단계를 따릅니다. 서로 다른 프로젝트 안에 클래스의 각 버전을 놓고, 비교할 VI 테스트 필드에 프로젝트 경로를 입력할 때 다음 구문을 사용해야 합니다:

```
ProjectName.lvproj\My
Computer\ClassName.lvclass:ClassName.ct1.
```

매우 큰 VI의 비교는 시간을 소모할 수 있습니다. 두 VI의 비교를 비교 진행 대화 상자에서 취소할 수 있습니다.

진행 막대는 비교 알고리즘의 단계를 보여주며, 아직도 찾아야 할 남아있는 다른 점의 수를 보여주는 것이 아닙니다. 비교가 완료되면 LabVIEW는 차이 대화 상자과 함께 두 VI의 프런트패널과 블록 다이어그램을 **디스플레이**합니다.



노트 또한 **LVCompare.exe**는 VI 비교 대화 상자를 사용하여 타사 소스 컨트롤 제공자를 사용하는 VI를 비교하거나 명령 라인으로부터 VI를 비교합니다.

비교가 완료된 후, 차이 대화 상자에서 리포트 생성을 클릭하여 VI 비교 리포트를 생성할 수 있습니다. 리포트 생성 대화 상자에서 리포트 타입을 선택하고 리포트를 웹 페이지(.xml), 텍스트 파일(.txt), (Windows) 또는 Microsoft Word 문서(.doc)로 저장할 경로를 지정할 수 있습니다.



노트

- 비교 노트를 저장하는 경로는 네트워크 경로가 아닌 로컬 디렉토리여야 합니다.
- Microsoft Word 리포트를 생성하려면 Microsoft Word가 설치되어 있어야 합니다.

VI 계층구조를 비교할 때에도 또한 비교 리포트를 생성할 수 있습니다.

VI 계층구조 비교하기

LabVIEW Professional Development System에 있는 VI 계층구조 비교 대화 상자를 사용하여 두 VI의 계층구조를 비교합니다. LabVIEW는 양쪽 계층구조에서 이름이 동일한 파일을

모두 비교하고, 이 파일들이 같은지 차이가 있는지 분류합니다. 동일한 이름과 경로의 VI와 SubVI를 공유된 VI로 간주합니다. 다른 모든 VI는 첫번째 VI 계층구조에 추가되거나 제거된 것으로 분류됩니다. 비교가 끝나면 VI 계층구조 비교 대화 상자는 차이점을 요약해서 디스플레이합니다. 계층구조의 VI 리스트박스는 VI와 어떻게 VI가 분류되었지에 대한 기호를 디스플레이합니다. 차이점을 가지고 있는 일련의 VI를 선택할 수 있으며 **VI 비교** 대화 상자를 사용하여 시각적으로 비교할 수 있습니다.

LabVIEW는 동일한 이름의 두 개의 파일을 비교할 수 없기 때문에 비교를 위해서 LabVIEW는 하나의 VI 이름을 바꿉니다. 첫번째 VI는 그대로 남아있으며 두번째 VI는 임시 디렉토리으로 이동되고 cmp_ 접두어가 추가됩니다. LabVIEW는 subVI의 이름을 바꾸지 않기 때문에 이름이 바뀐 VI는 첫번째 VI에서 로드된 subVI를 링크합니다.

다음 단계를 따라 두 VI의 계층구조를 비교합니다.

1. VI의 프런트패널이나 블록 다이어그램 메뉴에서 도구 >> 비교 >> VI 계층구조 비교를 선택하여 VI 계층구조 비교 대화 상자를 디스플레이합니다.
2. 파일 탐색을 클릭하여 파일 시스템에서 비교할 VI를 선택합니다. 메모리 탐색을 클릭하여 VI 선택 대화 상자를 디스플레이한 후, 메모리에서 비교할 VI를 선택합니다.
3. VI 계층구조 비교 대화 상자의 비교 섹션에서, 포함시킬 **비교 옵션**을 활성화합니다.
4. 계층구조 비교를 클릭하여 VI의 계층구조를 비교합니다. 비교의 결과는 계층구조의 VI와 설명 리스트박스에 나타납니다.
5. (옵션) 리포트 생성을 클릭하여 두 VI 계층구조의 차이점에 대한 리포트를 생성할 수 있습니다. 리포트 생성 대화 상자에서 리포트 타입을 선택하고 리포트를 웹 페이지(.xml), 텍스트 파일(.txt), (Windows) 또는 Microsoft Word 문서(.doc)로 저장할 경로를 지정할 수 있습니다.



노트

- 비교 노트를 저장하는 경로는 네트워크 경로가 아닌 로컬 디렉토리여야 합니다.
 - Microsoft Word 리포트를 생성하려면 Microsoft Word가 설치되어 있어야 합니다.
6. 계층구조의 VI 리스트박스의 VI를 클릭하여 설명 리스트박스에 계층구조 차이에 대한 설명을 디스플레이합니다.
 7. 계층구조의 VI 리스트박스의 VI를 더블 클릭하여 두 계층구조의 VI를 열고 **차이를 확인**합니다. 또한 차이 보이기를 클릭할 수 있습니다.
 8. 완료를 클릭하여 VI 계층구조 비교 대화 상자를 닫습니다.



노트 계층구조 읽기를 클릭하여 계층구조에서 VI를 분류할 수 있습니다. 비교가 필요한 VI가 비교되지 않음으로 분류됩니다. 아이টে를 더블 클릭하거나 차이 보이기를 클릭하여 각각의 VI를 선택적으로 비교할 수 있습니다.

<Ctrl- > 키를 눌러서 큰 VI 계층구조의 비교를 강제 종료합니다.

두 개의 비교된 VI에서 차이 보이기

LabVIEW Professional Development System에서 차이 대화 상자를 통해 두 개의 **비교된 VI**의 프런트패널과 블록다이어그램을 확인합니다. 두 VI를 비교한 후, VI 비교 대화 상자에서 비교를 클릭하여 차이 대화 상자를 엽니다.

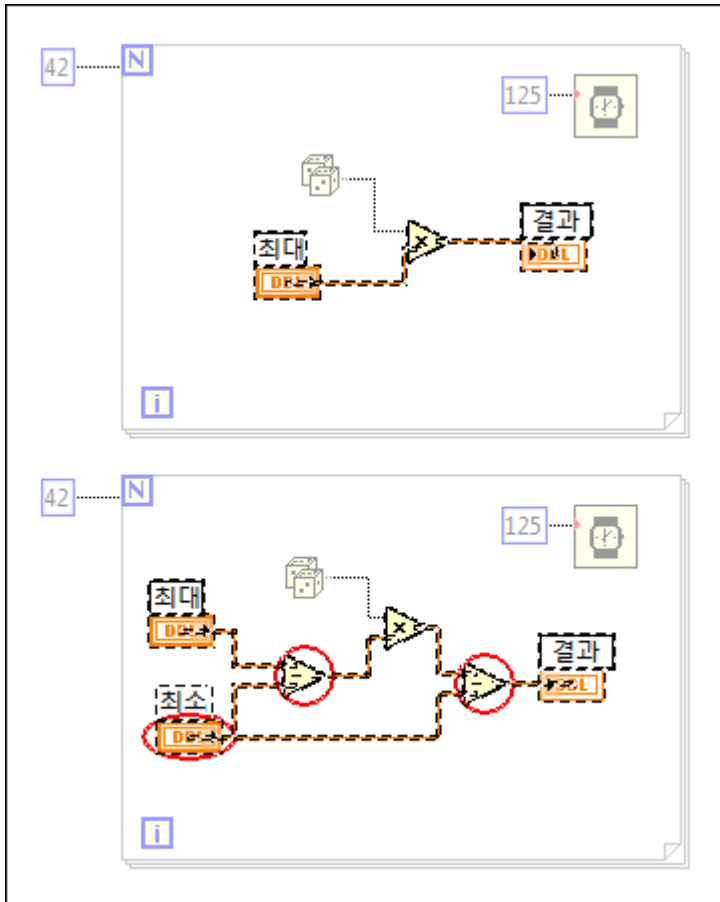
또한, VI 계층구조 비교 대화 상자에서 차이 보이기 버튼을 클릭하여 대화 상자에 접근할 수 있습니다.

차이 대화 상자는 차이의 목록과 선택된 차이의 상세한 설명을 포함합니다.

다음 단계를 따라 선택적으로 비교된 두 VI의 프런트패널과 블록다이어그램의 차이를 찾아냅니다.

1. 차이를 하이라이트하려면, 차이 리스트의 항목을 더블 클릭하거나 항목을 선택하고 차이 보이기 클릭합니다. 세부사항을 하이라이트하려면, 세부사항 리스트의 항목을 더블 클릭하거나 항목을 선택하고 세부사항 보이기를 클릭합니다. 확인 표시는 선택한 항목을 나타냅니다.
2. 추가 명령을 클릭한 후 좌우로 정렬 또는 상하로 정렬 버튼을 클릭하여 두 VI의 윈도우를 정렬합니다.
3. VI를 비교한 후 편집하면, 일부 차이는 더 이상 존재하지 않을 수 있습니다. 업데이트를 클릭하여 두 VI를 다시 비교하고 차이 리스트를 업데이트합니다.

추가 명령을 클릭한 후 차이를 원으로 표시를 선택하여 변경된 객체 주변에 빨간색 원을 그립니다. 차이를 하이라이트할 때 객체의 차이 부분이 선택됩니다. 다음 그림은 블록다이어그램 차이의 예를 보여줍니다.



그림에서 원으로 표시된 객체가 두 번째 VI에 삽입되었습니다. 객체를 선택했지만 원으로 표시되지 않은 경우에는 차이가 없습니다. 이것은 고정 객체로 선택되어 차이를 나타내기 위한 참고 포인트를 제공합니다. 흐리게 나타나는 객체는 차이를 표시하는 부분이 아닙니다. 이 비교에서는 VI 속성 및 외관 변경에 차이가 있는지 확인합니다.

VI 비교 옵션

VI 또는 VI 계층구조를 비교할 때 선택적으로 프런트패널과 블록다이어그램의 차이 찾기를 할 수 있습니다. 블록다이어그램의 경우 외관 차이를 무시할 수 있습니다. 외관 변화는 블록다이어그램의 실행에 영향을 미치지 않습니다. 프런트패널의 경우, 객체의 위치 및 크기 변경을 무시할 수 있습니다. 위치와 크기 변경은 객체가 앞에서 뒤로 이동한 것을 포함하며, 반대의 경우도 마찬가지입니다.

또한, VI 프로퍼티 대화 상자에서 설정과 같은 VI 속성의 차이를 구별할 수 있습니다.



관련 정보

[두 개의 비교된 VI에서 차이 보이기](#) (920 페이지 참조)

LVCompare.exe로 소스 컨트롤 설정하기

타사 **소스 컨트롤** 제공자를 설정하여 LVCompare.exe를 기본 비교 도구로 사용할 수 있습니다. LVCompare.exe는 VI 비교 대화 상자와 같습니다. LVCompare.exe를 사용하여 두 VI 사이의 차이를 대화식으로 비교할 수 있습니다.



노트

LVCompare.exe는 LabVIEW Professional Development System에서만 사용할 수 있습니다.

다음 단계에 따라 타사 소스 컨트롤 제공자를 설정하여, LVCompare.exe를 기본 비교 도구로 사용합니다.

1. LabVIEW를 실행 중인 컴퓨터에 타사 소스 컨트롤 제공자도 포함되어 있는지 확인하십시오. 추가적인 도움은 소스 컨트롤 관리자와 상의하십시오.
2. 소스 컨트롤 제공자가 LVCompare.exe를 사용하도록 지정합니다. LabVIEW는 LVCompare.exe를 플랫폼에 따라 서로 다른 디렉토리에 설치하며, 컴퓨터 설정에 따라 디렉토리가 달라질 수 있습니다:
 - (Windows) C:\Program Files\National Instruments\Shared\LabVIEW Compare\LVCompare.exe
 - (macOS) /Library/Application Support/National Instruments/LabVIEW Compare/LVCompare.app/Contents/MacOS/LVCompare
 - (Linux) /usr/local/bin/LVCompare
3. 다음 구문을 사용하여 LVCompare.exe가 필요로 하는 모든 명령문을 설정할 수 있습니다. lvcompare.exe <path to VI 1> <path to VI 2> [-lvpath <path to LabVIEW>][-noattr][-nofp][-nofppos][-nobd][-nobdcosm][-nobdpos]

노트 각 VI에 대해 지정된 경로는 절대 경로이거나 LVCompare.exe의 실행 위치에 따라 달라지는 상대적 경로입니다.

이 명령문의 의미는 다음과 같습니다:

- -lvpath는 비교 작업을 수행하려는 LabVIEW 버전에 대한 경로입니다.

노트 비교 작업을 수행할 LabVIEW의 버전을 지정하지 않는 경우, LabVIEW는 현재 등록된 LabVIEW 버전을 사용합니다. (macOS) LabVIEW는 컴퓨터에 설치된 최신 LabVIEW 버전을 사용합니다. (Linux) LabVIEW는 가장 최근에 설치된 LabVIEW 버전을 사용합니다.

- -noattr은 VI 속성을 비교하지 말라는 의미입니다.
- -nofp는 프런트패널을 비교하지 말라는 의미입니다.
- -nofppos는 프런트패널 객체의 크기나 위치를 비교하지 말라는 의미입니다.

- -nofp는 블록다이어그램을 비교하지 말라는 의미입니다.
- -nobdcosm은 위치나 크기의 변경을 포함하여 블록다이어그램 객체의 모양을 비교하지 말라는 의미입니다.
- -nofppos는 블록다이어그램 객체의 크기나 위치를 비교하지 말라는 의미입니다.

노트 이 인수는 LabVIEW 2021에서 제거되었습니다.

예를 들어, LabVIEW 2009를 사용하여 VI를 비교하는데 외관, 크기, 위치 차이는 비교하지 않도록 타사 소스 컨트롤 제공자를 설정하려면, 다음의 명령 라인을 사용합니다.

"C:\Program Files\National Instruments\Shared\LabVIEW Compare\LVCompare.exe" \$1 \$2 -lvpath "C:\Program Files\National Instruments\LabVIEW 2009\LabVIEW.exe" -nobdcosm -nofppos 이때 \$1 및 \$2는 비교하려는 VI의 경로에 대해 타사 소스 컨트롤 제공자가 정의하는 변수입니다.



노트 또한 명령 라인에 직접 이 명령문을 입력하여 LVCompare.exe를 시작할 수 있습니다.

1. (옵션) 남은 단계가 있다면 이를 완료하여 타사 소스 컨트롤 제공자를 설정합니다. 예를 들어, 일부 소스 컨트롤 제공자에서는 사용자가 특정 파일 확장자를 위한 비교 도구를 지정할 수 있습니다. 소스 컨트롤 제공자에서 직접 비교 도구를 지정할 수 있는 경우, *.vi 파일 확장자에 대한 LVCompare.exe를 지정하십시오. 추가적인 정보는 사용 중인 소스 컨트롤 제공자의 문서를 참조합니다.

LVMerge.exe로 소스 컨트롤 설정하기

또한 타사 **소스 컨트롤** 제공자를 설정하여 LVMerge.exe를 기본 병합 어플리케이션으로 사용할 수 있습니다. LVMerge.exe는 VI 병합 대화 상자와 같습니다. LVMerge.exe를 사용하여 자동적으로 또는 대화식으로 원본 VI와 원본 VI의 두 개정본들 사이의 차이를 해결합니다.



노트

LabVIEW Professional Development System이 있는 경우에만 LVMerge.exe를 사용할 수 있습니다.

다음 단계를 따라 타사 소스 컨트롤 제공자를 설정하여 LVMerge.exe를 기본 병합 어플리케이션으로 사용합니다.

1. LabVIEW를 실행하는 컴퓨터에 타사 소스 컨트롤 제공자가 설치되어 있으며 소프트웨어 프로젝트의 필수사양을 충족시키기 위해 설정되었는지 확인합니다. 추가적인 도움은 소스 컨트롤 관리자와 상의하십시오.
2. 소스 컨트롤 제공자가 LVMerge.exe를 사용하도록 지정합니다. LabVIEW는 플랫폼에 따라 LVMerge.exe를 다른 위치에 설치합니다. 다음 디렉토리는 LVMerge.exe의 기본 설치 경로이지만, 로컬 컴퓨터의 설정에 따라 달라질 수 있습니다:

- (Windows) C:\Program Files\National Instruments\Shared\LabVIEW Merge\LVMerge.exe
 - (macOS) /Library/Application Support/National Instruments/LabVIEW Merge/LVMerge.app/Contents/MacOS/LVMerge
 - (Linux) /usr/local/bin/LVMerge
3. (옵션) LVMerge.exe가 타사 소스 컨트롤 제공자에서 필요로 하는 어떤 인수든지 설정합니다. 다음 테이블은 사용 가능한 인수를 나열하고 있습니다.

병합 작업을 위해 사용되는 LabVIEW 버전의 경로를 지정합니다. 이 인수를 사용자가 제공하지 않는 경우, VI 병합 대화 상자는 기본으로 다음의 LabVIEW 버전을 사용합니다:

- (Windows) 현재 등록된 LabVIEW
- (macOS) 마지막으로 설치된 LabVIEW 버전
- (Linux) 마지막으로 설치된 LabVIEW

사용자가 VI 복사본을 체크아웃했던 절대 경로입니다(기본 VI).

소스 컨트롤에서 VI의 최신 개정판 복사본의 절대 경로입니다(상대편 VI).

사용자가 편집한 VI의 절대 경로입니다(내 VI).

병합된 VI를 저장할 절대 경로입니다.

예를 들어, 병합 작업을 수행할 LabVIEW 버전을 LabVIEW 8.6으로 설정하려는 경우에는 다음의 명령 라인을 사용합니다. LVMerge.exe "C:\Program Files\National Instruments\LabVIEW 8.6\LabVIEW.exe" c:\files\base.vi c:\files\theirs.vi c:\files\yours.vi c:\files\merged.vi. 명령 라인에서 첫 번째 경로는 선택 사항이며 LabVIEW 실행 파일의 위치로 연결됩니다.

1. (옵션) 남은 단계가 있다면 이를 완료하여 타사 소스 컨트롤 제공자를 설정합니다. 추가적인 정보는 타사 소스 컨트롤 제공자의 문서를 참조합니다.

VI 병합하기

병합할 VI 선택 대화 상자를 사용하여 자동적으로 또는 대화식으로 원본 VI와 2 개의 개정본들 사이의 차이를 해결합니다. 원본 VI 및 원본 VI를 기반으로 하는 두 개정본이 주어지면, LabVIEW는 변경 사항을 해결하고 병합하여 개별적인 병합 VI로 만들 수 있습니다. 두 VI 사이의 차이가 충돌하면, VI 병합 대화 상자를 사용하여 변경 내역을 대화식으로 해결할 수 있습니다.



노트

VI 병합 대화 상자는 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 따라 자동으로 또는 대화식으로 두 VI를 병합합니다.

1. LabVIEW는 같은 이름을 가진 두 VI를 불러올 수 없으므로, 병합하려는 VI의 이름을 변경해야 합니다.
2. 도구 » 병합 » VI 병합을 선택하여 병합할 VI 선택 대화 상자를 디스플레이합니다.
3. 기본 VI 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고 원본 VI를 찾은 다음에 확인을 클릭합니다.
4. 상대편 VI 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고 병합하려는 첫 번째 VI를 찾은 다음에 확인을 클릭합니다.
5. 내 VI 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고 병합하려는 두 번째 VI를 찾은 다음에 확인을 클릭합니다.
6. (옵션) 병합 옵션 섹션에 있는 옵션을 선택하여 LabVIEW가 VI의 원소를 병합하는 방법을 설정합니다.
7. 병합 버튼을 클릭합니다. 프론트패널 병합 윈도우 및 블록다이어그램 병합 윈도우, 병합된 VI, VI 병합 대화 상자가 나타납니다.
8. 기본적으로, LabVIEW는 해결되지 않은 변경사항 충돌 필드에서 각각의 변경 사항을 숨깁니다. 각 변경 사항 옆의 확장/축소 기호를 클릭하여 해당 변경 사항의 상세 설명을 볼 수 있습니다.
9. 각 변경사항의 경우, 내 또는 상대편 세부 항목 아래의 변경사항에 대한 설명을 클릭하여 내 VI 또는 상대편 VI의 세부사항을 선택합니다. LabVIEW는 각각의 해결되지 않은 변경사항 옆에 X를 놓습니다.



노트 프론트패널 변경사항 윈도우, 블록다이어그램 변경사항 윈도우, 또는 제목없는 병합된 VI에서 대부분의 변경 사항에 빨간 동그라미가 쳐져 있습니다.

1. 기본 선택, 상대편 것 선택, 내 것 선택 버튼 중에서 하나를 클릭하여, 수용하고 병합된 VI에 추가할 변경사항을 선택합니다. 각 변경사항을 해결하게되면, 해당 변경사항 옆에 확인 표시가 나타납니다.
2. 모든 변경사항을 해결한 후, 닫기 버튼을 클릭하여 VI 병합 대화 상자를 닫고, 저장 버튼을 클릭하여 병합된 VI를 저장합니다.



노트 상대편 VI의 변경사항과 내 VI의 변경사항을 병합하려면 기본 VI에 공통적인 교차영역이 있어야 합니다. 예를 들어, 상대편 VI와 내 VI의 대응 터널이 서로 일치하는지 VI 병합이 파악할 수 있으려면 터널이 기본 VI에 있어야 합니다.

또한 타사 **소스 컨트롤** 제공자를 설정하여, 기본 병합 어플리케이션으로 **LVMerge.exe**를 **사용**할 수 있습니다. LVMerge.exe는 병합할 VI 선택 대화 상자와 같습니다.

LLB를 병합하려면 병합할 LLB 선택 대화 상자를 사용하여 LLB의 여러 개정본 사이의 차이를 자동으로 또는 대화식으로 해결할 수 있습니다.

LLB 병합하기

병합할 LLB 선택 대화 상자를 사용하여 원본 **LLB**와 2개의 개정본 사이의 차이를 자동으로 또는 대화식으로 해결합니다. 원본 LLB와 이를 기반으로 하는 두 개정본이 주어지면, LabVIEW는 자동으로 변경사항을 해결하고 병합하여 별도의 병합된 LLB로 만들 수 있습니다.

니다. 두 LLB 사이의 차이가 충돌하면, LLB 병합 대화 상자를 사용하여 변경 사항을 대화식으로 해결할 수 있습니다.



노트

LLB 병합 대화 상자는 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 따라 자동으로 또는 대화식으로 두 LLB를 병합합니다.

1. 도구 >> 병합 >> LLB 병합을 선택하여 병합할 LLB 선택 대화 상자를 디스플레이합니다.
2. 기본 LLB 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고, 원본 LLB를 찾은 다음에 확인을 클릭합니다.
3. 상대편 LLB 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고, 기본 LLB와 병합시키려는 첫번째 LLB를 찾은 다음에 확인을 클릭합니다.
4. 내 LLB 텍스트 박스 옆에 있는 탐색 버튼을 클릭하고, 기본 LLB와 병합시키려는 두 번째 LLB를 찾은 다음에 확인을 클릭합니다.
5. (옵션) 병합 옵션 섹션에 있는 옵션을 선택하여 LabVIEW가 LLB의 원소를 병합하는 방법을 설정합니다.
6. 병합 버튼을 클릭합니다. LLB 병합 대화 상자가 나타납니다. LabVIEW는 각각의 LLB에서 찾은 모든 파일을 파일 열에 나열합니다. 상대편, 내, 병합된 열에 있는 각 파일 옆의 기호는 LabVIEW가 각 LLB에서 발견한 변경 사항의 타입을 표시합니다.



노트 내 LLB 및 상대편 LLB에 같은 프로젝트 라이브러리 파일 .lvlib에 적용되는 변경사항이 있으면 LabVIEW는 자동으로 이 변경사항을 해결할 수 없습니다. 이같은 변경사항을 해결하기 위해서는, .lvlib 파일의 하나의 개정 버전을 열고 직접 다른 개정 버전과 일치하도록 편집하는 방법이 권장됩니다.

1. 변경 사항이 나열된 행을 선택한 후 상대편 것 선택, 기본 선택, 내 것 선택 버튼 중에서 하나를 클릭하여, 수용하고 병합된 LLB에 추가할 변경 사항을 선택합니다. 각 변경 사항을 해결함에 따라, 선택된 변경 사항 옆에 확인 표시가 나타납니다.
2. 두 LLB가 동일한 VI에서 나온 다른 개정본을 포함하는 경우, VI 병합 버튼을 사용하여 개정본을 병합합니다. VI를 포함하는 행을 클릭하고 VI 병합 버튼을 클릭하여 병합할 VI 선택 대화 상자를 엽니다.
3. **VI 개정본 사이의 차이를 해결합니다.**
 - 파일을 병합하는 경우, 파일을 포함하는 행을 하이라이트하고 병합된 파일 보기를 클릭하여 생성한 파일을 볼 수 있습니다.
4. 모든 변경 사항을 해결한 후, 닫기 버튼을 클릭하여 LLB 병합 대화 상자를 닫고, 저장 버튼을 클릭하여 병합된 LLB를 저장합니다.

또한 타사 **소스 컨트롤** 제공자를 설정하여, 기본 병합 어플리케이션으로 **LVMerge.exe**를 사용할 수 있습니다. LVMerge.exe는 병합할 LLB 선택 대화 상자와 같습니다.

개별 VI를 병합하려면 병합할 VI 선택 대화 상자를 사용하여 VI의 여러 개정본 사이의 차이를 자동으로 또는 대화식으로 해결할 수 있습니다.

VI 메트릭스

VI 메트릭스 윈도우 사용하기

LabVIEW Professional Development System에서 지원하는 VI 메트릭스 윈도우는 어플리케이션의 복잡성을 측정하는 방법을 제공하는 것으로, 텍스트 기반의 프로그램 언어에서 많이 사용되는 SLOC(Source Lines of Code) 메트릭스와 유사합니다. VI 메트릭스 윈도우에서 VI에 관한 메트릭스를 볼 수 있어 매우 복잡한 VI의 영역을 찾는 데 유용합니다. 또한, 이 메트릭스를 사용하여 앞으로의 프로젝트를 예측하는 기초를 확립할 수 있습니다.

SLOC를 포함한 모든 메트릭은 복잡한 개략적인 측정이라는 것을 기억하십시오. 특정한 경우에 사용자가 찾은 어떤 것이 다른 것보다 더 중요할 수 있기 때문에 VI 메트릭스 윈도우를 사용하면 많은 메트릭스에 접근할 수 있습니다. 예를 들어, 특정 메트릭스를 조합하여 VI의 복잡성에 대한 더 좋은 아이디어를 얻어 사용자 인터페이스 VI를 결정할 수도 있습니다. 이 경우에는, VI에 대한 정보를 저장하고 결과를 분석하기 위해서 VI를 작성하고, VI의 복잡성에 대한 새로운 측정을 생성하기 위해 필드를 조합하여 사용자의 메트릭을 만들 수 있습니다.

National Instruments는 VI를 분석하는데 유용한 조합 또는 기타 메트릭스에 대한 의견을 듣고자 합니다. National Instruments 웹 사이트를 방문하여 의견을 제출해 주시기 바랍니다.

VI 메트릭스 윈도우를 사용하여 VI 분석하기

다음 단계를 따라 VI를 분석합니다.

1. 분석하려는 VI를 엽니다.
2. 도구 >> 프로파일 >> VI 메트릭스를 선택하여 VI 메트릭스 윈도우를 디스플레이합니다.



노트

VI 메트릭스는 LabVIEW Professional Development System에서만 사용 가능합니다.

1. VI 선택 풀다운 메뉴를 사용하여 프런트패널이 열린 VI의 리스트에서 선택합니다. 윈도우 아래의 통계 테이블에서, LabVIEW는 VI와 SubVI의 이름 그리고 각 VI의 메트릭스를 디스플레이합니다. 선택된 계층구조의 각 VI에 대해서, 윈도우는 VI가 포함하는 노드의 수를 보여줍니다. 이 숫자는 텍스트 기반의 프로그램 언어에서 일반적으로 사용되는 SLOC 메트릭과 비슷한 개략적인 메트릭입니다. 노드의 숫자는 함수, subVI, 구조, 프런트패널 객체의 터미널, 상수, 글로벌과 로컬 변수, 프로퍼티 노드를 포함합니다. 그러나 LabVIEW는 하나의 노드와 동일한 프로퍼티 노드를 가진 여러 프로퍼티 읽기를 카운트합니다. 사용자 인터페이스 확인란에 확인 표시를 하면, VI가 읽거나 쓰는 전체 프로퍼티의 개수를 확인할 수 있습니다. 노드의 개수는 [For 루프]의 반복 터미널이나 다중 시퀀스 구조의 시퀀스 로컬 터미널과 같은 구조의 하위 구성요소인 와이어, 터널, 또는 객체를 포함하지 않습니다.

2. (옵션) 통계 보이기 확인란에 확인 표시를 하여 다음 각 VI의 복잡성과 관련된 추가 정보를 디스플레이합니다: **블록다이어그램**, **사용자 인터페이스**, **로컬 및 글로벌 변수**, **공유 라이브러리**, **SubVI 인터페이스** 매트릭스.
3. (옵션) 저장 버튼을 클릭하여 VI 계층구조의 매트릭스 정보를 텍스트 파일에 저장합니다. 이 파일은 보이는 열의 매트릭스를 통계 테이블에 포함합니다. 파일은 사용자 VI의 개수와 labview\vi.lib VI의 개수도 포함합니다. LabVIEW는 정보를 탭으로 자리 구분한 포맷으로 저장하기 때문에, 스프레드시트에서 읽을 수 있으며 VI를 사용하여 쉽게 읽거나 분석이 가능합니다.

블록다이어그램 매트릭스

다이어그램 확인란에 확인 표시를 하면 LabVIEW Professional Development System에서 사용 가능한 블록다이어그램 매트릭스가 VI 매트릭스 윈도우에 나타납니다. 다음의 매트릭스가 VI와 구조의 블록다이어그램과 와이어를 지정합니다:

- 구조—For 루프, While 루프, 케이스 구조, 이벤트 구조, 플랫 시퀀스 구조, 다층 시퀀스 구조, 조건적 비활성화 구조와 다이어그램 비활성화 구조 개수입니다.
- 다이어그램—블록다이어그램의 개수입니다. 각각의 VI는 하나의 최상위 레벨 블록다이어그램을 가지고 있습니다. 또한 VI는 각 루프나 구조를 위한 하나의 서브 다이어그램을 가지고 있습니다.
- 최대 다이어그램 깊이—VI에서 블록다이어그램의 가장 깊은 하위 레벨입니다. VI가 구조를 가지고 있지 않다면 이것은 0의 깊이를 갖습니다.
- 다이어그램 폭(픽셀)—픽셀 단위의 블록다이어그램의 폭입니다.
- 다이어그램 높이(픽셀)—픽셀 단위의 블록다이어그램의 높이를 뜻합니다.
- 와이어 소스—VI에서 소스의 총 개수를 측정합니다. 각각의 와이어는 단일 소스를 가지고 있으나 여러 대상으로 분기할 수 있습니다. 와이어가 구조에 들어가거나 나오면, LabVIEW는 와이어가 통과하는 터널을 새로운 소스로 간주합니다.

공유 라이브러리 매트릭스

공유 라이브러리 호출 확인란에 확인 표시를 하면 LabVIEW Professional Development System에서 사용 가능한 공유 라이브러리 매트릭스가 VI 매트릭스 윈도우에 나타납니다. 다음 매트릭스는 블록다이어그램에서 라이브러리 함수 호출 노드를 지정합니다:

- 공유 라이브러리 호출—블록다이어그램에 있는 라이브러리 함수 호출 노드의 숫자.

글로벌/로컬 매트릭스

글로벌/로컬 확인란에 확인 표시를 하면 LabVIEW Professional Development System에서 사용 가능한 글로벌/로컬 매트릭스가 VI 매트릭스 윈도우에 나타납니다. 다음 매트릭스는 VI의 글로벌 및 로컬 변수를 지정합니다:

- 글로벌 읽기—블록다이어그램에서 읽는 글로벌 변수의 개수.

- 글로벌 쓰기—블록다이어그램에서 쓰는 글로벌 변수의 개수.
- 로컬 읽기—블록다이어그램에서 로컬 읽기 변수의 개수.
- 로컬 쓰기—블록다이어그램에서 로컬 쓰기 변수의 개수.

SubVI 인터페이스 메트릭스

SubVI 인터페이스 확인란에 확인 표시를 하면 LabVIEW Professional Development System에서 사용 가능한 SubVI 인터페이스 메트릭스가 VI 메트릭스 윈도우에 나타납니다. 다음의 메트릭스가 VI의 SubVI 및 컨트롤과 인디케이터를 지정합니다:

- 커넥터 입력—VI 커넥터 팬의 컨트롤 개수입니다.
- 커넥터 출력—VI 커넥터 팬의 인디케이터 개수입니다.

사용자 인터페이스 메트릭스

사용자 인터페이스 확인란에 확인 표시를 하면 LabVIEW Professional Development System에서 사용 가능한 사용자 블록다이어그램 메트릭스가 VI 메트릭스 윈도우에 나타납니다. 다음의 메트릭스가 VI 사용자 인터페이스를 지정합니다:

- 컨트롤—최상위 프런트패널 컨트롤의 개수입니다. 배열 클러스터는 단일 컨트롤로 세어집니다.
- 컨트롤—최상위 프런트패널 인디케이터의 개수입니다. 클러스터 또는 배열은 단일 인디케이터로 세어집니다.
- 프로퍼티 읽기—블록다이어그램에서 프로퍼티 읽기의 개수입니다. 동일한 프로퍼티 노드에 여러 프로퍼티 읽기가 있다면, 각 프로퍼티는 프로퍼티 읽기 개수를 증가시킵니다.
- 프로퍼티 쓰기—블록다이어그램에서 프로퍼티 쓰기의 개수입니다. 동일한 프로퍼티 노드에 여러 프로퍼티 쓰기가 있다면, 각 프로퍼티는 프로퍼티 쓰기 개수를 증가시킵니다.

취소 설정하기

다음 단계를 따라 취소 또는 다시 실행을 할 수 있는 동작의 개수를 설정합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 환경을 선택합니다.
3. VI에서 최대 실행 취소 단계 텍스트 박스 안에 숫자를 입력합니다. 입력할 수 있는 최대 숫자는 99입니다. 입력할 수 있는 최소 숫자는 1입니다.

4. 확인 버튼을 클릭합니다.



노트

각 VI는 각자 고유한 취소 인스턴스를 기록합니다. 그러므로, VI에서 최대 실행 취소 단계를 위해 입력한 숫자는 각 VI에 속해 있습니다. 이 숫자를 낮게 설정하면 메모리 리소스를 절약할 수 있습니다.

일부 인스턴스에서, LabVIEW는 모든 취소 정보를 삭제합니다. VI 호출자는 VI의 프런트패널이 변경되었을 때 자동적으로 업데이트합니다. LabVIEW는 다음과 같은 변경이 있을 때 호출자 VI에 있는 모든 취소 정보를 삭제합니다:

- **VI 커넥터 팬 패턴을 변경**하거나 또는 컨트롤의 데이터 타입을 변경합니다.
- 엄격한 데이터 타입 외형을 변경하거나, 데이터 타입 또는 엄격함을 변경하거나, 또는 사용자 지정 컨트롤 타입 정의를 변경합니다.
- 글로벌 VI에 있는 컨트롤의 데이터 타입이나 이름을 변경하거나 글로벌 VI에 대한 컨트롤을 추가하거나 제거합니다.



빠른 탐색 대화 상자의 바로 가기 설정하기

빠른 탐색 설정 대화 상자를 사용하여 객체에 대한 바로 가기를 지정할 수 있습니다. 빠른 탐색 대화 상자에서 객체 이름 전체를 입력하지 않아도 되도록, 객체에 대한 바로 가기 이름을 지정할 수 있습니다.

다음 단계를 완료하여 빠른 탐색 대화 상자에서의 바로 가기를 지정합니다.

1. <Ctrl>-<Space> 키를 누르거나 보기>>빠른 탐색을 선택하여 빠른 탐색 대화 상자를 디스플레이합니다. 중국어 키보드에서는, <Ctrl-Shift-Space> 키를 누릅니다. (macOS) <Command-Shift-Space> 키를 누릅니다. (Linux) <Alt-Space> 키를 누릅니다.



노트

빠른 탐색 설정 대화 상자를 사용하여 LabVIEW 프로젝트 아이템에 바로 가기를 할당할 수 없습니다.

1. 빠른 탐색 대화 상자에서 설정 버튼을 클릭하여 빠른 탐색 설정 대화 상자를 디스플레이합니다. 대화 상자를 어느 윈도우에서 열었는지에 따라 LabVIEW는 대화 상자의 프런트패널 또는 블록 다이어그램 페이지를 디스플레이합니다.
2. 패널 바로가기 또는 다이어그램 바로가기 텍스트 박스에 바로 가기를 입력합니다. 예를 들어, 케이스 구조와 같은 객체를 선택하여 바로 가기를 할당하려면 다이어그램 바로가기 텍스트 박스에 cs를 입력합니다.

노트 빠른 탐색 대화 상자를 사용하여 바로 가기를 생성할 경우, <Ctrl> 및 <Shift> 키와 같은 조합키를 사용할 수 없습니다.

3. 패널 바로 가기 객체 또는 다이어그램 바로 가기 객체 풀다운 메뉴에서 객체를 선택합니다.
4. 추가 버튼을 클릭하여 해당 객체에 대한 바로 가기를 지정합니다. 프런트패널 바로 가기 또는 블록다이어그램 바로 가기 테이블에 사용자가 지정한 모든 바로 가기가 나열됩니다.
5. (옵션) 바로 가기를 제거하려는 경우, 테이블에서 바로 가기를 선택하고 선택된 바로 가기 제거 버튼을 클릭합니다.
6. 확인 버튼을 클릭하여 대화 상자를 닫습니다.

할당한 바로 가기를 사용하려면, <Ctrl>-<Space> 키를 누르거나 보기>>빠른 탐색을 선택하여 빠른 탐색 대화 상자를 디스플레이합니다. 중국어 키보드에서는, <Ctrl-Shift-Space> 키를 누릅니다. (macOS) <Command-Shift-Space> 키를 누릅니다. (Linux) <Alt-Space> 키를 누릅니다. 객체에 대해 지정한 바로 가기를 입력합니다. 바로 가기 및 객체 이름(괄호 안)이 텍스트 박스에 나타납니다. 예를 들어, 케이스 구조에 대한 바로 가기를 cs로 지정한 경우, 텍스트 박스에 cs를 입력합니다. 텍스트 박스는 cs (케이스 구조)를 디스플레이합니다. 블록다이어그램을 클릭하여 블록다이어그램에 케이스 구조를 놓습니다.



노트 빠른 탐색 대화 상자에서 설정한 바로 가기를 빠른 탐색 대화 상자를 지원하는 다른 LabVIEW로 이전하려면, 현재의 **LabVIEW 설정 파일**에서 QuickDropPanelShortcuts 및 QuickDropDiagramShortcuts를 복사하여 이전할 LabVIEW의 LabVIEW 설정 파일에 붙여넣습니다.

완료되지 않은 태스크 관리하기

블록다이어그램의 라벨 또는 주석에 해시태그(#)를 사용하여 태스크 또는 해야 할 아이템의 리스트를 관리할 수 있습니다. 텍스트 앞에 해시태그를 놓는 것을 북마크라고 합니다. 블록다이어그램 주석에 해시태그를 추가하면, LabVIEW는 북마크를 감지하고 알고리즘 #다시쓰기와 같이 북마크 태그를 굵은체로 나타내어 나머지 텍스트와 구별합니다. 북마크를 사용하면 앞으로 더 개발해야 할 필요가 있는 불완전한 코드를 식별할 수 있습니다. 어플리케이션에 있는 모든 북마크를 찾으려면, 보기>>북마크 관리자를 선택하여 북마크 관리자 윈도우를 디스플레이합니다.



노트

컨트롤 또는 인디케이터 라벨에는 북마크를 사용할 수 없습니다.

템플릿과 샘플 프로젝트에서 VI 생성하기

파일>>프로젝트 생성을 선택하여 다음 템플릿과 샘플 프로젝트가 나열되어 있는 프로젝트 생성 대화 상자를 디스플레이합니다:

- 기본 상태 머신—코드의 섹션 및 실행 순서를 정의하는 템플릿.

- 유한 측정—기본 상태 머신 템플릿을 사용하여 한 개의 측정을 수집하는 샘플 프로젝트.
- 큐 메시지 핸들러—데이터 수집 또는 로깅과 같은 다른 태스크를 수행하는 동안 사용자 인터페이스를 모니터할 수 있도록 하는 템플릿.
- 연속 측정 및 로깅—큐 메시지 핸들러 템플릿을 사용하여 연속적으로 측정 결과를 수집하고 이를 디스크에 로그하는 샘플 프로젝트.
- 액터 프레임워크—서로 통신하는 여러 독립적인 태스크로 구성된 어플리케이션용 템플릿.
- 피드백형 증발식 냉각기—액터 프레임워크 템플릿을 사용하여 물 탱크를 냉각하는 샘플 프로젝트. 이때 냉각기는 시뮬레이션 또는 실제 하드웨어로 구현하며, 수동 및 자동 조정 기능이 있고, 두 개 중 하나의 사용자 인터페이스를 사용하여 볼 수 있습니다.
- 터치 패널 프로젝트—Windows Embedded Standard OS가 실행되는 터치 패널 타겟에 대해 어플리케이션을 정의하는 템플릿. 이 템플릿에는 터치 패널에서 실행되는 터치 패널 VI와 호스트 컴퓨터에서 실행되는 VI가 포함되어 있습니다. 호스트 컴퓨터에서 실행되는 VI는 큐 메시지 핸들러 템플릿을 사용합니다.



노트

LabVIEW 애드온이 추가적인 템플릿 및 샘플 프로젝트를 설치할 수도 있습니다.

템플릿이 일반적으로 사용되는 디자인 패턴을 LabVIEW 코드에서 구현하는 방법을 보여주는 반면, 샘플 프로젝트는 특정 어플리케이션에서 템플릿을 사용하는 방법을 보여줍니다. 각 템플릿과 샘플 프로젝트에는 확장 가능한 견고한 어플리케이션의 개발을 시작하는데 필수적인 프로젝트 라이브러리, 함수, 구조 및 사용자 인터페이스가 포함되어 있습니다.

어플리케이션 디자인 패턴에 대한 더 자세한 정보는 ni.com을 참조하십시오.

템플릿 VI 생성하기

사용자 템플릿 VI를 생성하면 비슷한 작업을 할 때마다 매번 프론트패널이나 블록다이어그램에 같은 성분을 추가하지 않아도 됩니다.

다음 단계를 완료하여 기존 VI 템플릿을 기초로 하여 새 템플릿을 생성합니다.

1. 파일>>새로 만들기를 선택하여 새로 만들기 대화 상자를 디스플레이합니다.
2. 새로 생성 리스트의 템플릿 디렉토리에서 만들려는 템플릿과 유사한 템플릿 VI를 선택합니다.
3. 확인 버튼을 클릭합니다. 템플릿 VI의 프론트패널과 블록다이어그램이 나타납니다.



노트

템플릿에서 VI를 생성할 때, LabVIEW는 옵션 대화 상자에서 선택한 옵션을 새 VI에 적용하지 않습니다.

4. VI를 편집합니다.

팁 템플릿을 템플릿 VI에 SubVI로 추가할 수 있습니다. 템플릿을 SubVI로 추가하려면, VI 템플릿을 열고 프런트패널 또는 블록다이어그램 윈도우의 오른쪽 위 코너에 있는 VI 아이콘을 최상위 템플릿 VI의 블록다이어그램으로 끕니다. 템플릿 SubVI를 가지는 템플릿으로부터 VI를 생성하고 이 VI를 저장하면, LabVIEW는 각 템플릿 SubVI를 VI로 저장하도록 입력요청합니다.

1. 템플릿으로 VI를 저장합니다.

1. 파일»다른 이름으로 저장을 선택하여 파일 대화 상자를 디스플레이합니다.

노트 이미 템플릿 VI를 저장한 경우, 대신 파일»다른 이름으로 저장을 선택하여 다른 이름으로 저장 대화 상자를 디스플레이합니다. 복사와 복사본 열기 옵션을 선택하고 계속 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.

2. (옵션) LabVIEW가 설치한 템플릿 VI와 함께 템플릿 VI가 새로 만들기 대화 상자에 나타나는 경우 labview\templates 디렉토리를 탐색합니다. 블록다이어그램 미리보기를 설정하려면, 블록다이어그램의 이미지를 labview\templates 디렉토리에 .png 파일로 저장합니다. VI의 이름에 d를 추가하여 .png 파일이름을 생성합니다. 예를 들어, test.vi의 블록다이어그램 미리보기를 설정하려는 경우, 블록다이어그램의 이미지를 labview\templates 디렉토리에 Testd.png 파일로 저장합니다. VI 템플릿이 새로 생성 리스트에서 선택되면 이 블록다이어그램 미리보기와 **VI 설명**이 설명 섹션에 나타납니다.

주의 사용자 VI 및 다른 파일을 **LabVIEW와 함께 설치되는 .lib 파일**에 저장하지 마십시오.

3. (Windows) 저장할 타입 풀다운 메뉴에서 템플릿 VI를 선택합니다. (macOS와 Linux) 템플릿 VI 파일 확장자 .vit를 포함합니다.

4. 템플릿 VI의 이름을 입력한 후 확인 버튼을 클릭합니다.

또한, 파일 시스템에서 .vi 확장자를 .vit 확장자로 변경하여 기존의 VI를 템플릿 VI로 변환할 수 있습니다.

VI 및 다른 파일 타입에서 컴파일된 코드 분리하기

VI에서 컴파일된 코드를 분리하여 소스만 있는 VI를 생성하는 이유는 다음과 같습니다:

- 소스 컨트롤 단순화—VI 계층구조 또는 LabVIEW 프로젝트 안의 모든 파일에서 컴파일된 코드를 분리할 경우, 파일 하나가 변경되었을 때 계층구조 또는 프로젝트의 다른 곳에서 저장되지 않은 변경이 생기지 않습니다. 그렇기 때문에 변경하려는 VI만 체크아웃하고 이 변경된 VI를 호출하는 모든 VI를 체크아웃할 필요가 없습니다.
- 소스 컨트롤에 있는 VI를 LabVIEW 새 버전으로 업그레이드하기 위해 준비—소스 컨트롤에서 VI 체크아웃하지 않고 소스만 있는 VI를 새 LabVIEW 버전으로 업그레이드할 수 있습니다.
- VI 로드 시간 향상—LabVIEW는 일반 VI보다 소스만 있는 VI를 더 빨리 로드할 수 있습니다. 이러한 장점을 최대화하려면 VI 계층구조 또는 프로젝트의 모든 파일에서 컴파일된 코드를 분리하십시오..

LabVIEW 런타임 엔진을 사용하여 로드하거나 실행하려는 VI에서는 컴파일된 코드를 분리하지 마십시오. 런타임 엔진은 분리한 컴파일된 코드를 저장하는 컴파일된 객체 캐시에 접근할 수 없기 때문에 이러한 VI를 실행할 수 없습니다. 소스 컨트롤에 체크인한 VI 계층구조가 런타임 엔진에서 실행되도록 하려면, 소스 배포를 VI와 함께 만드십시오.

어떤 파일에서 컴파일된 코드 분리할지 선택하기

- 기본으로 모든 새 파일 선택 옵션 대화 상자에 있는 환경 페이지에서 새 파일에서 컴파일된 코드 분리 확인란에 확인 표시를 합니다.
- 기본으로 단일 프로젝트 안의 모든 프로젝트 아이템 선택 프로젝트 프로퍼티 대화 상자에 있는 프로젝트 페이지에서 새 프로젝트 아이템에서 컴파일된 코드 분리 확인란에 확인 표시를 합니다.



노트

이 옵션은 이미 존재하는 파일에는 영향을 미치지 않으며, 이 옵션을 선택한 후에 기존의 파일을 프로젝트에 추가하더라도 기존 파일에서 컴파일된 코드를 분리하지 않습니다.

- 단일 프로젝트에 있는 기존 프로젝트 아이템 선택 프로젝트 프로퍼티 대화 상자의 프로젝트 페이지에서 기존 아이템 표시 버튼을 클릭합니다. 그 후 프로젝트에서 컴파일된 코드를 분리할 아이템을 선택할 수 있습니다.
- 단일 VI VI 프로퍼티 대화 상자의 일반 페이지에서 소스 파일에서 컴파일된 코드 분리 확인란에 확인 표시를 합니다. 또한 VI의 컴파일된 코드 포함 프로퍼티를 사용하여 프로그램적으로 단일 VI에서 컴파일된 코드를 분리할 수 있습니다.
- 단일 프로젝트 라이브러리, LabVIEW 클래스 또는 X컨트롤 파일의 프로퍼티 페이지에서 소스 파일에서 컴파일된 코드 분리 확인란에 확인 표시를 합니다. 또한 프로젝트 라이브러리의 컴파일된 코드 포함 프로퍼티를 사용하여 객체 캐시에 프로젝트 라이브러리의 코드를 프로그램적으로 저장할 수 있습니다.

노트 LabVIEW는 이러한 파일 타입에서 실제로 컴파일된 코드를 분리하지 않습니다. 대신 이 옵션을 선택하면, LabVIEW는 소스만 있는 파일의 코드를 객체 캐시에 저장하여 소스만 있는 VI와 같은 혜택을 얻습니다.



관련 정보

[VI 및 다른 파일 타입에서 컴파일된 코드 분리하여 소스 컨트롤하기](#) (339 페이지 참조)

[LabVIEW 런타임 엔진 사용하기](#) (350 페이지 참조)

[컴파일된 객체 캐시](#) (73 페이지 참조)

[어플리케이션 개발하기 및 배포하기](#) (1448 페이지 참조)

비동기적으로 VI 호출하기

다음의 일반 사용 패턴을 포함한 비동기적으로 VI 호출하기에 대한 설명과 고려사항:

- 호출 및 수집: 비동기적으로 VI 호출하기 및 결과 수집하기
- 호출 및 무시: 결과 수집 없이 비동기적으로 VI 호출하기

- 병렬 실행을 위해 VI의 여러 인스턴스를 비동기적으로 호출하기



관련 정보

[비동기적으로 VI 호출하기](#) (934 페이지 참조)

[비동기적으로 VI 호출하기 및 결과 수집하기](#) (935 페이지 참조)

[결과 수집 없이 비동기적으로 VI 호출하기](#) (937 페이지 참조)

[병렬 실행을 위해 VI의 여러 인스턴스를 비동기적으로 호출하기](#) (938 페이지 참조)

비동기적으로 VI 호출하기 및 결과 수집하기

비동기적으로 호출하려는 VI의 참조를 열 때, 이후 VI의 결과를 수집할 것인지 결정해야 합니다. 이 토픽은 비동기화 호출 및 수집 VI 호출을 시작하는 방법과 이후 결과 수집 방법에 대해 설명합니다. 다른 프로그래밍 언어는 이러한 사용 모델을 fork-and-join으로 설명합니다. 또는 [결과 수집 없이 VI를 호출](#)할 수 있습니다.

사용 객체

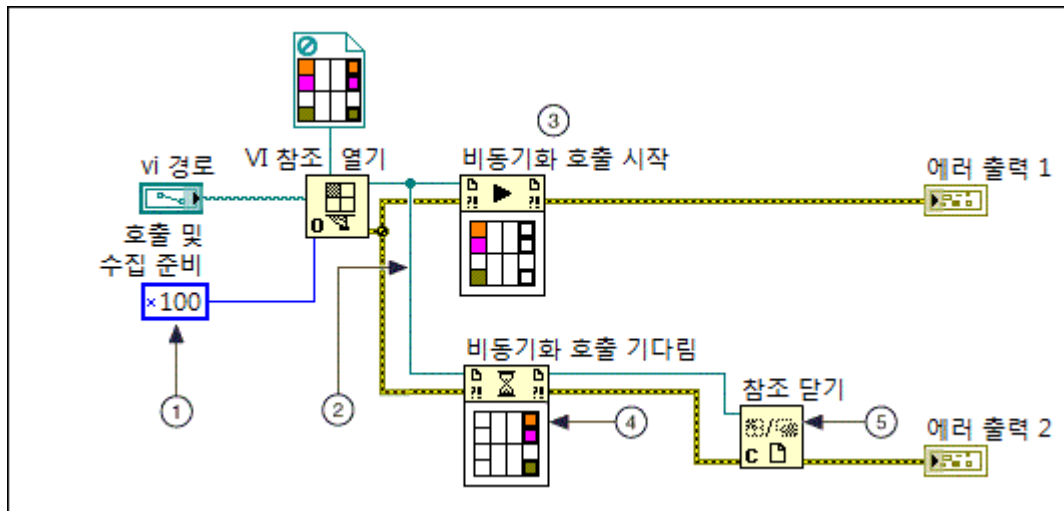
다음의 블록 다이어그램 객체를 사용하여 다이나믹하게 비동기화 VI 호출을 시작한 후 나중에 해당 VI의 출력을 수집합니다:

업격한 타입의 VI 참조	VI 참조 열기	비동기화 호출 시작	비동기화 호출 기다림	참조 닫기

생성

실행 단계

다음의 블록 다이어그램을 생성하여 비동기적으로 VI를 호출한 후 나중에 결과를 수집합니다.



위의 블록다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.



노트

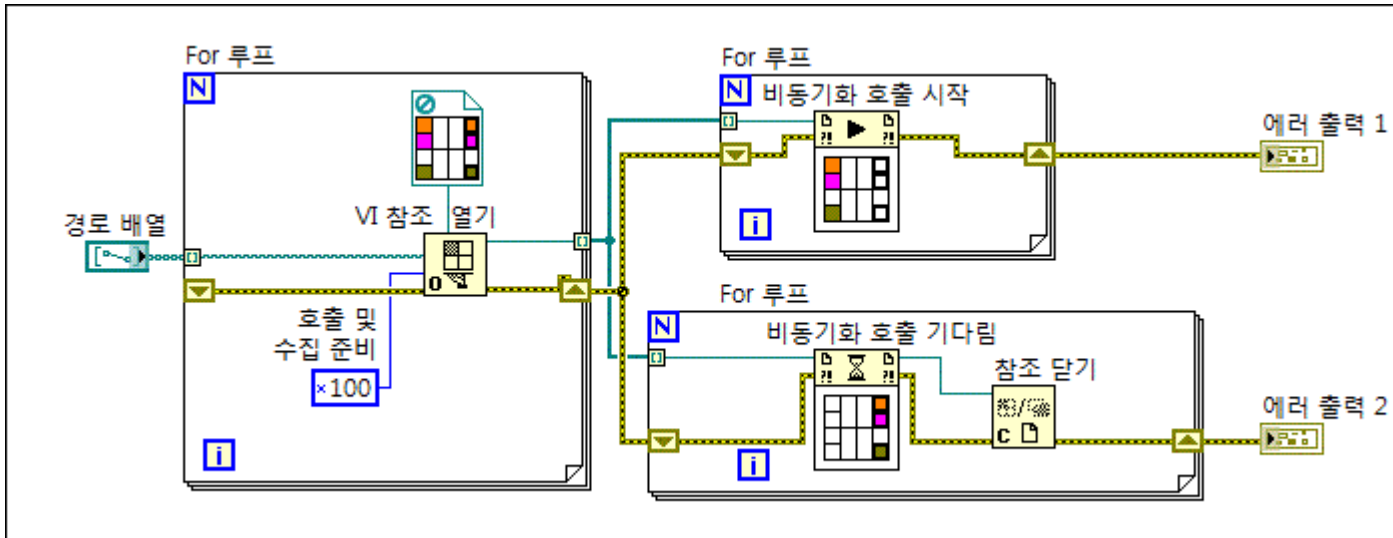
LabVIEW 도움말은 [VI 참조 열기] 함수의 모든 옵션 플래그를 16진수 형으로 디스플레이합니다. 그러므로, 항상 16진수로 숫자형 상수 입력의 **디스플레이 포맷을 변경**하여 올바른 값을 전달하도록 합니다.

유의점 및 권장사항

- VI 강제 종료하기— 다음과 같이 사용하던 VI를 강제 종료하는 경우 얻게 될 결과에 주의하십시오:
 - 호출 및 수집 VI 참조가 열려 있는 VI를 강제 종료하는 경우, LabVIEW는 자동으로 참조를 닫고 VI의 모든 실행 중인 인스턴스를 강제 종료합니다.
 - 호출 및 수집 VI 참조의 특정 인스턴스를 강제 종료하는 경우, 정확히 하나의 비동기화 호출 기다림 노드가 호출된 VI의 강제 종료를 알리는 에러를 반환합니다. 같은 VI 참조의 호출을 기다리는 모든 다른 비동기화 호출 기다림 노드는 계속해서 기다립니다.
 - 호출 및 수집 VI 참조에 대해 강제 종료 메소드를 호출하는 경우, LabVIEW는 현재 실행 중인 모든 VI의 인스턴스를 강제 종료합니다. 상응하는 수의 비동기화 호출 기다림 노드가 에러를 반환합니다.
- 호출 및 수집 VI 참조가 열려 있는 VI가 실행을 마치면, LabVIEW는 자동으로 참조를 닫고 VI의 모든 실행 중인 인스턴스를 강제 종료합니다.
- 같은 VI 참조에 대해 여러 비동기화 호출을 시작할 경우, 어떤 VI 호출이 특정 비동기화 호출 기다림 노드에 출력을 반환하는지 직접 확인할 수 없습니다. 어떤 VI 인스턴스가 반환되었는지 알 필요가 있을 경우, **VI 호출에 식별자를 지정**합니다.

변동: 고유한 VI 참조에 대한 여러 호출로부터 출력 수집하기

다음 블록다이어그램은 여러 VI를 비동기적으로 호출한 후 그 출력을 수집하는 템플릿을 제공합니다. 각 비동기화 호출은 고유한 VI 참조와 연관되기 때문에, 각 비동기화 호출 기다림 노드는 특정 VI가 끝날 때까지 기다립니다. 서브패널의 내용물이 개별 subVI의 프론트패널인 사용자 인터페이스를 생성할 때 이 템플릿이 특별히 유용합니다.



예제

VI를 비동기적으로 호출하고 나중에 그 결과를 수집하는 예제는 labview\examples\Application Control\VI Server\Asynchronous Call By Reference 디렉토리에 있는 Asynchronous Call and Collect VI를 참조하십시오.

 예제 열기  관련 예제 찾기

결과 수집 없이 비동기적으로 VI 호출하기

비동기적으로 호출하려는 VI의 참조를 열 때, 이후 VI의 결과를 수집할 것인지 결정해야 합니다. 이 토픽에서는 언제 또는 어떤 값을 VI가 반환하는지 추적하지 않고 비동기화 호출 및 무시 VI 호출을 시작하는 방법에 대해 설명합니다. 다른 프로그래밍 언어는 이러한 사용 모델을 fork-and-forget으로 설명합니다. 다른 방법으로, **VI를 호출하고 나중에 그 결과를 수집**할 수 있습니다.

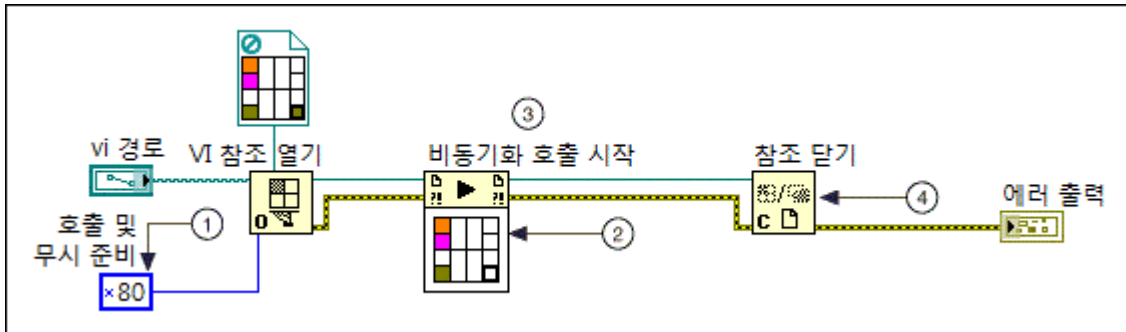
사용 객체

다음의 블록 다이어그램 객체를 사용하여 결과 수집 없이 VI를 비동기적으로 호출합니다:

엄격한 타입의 VI 참조	VI 참조 열기	비동기화 호출 시작	참조 닫기
			
생성			

실행 단계

다음의 블록 다이어그램을 생성하여 결과 수집 없이 비동기화 VI 호출을ダイナミック하게 시작합니다.



위의 블록 다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.



노트

LabVIEW 도움말은 [VI 참조 열기] 함수의 모든 옵션 플래그를 16진수 형으로 디스플레이합니다. 그러므로, 항상 16진수로 숫자형 상수 입력의 **디스플레이 포맷을 변경**하여 올바른 값을 전달하도록 합니다.

유의점 및 권장사항

- 종료하지 않는 VI는 호출하지 마십시오. 호출 및 무시 VI 참조에서 호출을 시작하면, LabVIEW는 호출 VI가 실행을 마칠 때 타겟 VI를 강제 종료하지 않습니다. 그 대신 호출 VI가 계속 실행 중인지 여부에 상관없이 타겟 VI는 완료될 때까지 실행됩니다. 그러므로, 타겟 VI가 무한 루프를 포함하거나 다른 이유로 실행을 멈출 수 없는 경우, 사용자가 LabVIEW를 닫을 때까지 메모리에 남아 성능에 부정적인 영향을 미칠 수도 있습니다.
- 호출 및 무시 VI 참조에 대해 강제 종료 메소드를 호출하는 경우, LabVIEW는 현재 실행 중인 모든 VI의 인스턴스를 강제 종료합니다.

예제

결과를 수집하지 않고 VI를 비동기적으로 호출하는 예제는 labview\examples\Application Control\VI Server\Asynchronous Call By Reference 디렉토리에 있는 Asynchronous Call and Forget VI를 참조하십시오.



예제 열기



관련 예제 찾기

병렬 실행을 위해 VI의 여러 인스턴스를 비동기적으로 호출하기

같은 VI에 대해 여러 비동기화 호출을 시작하면, LabVIEW는 호출을 직렬화하고, 강제로 한 번에 하나씩 실행시키도록 기본 설정되어 있어, 누적 실행 시간이 더 길어집니다. 다

음 과정에 따라 호출을 병렬로 실행하여, 어떤 호출도 다른 호출의 실행을 지연하는 일이 없도록 합니다.

사용 객체

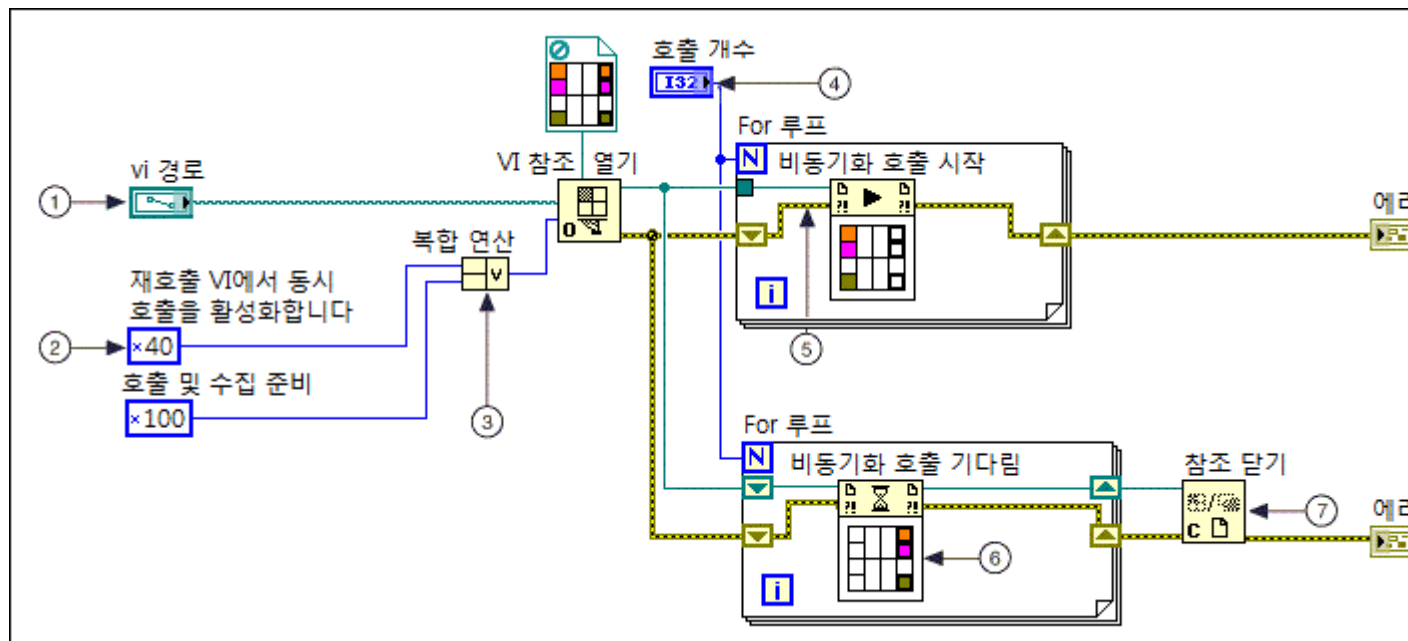
다음의 블록다이어그램 객체를 사용하여 다이나믹하게 **재호출 VI**에 대한 비동기화 호출을 시작한 후 나중에 해당 VI의 출력을 수집합니다:

엄격한 타입의 VI 참조	VI 참조 열기	비동기화 호출 시작	비동기화 호출 기다림	참조 닫기
				

만들기

실행 단계

다음의 블록다이어그램을 생성하여 재호출 VI에 대한 여러 비동기화 호출을 시작한 후 결과를 수집합니다.



위의 블록다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.



노트

LabVIEW 도움말은 [VI 참조 열기] 함수의 모든 옵션 플래그를 16진수 형으로 디스플레이합니다. 그러므로, 항상 16진수로 숫자형 상수 입력의 **디스플레이 포맷을 변경**하여 올바른 값을 전달하도록 합니다.



노트 **비동기화 호출의 결과를 무시**하려면 [비동기화 호출 기다림] 노드를 생략하고 0x100 옵션 플래그 대신에 0x80 옵션 플래그를 포함시킵니다.

유의점 및 권장사항

VI 서버 프로퍼티 및 메소드는 비동기화 VI 호출의 병렬 인스턴스를 수정할 수 없습니다. 0x40 VI 참조에서 VI 서버 프로퍼티나 메소드를 호출하는 경우, 해당 프로퍼티나 메소드는 동기화 호출 시작 노드가 실제로 호출하는 VI 복제를 수정할 수 없습니다. 그 대신 프로퍼티나 메소드는 타겟 VI 원본에 영향을 미칩니다. 비동기화 호출 시작 노드가 실제로 호출하는 VI 복제에 VI 서버 프로퍼티나 메소드를 적용하려면, 프로퍼티나 메소드를 타겟 VI 내에서 호출합니다.

예제

VI의 병렬 인스턴스를 비동기적으로 호출하여 실행 시간을 줄이는 예제는 labview\examples\Application Control\VI Server\Asynchronous Call By Reference 디렉토리에 있는 Asynchronous Call and Collect (Using Option 0x40) VI를 참조하십시오.



예제 열기



관련 예제 찾기

동일한 태스크를 해결하기 위한 몇 가지 동기화 및 비동기화 접근 방식의 비교는 labview\examples\Application Control\VI Server\Asynchronous Call By Reference 디렉토리에 있는 Benchmarking Asynchronous Calls VI를 참조하십시오.



예제 열기

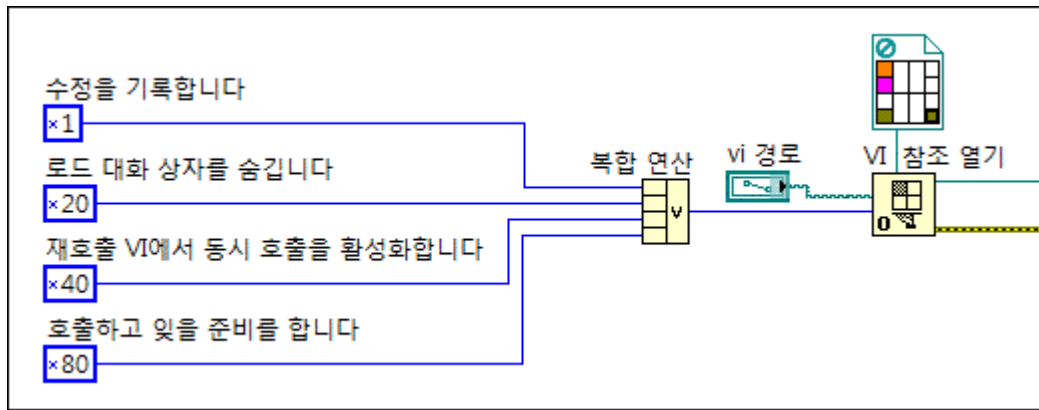


관련 예제 찾기

옵션 플래그를 단일 옵션값으로 결합하기

[VI 참조 열기] 함수의 옵션 입력은 참조 출력의 특정한 특성과 동작을 각각 지정하는 옵션 플래그의 조합을 받습니다. 각 옵션 플래그는 고유한 16진수로 나타납니다. 옵션 플래그가 숫자형이기 때문에, 옵션 플래그 몇 개를 하나의 숫자로 결합해서 반환된 VI 참조의 여러 특성을 지정할 수 있습니다.

[VI 참조 열기] 함수의 옵션 입력에 대해 여러 옵션 플래그를 하나의 값으로 결합하려면, 원하는 옵션 플래그를 사용하여 다음 블록 다이어그램을 생성합니다. 보다 자세한 내용은 다음 과정을 참조하십시오.



다음 단계를 따라 옵션 입력에 대해 여러 옵션 플래그를 하나의 값으로 결합합니다.

1. [VI 참조 열기] 함수의 참조 출력에 어떤 특성을 지정할 것인지 결정합니다.

설명

수정사항을 기록합니다.

편집을 위해 템플릿을 엽니다.

사용자에게 변경 저장을 요청합니다.

재호출 실행을 준비합니다.

찾을 수 없는 SubVI의 위치를 알아내도록 요청합니다.

로드 대화 상자를 숨깁니다.

재호출 VI에서 동시 호출 활성화

호출 및 무시를 준비합니다.

호출 및 수집을 준비합니다.

특정한 옵션 플래그에 대한 더 자세한 내용은 전체 옵션 테이블을 참조하십시오.

2. 원하는 각 특성에 대해 숫자형 상수를 블록다이어그램에 추가합니다.

팔레트에서 찾기

3. 각 숫자형 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 기수를 선택합니다.
4. 기수를 클릭한 후 바로 가기 메뉴에서 16진수를 선택합니다.
5. 각 숫자형 상수의 값을 원하는 특성에 대응하는 옵션 플래그에 따라 설정합니다.
6. [복합 연산] 함수를 블록다이어그램에 추가합니다.

팔레트에서 찾기

7. 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모드 변경 >> OR를 선택하여 연산 모드를 OR로 설정합니다.

8. 크기 조정 핸들 하나을 클릭하고 끌어서 [복합 연산] 함수를 확장하여 각 옵션 플러그에 대해 충분한 입력 터미널을 디스플레이합니다.
9. 각 숫자형 상수를 [복합 연산] 함수의 입력 터미널에 연결합니다.
10. [복합 연산] 함수의 출력을 [VI 참조 열기] 함수의 옵션 입력에 연결합니다.

VI 서버 설정하기

다음 단계를 따라 VI 서버를 설정합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. (옵션) 기본 VI 서버 설정은 대부분의 어플리케이션에 적절합니다. 이 기본 설정을 변경하려는 경우 항목 리스트에서 VI 서버를 선택합니다. 기본 설정은 LabVIEW를 호출하기 위해서 ActiveX를 허용하지만 프로토콜로 TCP/IP를 허용하지는 않습니다. TCP/IP를 다른 LabVIEW 어플리케이션과 함께 사용하려면, TCP/IP 확인란에 확인 표시를 하고 머신 접근 셋팅을 설정합니다.
3. VI에 대한 브라우저 접근을 컨트롤하거나 원격 어플리케이션이 컨트롤할 수 있는 VI를 지정하려는 경우, 머신 접근을 설정하고 VI 반출 리스트 옵션을 설정합니다.
4. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.



노트

LabVIEW 프로젝트의 타겟이 VI 서버를 지원하는 경우, 타겟의 프로퍼티 대화 상자를 사용하여 타겟에 VI 서버를 설정할 수 있습니다. 예를 들어, 내 컴퓨터 프로퍼티 대화 상자를 사용하여 내 컴퓨터 타겟의 VI 서버를 설정합니다.

VI 호출 설정하기

VI 호출 설정 대화 상자를 사용하여 언제 subVI를 로드하는지 설정할 수 있습니다.

다음 단계를 따라 VI 호출을 설정합니다.

1. 블록 다이어그램에 VI를 놓습니다.
2. VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 호출 설정을 선택하여 VI 호출 설정 대화 상자를 디스플레이합니다.



노트

이 아이템은 VI 서버 호출을 지원하는 **어플리케이션 인스턴스**의 열린 VI에서만 사용 가능합니다. 함수, 익스프레스 VI, 다형성 VI 또는 FPGA 타겟처럼 VI 서버 호출을 지원하지 않는 어플리케이션 인스턴스의 열린 VI에서는 사용할 수 없습니다.

1. SubVI를 로드할 때 다음 옵션들 중의 하나를 선택합니다:
 - 호출과 함께 로드—호출자 VI가 로드함과 동시에 SubVI를 로드합니다. 이는 SubVI가 로드를 하는 방식과 동일합니다.

- 개별 호출마다 다시 로드—SubVI가 이미 메모리 상에 있지 않은 경우, 호출자 VI가 SubVI를 호출할 때마다 SubVI를 로드합니다. 이 옵션을 선택하더라도, 호출 이후에 SubVI가 메모리 상에 남아있다고 보장되는 것은 아닙니다.
- 첫번째 호출에서 로드하고 유지—호출자 VI가 첫번째로 SubVI를 호출할 때에만 SubVI를 로드합니다. 호출자 VI의 크기가 큰 경우, 대화 상자에서 첫번째 호출에서 로드하고 유지 옵션을 선택하여 로드 시간과 메모리를 절약할 수 있습니다. 이 옵션을 선택하면 호출자 VI가 SubVI를 필요로 할 때까지 SubVI가 로드되지 않습니다. 개별 호출마다 다시 로드 옵션 또는 첫번째 호출에서 로드하고 유지 옵션을 선택하는 경우, 호출 VI의 블록다이어그램이 메모리에 있으면 LabVIEW는 계속 SubVI를 로드합니다. 호출자 VI를 편집 모드에서 열 경우, LabVIEW는 호출자 VI의 블록다이어그램을 로드하므로 subVI도 함께 로드됩니다. SubVI가 로드되는 것을 피하려면 호출자 VI의 블록다이어그램을 열지 마십시오. 호출자 VI를 subVI로 호출하고, 호출자 VI의 프런트패널과 블록다이어그램을 닫습니다.



노트 개별 호출마다 다시 로드 또는 첫 번째 호출에서 로드하고 유지를 선택하면, SubVI를 호출하도록 설정된 [참조에 의한 호출] 노드로 SubVI가 대체됩니다. 참조에 의한 호출 노드의 기호는 사용자가 선택한 모드에 따라 달라집니다. 이 옵션 중 어느 하나라도 선택한다면 더 이상 subVI를 찾을 수 없게 됩니다. 이 옵션들은 subVI로의 정적 링크를 제거합니다.

1. 호출 설정을 저장하기 위해서 확인 버튼을 클릭합니다.

SubVI로부터 프런트패널 객체를 프로그램적으로 컨트롤하기

다음 단계를 따라 컨트롤 참조를 SubVI에 전달하고 그것을 참조된 컨트롤이나 인디케이터의 프로퍼티나 메소드를 변경시키기 위해 사용합니다.

1. 참조를 생성하려는 Main VI의 프런트패널 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>참조를 선택합니다. LabVIEW는 엄격한 타입의 컨트롤 참조를 마우스 커서에 연관시킵니다. Main VI의 블록다이어그램에 참조를 놓습니다. 또한, 컨트롤 팔레트에서 **컨트롤 참조 번호**를 선택하고, SubVI의 프런트패널에 추가하고, 같은 타입의 객체를 프런트패널의 컨트롤 참조 번호 셀에 참조된 객체로 추가하여, 컨트롤 참조를 생성할 수 있습니다. 이것은 엄격한 타입이 아닌 컨트롤 참조를 생성합니다.
2. SubVI를 열고 새 VI를 생성하여 컨트롤 참조를 전달합니다. Main VI의 블록다이어그램으로부터, 컨트롤 참조를 SubVI의 프런트패널로 끌어와서 SubVI 프런트패널에 컨트롤 참조를 위한 컨트롤 참조 번호 컨트롤을 생성합니다. 상응하는 컨트롤 참조 터미널이 SubVI 블록다이어그램에 나타납니다.



노트

블록다이어그램으로 컨트롤 참조를 끌어오면, 컨트롤 참조 번호 컨트롤을 생성하기 보다는 컨트롤과 컨트롤 참조를 복사합니다.

1. 다음 단계를 따라 프로퍼티 노드나 인보크 노드를 생성하여 컨트롤 참조와 연관된 프런트패널 객체의 프로퍼티나 메소드를 변경합니다.

1. VI의 블록다이어그램의 컨트롤 참조에서 마우스 오른쪽 버튼을 클릭합니다.
2. 생성>>프로퍼티(타입 이름) 또는 생성>>메소드(타입 이름)을 선택하고 바로 가기 메뉴에서 프로퍼티나 메소드를 선택합니다.
3. SubVI의 블록다이어그램에 노드를 놓기 위해 클릭합니다.
4. 컨트롤 참조를 프로퍼티 노드나 인보크 노드의 참조 터미널에 연결합니다. 프로퍼티 노드나 인보크 노드는 Main VI에서 컨트롤을 참조합니다.

또한, 함수 팔레트에서 프로퍼티 노드나 인보크 노드를 선택하여 블록다이어그램에 위치시킵니다. 노드에서 마우스 오른쪽 버튼을 클릭하고 프로퍼티나 메소드 바로 가기 메뉴에서 프로퍼티나 메소드를 선택합니다.

2. **커넥터 팬에서 터미널을 설정**하는데 이것은 컨트롤 참조를 포함합니다.
3. Main VI에서 SubVI를 블록다이어그램에 추가하고 컨트롤 참조를 SubVI의 컨트롤 참조 터미널에 연결합니다.

SubVI에서 프로그램적으로 프런트패널 객체를 컨트롤하는 예제는 labview \examples\Application Control\VI Server\Control References에 있는 Control References VI를 참조하십시오.



예제 열기



관련 예제 찾기

다이나믹하게 VI를 호출하는 엄격한 타입의 VI 참조 생성하기

엄격한 타입의 VI 참조, 즉 **엄격한 타입 정의**를 포함하는 참조를 생성하여, 참조에 의한 호출 노드나 비동기화 호출 시작 노드로 다이나믹하게 VI를 호출합니다. 엄격한 타입의 VI 참조는 호출하는 VI와 그 커넥터 팬을 모두 식별합니다.

다음 단계를 따라 참조에 의한 호출 노드 또는 비동기화 호출 시작 노드와 함께 사용할 엄격한 타입의 VI 참조를 생성합니다.

1. 블록다이어그램에 [VI 참조 열기] 함수를 추가합니다.
2. 엄격한 타입의 참조를 생성하기 위해 사용하려는 VI의 경로를 [VI 참조 열기] 함수의 VI 경로 입력에 연결합니다.
3. [VI 참조 열기] 함수의 타입 지정자 VI 참조 번호 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택합니다.
4. 타입 지정자 VI 참조 번호 상수에서 마우스 오른쪽 버튼을 클릭한 후, VI 서버 클래스 선택>>탐색을 선택하고 파일 대화 상자가 나타나면 VI를 선택합니다. 또한, 프런트패널이나 블록다이어그램 윈도우의 오른쪽 위 코너에서 VI 아이콘을 선택하고 아이콘을 타입 지정자 VI 참조 번호 상수로 끌 수 있습니다. VI를 선택한 후, 참조

가 엄격한 타입이라는 것을 알리는 취소선 아이콘 이 참조의 왼쪽 위 모서리에 나타납니다.



노트

선택한 커넥터 팬은 반드시 [VI 참조 열기] 함수의 vi 경로 입력에 연결한 경로에 있는 VI의 커넥터 팬과 일치해야 합니다. 같은 VI를 두 입력 모두에 사용할 수 있습니다.

이제 [VI 참조 열기] 함수의 vi 참조 출력은 엄격한 타입의 VI 참조입니다.

VI 서버 어플리케이션 생성하기

LabVIEW VI 서버를 사용하여 프로그램적으로 프런트패널 객체를 컨트롤하고, VI, LabVIEW를 컨트롤하며, 컴퓨터에 또는 네트워크를 통하여 원격으로ダイナミック하게 VI를 로드하고, 편집하고, 실행할 수 있습니다. VI에 대한 브라우저 접근을 컨트롤할 수 있고 원격 어플리케이션이 어떤 VI를 컨트롤할 수 있는지 설정할 수 있습니다.

다음 단계를 따라 VI 서버 어플리케이션을 생성합니다.

1. TCP/IP 프로토콜을 허용하도록 VI 서버를 설정합니다.
2. [어플리케이션 참조 열기] 함수를 사용하여 로컬 또는 원격 어플리케이션 인스턴스의 참조를 엽니다.



노트

LabVIEW 프로젝트 또는 LabVIEW 프로젝트의 타겟에서 작업하는 경우처럼, 여러 개의 어플리케이션 인스턴스가 동시에 열려 있는 경우, 여러 VI 서버가 서로 다른 포트에서 리스할 수 있습니다. 컴퓨터 이름과 포트 또는 서비스 이름을 지정하여 특정한 어플리케이션 인스턴스에 대한 어플리케이션 참조를 엽니다.

1. [VI 참조 열기] 함수를 사용하여 어플리케이션 인스턴스를 위해 이미 메모리에 존재하는 로컬 또는 원격 컴퓨터의 VI의 참조를 열거나 디스크에서 다이내믹하게 VI를 로드합니다. 그러나, VI 경로에 대한 입력이 파일 경로인 경우, 디스크로부터 VI를 로드할 수 있도록 사용자 인터페이스가 유효 상태가 될 때까지 이 함수는 대기한다는 점을 유념하십시오.
2. [프로퍼티 노드]를 사용하여 프로퍼티를 얻고 설정하거나 [인보크 노드]를 사용하여 메소드를 불러옵니다.
 - 프로퍼티 노드
 - 인보크 노드

또한, 참조에 의한 호출 노드를 사용하여 다이내믹하게 로드된 VI를 호출할 수 있습니다.

3. [참조 닫기] 함수를 사용하여 열려 있는 모든 참조를 닫습니다.

VI 서버를 사용한 예제는 labview\examples\viserver를 참조하십시오.

관련 예제 찾기



관련 정보

[SubVI로부터 프런트패널 객체를 프로그램적으로 컨트롤하기](#) (943 페이지 참조)

[웹 브라우저 접근 허용 및 거부하기](#) (1518 페이지 참조)

[어떤 VI를 원격으로 조정할 수 있는지 지정하기](#) (947 페이지 참조)

[VI 서버 설정하기](#) (942 페이지 참조)

[어플리케이션 인스턴스로 작업하기](#) (308 페이지 참조)

[LabVIEW에서 프로젝트 관리하기](#) (290 페이지 참조)

[다이나믹하게 VI 로딩 및 호출하기](#) (170 페이지 참조)

프로퍼티 노드와 인보크 노드에서 컨트롤 참조 SubVI 생성하기

다음 단계에 따라 프로퍼티 노드 또는 인보크 노드에서 직접 컨트롤 참조를 포함하는 SubVI를 생성합니다.

1. 컨트롤 참조 SubVI를 원하는 프런트패널 객체 또는 해당 블록 다이어그램 터미널에서 마우스 오른쪽 버튼을 클릭한 후 생성»프로퍼티 노드 또는 생성»인보크 노드를 선택하고 바로 가기 메뉴에서 프로퍼티 또는 메소드를 선택하여 객체에 함축적으로 연결된 **프로퍼티 노드 또는 인보크 노드를 생성**합니다.
2. 위치 도구를 사용하여 새로운 프로퍼티 노드 또는 인보크 노드를 선택하고 편집»SubVI 생성을 선택합니다. LabVIEW는 Main VI의 블록 다이어그램의 컨트롤에 대한 컨트롤 참조를 생성하고, 프로퍼티 노드 또는 인보크 노드에 연결된 컨트롤 참조 번호를 포함하는 SubVI를 생성하고, Main VI의 컨트롤 참조를 새로운 SubVI의 컨트롤 참조 번호 입력 터미널에 연결합니다.



노트

한 번에 여러개의 프로퍼티 노드 또는 인보크 노드를 선택하여 여러개의 컨트롤 참조 번호와 프로퍼티 노드 또는 인보크 노드를 가지는 SubVI를 생성합니다.

1. SubVI를 더블 클릭하여 엽니다. 프로퍼티 노드 또는 인보크 노드와 SubVI의 다른 원소를 다른 VI를 수정할 때와 같이 수정합니다.

다이나믹하게 VI 호출하기

다음 단계를 따라 [참조에 의한 호출] 노드로 다이나믹하게 VI를 호출합니다.



노트

비동기적으로 VI를 호출하려는 경우, [참조에 의한 호출] 노드 대신 [비동기화 호출 시작] 노드를 사용합니다.

1. 블록다이어그램에 [VI 참조 열기] 함수를 추가합니다.
2. [VI 참조 열기] 함수를 사용하여 엄격한 타입의 참조를 생성합니다.
3. 블록다이어그램에 [참조에 의한 호출] 노드를 추가합니다.
4. [VI 참조 열기] 함수의 VI 참조 출력을 [참조에 의한 호출] 노드의 참조 입력에 연결합니다.
5. 블록다이어그램에 [참조 닫기] 함수를 추가합니다.
6. [참조에 의한 호출] 노드의 참조 출력 출력을 [참조 닫기] 함수의 참조 입력에 연결합니다.
7. 블록다이어그램의 모든 에러 입력과 에러 출력 터미널을 연결합니다.



관련 정보

[다이나믹하게 VI 로딩 및 호출하기](#) (170 페이지 참조)

[비동기적으로 VI 호출하기](#) (171 페이지 참조)

[다이나믹하게 VI를 호출하는 엄격한 타입의 VI 참조 생성하기](#) (944 페이지 참조)

[VI에서 에러 I/O 설정하기](#) (1388 페이지 참조)

어떤 VI를 원격으로 조정할 수 있는지 지정하기

옵션 대화 상자의 VI 서버 페이지에서 접근 가능한 서버 리소스 섹션에 있는 모든 박스에 확인 표시를 하여 원격 어플리케이션이 VI를 컨트롤하도록 허용할 수 있습니다. VI 반출 리스트 섹션을 사용하여 원격 어플리케이션이 컨트롤할 VI를 지정합니다. 개별적인 VI나 그룹 VI, 또는 디렉토리 경로에 대한 접근을 허용하고 거부하는 VI 반출 리스트를 생성합니다. 또한, VI 접근 리스트 프로퍼티를 사용하여 VI에 대한 접근을 프로그램적으로 허용하거나 거부할 수 있습니다.

원격 어플리케이션이 VI를 컨트롤하려고 시도할 때, VI 서버는 VI 이름을 VI 반출 리스트에 있는 엔트리와 비교해서 접근을 허용할 것인지 결정합니다. VI 반출 리스트의 엔트리가 VI 이름과 일치하면, VI 서버는 사용자가 어떻게 엔트리를 설정했는지에 따라 해당 VI의 이미지에 대한 접근을 허용하거나 거부합니다. 기본적으로, 원격 어플리케이션은 모든 VI를 컨트롤할 수 있습니다. 새로운 VI 반출 리스트 엔트리를 추가하거나, 기존의 엔트리를 편집하거나, 엔트리를 제거할 때, 엔트리에 올바른 구문, 순서, 와일드카드를 사용합니다.



관련 정보

[VI의 구문 열거하기](#) (1528 페이지 참조)

[새 VI 반출 리스트 엔트리 추가하기](#) (1410 페이지 참조)

[기존 VI 반출 리스트 엔트리 편집하기](#) (1410 페이지 참조)

[VI 반출 리스트에서 VI 삭제하기](#) (1410 페이지 참조)

[VI의 구문 열거하기](#) (1528 페이지 참조)

VI 스크립팅

VI 스크립팅

필수: VI 스크립팅

여러 일반 스크립팅 태스크의 수행 방법을 알아봅니다.

- VI 스크립팅 활성화하기에 대한 설명
- 타겟 VI 검사하기, 편집하기 및 생성하기에 대한 설명과 예제
- VI 스크립팅을 사용한 새 VI 생성, 블록다이어그램에 객체 추가, 추가한 객체의 위치 변경, 객체 간 연결에 대한 기본 길라잡이



관련 정보

[VI 스크립팅 활성화하기](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅 활성화하기](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅 길라잡이](#) (978 페이지 참조)

VI 스크립팅 활성화하기

VI 스크립팅 팔레트를 디스플레이하고 관련된 프로퍼티와 메소드를 사용하려면, [VI 스크립팅](#)을 활성화해야 합니다. 다음 단계를 따라 LabVIEW에서 VI 스크립팅을 활성화합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. VI 스크립팅 함수, 프로퍼티, 메소드 보이기 확인란에 확인 표시를 합니다.



노트

기본적으로 기본 도움말 윈도우에 추가적인 VI 스크립팅 정보 디스플레이 확인란이 선택되어 있습니다. 이 옵션은 VI를 탐색하고 편집하는데 사용할 수 있는 [VI 스크립팅 커넥터 팬 정보](#)를 디스플레이합니다.

4. 확인 버튼을 클릭하여 변경을 적용하고 옵션 대화 상자를 닫습니다.

일반적인 VI 스크립팅 태스크

VI 스크립팅을 사용하여 타겟 VI를 검사하거나, 편집하거나, 생성할 때, 여러 작은 VI 스크립팅 태스크를 결합하여 보다 큰 스크립팅 목적을 달성하는 경우가 종종 있습니다. 이 토픽에서는 가장 일반적인 VI 스크립팅 태스크 일부와 그러한 태스크를 완료하는데 사용할

수 있는 함수, 프로퍼티, 메소드에 대한 정보 및 예제에 해당하는 링크들이 나열됩니다. 보다 복잡한 일부 태스크는 별도의 토픽에서 찾을 수 있습니다.

VI 및 VI 객체의 참조 얻기

- 타겟 VI의 참조 얻기
- 모르는 타겟 VI에서 객체의 참조 얻기
- 알고 있는 VI에서 객체의 참조 얻기
- 노드에서 와이어로 탐색하기
- 와이어에서 노드로 탐색하기
- 구조 탐색하기
- VI의 프런트패널과 블록다이어그램 사이에서 탐색하기

VI 및 VI 객체 생성하기

- 새 VI 생성하기
- 템플릿에서 새 VI 생성하기
- 프런트패널 또는 블록다이어그램에 객체 추가하기
- 객체 복사하기
- 터미널에서 상수, 컨트롤 또는 인디케이터 생성하기
- 여러 다이어그램이 있는 구조의 프레임을 추가하고, 복제하고, 제거하기
- 객체를 다른 객체에 연결하기

VI 및 VI 객체 검사하기 및 편집하기

- 커넥터 팬 설정하기
- 객체 이동하기
- 객체 대체하기
- 객체 삭제하기
- 잘라내고, 복사하고, 붙여넣을 객체 선택하기
- 노드의 라벨과 상호작용하기

부수적인 내역

- 작업 취소하기
- 블록다이어그램 정리
- 객체 태그 사용하기
- VI의 프런트패널이나 블록다이어그램 보이기
- 참조 닫기

VI의 프런트패널과 블록다이어그램 사이에서 탐색하기

다음과 같은 프로퍼티를 사용하여 VI의 프런트패널과 블록다이어그램 사이를 프로그램적으로 탐색합니다:

- 컨트롤:터미널—프런트패널 컨트롤에서 그에 대응하는 블록다이어그램의 터미널로 탐색 가능
- 컨트롤 터미널:컨트롤—블록다이어그램 컨트롤 터미널에서 그에 대응하는 프런트패널 컨트롤로 탐색 가능

새 VI 생성하기

[새 VI] 함수를 사용하여 프로그램적으로 새 VI를 생성합니다.

프로그램적으로 새 VI를 생성하고 여는 방법에 대한 자세한 설명은 VI 스크립팅 길라잡이를 참조하십시오.

터미널에서 상수, 컨트롤 또는 인디케이터 생성하기

다음 메소드를 사용하여 터미널에서 상수, 컨트롤 또는 인디케이터를 프로그램적으로 생성합니다:

- 터미널:상수 생성
- 터미널:컨트롤 생성
- 터미널:인디케이터 생성

여러 다이어그램이 있는 구조의 프레임을 추가하고, 복제하고, 제거하기

다음 메소드를 사용하여 여러 다이어그램이 있는 구조의 프레임과 프로그램적으로 상호작용합니다:

- 멀티프레임 구조:프레임 추가
- 멀티프레임 구조:프레임 복제
- 멀티프레임 구조:프레임 제거
- 플랫 시퀀스:프레임 추가
- 플랫 시퀀스:프레임 제거

객체를 다른 객체에 연결하기

다음 메소드를 사용하여 프로그램적으로 객체를 다른 객체에 연결합니다:

- 터미널:와이어 연결
- 노드:와이어(복수) 연결

터미널:와이어 연결 메소드를 사용하여 두 객체를 연결하는 예제는 labview \examples\Application Control\VI Scripting\Creating VIs 디렉토리에 있는 Creating New VI From Scratch VI를 참조하십시오.

 예제 열기  관련 예제 찾기

노드:와이어(복수) 연결 메소드를 사용하여 두 객체를 서로 연결하는 예제는 VI 스크립팅 길라잡이를 참조하십시오.

객체를 이동하거나 복사하기

G객체:이동 메소드를 사용하여 객체를 프로그램적으로 이동합니다.

객체 대체하기

다음 중 하나의 메소드를 사용하여 프로그램적으로 객체를 대체합니다:

- G객체:대체
- G객체:속성 없음 대체

대체 메소드를 사용하여 객체를 대체하는 예제는 labview\examples \Application Control\VI Scripting\Creating VIs 디렉토리에 있는 Creating New VI From Template VI를 참조하십시오.

 예제 열기  관련 예제 찾기

객체 삭제하기

일반:삭제 메소드를 사용하여 객체를 프로그램적으로 삭제합니다.

삭제 메소드를 사용하여 와이어를 삭제하는 예제는 labview\examples \Application Control\VI Scripting\Managing References 디렉토리에 있는 Closing References VI를 참조하십시오.

 예제 열기  관련 예제 찾기

잘라내고, 복사하고, 붙여넣을 객체 선택하기

다음과 같은 프로퍼티를 사용하여 선택한 객체에 프로그램적으로 접근합니다:

- G객체:선택됨
- 패널:선택 리스트[]
- 최상위 레벨 다이어그램:선택 리스트[]

다음과 같은 메소드를 사용하여 선택사항을 프로그램적으로 편집합니다:

- 패널 메소드
- 최상위 레벨 다이어그램 메소드

노드의 라벨과 상호작용하기

노드:라벨 프로퍼티를 사용하여 노드의 라벨과 프로그램적으로 상호작용합니다.

작업 취소하기

다음 메소드를 사용하여 VI 스크립팅 작업을 프로그램적으로 취소합니다:

- VI:트랜잭션:실행 취소
- VI:트랜잭션:다시 실행
- VI:트랜잭션:실행 취소 시작
- VI:트랜잭션:실행 취소 끝
- VI:트랜잭션:실패

트랜잭션 메소드를 사용하여 프로그램적으로 작업을 취소하는 예제는 labview \examples\Application Control\VI Scripting\Supporting Undo 디렉토리에 있는 Supporting Undo VI를 참조하십시오.

 예제 열기  관련 예제 찾기

블록다이어그램 정리

VI:블록다이어그램:정리 메소드를 사용하여 타겟 VI의 블록다이어그램을 정리합니다.

객체 태그 사용하기

다음 메소드를 사용하여 객체 태그를 프로그램적으로 설정합니다:

- 일반:태그.태그 얻기
- 일반:태그.태그 설정

태그 메소드를 사용하여 객체에 태그를 추가하고 태그된 객체를 찾는 예제는 labview \examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Tagging VI를 참조하십시오.

 예제 열기  관련 예제 찾기

VI의 프런트패널이나 블록다이어그램 보이기

다음과 같은 프로퍼티와 메소드를 사용하여 타겟 VI의 프런트패널이나 블록다이어그램을 프로그램적으로 보이게 하거나 위치를 정합니다:

- 프런트패널:열기
- 프런트패널 윈도우:윈도우 경계
- 블록다이어그램 윈도우:열기
- 블록다이어그램 윈도우:윈도우 경계
- 블록다이어그램 윈도우:원형

새로 생성된 VI의 프런트패널과 블록다이어그램을 열고 위치를 정하는 예제는 `labview\examples\Application Control\VI Scripting\Creating VIs` 디렉토리에 있는 `Creating New VI From Scratch VI`를 참조하십시오.

 예제 열기  관련 예제 찾기



관련 정보

[VI 참조 얻기](#) (954 페이지 참조)

[VI 스크립팅을 사용하여 모르는 타겟 VI의 객체 참조 얻기](#) (958 페이지 참조)

[VI 스크립팅을 사용하여 이미 알고 있는 타겟 VI의 객체 참조 얻기](#) (956 페이지 참조)

[VI 스크립팅을 사용하여 노드에서 와이어로 탐색하기](#) (972 페이지 참조)

[VI 스크립팅을 사용하여 와이어에서 노드로 탐색하기](#) (974 페이지 참조)

[VI 스크립팅을 사용하여 구조 탐색하기](#) (977 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅을 사용하여 템플릿에서 새 VI 생성하기](#) (966 페이지 참조)

[VI 스크립팅을 사용하여 VI에 객체 추가하기](#) (961 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅을 사용하여 커넥터 팬 설정하기](#) (963 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[일반적인 VI 스크립팅 태스크](#) (948 페이지 참조)

[VI 스크립팅을 사용하여 VI 및 VI 객체의 참조 달기](#) (969 페이지 참조)

[길라잡이: VI 스크립팅을 사용하여 VI를 생성하고 열기](#) (979 페이지 참조)

[길라잡이: VI 스크립팅을 사용해서 객체 연결하기](#) (984 페이지 참조)

VI 참조 얻기

필수: [VI 스크립팅](#)

VI 스크립팅을 사용하여 VI 및 VI 객체를 검사하거나, 수정하거나, 생성하려면, 보통의 경우 상호작용하려는 타겟 VI의 참조를 얻어 시작해야 합니다. 그런 다음, 해당 참조를 사용하여 타겟 VI 내 객체의 참조를 얻거나 타겟 VI에 새 객체를 추가할 수 있습니다.

특정 용도에 맞게 VI 참조를 얻는 최상의 방법을 확인하려면 다음 테이블을 참조하십시오.

타겟 VI의 타입	타겟 VI의 참조 얻는 방법
편집 시그 경로를 아는 기존 타겟 VI	정적 VI 참조 
실행 시 선택하려는 기존 타겟 VI [참조에 의한 호출] 노드로 호출하려는 타겟 VI	VI 참조 열기 
새 VI	새 VI 
템플릿 VI에서 복사한 내용이 포함된 새 VI	템플릿에서 새 VI를 생성합니다.

유의점 및 권장사항

- 가능한 **복잡한 새 VI는 템플릿에서 생성** — 한 번에 하나씩 각 구성요소 객체를 개별적으로 생성하여 프로그램적으로 새 VI를 생성할 수도 있지만, 템플릿에서 새 VI를 생성한 후 코드의 일부를 약간 수정하는 것이 훨씬 더 용이합니다.
- 타겟 VI의 디스플레이는 옵션—VI를 프로그램적으로 검사하거나 편집하기 위해 타겟 VI의 프런트패널이나 블록다이어그램을 디스플레이할 필요는 없습니다. 그러나 다음과 같은 프로퍼티와 메소드를 사용하여 타겟 VI의 서로 다른 부분을 디스플레이할 수 있습니다:
 - 프런트패널:열기
 - 블록다이어그램 윈도우:열기
- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 객체 참조 닫기**를 참조하십시오.

VI 스크립팅을 사용하여 VI 객체 참조 얻기

필수: VI 스크립팅

VI 스크립팅을 사용하여 프런트패널 또는 블록다이어그램 객체를 검사하거나, 수정하거나, 생성할 때, 상호작용하려는 객체의 참조를 얻어야 합니다. 그런 다음 이 참조를 프로퍼티노드 또는 인보크 노드에 연결하여 참조된 객체에 대한 정보를 얻거나 설정할 수 있습니다.

특정 용도에 맞는 VI 참조 얻기의 가장 좋은 방법을 확인하려면 다음 리스트를 참조하십시오. 리스트에 열거된 방식들은 아래로 내려갈수록 난이도가 높아집니다.

- [VI 객체 참조 열기] 함수를 사용합니다. [VI 객체 참조 열기] 함수를 사용하면 타겟 VI에서 라벨로 특정 객체를 검색할 수 있습니다. 템플릿 VI와 같이 **내용을 아는 타겟 VI에서 객체의 참조를 얻으려면** 이 방식을 사용합니다.
- [G객체의 탐색] VI를 사용합니다. [G객체의 탐색] VI를 사용하면 타겟 VI에서 특정 클래스의 인스턴스인 객체를 모두 검색할 수 있습니다. **모르는 타겟 VI에서 객체의 참조를 얻으려면** 이 방식을 사용합니다. 또한 동일한 클래스의 인스턴스인, 타겟 VI의 여러 객체와 상호작용하려면 이 방식을 사용합니다.
- 타겟 VI 내의 객체 간 연결을 탐색합니다. 타겟 VI 내의 **객체 간 연결 탐색**을 통해 한 객체의 참조를 사용하여 연결된 객체의 참조를 얻을 수 있습니다. 전술한 방식으로 직접 접근할 수 없는 객체의 경우 그 참조를 얻으려면 이 방식을 사용하십시오.

유의점 및 권장사항

- 열어 놓은 참조를 모두 닫습니다. 지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 객체 참조 닫기**를 참조하십시오.

VI 스크립팅을 사용하여 이미 알고 있는 타겟 VI의 객체 참조 얻기

필수: VI 스크립팅

템플릿에서 새 VI를 생성할 때, 사용자는 보통 LabVIEW가 템플릿에서 새 VI로 복사한 코드를 조금 수정하려 합니다. 이러한 수정을 위해서는, 검사하거나 수정하려는 새 VI에서 객체의 참조를 얻어야 합니다. 이 객체 참조를 얻고 난 후, 이들을 프로퍼티 노드나 인보크 노드에 연결하여, 참조된 객체에 대한 정보를 얻거나 설정할 수 있습니다. 새 VI에서 객체의 참조를 얻는 가장 쉬운 방법은 [VI 객체 참조 열기] 함수를 사용하는 것입니다. 그러나, 이 함수를 사용하려면, 새 VI의 원하는 객체에 고유 라벨이 있어야 합니다. 원하는 객체에 고유 라벨이 없는 경우, 보다 복잡한 또 다른 과정을 거쳐야만 **라벨이 없거나 모르는 이런 객체의 참조를 얻을** 수 있습니다.

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

VI 객체 참조 열기	클래스 지정자 상수	참조 닫기
		

실행 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Obtaining Known Object Reference VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 상호작용하려는 객체를 포함하는 **타겟 VI의 참조를 얻습니다.**
2. 참조를 얻으려는 객체의 식별 정보를 결정합니다.

1. 클래스 타입—VI 스크립팅과 함께 사용할 수 있는 프런트패널 또는 블록 다이어그램 객체는 모두 VI 서버 클래스의 멤버이므로, 해당 객체의 참조는 상응하는 클래스에 속하게 됩니다. 접근하려는 객체의 클래스 이름을 결정하려면 **VI의 조직**을 참조하십시오. 클래스 지정자 상수를 사용하여 [VI 객체 참조 열기] 함수의 객체 클래스를 나타냅니다.

예제 세부사항

예제에서, Edit Me 컨트롤은 숫자 클래스의 멤버입니다.

2. 소유자—프런트패널 및 블록 다이어그램의 모든 객체에는 소유자가 있습니다. 가장 단순한 경우, 객체의 소유자는 VI의 프런트패널이거나 블록 다이어그램입니다. 그러나, 객체가 루프, 케이스 구조, 시퀀스 구조 또는 클러스터와 같은 하위 구조 내에 위치하는 경우, 하위 구조가 객체를 소유합니다.

예제 세부사항

예제에서, Edit Me 컨트롤은 타겟 VI의 프런트패널에 위치하므로, 타겟 VI의 프런트패널은 이 컨트롤을 소유합니다. 따라서, 스크립팅 VI는 VI:프런트패널 프로퍼티를 사용하여 타겟 VI의 프런트패널에 대한 참조를 얻습니다.



노트

Control 및 ControlTerminal 클래스는 자주 혼동됩니다. ControlTerminal 객체는 대응하는 프런트패널 Control 객체를 블록 다이어그램에서 나타낸 것입니다. 따라서, 타겟 VI의 프런트패널은 Edit Me Control 객체를 소유하고, 타겟 VI의 블록 다이어그램은 관련 Edit Me ControlTerminal 객체를 소유합니다. 사용자는 Control 객체의 숫자형을 변경할 수 있지만 ControlTerminal 객체의 숫자형은 변경할 수 없습니다.

3. 라벨—이것은 타겟 VI에서 원하는 객체에 지정된 이름입니다.

예제 세부사항

예제에서, 변경할 컨트롤의 라벨은 Edit Me입니다.

3. [VI 객체 참조 열기] 함수를 사용하여 단계 2의 식별 정보를 함수에 연결함으로써 원하는 객체의 참조를 얻습니다.
4. [VI 객체 참조 열기] 함수의 객체 참조 번호 출력을 프로퍼티 노드 또는 인보크 노드에 연결하여 참조된 객체에 대한 정보를 얻고 설정합니다.

예제 세부사항

예제에서, 스크립팅 VI는 숫자형:형 프로퍼티를 사용하여 Edit Me 컨트롤의 숫자형을 설정합니다.

5. [참조 닫기] 함수를 사용하여 모든 열린 객체 참조를 사용 후 닫습니다.

유의점 및 권장사항

- 적어도 한 번 블록다이어그램 라벨을 디스플레이—[VI 객체 참조 열기] 함수는 라벨이 디스플레이되지 않은 객체는 찾을 수 없습니다. 블록다이어그램 객체가 라벨을 디스플레이하지 않도록 기본설정되기 때문에, [VI 객체 참조 열기] 함수를 사용해 찾으려는 블록다이어그램 객체가 있다면 무엇이든 그 라벨을 직접 디스플레이해야 합니다. 보이지 않는 객체의 라벨을 디스플레이하려면, 객체에서 마우스 오른쪽 버튼을 클릭한 후 보이는 아이tem>>라벨을 선택합니다. 최소한 한 번 라벨을 디스플레이하면, 라벨을 다시 보이지 않게 하더라도 [VI 객체 참조 열기] 함수가 관련 객체를 찾을 수 있습니다.
- 고유 라벨 생성—LabVIEW는 블록다이어그램 객체의 경우 고유 라벨을 강제하지 않습니다. 같은 클래스에 속하는 객체의 경우 대부분 같은 라벨이 붙도록 기본설정되어 있습니다. 예를 들어, 블록다이어그램에 For 루프 2개를 추가하는 경우, For 루프라는 라벨이 각 루프에 붙게 됩니다. 다른 객체와 라벨을 공유하는 어느 객체에 [VI 객체 참조 열기] 함수를 사용하는 경우, LabVIEW는 처음으로 찾는 객체의 참조를 반환합니다. 각 객체에 고유 라벨을 지정하면 LabVIEW가 원하는 참조를 반환하도록 할 수 있습니다.
- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 객체 참조 닫기**를 참조하십시오.

추가 예제

알고 있는 VI에서 참조를 얻는 예제는 labview\examples\Application Control\VI Scripting\Creating VIs 디렉토리에 있는 Creating New VI From Template VI를 참조하십시오.

 예제 열기  관련 예제 찾기

VI 스크립팅을 사용하여 모르는 타겟 VI의 객체 참조 얻기

필수: VI 스크립팅

VI 스크립팅을 사용하여 실행 시까지 그 내용을 알지 못하는 타겟 VI를 검사하거나 수정할 수 있습니다. 다른 모든 VI 스크립팅 어플리케이션과 같이, 타겟 VI에서 조작하려는 객체가 있으면 그 객체의 참조를 얻어야 합니다. 그러나, 실행 시까지 타겟 VI의 내용을 알지 못하기 때문에, [VI 객체 참조 열기] 함수를 사용하여 객체를 라벨로 찾을 수 없습니다. 그 대신, [G객체의 탐색] VI를 사용하여 같은 클래스에 속한 객체 그룹을 찾아낼 수 있습니다. 그 후 반환된 객체 배열을 반복하여 조작하려는 객체를 찾아냅니다.

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

G객체의 탐색	보다 구체적인 클래스로	참조 닫기



실행 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Obtaining Unknown Object References VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 상호작용하거나 편집하려는 객체를 포함하는 **타겟 VI의 참조를 얻습니다.**

예제 세부사항

예제에서, [VI 참조 열기] 함수는 타겟 VI의 경로 컨트롤에 의해 지정된 VI의 참조를 제공합니다. 이를 실행하면 다른 VI 경로를 지정하여 스크립팅 VI를 모든 타겟 VI에 적용할 수 있습니다. 또한 사용자가 스크립팅 VI를 VI 세트에 프로그램적으로 적용하려는 경우, 스크립팅 VI를 subVI로 변환할 수 있습니다.

2. 참조를 얻으려는 객체의 클래스와 위치를 결정합니다.
 1. VI 포함하기—이 참조를 [G객체의 탐색] VI의 VI 참조 번호 입력에 연결합니다.
 2. VI 내 위치—타겟 VI 내에서, 객체는 For 루프 또는 케이스 구조와 같은 구조의 프런트패널, 블록다이어그램, 클러스터 또는 다이어그램에 위치할 수 있습니다. 타겟 트래버스 입력에서 상수를 생성하여 이러한 위치 중 [G객체의 탐색] VI가 검색할 위치를 지정해야 합니다.

예제 세부사항

예제에서, 스크립팅 VI는 블록다이어그램 객체인 컨트롤 터미널을 조작해야 합니다. 그러므로 스트립팅 VI는 BD를 스크립팅 VI가 검색할 컨테이너로 지정합니다.



노트

관심 대상 객체가 구조의 클러스터나 다이어그램에 있는 경우, 타겟 트래버스 입력의 후보 대상이 열거된 리스트에서 기타를 선택합니다. 두 경우 모두, 컨테이너 클러스터 또는 다이어그램의 참조를 [G객체의 탐색] VI의 다른 참조 번호 입력에 연결합니다.

3. 객체의 클래스 이름—조작하려는 객체의 클래스 이름을 결정하려면 **일반 VI 객체의 조직**을 참조하십시오.

예제 세부사항

예제에서, 스크립팅 VI가 터미널을 컨트롤하는 참조를 얻기 때문에, 스크립팅 VI는 컨트롤 터미널의 클래스 이름인 ControlTerminal을 사용합니다.

3. [G객체의 탐색] VI를 사용하여 단계 2의 정보를 [G객체의 탐색] VI에 연결함으로써 원하는 객체의 참조를 얻습니다.

- 반환된 참조 배열을 반복하여 객체를 개별적으로 조작합니다.

예제 세부사항

예제에서, 스크립팅 VI는 For 루프를 사용하여 참조 배열의 각 원소에 접근합니다.

- [보다 구체적인 클래스로] 함수를 사용하여 각각의 반환된 참조를 보다 구체적인 클래스로 캐스트합니다. [G객체의 탐색] VI는 클래스 이름 입력에 의해 지정된 클래스에 속하는 참조 대신에 G객체 참조의 배열을 반환합니다. 그러나 안전하게 [보다 구체적인 클래스로] 함수를 사용하여 클래스 이름에 의해 지정된 클래스로 반환된 참조를 캐스트할 수 있습니다. 참조를 보다 구체적인 클래스로 캐스트하면, 참조에서 보다 구체적인 프로퍼티나 메소드를 호출할 수 있습니다.

예제 세부사항

예제에서, 스크립팅 VI는 각각의 반환된 참조를 ControlTerminal로 캐스트합니다. 컨트롤 터미널:라벨.보이기 프로퍼티를 참으로 설정해야만 컨트롤 터미널 라벨을 보이게 할 수 있고, 라벨.보이기 프로퍼티가 G객체에 사용 가능하지 않기 때문에 이러한 캐스트가 필요합니다. [G객체의 탐색] VI가 특별히 ControlTerminal 객체를 검색하기 때문에, 이와 같이 캐스트해도 에러가 발생하지 않습니다.

- 참조를 프로퍼티 노드나 인보크 노드에 연결하여 참조된 객체에 대한 정보를 얻거나 설정합니다.

예제 세부사항

예제에서, 스크립팅 VI는 프로퍼티 노드를 사용하여 각 ControlTerminal 참조의 라벨.보이기 프로퍼티를 참으로 설정하고, 이로써 대응하는 컨트롤 터미널의 라벨이 보이게 됩니다.

- [참조 닫기] 함수를 사용하여 모든 열린 객체 참조를 사용 후 닫습니다.

유의점 및 권장사항

- 필요한 경우 각 참조의 클래스 확인—직상위 클래스의 이름을 [G객체의 탐색] VI에 연결하면, 직상위 클래스는 같지만 직하위 클래스는 다른 참조의 배열을 얻게 됩니다. 예를 들면, [G객체의 탐색] VI를 사용하여 루프 클래스에 속하는 모든 객체를 검색하면, VI가 For 루프, While 루프 모두에 해당하는 참조를 반환합니다. 다음 두 가지 방식 중 하나로 참조의 클래스를 확인할 수 있습니다.
 - [보다 구체적인 클래스로] 함수를 사용하여 특정 클래스로 참조를 캐스트합니다. 함수가 에러를 반환하지 않으면, 참조는 해당 특정 클래스에 일치하는 것입니다.
 - 개별 반환된 참조의 클래스 이름 프로퍼티를 맞추려는 클래스 이름과 비교합니다.
- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 [VI 및 객체 참조 닫기](#)를 참조하십시오.

추가 예제

[G객체의 탐색] VI 사용 예제는 labview\examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Using Traverse VI를 참조하십시오.

 예제 열기  관련 예제 찾기

VI 스크립팅을 사용하여 VI에 객체 추가하기

필수: **VI 스크립팅**

VI 스크립팅을 사용하여 객체를 VI의 프런트패널이나 블록다이어그램에 프로그램적으로 추가할 수 있습니다. 해당되는 객체는 다음과 같습니다:

- 프런트패널 컨트롤, 사용자 정의된 사용자 컨트롤 포함
- 함수, 구조, subVI 호출을 포함한 블록다이어그램 노드
- 클러스터, 배열, 구조의 원소

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

새 VI 객체	클래스 지정자 상수	참조 닫기
		

실행 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Creating Objects 디렉토리에 있는 Adding Objects VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. **타겟 VI의 참조를 얻습니다.**
2. 생성하려는 새 객체의 정의 정보를 결정합니다.
 - 소유자 참조—프런트패널 또는 블록다이어그램의 모든 객체에는 소유자가 있습니다. 가장 단순한 경우, 객체의 소유자는 VI의 프런트패널이거나 블록다이어그램입니다. 그러나, 객체가 루프, 케이스 구조, 시퀀스 구조 또는 클러스터와 같은 하위 구조 내에 위치하는 경우, 하위 구조가 객체를 소유합니다. **추가 예제** 섹션은 객체를 루프, 배열, 클러스터에 추가하는 예제를 포함합니다.

예제 세부사항

예제의 목적은 타겟 VI의 블록다이어그램에서 새로운 [빠기] 함수를 생성하는 것입니다. 더 엄밀하게 따지면 타겟 VI의 블록다이어그램이 소유자이지만, 스크립팅 VI는 타겟 VI의 참조를 새로운 [빠기] 함수의 소유자로 사용합니다. 그러나 [새 VI 객체] 함수는 [빠기] 함수와 같은 객체가 블록다이어그램에만 나타날 수 있는 때를 인식하여 그에 따라 새 객체를 놓습니다.

- 클래스 이름—VI 스크립팅으로 생성할 수 있는 프런트패널 또는 블록다이어그램 객체는 모두 VI 서버 클래스의 멤버입니다. 생성하려는 객체의 클래스 이름을 결정하려면 **VI의 조직**을 참조하십시오. 클래스 지정자 상수를 사용하여 [새 VI 객체] 함수의 객체 클래스를 나타냅니다.

예제 세부사항

예제에서, [빠기] 함수는 함수 클래스의 멤버입니다.

- 스타일—모든 LabVIEW 고유 객체에는 스타일이 있어서 객체가 정확히 어떤 종류인지 추가로 정의합니다. 컨트롤:스타일 ID 프로퍼티 또는 노드:스타일 프로퍼티를 사용하여 생성하려는 객체의 스타일을 확인할 수 있습니다.



노트

SubVI나 사용자 정의 컨트롤을 타겟 VI에 추가하는 경우에는 스타일을 지정할 필요가 없습니다.

예제 세부사항

예제에서, 스크립팅 VI는 [빠기] 함수를 생성하며, 따라서 스타일이 빠기입니다. 노드의 스타일은 종종 노드의 클래스와 다릅니다.

- 위치—객체를 생성하면 모두 그 위치를 지정해야 합니다. 이 위치는 객체의 소유자가 어디서 비롯되는지와 관련이 있습니다.

예제 세부사항

예제에서, 스크립팅 VI는 [빠기] 함수의 위치를 (100, 200)으로 지정합니다.

- (옵션) subVI 또는 사용자 정의된 사용자 컨트롤의 경로—subVI 및 사용자 정의된 사용자 컨트롤의 경우 소스 파일의 경로를 [새 VI 객체] 함수의 경로 입력에 연결해야 합니다. subVI의 호출을 생성하는 예제는 **추가 예제** 섹션을 참조하십시오.
- (옵션) 새 객체의 크기—수동으로 크기를 조정할 수 있는 객체의 경우 모두 그 크기를 지정할 수 있습니다.

예제 세부사항

예제에서, 스크립팅 VI는 [빠기] 함수를 생성하며, 이 함수는 크기 조정이 가능합니다.

- [새 VI 객체] 함수를 사용하여 단계 2의 정보를 함수에 연결함으로써 새 객체를 타겟 VI에 추가합니다.
- (옵션) 타겟 VI에서 **새로 생성된 객체를 기존 객체에 프로그램적으로 연결**합니다. VI 스크립팅으로 객체를 연결하는 예제는 **추가 예제** 섹션을 참조하십시오.
- [참조 닫기] 함수를 사용하여 모든 열린 객체 참조를 사용 후 닫습니다.

유의점 및 권장사항

- 가능한 **복잡한 새 VI는 템플릿에서 생성** — 한 번에 하나씩 각 구성요소 객체를 생성하여 프로그램식으로 새 VI를 생성할 수도 있지만, 템플릿에서 새 VI를 생성한 후 코드의 일부를 약간 수정하는 것이 훨씬 더 용이합니다. **추가 예제** 섹션의 Creating New VI From Scratch VI는 간단한 타겟 VI의 경우에도 한 번에 한 객체씩 생성할 때 광범위한 스크립팅 코드가 필요하다는 것을 보여줍니다. 템플릿에서 새 VI를 생성하면 그런 복잡한 스크립팅 코드가 필요 없습니다.
- 가능한 복잡한 컨트롤은 템플릿에서 생성 — 한 번에 한 원소씩 복잡한 컨트롤을 프로그램적으로 생성할 수도 있지만, 사용자 컨트롤을 직접 생성하고, 앞의 과정에 따라 컨트롤의 인스턴스를 추가한 후, 필요에 따라 컨트롤의 일부를 개별적으로 수정하는 것이 더 용이합니다. 사용자 컨트롤에 대한 참조가 있는 경우, VI:참조에서 생성 메소드를 사용하여 컨트롤의 복사본을 생성할 수도 있습니다.
- 열어 놓은 참조 모두 닫기 — 지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 VI 객체 참조 닫기**를 참조하십시오.

추가 예제

배열 컨트롤, 숫자형 인디케이터, SubVI 호출을 타겟 VI에 추가하는 예제는 labview\examples\Application Control\VI Scripting\Creating VIs 디렉토리에 있는 Creating New VI From Scratch VI를 참조하십시오.

 **예제 열기**  **관련 예제 찾기**

[더하기] 함수를 While 루프와 같은 다른 객체의 안에 추가하는 예제는 labview\examples\Application Control\VI Scripting\Creating Objects 디렉토리에 있는 Drop Add Function Inside While Loop VI를 참조하십시오.

 **예제 열기**  **관련 예제 찾기**

객체를 클러스터에 추가하는 예제는 labview\examples\Application Control\VI Scripting\Creating Objects 디렉토리에 있는 Drop Digital Numeric Inside Cluster VI를 참조하십시오.

 **예제 열기**  **관련 예제 찾기**

VI 스크립팅을 사용하여 커넥터 팬 설정하기

필수: VI 스크립팅

VI 스크립팅을 사용하여 커넥터 팬과의 프로그램적인 상호작용을 통해 다음 작업을 수행할 수 있습니다:

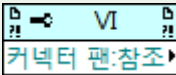
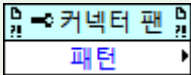
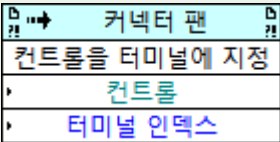
- subVI의 커넥터 팬 패턴 선택하기
- 커넥터 팬의 터미널에 컨트롤 지정하기

- 커넥터 팬 탐색하기
- subVI의 터미널이 필수인지, 옵션인지 또는 권장인지 정의하기

이 토픽에서는 처음 두 가지 태스크를 집중적으로 다룹니다. 커넥터 팬과의 프로그램적 상호작용에 대한 추가 정보는 커넥터 팬 메소드를 참조하십시오.

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

VI:커넥터 팬:참조	커넥터 팬:패턴	커넥터 팬:컨트롤을 터미널에 지정	커넥터 팬 패턴 참조 예
			

 예제 열기

다음 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Connector Pane 디렉토리에 있는 Add Terminals to Connector Pane VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 검사하거나 수정하려는 커넥터 팬이 속한 **VI의 참조를 얻습니다.**

예제 세부사항

예제에서, 스크립팅 VI는 [VI 참조 열기] 함수를 사용하여 특정 VI의 참조를 얻습니다.

2. VI:커넥터 팬:참조 프로퍼티를 사용하여 VI에 속한 커넥터 팬의 참조를 얻습니다.
3. VI에 대해 원하는 커넥터 팬 패턴을 결정합니다.

커넥터 팬 패턴은 4800과 4835 사이의 패턴 인덱스로 표시됩니다. 커넥터 팬은 패턴 4815를 사용하도록 기본설정됩니다.

다음 중 한 가지 방식을 사용하여 사용할 패턴 인덱스를 확인할 수 있습니다.

- 인덱스에 따른 모든 패턴 매핑은 labview\examples\Application Control\VI Scripting\Connector Pane 디렉토리에 있는 Connector Pane Pattern Reference VI를 참조하십시오.



Connector Pane Pattern Reference VI 열기

- 수정 중인 VI에 대해 사용하려는 커넥터 팬 패턴이 속한 VI의 기본 도움말을 봅니다. **기본 도움말 윈도우에 추가 VI 스크립팅 정보를 디스플레이**하는 경우, 패턴 인덱스는 VI의 경로 끝에 나타납니다.
- 다른 VI의 커넥터 팬:패턴 프로퍼티를 사용하여 해당 VI의 패턴 인덱스를 프로그램적으로 얻습니다.

예제 세부사항

예제에서, 스크립팅 VI를 통해 사용자는 실행 시 커넥터 팬 패턴을 선택할 수 있습니다. 따라서, 스크립팅 VI는 컨트롤을 사용하여 커넥터 팬 패턴을 결정합니다.



노트

커넥터 팬 패턴 컨트롤은 모든 정수를 받기 때문에, 이 예제는 해당 컨트롤의 값을 확인하여 값이 허용 가능한 패턴 인덱스 범위인 4800..4835 사이가 되도록 합니다. 단순화를 위해 예제는 이 단계를 생략합니다.

4. 커넥터 팬:패턴 프로퍼티를 사용하여 선택한 패턴으로 커넥터 팬을 설정합니다.
5. 커넥터 팬 터미널에 지정하려는 **컨트롤의 참조를 얻습니다.**

예제 세부사항

예제에서, 스크립팅 VI는 타겟 VI의 프런트패널의 패널:컨트롤[] 프로퍼티를 사용하여 모든 프런트패널 컨트롤의 참조를 얻습니다.

6. 이 컨트롤을 커넥터 팬 터미널에 따라 어떻게 매핑할지 결정합니다.

사용자가 볼 수 있는 커넥터 팬에 컨트롤과 인디케이터를 지정하는 가이드라인은 **LabVIEW Style Checklist**를 참조하십시오.

예제 세부사항

예제에서, 스크립팅 VI의 유일한 목적은 모든 컨트롤을 터미널에 지정하는 것입니다. 따라서, 스크립팅 VI에는 컨트롤을 터미널에 매핑하는 것과 관련한 기준이 없습니다.

7. 커넥터 팬:컨트롤을 터미널에 지정 메소드를 사용하여 커넥터 팬의 원하는 터미널에 각 컨트롤을 지정합니다.

선택한 패턴에서 각 터미널에 대응하는 터미널 인덱스를 확인하려면, 단계 3에서 언급한 Connector Pane Pattern Reference VI를 참조하십시오.

예제 세부사항

예제에서, 스크립팅 VI는 패널:컨트롤[] 배열 전체에서 반복되고, 커넥터 팬의 다음 터미널 인덱스로 각 컨트롤을 지정합니다. 스크립팅 목적상 컨트롤이 커넥터 팬에 특정한 방법으로 연결될 필요가 없기 때문에 이 방식에 특정 터미널 인덱스에 대한 지식이 필요하지는 않습니다.

노트 예제에서 선택한 커넥터 팬 패턴의 터미널 수가 프런트패널에 있는 컨트롤보다 더 적을 수 있기 때문에, 커넥터 팬:컨트롤을 터미널에 지정 메소드가 에러를 반환할 가능성도 있습니다. 이러한 에러를 피하는 방법에 대한 정보는 이 토픽의 [유의점 및 권장사항](#) 섹션을 참조하십시오.

8. [참조 닫기] 함수를 사용하여 모든 열린 객체 참조를 사용 후 닫습니다.

예제 세부사항

예제에서, 스크립팅 VI는 프런트 패널, 컨트롤, 커넥터 팬, 타겟 VI의 참조를 닫습니다.

유의점 및 권장사항

- 컨트롤 지정이 이미 존재하는 커넥터 팬 패턴을 변경할 때 주의 필요—커넥터 팬 패턴을 다른 패턴으로 대체하면, LabVIEW는 이전 패턴의 터미널에 지정되었던 컨트롤을 새 패턴의 터미널로 다시 지정합니다. 그러나, 새로운 지정으로 이전 패턴의 터미널 인덱스나 상대적인 위치가 보존되는 것은 아닙니다. 그러므로 최근 패턴 변경 이후 얻은 터미널 지정에 대한 정보를 참고로 합니다.
- 사용하려는 터미널 인덱스를 수용할 만큼 충분한 터미널이 커넥터 팬 패턴에 있는지 확인—참조된 커넥터 팬의 패턴에 있는 터미널이 메소드에 연결된 터미널 인덱스를 수용하지 못하는 경우, 커넥터 팬:컨트롤을 터미널에 지정 메소드가 에러를 반환합니다. 관련 커넥터 팬의 커넥터 팬:연결 터미널의 개수 프로퍼티를 사용하여 사용하려는 터미널 인덱스에 필요한 충분한 터미널이 커넥터 팬에 있는지 확인합니다.
- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 [VI 및 VI 객체 참조 닫기](#)를 참조하십시오.

VI 스크립팅을 사용하여 템플릿에서 새 VI 생성하기

필수: VI 스크립팅

[새 VI 객체] 함수 및 터미널:와이어 연결 메소드를 사용하여 복잡한 VI를 한 번에 한 객체씩 프로그램적으로 생성할 수도 있지만, 그러한 방식에는 복잡하고 광대한 스크립팅 코드가 필요해 작성하는데 시간이 많이 소요되고 읽기 어렵습니다. 생성하려는 VI에 아주 근사한 템플릿 VI를 작성하는 것이 더 용이한 경우가 많습니다. 이때 VI 스크립팅을 사용하여 템플릿 VI의 복사본을 생성하고 복사한 코드의 일부를 약간 편집하면 스크립팅 목적을 달성할 수 있습니다.

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

새 VI	참조 닫기



다음 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Creating VIs 디렉토리의 Creating New VI From Template VI를 열어 보십시오.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 원하는 타겟 VI의 형태에 가능한 가장 가까운 템플릿 VI를 디자인합니다.

템플릿 디자인에 다음 구성요소를 포함시킵니다:

- 한 번에 하나씩 프로그램적으로 생성하지 않을 모든 객체
- 아무것도 없는 상태에서 처음부터 프로그램적으로 생성하지 않을 연결 모두
- 실행 시 지정하려는 객체에 대한 자리 표시자 객체 스크립팅 VI에서 처음부터 연결을 프로그램적으로 생성하는 대신, 자리 표시자 객체를 통해 템플릿에서 자리 표시자의 터미널을 연결할 수 있습니다.
- 스크립팅 VI로 위치를 찾거나 수정하려는 모든 객체에서 눈에 보이는 라벨

예제 세부사항

예제에서, 템플릿은 배열 입력, 숫자형 출력, 자리 표시자 subVI, 이러한 필수 객체의 연결에 필요한 와이어 연결을 포함하는 디자인을 보여줍니다. 실행 시, 스크립팅 VI는 사용자가 지정한 계산을 수행하는 subVI로 자리 표시자 subVI를 대체할 수 있습니다.



노트

의도적으로 자리 표시자 SubVI는 가능한 대체 SubVI 모두에 대해 동일한 커넥터 팬 패턴을 가집니다. 일치하는 커넥터 팬 패턴 때문에 스크립팅 VI는 터미널을 다시 연결하지 않고도 쉽게 자리 표시자 subVI를 대체할 수 있습니다. 템플릿 VI의 자리 표시자 객체가 대체 객체와 다른 커넥터 팬을 지니는 경우, 의도한 바와 달리 LabVIEW가 대체 객체에 데이터를 연결하지 않을 수도 있습니다.

2. [새 VI] 함수로 템플릿 VI의 경로를 함수에 연결하여 템플릿 VI의 복사본을 생성합니다.

경로 컨트롤, 경로 상수 또는 템플릿 VI 참조의 VI:VI 경로 프로퍼티를 사용하여 템플릿 VI의 경로를 지정할 수 있습니다.

예제 세부사항

예제에서, VI:VI 경로 프로퍼티는 템플릿 VI의 경로를 제공합니다.

3. 템플릿 VI에서 **수정하려는 객체의 참조를 얻습니다**. 템플릿 VI에서 객체의 참조를 얻는 가장 일반적인 방법은 [VI 객체 참조 열기] 함수를 사용하는 것입니다.

예제 세부사항

예제에서, 스크립팅 VI는 자리 표시자 subVI를 스크립팅 VI의 사용자가 실행 시에 지정하는 작업을 수행하는 subVI로 대체해야 합니다. 자리 표시자 subVI에는 고유 레벨이 있기 때문에, [VI 객체 참조 열기] 함수는 해당 subVI의 참조를 얻을 수 있습니다.

4. 스크립팅 목적에 부합하도록 참조된 객체를 검사하거나 수정합니다.

VI 스크립팅을 사용한 VI 편집 방법에 대한 추가 정보는 **일반적인 스크립팅 태스크 리스트**를 참조하십시오.

예제 세부사항

예제에서, 스크립팅 VI는 [대체] 메소드를 사용하여 자리 표시자 subVI를 사용자 지정 계산 작업 수행이 가능한 subVI로 대체합니다. 실행 시 동작 라디오 버튼의 값에 따라, 케이스 구조는 자리 표시자 subVI를 더하기 subVI와 곱하기 subVI 중 무엇으로 대체할지 선택합니다.

5. [참조 닫기] 함수를 사용하여 모든 열린 객체 참조를 사용 후 닫습니다.

예제 세부사항

예제에서, 스크립팅 VI는 타겟 VI의 참조뿐만 아니라 대체 subVI의 참조를 닫습니다. 그러나 스크립팅 VI는 원래 자리 표시자 subVI의 참조는 닫지 않으며, 이는 대체 메소드가 해당 참조를 무효화하기 때문입니다.

유의점 및 권장사항

- 원하는 참조를 모아서 반환하는 중간 subVI 생성—템플릿에 수정을 많이 하는 VI 스크립팅 어플리케이션의 경우, 참조를 subVI로 모으는 코드를 따로 분리하여 사용자 코드를 명확하게 할 수 있습니다. 수정하려는 참조를 반환하는 subVI를 따로 생성하려면, 최상위 레벨의 스크립팅 VI가 아닌 새 subVI에서 단계 2부터 단계 4까지 수행합니다. 그러면 최상위 레벨의 스크립팅 VI가 새 subVI를 호출하고 반환된 참조로 수정합니다.
- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 객체 참조 닫기**를 참조하십시오.

추가 예제

한 번에 한 객체씩 새 VI를 생성하는 것에 대한 예제는 labview\examples\Application Control\VI Scripting\Creating VIs 디렉토리의 Creating New VI From Scratch VI를 참조하십시오. 이 예제의 복잡한 정도 그리고 템플릿을 사용하지 않고 작업을 완료하는데 필요한 참조의 수에 유의합니다.

 [예제 열기](#)  [관련 예제 찾기](#)

VI 스크립팅을 사용하여 VI 및 VI 객체의 참조 닫기

필수: VI 스크립팅

어떤 VI 스크립팅 태스크를 수행하든 생성, 편집 또는 검사하려는 VI 및 객체의 참조를 열어야 합니다. 참조를 반환하는 VI, 함수, 프로퍼티, 메소드를 사용할 때, 참조를 생성하거나 열게 됩니다. VI 참조가 열려있는 동안 LabVIEW는 관련된 VI를 메모리에서 제거할 수 없습니다. 그러므로, 다수의 참조가 열려 있으면, 다수의 VI와 객체가 메모리에 있게 되고, 이는 LabVIEW가 더 느리게 작동하는 원인이 되며, 사용 가능한 메모리가 없는 경우 LabVIEW에서 새 VI와 객체가 생성되지 않을 수도 있습니다. 이러한 성능상의 문제를 방지하려면, 스크립팅 VI 내에서 열어놓은 VI 또는 객체의 참조를 닫습니다.

다음 타입의 참조는 열려 있어도 성능에 영향을 끼치지 않으므로 닫을 필요가 없습니다.

- G객체에서 상속한 참조
- 이 VI 또는 이 어플리케이션의 정적 참조

[참조 닫기] 함수를 사용하여 생성하거나 열어 놓은 참조를 닫습니다. 다음 작업은 작업 동작의 일부로 참조를 닫기 때문에 [참조 닫기]를 호출할 필요가 없습니다.

작업	결과
일반:삭제 메소드 사용하기	삭제하는 객체의 참조를 닫음
G객체:대체 메소드 사용하기	대체하는 객체의 참조를 닫음
VI의 참조 닫기	VI 내 객체의 참조를 무효화할 수 있음

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

참조 닫기



실행 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Managing References 디렉토리에 있는 Closing References VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 스크립팅 VI의 블록다이어그램에 있는 VI 및 VI 객체의 참조를 확인합니다. 앞에서 답을 필요가 없다고 언급된 참조 타입은 무시합니다.

예제 세부사항

예제에서, 스크립팅 VI는 다음 위치에서 참조를 엽니다:

- [VI 참조 열기]—타겟 VI의 참조 열기
 - [G객체의 탐색]—타겟 VI에서 각 함수 객체의 참조 열기
 - 함수:터미널[]—각 함수의 모든 터미널에 대한 참조 열기
 - 터미널:연결된 와이어—각 터미널에 연결된 와이어의 참조 열기
 - 와이어:터미널[]—와이어의 모든 터미널에 대한 참조 열기
 - 함수:대체—대체 함수의 참조 열기
 - 함수:터미널[]—대체 함수의 모든 터미널에 대한 참조 열기
2. LabVIEW가 각 참조를 언제 마지막으로 사용할지 결정합니다.
- 참조의 데이터 흐름을 추적하는 경우, LabVIEW에서는 노드의 터미널 참조 사용 시 해당 노드에 열린 참조가 필요합니다.

예제 세부사항

예제에서, 열린 참조는 다음 위치에서 마지막으로 사용됩니다:

- 타겟 VI 참조—VI의 끝 다른 열린 참조가 타겟 VI 내의 객체를 대상으로 하기 때문에 LabVIEW는 이 참조를 VI 전체에 걸쳐 사용합니다.
 - 함수 참조—함수:대체 메소드 이후
 - 함수 터미널 참조—터미널:연결된 와이어 메소드 이후
 - 와이어 참조—와이어:삭제 메소드 이후
 - 와이어 터미널 참조—터미널:와이어 연결 메소드 이후
 - 대체 함수 참조—함수:터미널[] 메소드 이후
 - 대체 함수의 터미널 참조—터미널:와이어 연결 메소드 이후
3. LabVIEW가 마지막으로 VI 또는 해당 객체 참조 중 하나에 대한 참조를 사용한 후, 곧바로 [참조 닫기] 함수에 각각의 열린 VI 참조를 연결합니다. 여러 클러스터가 제공하는 데이터 흐름을 사용하여 LabVIEW가 각 [참조 닫기] 함수를 언제 실행하는지 컨트롤합니다. 참조를 닫을 때에는 [결과](#)에 주의하십시오.

유의점 및 권장사항

- 해당 VI의 내의 모든 객체 참조를 사용한 후 객체 참조 닫기—VI의 참조를 닫으면 해당 VI 내에 위치한 객체의 열린 참조가 모두 무효화될 수 있습니다. VI 참조와 해당 VI 내에 위치한 객체의 모든 참조를 사용하고 난 후에만 VI 참조를 안전하게 닫을 수 있습니다.
- 가능한 빨리 참조 닫기—열린 참조의 개수가 너무 많으면 LabVIEW의 실행 속도가 더 느려집니다. 작은 어플리케이션의 경우 이러한 효과가 무시할 만하지만, 큰 어플리케이션은 눈에 띄게 성능이 나빠 집니다. 프로그램에서 일찍 참조를 닫으면, 참조가 불필요하게 메모리에 남아있지 않게 됩니다.

- 가능하면 VI 끝나기 전에 참조 닫기—LabVIEW는 실행을 마친 VI에서 참조를 자동으로 닫기 때문에, VI가 끝나고 참조를 닫는 것은 성능 향상에 보탬이 되지 않습니다.

추가 예제

VI 스크립팅 사용 중 여러 참조 닫기의 또 다른 예제는 labview\examples\Application Control\VI Scripting\Creating VIs 디렉토리의 Creating New VI From Scratch VI를 참조하십시오.

 예제 열기  관련 예제 찾기

VI 스크립팅을 사용하여 VI 탐색하기

필수: VI 스크립팅

VI의 블록 다이어그램 객체는 터미널과 와이어로 연결되기 때문에, 한 객체의 참조를 사용하여 연결된 다른 객체에 접근할 수 있습니다. 이와 같은 타겟 VI 탐색 방법은 다음과 같은 경우에 사용할 수 있습니다:

- 객체 간 연결 검사하기 또는 편집하기
- [VI 객체 참조 열기] 함수 또는 [G객체의 탐색] VI로 직접 접근할 수 없는 객체의 참조 얻기

LabVIEW에서 직접 객체를 서로 연결할 수 있는 경우, 대개 그러한 연결을 프로그램적으로 탐색할 수 있게 하는 프로퍼티나 메소드를 LabVIEW에서 제공합니다. 따라서, 모든 탐색 작업에는 시작 객체의 참조를 얻은 후 필요한 프로퍼티와 메소드를 사용하여 연결된 객체의 참조를 얻는 작업이 수반됩니다. 규모가 큰 탐색 작업의 경우 다음과 같은 하위 탐색 작업으로 나눌 수 있습니다:

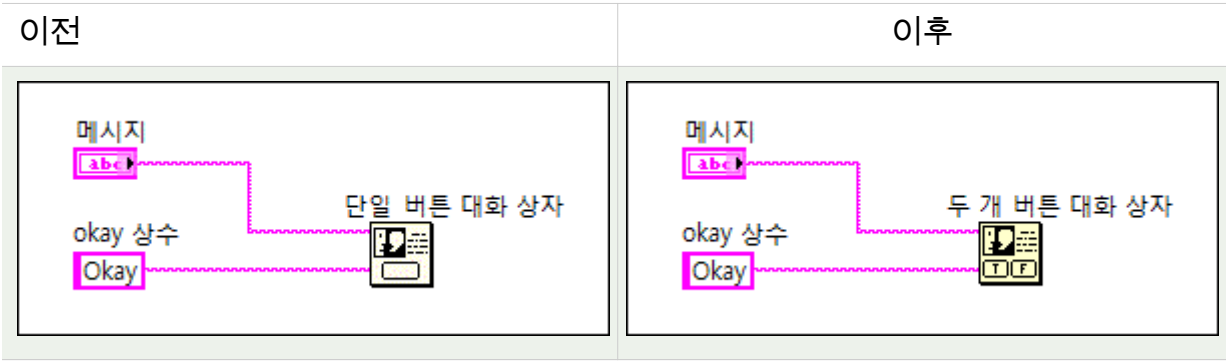
- 탐색을 시작하는 출발점 확인하기
- **노드에서 와이어로 탐색하기**
- **와이어에서 노드로 탐색하기**
- **구조 탐색하기**
- 블록 다이어그램과 프런트패널 사이에서 탐색하기

다음의 예는 위의 하위 탐색 작업을 일부 결합하여 보다 큰 탐색 문제를 해결하는 방법을 보여줍니다.

작업 예: 함수를 대체하면서 와이어 연결 보존하기

모든 타겟 VI에서 [단일 버튼 대화 상자] 함수를 [두 버튼 대화 상자] 함수로 대체하는 스크립팅 VI를 작성할 수 있습니다. 그러나, 이렇게 대체한 후, [단일 버튼 대화 상자] 함수의

버튼 이름 입력에 연결된 객체가 새로운 [두 버튼 대화 상자] 함수의 참 버튼 이름 입력에 연결되도록 해야 합니다. 다음 이미지는 어떻게 변경되어야 하는지 보여줍니다.



이 작업 예에서 탐색이 필요한 이유

[단일 버튼 대화 상자] 함수를 대체하려면, 함수의 참조만 필요합니다. 그러나, 원래 와이어 연결을 [두 버튼 대화 상자] 함수에서 계속 유지하려면, [단일 버튼 대화 상자] 함수의 버튼 이름 입력에 연결된 객체의 참조도 필요합니다. 이 문자열 참조는 [단일 버튼 대화 상자]와 연결된 객체 사이의 연결을 탐색하여 얻을 수 있습니다.

탐색을 통해 이 작업 예를 완료하는 방법

다음은 주로 [단일 버튼 대화 상자] 함수에서 그 함수의 버튼 이름 입력에 연결된 객체로 탐색하는데 필요한 하위 탐색 작업의 개요입니다. 각 작업의 실행 방법에 대한 구체적인 정보는 개별 작업 토픽을 참조하십시오.

1. 탐색의 출발점 확인—관심 대상인 모든 타겟 VI에 [단일 버튼 대화 상자] 함수가 있기 때문에, 탐색의 출발점으로 [단일 버튼 대화 상자] 함수를 사용할 수 있습니다.
2. **노드에서 와이어로 탐색**—[단일 버튼 대화 상자] 함수의 참조를 사용하여 함수의 버튼 이름 입력에 연결된 와이어의 참조를 얻을 수 있습니다.
3. **와이어에서 노드로 탐색**—와이어 참조를 사용하여 소스 노드의 참조를 얻을 수 있습니다.

이 3가지 탐색 작업을 마치면, 작업 완료에 필요한 참조에 모두 접근이 가능합니다.

이 스크립팅 태스크의 완전한 솔루션은 labview\examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리의 Navigation Overview VI를 참조하십시오.

 [예제 열기](#)  [관련 예제 찾기](#)

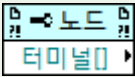
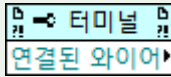
VI 스크립팅을 사용하여 노드에서 와이어로 탐색하기

필수: **VI 스크립팅**

노드의 참조를 사용하여 연결된 와이어의 참조를 얻으려면 언제나 노드에서 와이어로 탐색합니다. 노드에서 와이어로 탐색하는 것은 하위 탐색 작업으로 다른 하위 탐색 작업과 결합하여 더 큰 탐색 목적을 달성할 수 있습니다. 노드에서 와이어로 탐색하는 것이 다른 탐색 작업과 어떻게 상호작용하는지 더 자세히 알려면 [탐색 개요](#)를 참조하십시오.

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

노드:터미널[]	보이는 터미널 인덱스	배열 인덱스	터미널:연결된 와이어	참조 닫기
				

다음 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Navigating Nodes and Wires VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 노드의 참조를 얻습니다.

예제 세부사항

예제에서, 스크립팅 VI는 [G객체의 탐색] VI를 사용하여 타겟 VI에 위치한 모든 함수 노드의 참조를 얻습니다. 케이스 구조는 각 함수 노드의 스타일을 확인하여 그 스타일이 단일 버튼 대화 상자인 함수 노드에서만 탐색이 지속되도록 합니다.

2. 노드:터미널[] 프로퍼티를 사용하여 노드의 터미널에 대한 참조를 얻습니다.

예제 세부사항

예제에서, 참조가 함수 참조이기 때문에 노드:터미널[] 프로퍼티가 함수:터미널[]로 나타납니다. 함수가 노드의 서브클래스이기 때문에, 함수는 노드의 터미널[] 프로퍼티를 상속합니다.

3. 노드를 대상 와이어에 연결하는 터미널의 터미널 인덱스를 확인하려면 노드의 기본 도움말 윈도우를 참조하십시오.

기본 도움말 윈도우의 커넥터 팬이 대괄호로 묶인 숫자를 각 터미널 옆에 디스플레이하도록 하려면, [기본 도움말 윈도우에 추가 VI 스크립팅 정보를 디스플레이](#)해야 합니다. 이 숫자는 상관되는 터미널의 참조가 저장된 터미널[] 배열의 인덱스를 나타냅니다.

예제 세부사항

예제에서, 스크립팅 VI는 [단일 버튼 대화 상자] 함수의 버튼 이름 입력의 인덱스를 필요로 합니다. [단일 버튼 대화 상자] 함수의 기본 도움말 윈도우에 따르면, 해당 입력 터미널의 인덱스는 1입니다.

4. 원하는 터미널 인덱스와 함께 [배열 인덱스] 함수를 사용하여 터미널[] 배열에서 원하는 터미널의 참조를 얻습니다.
5. 반환된 터미널 참조의 터미널:연결된 와이어 프로퍼티를 사용하여 해당 터미널에 연결된 와이어의 참조를 얻습니다.
6. 반환된 와이어 참조를 사용하여 와이어를 수정하거나 **계속 탐색**합니다.

예제 세부사항

예제에서, 스크립팅 VI는 와이어 참조를 사용하여 와이어의 와이어:터미널[] 프로퍼티를 사용함으로써 **와이어에서 노드로 탐색**을 계속합니다.

7. [참조 닫기] 함수를 사용하여, 노드 참조, 터미널 참조, 와이어 참조를 사용 후 닫습니다.

유의점 및 권장사항

- **기본 도움말 윈도우에서 터미널 인덱스 디스플레이**—원하는 터미널의 인덱스를 알면, 노드:터미널[] 배열에서 원하는 터미널에 접근할 수 있습니다.
- 노드:터미널[] 배열에서 유효하지 않은 참조에 주의—노드:터미널[] 배열은 노드의 각 터미널에 대해 하나의 원소를 포함합니다. 그러나 각 원소는 해당 터미널이 연결된 경우에만 유효한 터미널 참조입니다. 많은 VI와 함수에 옵션 입력이 있고 이는 항상 연결되는 것이 아니기 때문에, 터미널 배열에는 해당 터미널에 대해 유효하지 않은 참조가 종종 포함됩니다. 그러므로 특정 터미널이 연결되었는지 확실하지 않은 경우, 터미널 배열에서 해당 원소를 확인하여 스크립팅 VI의 다른 곳에서 사용을 시도하기 전에 유효하지 않은 참조인지 점검합니다.
- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 객체 참조 닫기**를 참조하십시오.

추가 예제

노드에서 와이어로 탐색하기, 구조 탐색하기를 포함하는 전체 탐색 예제는 labview\examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리의 Navigation Overview VI를 참조하십시오.

 예제 열기  관련 예제 찾기

VI 스크립팅을 사용하여 와이어에서 노드로 탐색하기

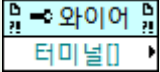
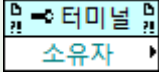
필수: VI 스크립팅

와이어 참조를 사용하여 와이어에 연결된 노드의 참조를 얻을 때마다 와이어에서 노드로 탐색합니다. 와이어에서 연결된 노드로 탐색하는 것은 하위 탐색 작업으로 다른 하위 탐색 작업과 결합하여 더 큰 탐색 목적을 달성할 수 있습니다. 와이어에서 노드로 탐색하

는 것이 다른 탐색 작업과 어떻게 상호작용하는지 더 자세히 알려면 [탐색 개요](#)를 참조하십시오.

사용 객체

이러한 VI 스크립팅 태스크에는 다음 객체가 필요합니다:

와이어:터미널[]	배열 인덱스	터미널:소유자	참조 닫기
			

다음 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview \examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Navigating Nodes and Wires VI를 엽니다. 이 예제는 또한 노드에서 와이어로 탐색하는 과정을 설명합니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 와이어의 [참조를 얻습니다](#).

와이어에서 노드로 탐색할 때, 이 탐색의 결과로 보통 와이어 참조를 얻습니다.

예제 세부사항

예제에서, 스크립팅 VI는 [단일 버튼 대화 상자] 함수에서 해당 함수의 버튼 이름 입력에 연결된 와이어로 탐색함으로써 초기 와이어 참조를 얻습니다.

2. 와이어의 와이어:터미널[] 프로퍼티를 사용하여 와이어의 터미널에 대한 참조를 얻습니다.
3. 어떤 와이어의 터미널이 탐색 목적에 가장 잘 부합하는지 확인합니다.

일부 탐색 작업에서는, 와이어에서 하나의 터미널만 고려 대상이 될 수 있습니다. 다른 탐색 작업에서는, 와이어의 모든 터미널이 검사나 탐색의 고려 대상이 될 수 있습니다.

예제 세부사항

예제에서, 스크립팅 VI는 와이어의 소스 터미널만 필요로 합니다.

4. 단계 2의 터미널[] 배열과 함께 [배열 인덱스] 함수를 사용하여 원하는 터미널의 참조를 얻습니다.

터미널[] 배열의 첫번째 원소는 항상 와이어의 소스 터미널에 대한 참조입니다. 와이어가 터미널 2개만 연결하는 경우, 배열에서 두번째 원소는 와이어의 싱크 터미

널에 대한 참조입니다. 와이어에 여러 싱크 터미널이 있는 경우, 배열은 이들을 배열 인덱스 1로 시작해 일정한 순서 없이 저장합니다.

와이어에 여러 싱크 터미널이 있는 경우, For 루프로 배열의 각 터미널 참조를 반복할 수 있습니다. 각 터미널 참조에 대해 단계 5를 반복합니다.

예제 세부사항

예제에서, 스크립팅 vi는 와이어의 소스 터미널에 대해서만 참조를 필요로 합니다. 소스 터미널은 항상 와이어의 터미널[] 배열에서 시작 위치에 저장되기 때문에, 스크립팅 vi는 [배열 인덱스] 함수를 사용하여 터미널[] 배열의 인덱스 0에서 터미널의 참조를 얻습니다.

5. 반환된 터미널 참조의 터미널:소유자 프로퍼티를 사용하여 연결된 노드의 참조를 얻습니다.
6. (옵션) [보다 구체적인 클래스로] 함수를 사용하여 반환된 참조를 실제 클래스로 캐스트합니다. 이 단계를 통해 다음과 같은 혜택을 얻을 수 있습니다:
 - 클래스 확인—[보다 구체적인 클래스로] 함수는 객체가 해당 클래스에 속하는 경우에만 참조된 객체를 더 구체적인 클래스로 캐스트합니다. 그러므로 [보다 구체적인 클래스로] 함수를 사용하여 참조의 클래스를 확인할 수 있습니다. 참조된 객체와 상호작용하려 할 때, 이 객체가 특정 클래스에 속하는 경우에만, 이러한 확인 작업이 필요합니다.
 - 추가 프로퍼티와 메소드에 접근 가능—참조를 보다 구체적인 클래스로 캐스트하면, 더 구체적인 참조의 프로퍼티와 메소드를 사용할 수 있습니다. 이전 단계에서 터미널:소유자 프로퍼티는 항상 G객체 참조를 반환하는데, 이는 오직 한정된 세트의 프로퍼티와 메소드를 지원하는 클래스입니다.

예제 세부사항

예제에서, 스크립팅 vi는 반환된 노드가 문자열 상수인 경우에만 이를 수정할 수 있습니다. 그러므로 스크립팅 vi는 참조를 문자열 상수로 캐스트하려 합니다. 케이스 구조는 [보다 구체적인 클래스로] 함수가 에러를 반환하는지 그리고 반환된 객체가 문자열 상수인지 확인합니다.

7. 노드 참조를 사용하여 노드를 편집하거나 탐색을 계속할 수 있습니다.

예제 세부사항

예제에서, 스크립팅 vi는 노드를 사용하여 탐색을 계속하지 않고 노드를 편집합니다. 스크립팅 vi는 문자열 상수:값 프로퍼티를 사용하여 문자열 상수인 노드의 값을 0kay!로 설정합니다.

8. [참조 닫기] 함수를 사용하여, 와이어 참조, 터미널 참조, 노드 참조를 사용 후 닫습니다.

유의점 및 권장사항

- 터미널:소유자 프로퍼티에 의해 반환된 참조 타운캐스트—터미널:소유자 프로퍼티는 항상 일반 객체의 참조를 반환합니다. 그러나 터미널이 보다 구체적인 객체 클래스에 와이어를 연결하고 있는 경우, [보다 구체적인 클래스로] 함수를 사용하여 보다 구체적인 클래스로 반환된 참조를 다운캐스트할 수 있습니다. 보다 구체

적인 클래스로 참조를 다운캐스트하면, 참조에서 보다 구체적인 프로퍼티나 메소드를 호출할 수 있습니다.

- 열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 [VI 및 객체 참조 닫기](#)를 참조하십시오.

추가 예제

노드에서 와이어로 탐색하기를 포함하는 전체 탐색 예제는 labview\examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리의 Navigation Overview VI를 참조하십시오.

 예제 열기  관련 예제 찾기

VI 스크립팅을 사용하여 구조 탐색하기

필수: [VI 스크립팅](#)

구조의 특정 부분에 대한 참조를 검사하거나, 편집하거나, 생성할 때마다 구조를 탐색합니다. 구조의 원소는 와이어가 아닌 구조 자체에 의해 연결되기 때문에 구조를 탐색하는 것은 독특한 탐색 작업입니다. 그러므로 구조를 탐색하려면, 구조를 구성하는 서로 다른 원소뿐만 아니라 이러한 원소의 참조를 반환하는 프로퍼티에 대해서도 알고 있어야 합니다. 구조 탐색을 다른 하위 탐색 작업과 결합하여 더 큰 탐색 목적을 달성할 수 있습니다. 구조를 탐색하는 것이 다른 탐색 작업과 어떻게 상호작용하는지 더 자세히 알려면 [탐색 개요](#)를 참조하십시오.

사용 객체

구조를 탐색하려면, 구조의 프로퍼티를 사용하여 구조 조직의 다른 부분들에 대한 참조를 얻습니다. 구조 탐색을 위해 필요한 프로퍼티는 탐색하는 구조에 따라 달라집니다. 특정 구조를 탐색하는데 필요한 프로퍼티를 확인하려면 다음 구조 조직을 참조하십시오.

- [루프 프로퍼티](#)
- [케이스 구조 프로퍼티](#)
- [Timed 시퀀스 구조 프로퍼티](#)

다음 단계

시작하기 전: 다음 과정을 각 단계별로 보여주는 예제를 확인하려면 labview\examples\Application Control\VI Scripting\Finding and Modifying Objects 디렉토리에 있는 Navigating Structures VI를 엽니다.



이 특정 예제에 대한 추가 정보를 보려면 과정을 확장합니다.

1. 탐색하려는 구조의 **참조를 연습니다**.

예제 세부사항

예제에서, 스크립팅 VI는 [G객체의 탐색] VI를 사용하여 타겟 VI에 위치한 모든 케이스 구조의 참조를 연습니다.

2. 구조 참조를 사용하여 전체 탐색 작업에 적합한 구조의 원소에 대한 참조를 연습니다.

터널, 시프트 레지스터, 케이스 구조 프레임 이름 등이 구조의 원소로 포함될 수 있습니다. 서로 다른 타입의 구조를 구성하는 부분들에 대한 시각적인 설명은 **일반 VI 객체의 조직**을 참조하십시오.

예제 세부사항

예제에서, 스크립팅의 목적은 케이스 선택자에 연결된 객체를 탐색하고 분석하는 것입니다. 따라서, 스크립팅 VI는 케이스 구조:선택자 프로퍼티를 사용하여 각 케이스 구조의 선택자 터널에 대한 참조를 연습니다. 이 참조는 연결된 객체에 대한 탐색을 계속하도록 합니다.

3. 구조 원소의 참조를 사용하여 원소를 편집하거나 탐색을 계속합니다.

구조 조직이 보여주는 것처럼, 구조의 일부 원소는 다른 구조 원소에 대한 접근을 가능하게 합니다. 예를 들어, 터널의 참조를 사용하여 해당 터널의 안쪽 및 바깥쪽 터널의 참조를 얻을 수 있습니다.

예제 세부사항

예제에서, 스크립팅 VI는 케이스 선택자의 참조를 사용하여 **노드(케이스 선택자)에서 연결된 와이어로 탐색**을 계속합니다. 스크립팅 VI는 케이스 선택자의 터널:외부 터미널 프로퍼티를 사용하여 케이스 선택자의 외부 터미널에 대한 참조를 연습니다. 스크립팅 VI가 연결된 와이어의 참조를 얻으려면 터미널 참조를 필요로 합니다.

4. [참조 닫기] 함수를 사용하여 구조 또는 구조 원소의 모든 참조를 사용 후 닫습니다.

유의점 및 권장사항

열어 놓은 참조 모두 닫기—지나치게 많은 수의 참조가 열려 있으면 VI의 성능에 부정적인 영향을 미칩니다. 성능에 대한 세부사항은 **VI 및 객체 참조 닫기**를 참조하십시오.

추가 예제

없음

VI 스크립팅 길라잡이

필수: **VI 스크립팅**

이 모음에는 **VI 스크립팅** 함수, 프로퍼티, 메소드를 사용하여 새 VI를 생성하고, 블록 다이어그램에 객체를 추가하고, 추가한 객체의 위치를 변경하고, 객체를 서로 연결하는 길라잡이가 포함됩니다.

길라잡이를 시작합니다.

길라잡이에 설명된 프로그래밍 개념에 대한 자세한 내용은 **일반적인 VI 스크립팅 태스크**에서 확인하십시오.



(Windows) 관련된 토픽을 보려면 이 윈도우의 맨 위에 있는 도구 모음에서 왼쪽 그림과 같은 위치 보기 버튼을 클릭합니다. LabVIEW 도움말의 내용 탭에서 이 토픽이 하이라이트되어 사용자가 관련 토픽을 탐색할 수 있게 합니다.

길라잡이: VI 스크립팅을 사용하여 VI를 생성하고 열기

다음 단계를 완료하여 VI와 VI 스크립트를 생성하여 타겟 VI를 여십시오.

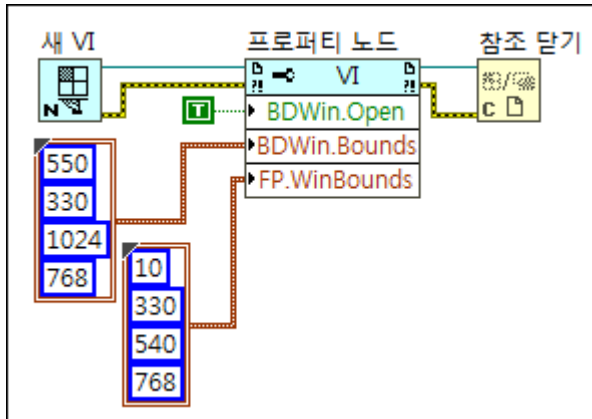


노트

- 이 토픽은 **VI 스크립팅** 개요에 익숙하다고 가정합니다.
- VI 스크립팅 VI 및 함수를 디스플레이하고 이와 연관된 프로퍼티와 메소드를 사용하려면 **VI 스크립팅을 활성화**해야 합니다.

- 파일>>새로 만들기>>VI를 선택하여 새 VI를 생성합니다.
- 새 VI 함수를 블록다이어그램에 추가합니다.
- 프로퍼티 노드를 새 VI 함수의 오른쪽에 추가합니다.
- 새 VI 함수의 vi 참조 번호 출력을 프로퍼티 노드의 참조 입력에 연결합니다.
- 세 개의 터미널을 포함하도록 **프로퍼티 노드의 크기를 재조정**합니다.
- 다음의 프로퍼티를 프로퍼티 노드에 추가 하십시오.
 - 블록다이어그램 윈도우:열기—프로퍼티 노드의 첫 번째 터미널을 클릭하여 바로가기 메뉴에서 블록다이어그램 윈도우>>열기를 선택합니다. 블록다이어그램 윈도우 프로퍼티가 VI 스크립팅 클래스에 속하기 때문에 프로퍼티 노드는 블록 다이어그램 윈도우 프로퍼티가 추가될 때 파란색으로 변합니다.
 - 블록다이어그램 윈도우:윈도우 경계—프로퍼티 노드의 두 번째 터미널을 클릭하여 바로가기 메뉴에서 블록다이어그램 윈도우>>윈도우 경계를 선택합니다.
 - 프론트패널 윈도우:패널 경계—프로퍼티 노드의 세 번째 터미널을 클릭하여 바로가기 메뉴에서 프론트패널 윈도우>>패널 경계를 선택합니다.
- 프로퍼티 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 것을 쓰기로 변경을 선택합니다.
- 각각의 프로퍼티 노드에 상수를 생성하고 10번째 단계의 블록다이어그램과 일치하도록 상수를 설정합니다.
- [참조 닫기] 함수를 프로퍼티 노드의 오른쪽에 추가합니다.

10. 다음 블록다이어그램과 일치하도록 와이어를 모두 연결합니다.



11. ScriptingTutorial.vi로 **VI**를 저장합니다.
 12. **VI**를 실행합니다.

LabVIEW는 제목없는 프런트패널과 블록다이어그램 윈도우를 생성하고 엽니다.

이 길라잡이를 진행하려면 **길라잡이:VI 스크립팅을 사용해서 VI에 객체 추가하기** 토픽을 진행하십시오.

길라잡이:VI 스크립팅을 사용하여 VI에 객체 추가하기

프런트패널과 블록다이어그램에 VI들과 VI 스크립팅의 객체들을 추가할 수 있습니다. 이 길라잡이는 객체를 블록다이어그램 추가합니다. 다음 단계를 따라 VI 스크립팅을 사용해서 VI에 객체를 추가합니다.



노트

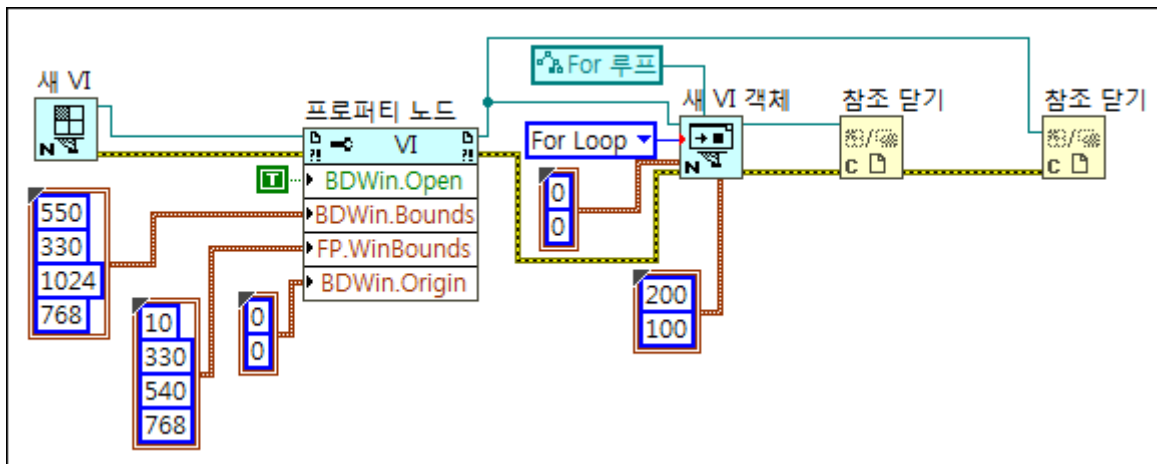
- 이 토픽은 **VI 스크립팅** 개요에 익숙하다고 가정합니다.
- VI 스크립팅 VI 및 함수를 디스플레이하고 이와 연관된 프로퍼티와 메소드를 사용하려면 **VI 스크립팅을 활성화**해야 합니다.

- 이전 연습문제**를 이미 완료 하지 않았다면 완료하십시오. 이전 연습문제에서 제목없는 VI가 열려 있다면 VI를 종료하십시오.
- 참조 달기 함수에서 와이어를 제거하고 블록다이어그램에 여백이 있을수 있도록 함수를 오른쪽으로 이동시키십시오.
- 블록다이어그램 윈도우:원형 프로퍼티를 프로퍼티 노드에 추가하고 수평 및 수직 입력값이 0 인 상수를 생성합니다. 블록다이어그램 왼쪽 위 코너의 수직 및 수평 좌표를 설정합니다. 이 연습문제 이후에 For 루프의 지점을 왼쪽상단에 나타낼수 있도록 지정하십시오.
- [새 VI 객체] 함수를 프로퍼티 노드 오른쪽에 추가하십시오.
- 프로퍼티 노드의 참조 출력 출력을 [새 VI 객체] 함수의 소유자 참조 번호 입력에 연결합니다.
- [새 VI 객체] 함수의 vi 객체 클래스 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택하여 클래스 지정자 상수를 생성합니다.

- 클래스 지정자 함수를 클릭하고 일반 >> G 객체 >> 노드 >> 구조 >> 루프 >> For 루프를 바로가기 메뉴에서 선택합니다.

노트 이 연습문제에서 For 루프와 블록다이어그램을 추가합니다. [올바른 참조](#)를 통해 프런트패널이나 블록다이어그램의 객체를 추가하거나 조작 할 수 있습니다.

- [새 VI 객체] 함수의 스타일 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로가기 메뉴에서 생성 >> 상수를 선택하고, 상수의 풀다운 메뉴에서 For 루프를 선택합니다.
- [새 VI 객체] 함수의 위치 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로가기 메뉴에서 생성 >> 상수를 선택하고, 수평 및 수직 입력의 값으로 0을 입력합니다. 이 단계는 좌표를 입력하여 블록다이어그램에서 새 객체의 위치를 지정합니다.
- [새 VI 객체] 함수의 경계 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로가기 메뉴에서 생성 >> 상수를 선택하고, 폭 및 높이 입력에 각각 200과 100을 입력합니다. 이 단계는 For 루프의 크기를 지정합니다.
- 다른 참조 달기 함수를 이미 배치되어 있는 참조 달기 함수 오른쪽에 추가합니다. VI의 참조와 For 루프를 달는것은 메모리 손실이 없을 수 있도록 합니다. VI가 For 루프의 객체를 포함하고 있기 때문에 VI 참조를 For 루프의 참조를 달은 후에 달 수 있도록 합니다.
- 다음 블록다이어그램과 일치하도록 와이어를 모두 연결합니다.



- VI를 저장합니다.
- VI를 실행합니다.

LabVIEW는 제목없는 프런트패널과 블록다이어그램 윈도우를 생성하고 엽니다. 블록다이어그램은 For 루프를 왼쪽 상단에 포함하고 있습니다.

참조된 객체 생성의 추가적인 옵션

다음 옵션들을 참조도니 객체들을 생성하는데 고려해 봅니다.

- 프런트패널이나 블록다이어그램에 사용자 정의 객체를 추가하려면, 사용자 정의 객체의 경로를 [새 VI 객체] 함수에 연결하거나 참조에서 생성 메소드를 사용하십시오.

- 보다 구체적인 클래스 함수를 이용해서 일반 객체 참조를 컨트롤과 같은 보다 구체적인 참조로 전환시키십시오. 이렇게 하면 보다 구체적인 참조 타입의 프로퍼티를 사용할 수 있습니다.

이 길라잡이를 진행하려면 **길라잡이:VI 스크립팅을 사용해서 VI에 객체 추가하기** 토픽을 진행하십시오.

길라잡이:VI 스크립팅을 사용해서 VI의 객체 위치 변경

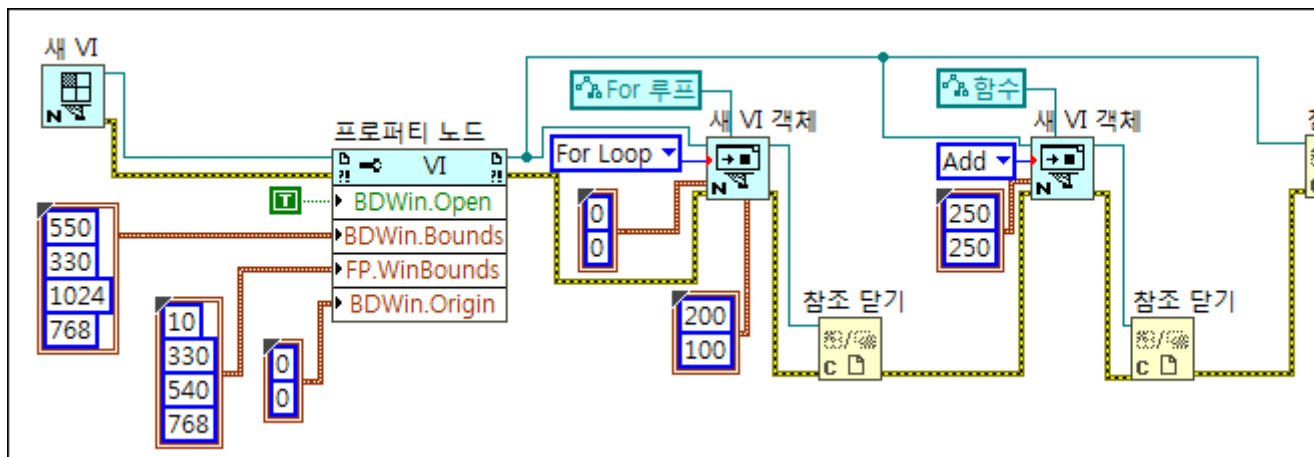
프런트패널과 블록다이어그램에 VI들과 VI 스크립팅의 객체들의 위치를 변경할 수 있습니다. 이 길라잡이는 블록다이어그램의 객체의 위치를 변경합니다. 다음 단계들을 완료하기 위해 VI 스크립팅을 이용해서 블록다이어그램의 VI들의 객체들을 위치 변경하십시오.



노트

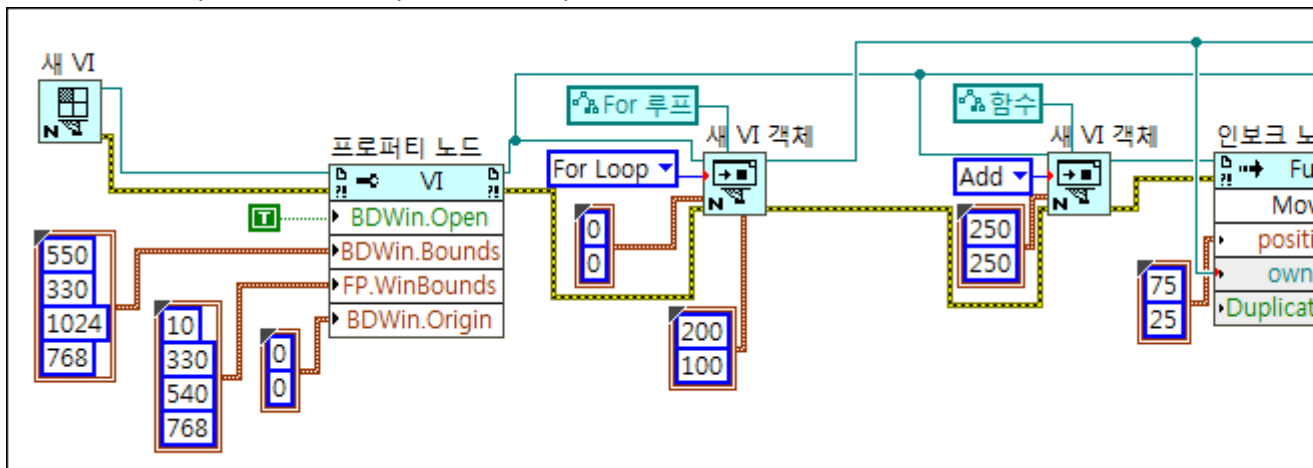
- 이 토픽은 **VI 스크립팅** 개요에 익숙하다고 가정합니다.
 - VI 스크립팅 VI 및 함수를 디스플레이하고 이와 연관된 프로퍼티와 메소드를 사용하려면 **VI 스크립팅을 활성화**해야 합니다.
1. **이전 연습문제**를 이미 완료 하지 않았다면 완료하십시오. 이전 연습문제에서 제 목없는 VI가 열려 있다면 VI를 종료하십시오.
 2. VI 참조를 닫는[참조 닫기] 함수에서 와이어를 제거하고 블록다이어그램에 여백 이 있을 수 있도록 함수를 오른쪽으로 이동시키십시오.
 3. 블록다이어그램의 타겟 VI에 [더하기] 함수를 생성할 수 있도록 [새 VI 객체] 함수를 추가하십시오. 새 VI 객체 함수를 설정할 수 있도록 다음 블록다이어그램에서 보이는 것과 같이 연결을 완성하십시오. 모든 참조가 닫혔는지를 확인합니다. 소유자 참조번호를 VI로 지정합니다.

팁 클래스 지정자 상수에서 함수를 지정하려면 바로 가기 메뉴에서 일반>>G객체>>노드>>함수>>함수를 선택합니다.



4. **VI를 저장합니다.**

5. **VI를 실행**합니다. LabVIEW는 제목없는 프런트패널과 블록다이어그램 윈도우를 생성하고 엽니다. 블록다이어그램의 왼쪽 상단에는 For 루프, 하단에는 [더하기] 함수가 포함되어 있습니다. For 루프를 생성하는 [새 VI 객체] 함수의 객체 참조번호 입력을 [더하기] 함수를 생성하는 [새 VI 객체] 함수의 소유자 참조번호 입력에 연결하여, For 루프 안에 [더하기] 함수를 놓을 수 있습니다. 그런 다음, [더하기] 함수 생성에 사용되는 [새 VI 객체] 함수의 위치 입력에서, For 루프 내의 좌표 위치를 지정할 수 있습니다. 길라잡이의 다음 단계인 블록다이어그램 객체의 위치 변경을 위한 다른 옵션을 확인하려면, 이동 메소드를 사용하여 [더하기] 함수를 For 루프로 이동시킵니다.
6. LabVIEW의 제목없는 VI가 아직 열려 있을경우에 VI를 종료합니다.
7. 참조 닫기 함수에서 세 함수의 연결을 모드 끄고 블록다이어그램에 여백이 있을 수 있도록 함수를 오른쪽으로 이동시키십시오. 블록다이어그램 그림의 13번째 단계를 참조하십시오.
8. [더하기] 함수를 생성하는 [새 VI 객체] 함수의 오른쪽에 인보크 노드를 추가합니다.
9. [새 VI 객체] 함수의 객체 참조 번호 출력을 인보크 노드의 참조 입력에 연결합니다.
10. 인보크 노드의 메소드 터미널을 클릭한 후 바로가기 메뉴에서 이동을 선택합니다.
11. [더하기] 함수를 이동할 블록다이어그램 객체를 지정하려면, For 루프를 생성하는 [새 VI 객체] 함수의 객체 참조 번호 출력을 인보크 노드의 소유자 입력에 연결합니다.
12. [더하기] 함수를 이동할 For 루프 내의 위치를 지정하려면, 인보크 노드의 위치 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>> 상수를 선택하고, 왼쪽 및 위쪽 입력의 값으로 각각 75와 25를 입력합니다.
13. 다음 블록다이어그램과 일치하도록 와이어를 모두 연결합니다.



14. VI를 저장합니다.
15. VI를 실행합니다.

LabVIEW는 제목없는 프런트패널과 블록다이어그램 윈도우를 생성하고 엽니다. 블록다이어그램은 왼쪽 상단에 For 루프를 포함하고, For 루프 안에는 [더하기] 함수가 있습니다.



노트 삭제 메소드를 사용하여 VI 스크립팅을 사용하는 객체를 삭제합니다. 대체 메소드를 사용해서 객체를 대체할 수 있습니다. 객체를 삭제한 후 깨진 와이어는 블록 다이어그램:깨진 와이어 제거 메소드를 호출하여 정리합니다.

이 길라잡이를 진행하려면 [길라잡이:VI 스크립팅을 사용해서 연결하기](#) 토픽을 진행하십시오.

길라잡이: VI 스크립팅을 사용해서 객체 연결하기

VI 스크립팅을 사용해서 블록 다이어그램에서 객체를 프로그램적으로 연결합니다. 블록 다이어그램에 놓는 객체의 대부분은 노드입니다. 예를 들어, 함수, VI, 구조는 노드입니다. 일반적으로 노드는 연결 가능한 여러 개의 터미널을 가집니다. 와이어(복수) 연결 메소드를 사용하여 **한 개 또는 여러 개의 와이어를 노드에 연결**합니다. 와이어 연결 메소드를 사용하여 **한 개의 와이어를 한 개 터미널에 연결**합니다.



노트

- 이 토픽은 [VI 스크립팅](#) 개요에 익숙하다고 가정합니다.
- VI 스크립팅 VI 및 함수를 디스플레이하고 이와 연관된 프로퍼티와 메소드를 사용하려면 [VI 스크립팅을 활성화](#)해야 합니다.

노드 연결하기

다음 단계를 따라 VI 스크립팅을 사용해서 두 개의 [더하기] 함수를 연결합니다.



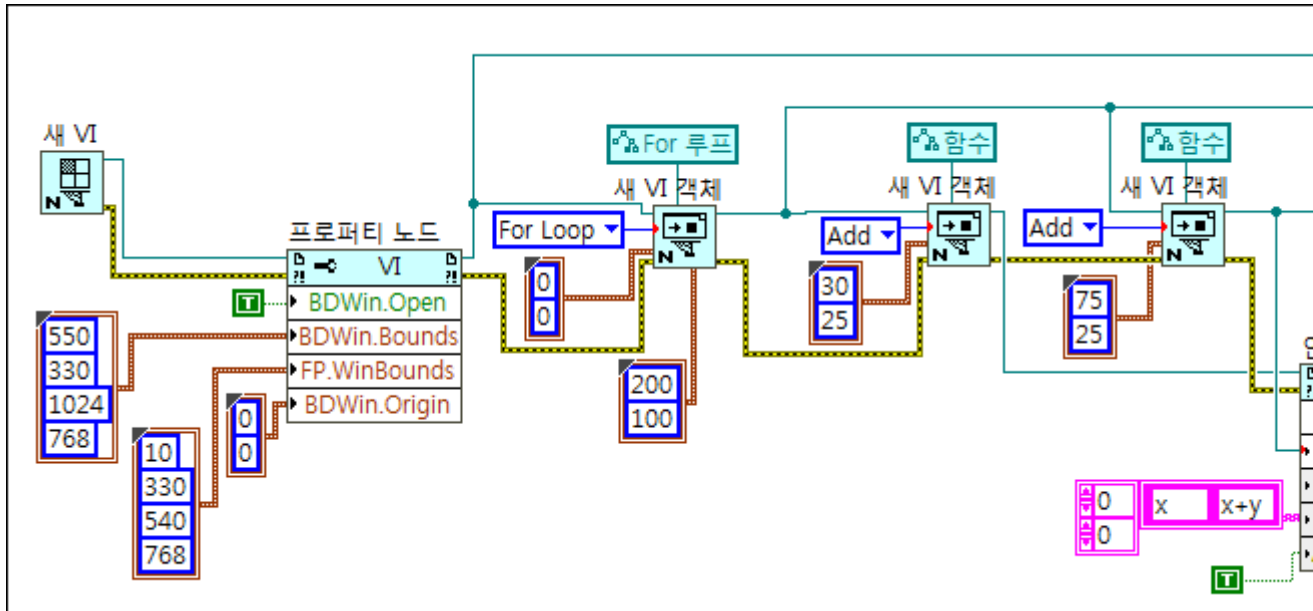
노트 노드는 일반 >> G객체 >> 노드 클래스에 있습니다.

1. [이전 연습문제](#)를 이미 완료 하지 않았다면 완료하십시오. 이전 연습문제에서 제목없는 VI가 열려 있다면 VI를 종료하십시오.
2. 블록 다이어그램에서 이동 메소드와 관련 와이어를 삭제합니다.
3. 다음 단계를 따라 For 루프에 [더하기] 함수를 놓습니다.
 1. [더하기] 함수를 생성하는 [새 VI 객체] 함수에 대한 수평과 수직 입력에 각각 30과 25를 입력합니다.
 2. [더하기] 함수를 생성하는 새 VI 객체 함수의 소유자 참조 번호 입력에서 와이어와 깨진 와이어를 제거합니다.
 3. For 루프를 생성하는 [새 VI 객체] 함수의 객체 참조 번호 출력을 [더하기] 함수를 생성하는 [새 VI 객체] 함수의 소유자 참조 번호 입력에 연결합니다.
4. For 루프 안에 각각 75와 25의 수평 및 수직 입력값을 가진 [더하기] 함수를 생성합니다. 보기는 블록 다이어그램 12번째 단계를 참조하십시오.
5. 블록 다이어그램에서 이전 단계에서 추가한 새 VI 객체 함수의 오른쪽에 인보크 노드를 추가합니다.
6. 첫번째 [더하기] 함수를 생성하는 [새 VI 객체] 함수의 객체 참조 번호 출력을 인보크 노드의 참조 입력에 연결합니다.

7. 인보크 노드를 클릭한 후 바로 가기 메뉴에서 와이어(복수) 연결을 선택합니다.
8. 두번째 [더하기] 함수를 생성하는 [새 VI 객체] 함수의 객체 참조 번호 출력을 와이어(복수) 연결 메소드의 소스 연결 입력에 연결합니다.
9. 와이어(복수) 연결 메소드의 와이어링 스펙 입력에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 생성>> 상수를 선택합니다.
10. 와이어링 스펙 상수에 와이어에 연결하려는 입력과 출력의 이름을 입력하며, 이 경우 이름은 각각 x와 x+y입니다.

노트 또는 경우에 따라 연결하려는 터미널의 이름을 직접 지정하지 않고 자동 와이어링을 할 수도 있습니다. 자동 와이어? 입력을 참으로 설정하여 LabVIEW가 소스 객체를 호환 가능한 인접한 객체에 자동으로 연결하도록 합니다.

11. LabVIEW가 와이어에 대해 최적의 경로를 찾도록 지정하려면 와이어(복수) 연결 메소드의 자동 와이어? 입력에서 마우스 오른쪽 버튼을 클릭하여 상수를 생성한 후, 상수를 참으로 설정합니다.
12. 객체의 참조를 닫고 다음의 블록다이어그램과 일치하도록 모든 와이어를 연결합니다.



13. VI를 저장합니다.
14. VI를 실행합니다.

LabVIEW는 제목없는 프런트패널과 블록다이어그램 윈도우를 생성하고 엽니다. 블록다이어그램 왼쪽 위 코너에 For 루프가 있으며, For 루프에는 두 개의 [더하기] 함수가 있습니다. [더하기] 함수는 함께 연결되어 있습니다.

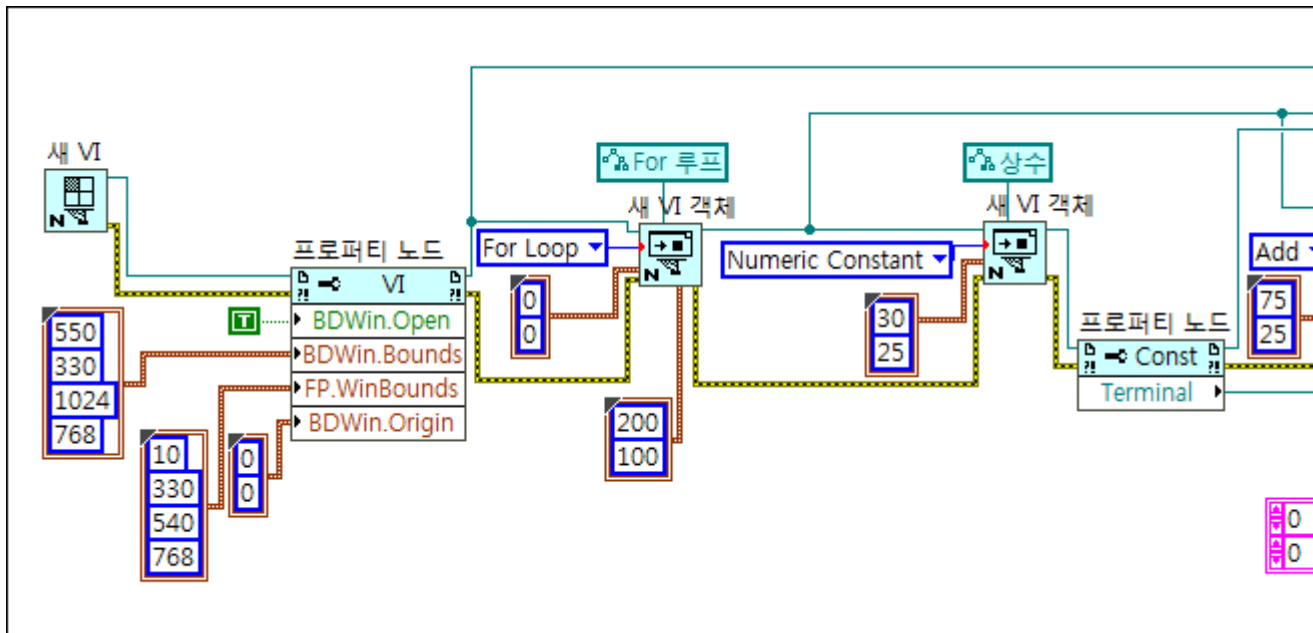
객체 연결하기

이 연습에서는 터미널을 노드에 연결합니다. 터미널에 연결하려면 와이어 연결 메소드를 사용합니다. 다음 단계를 따라 VI 스크립팅을 사용해서 상수를 [더하기] 함수에 연결합니다.



노트 숫자형 상수에 대한 터미널 참조는 일반>>G객체>>상수 클래스에 있습니다.

1. **노드 연결하기** 연습을 이미 완료하지 않았다면 완료하십시오. 이전 연습에서 생성한 제목없는 VI가 열려 있다면 이 VI를 닫습니다.
2. 첫번째 [더하기] 함수를 생성하는 [새 VI 객체] 함수에서 스타일 입력을 숫자형 상수로 바꾸고, VI 객체 클래스 입력을 상수로 바꾼 후, 객체 참조 번호와 에러 출력 출력에서 와이어를 제거합니다.
3. 블록다이어그램에서 숫자형 상수를 생성하는 [새 VI 객체] 함수의 오른쪽에 프로퍼티 노드를 추가합니다. 보기는 블록다이어그램의 9번째 단계를 참조하십시오.
4. 숫자형 상수를 생성하는 [새 VI 객체] 함수의 객체 참조 번호 출력을 프로퍼티 노드의 참조 입력에 연결합니다.
5. 프로퍼티 노드를 클릭하고 바로 가기 메뉴에서 터미널>>참조를 선택합니다.
6. 프로퍼티 노드의 터미널 출력을 기존 인보크 노드의 참조 입력에 연결합니다.
7. 인보크 노드를 클릭한 후 바로 가기 메뉴에서 와이어 연결을 선택합니다.
8. 와이어 연결 메소드의 와이어링 스펙 입력값을 와이어에 연결하려는 입력과 출력의 이름으로 바꾸며, 이 경우 이름은 x와 x입니다.
9. 다음 블록다이어그램과 일치하도록 와이어를 모두 연결합니다.



10. VI를 저장합니다.
11. VI를 실행합니다.

LabVIEW는 제목없는 프런트패널과 블록다이어그램 윈도우를 생성하고 엽니다. 블록다이어그램 왼쪽 위 코너에 For 루프가 있으며, For 루프에는 숫자형 상수와 [더하기] 함수가 있습니다. 숫자형 상수와 [더하기] 함수는 함께 연결되어 있습니다.

새 VI 보이기 엔트리 추가하기

다음 단계를 따라 VI 보이기 리스트에 새 엔트리를 추가합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 옵션 대화 상자의 항목 리스트에서 웹 서버를 선택합니다.
3. VI 보이기 섹션의 VI 보이기 텍스트 박스에 VI 또는 디렉토리 경로의 이름을 입력합니다. 또한, VI 이름이나 디렉토리 경로에 **와일드 카드를 사용**할 수 있습니다. LabVIEW 프로젝트의 일부인 VI를 지정하려면, 해당하는 경우 반드시 프로젝트 이름, 프로젝트 라이브러리, VI의 경로에 있는 타겟을 포함해야 합니다. 예를 들어, MyVI.vi이 My Computer라는 타겟 아래의 MyProject.lvproj라고 불리는 프로젝트에 있을 경우, VI 이름으로 MyProject.lvproj/My Computer/MyVI.vi를 입력합니다. 그 VI가 MyLibrary라고 불리는 프로젝트 라이브러리에 있으면, 이 라이브러리는 MyProject.lvproj/MyComputer/MyLibrary.lvlib:MyVI.vi로 경로 내의 프로젝트 라이브러리에 있습니다. 사용자 VI가 프로젝트나 프로젝트 라이브러리에 없으면, 추가적인 정보 없이 VI 이름을 입력할 수 있습니다.
4. 선택된 아이템에 대한 접근을 허용하기 위해서 접근 허용 버튼을 클릭합니다. 선택된 아이템에 대한 접근을 거부하기 위해서 접근 거부 버튼을 클릭합니다. 프런트패널을 원격으로 제어하는 경우, 여러 클라이언트가 VI 제어를 기다릴 때 원격 클라이언트가 VI를 제어할 수 있는 **시간 제한을 설정**할 수 있습니다. 접근을 허용할 때 녹색 확인 표시가 아이템의 왼쪽에 나타나고, 접근을 거부할 때 붉은색 X가 나타납니다.
5. 엔트리에 녹색 확인 표시나 붉은색 X마크가 그 이름 옆에 있지 않을 때, 엔트리에 대한 구문이 맞지 않습니다. 엔트리를 편집하거나 제거 버튼을 클릭하여 엔트리를 삭제합니다.
6. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

기존의 VI 보이기 엔트리 편집하기

다음 단계를 따라 VI 보이기 리스트에서 VI 이름 또는 디렉토리 경로를 편집합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 웹 서버를 선택합니다.
3. VI 보이기 섹션의 VI 보이기 리스트에서 변경하려는 VI 이름 또는 디렉토리 경로를 선택합니다. 이름 또는 경로가 텍스트 박스에 나타납니다.
4. 텍스트 박스에서 이름 또는 경로를 편집하거나 접근 허용 또는 접근 거부 라디오 버튼을 선택하여 접근을 허용하거나 거부합니다. 또한, 여러 클라이언트가 VI 제어를 기다릴 때 원격 클라이언트가 VI를 제어할 수 있는 **시간 제한을 설정**할 수 있습니다. 기본은 300초입니다. VI 이름이나 디렉토리 경로에 **와일드 카드를 사용**할 수 있습니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

보이는 VI 엔트리 삭제하기

다음 단계를 따라 보이는 VI 리스트에서 VI 이름 또는 디렉토리 경로를 삭제합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 웹 서버를 선택합니다.
3. VI 보이기 섹션의 VI 보이기 리스트에서 삭제하려는 VI 이름 또는 디렉토리 경로를 선택합니다.
4. 제거 버튼을 클릭합니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

프로그램적으로 모달이 아닌 윈도우 호출하기

다른 프로세스가 실행 중일 때에는 LabVIEW 콜백 VI나 DLL에서 모달 윈도우가 아닌 윈도우를 열 수 없습니다. 콜백 VI나 DLL이 모달이 아닌 윈도우를 호출하도록 하려면 사용자는 반드시 다음의 단계를 따라서 프로그램적으로 실행해야 합니다:

1. 블록다이어그램에 [VI 참조 열기] 노드를 추가합니다.
2. 열고자 하는 모달이 아닌 윈도우의 경로에 vi 경로 입력을 연결합니다.



노트

VI 기반의 윈도우 경로만 vi 경로 입력에 연결할 수 있습니다. **라이브러리 함수 호출 노드를 사용하여** VI 기반이 아닌 윈도우를 엽니다. LabVIEW 콜백 VI 또는 DLL로부터 윈도우를 호출하기 전에, 윈도우가 모달로 설정되어 있는지 반드시 확인해야 합니다.

1. 블록다이어그램에 인보크 노드를 추가합니다.
2. VI 참조 열기 노드의 vi 참조 출력을 인보크 노드의 참조 입력에 연결합니다.
3. 인보크 노드에서 메소드로 VI 실행을 선택합니다.
4. 거짓 불리언 상수를 메소드의 완료까지 기다림 입력에 연결합니다.

웹 서비스를 VI의 라이브러리로 반입하기

웹 서비스 반입 마법사를 사용하여 웹 서비스에 있는 메소드를 기반으로 하는 래퍼 VI를 생성하거나 업데이트합니다. 웹 서비스 설명 언어(WSDL)에 대한 유효한 URL을 제공해야

하며, WSDL은 ASP.NET에서 올바르게 검증해야 합니다. WSDL은 웹 서비스와 기능을 설명하는데 사용되는 XML 포맷의 언어입니다.



노트

- 웹 서비스 반입 마법사를 사용하려면 .NET Framework 4.0이 설치되어 있어야 합니다.
- 반입하려는 웹 서비스의 URL을 변경하는 경우, 다음 단계에 따라 웹 서비스의 VI를 다시 생성해야 합니다.

다음의 단계를 따라 웹 서비스에 있는 메소드를 기반으로 하는 VI의 라이브러리를 생성하거나 업데이트합니다.

1. 도구 » 반입 » 웹 서비스를 선택하여 웹 서비스 반입 마법사를 실행합니다.
2. URL을 반입하고자 하는 웹 서비스에 대한 WSDL에 입력합니다.
3. 다음 버튼을 클릭하고 마법사의 다음 페이지를 디스플레이합니다.
4. (옵션) 인증 또는 프록시 정보를 입력합니다.
5. 다음 버튼을 클릭하고 마법사의 다음 페이지를 디스플레이합니다.
6. 생성하려는 프로젝트 라이브러리 .lvlib 파일의 이름을 입력합니다.
7. 대상 디렉토리를 입력합니다. 이것은 프로젝트 라이브러리 .lvlib 파일을 저장하고자 하는 위치를 말합니다. 폴더를 탐색하여 선택할 수 있습니다.
8. 다음 버튼을 클릭하고 마법사의 다음 페이지를 디스플레이합니다.
9. 반입할 메소드를 선택합니다. 선택한 메소드에 대해 LabVIEW는 동일한 기능을 가진 VI를 생성합니다.
10. 다음 버튼을 클릭하고 마법사의 요약 페이지를 디스플레이합니다. LabVIEW가 래퍼 VI를 생성하기 전에 지정한 웹 서비스 정보를 이 페이지를 통해 검토합니다. 뒤로 버튼을 사용하여 변경할 수 있습니다.
11. 웹 서비스 메소드를 반입하고 메소드에 대한 래퍼 VI를 생성할 준비가 되어있으면 생성 버튼을 클릭하십시오. 생성이 완료되면 완료 페이지가 나타납니다.
12. 마법사가 닫혀있을 때 생성된 라이브러리 파일을 열거나 리포트를 보고자 한다면 적절한 옵션을 선택합니다.
13. 마침 버튼을 클릭하여 마법사를 닫고 지정한 파일을 선택적으로 열 수 있습니다.

객체

객체 정렬하기

정렬 눈금을 사용하여 **객체를 놓을 때 객체를 정렬**합니다.

객체를 놓은 후에 축을 따라서 객체를 정렬하려면 정렬하려는 객체를 선택하고, 도구 모음의 객체 정렬 풀다운 메뉴 를 사용하거나 편집 » 아이템 정렬을 선택합니다. 위쪽이나 아래쪽 경계 또는 중심 포인트를 사용하는 수평 축을 따라서 객체를 정렬할 수 있습니다. 왼쪽이나 오른쪽 경계 또는 중심 포인트를 사용하는 수직 축을 따라서 객체를 정렬

할 수 있습니다. 선택한 정렬 옵션은 변경하기 이전까지 적용됩니다. 풀다운 메뉴에서 아이콘 주위의 어두운 경계는 현재 정렬 옵션을 나타냅니다.

다음 단계를 따라 객체를 정렬합니다.

1. 정렬하려는 **객체를 선택**합니다.
2. 프런트패널과 블록다이어그램에서 도구 모음의 텍스트 셋팅 풀다운 메뉴 옆에 위치한 객체 정렬 풀다운 메뉴를 선택합니다.
3. 다음 옵션 중 선택합니다.
 - 위쪽 경계-선택된 객체의 모든 위쪽 경계를 가장 위쪽에 있는 객체로 정렬합니다.
 - 수직 중심-위 아래로 가장 긴 객체의 중간 포인트로 객체를 정렬합니다.
 - 아래쪽 경계-선택된 객체의 모든 아래쪽 경계를 가장 아래쪽에 있는 객체로 정렬합니다.
 - 왼쪽 경계-선택된 객체의 모든 왼쪽 경계를 가장 왼쪽에 있는 객체로 정렬합니다.
 - 수평 중심-왼쪽 오른쪽으로 가장 긴 객체의 중간 포인트로 객체를 정렬합니다.
 - 오른쪽 경계-선택된 객체의 모든 오른쪽 경계를 가장 오른쪽에 있는 객체로 정렬합니다.

객체의 다른 그룹에 정렬 동작을 반복하려면, <Ctrl-Shift-A> 키를 누릅니다. (macOS) <Command-Shift-A> 키를 누릅니다. (Linux) <Alt-Shift-A> 키를 누릅니다.

또한, 편집>>아이템 정렬을 선택하여 객체의 다른 그룹에 정렬 동작을 반복할 수 있습니다.

객체와 배경 색칠하기

프런트패널 객체, 프런트패널 팬과 블록다이어그램 작업 공간의 색을 추가하거나 변경할 수 있습니다. 일부 객체는 색을 따로 칠할 수 있는 전경과 배경을 가집니다. 예를 들어, 클래식 숫자형 팔레트의 노브의 전경색은 주요 다이얼 영역의 색이며, 배경색은 도드라진 테두리의 색입니다. **사용자 색을 생성**하고, **한 객체에서 다른 객체로 색을 복사**하고, **투명한 객체를 생성**할 수 있습니다.

다음 단계를 따라 객체의 색, 프런트패널 배경색 또는 블록다이어그램 작업 공간의 배경색을 변경합니다.

1. 색칠 도구를 사용하여 객체 또는 작업 공간에서 마우스 오른쪽 버튼을 클릭하여 색 선택기를 디스플레이합니다. **자동 도구 선택이 활성화**된 경우, 도구 팔레트에서 색칠 도구를 선택하여 자동 도구 선택을 비활성화합니다.



팁

<Shift> 키를 누르고 마우스 오른쪽 버튼을 클릭하여 커서의 위치에서 도구 팔레트의 임시 버전을 디스플레이할 수 있습니다.

색 선택기의 위쪽 부분에 회색조 스펙트럼과 **투명한 객체를 생성**할 때 사용할 수 있는 상자가 있습니다. 두 번째 스펙트럼에는 배경과 프런트패널 객체에 사용하기 적당한 부드러운 색이 포함되어 있습니다. 세 번째 스펙트럼은 하이라이트에 적당한 색을 포함합니다. 색 상자의 첫 번째 행은 **사용자 정의된 색**을 디스플레이합니다. 색 상자의 두 번째 행은 가장 최근에 선택된 색과 **시스템 색**을 디스플레이합니다. 시스템 색을 사용하면 프런트패널의 모양을 VI를 실행하는 컴퓨터의 시스템 색에 맞게 설정할 수 있습니다. 시스템 색은 배경 색과 텍스트 색이 포함된 세 가지 색의 쌍입니다. 패널, 객체, 윈도우, 메뉴와 같은 아이템의 하이라이트 색입니다. 색 선택기의 아래쪽에 있는 상자는 현재 선택된 색을 디스플레이합니다. 전경 색과 배경 색이 있는 객체는 이 상자의 왼쪽 절반에 전경 색을, 오른쪽 절반에 배경 색을 디스플레이합니다.

2. 색을 선택하고 객체 또는 배경을 선택된 색으로 미리보기 위해서 커서를 색 선택기 위로 이동합니다. 커서를 색 위로 이동할 때, LabVIEW는 색의 RGB 값을 색 선택기의 아래 부분에 디스플레이합니다. 객체가 전경 색과 배경 색을 가지는 경우, 전경 색을 선택하고 설정하도록 <F> 키를 누르거나 배경 색을 선택하고 설정할 수 있도록 키를 누릅니다. 또한, 스페이스 바를 눌러서 전경 색과 배경 색 사이를 토글할 수 있습니다.
3. (옵션) 색 선택기의 오른쪽 하단 코너의 더 많은 색 버튼을 클릭한 후 OS의 색 유틸리티를 사용하여 **특정 색을 선택**합니다.
4. 객체 또는 배경에 적용하려는 색을 클릭합니다. 색칠 작업을 취소하려면, <Esc> 키를 누릅니다.

객체 복사하기 및 붙여넣기

두 가지 방법으로 객체를 복사하고 붙여넣을 수 있습니다. **객체를 선택**하고 편집>>복사와 편집>>붙여넣기를 선택하거나, 객체를 복제할 수 있습니다.

다음 단계를 따라 객체를 복제합니다.

1. 객체를 클릭하기 위해 위치 도구를 사용하는 동안 <Ctrl> 키를 누릅니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.
2. 새 위치로 복사본을 끄는 동안 계속 키를 누릅니다.

(Linux) 또한, 중간 마우스 버튼으로 객체를 클릭하여 새 위치로 객체를 끌어서 객체를 복제할 수 있습니다.

라벨이 있는 객체를 복제하거나 복사할 때, LabVIEW는 원래 객체의 이름을 복사하여 숫자를 추가합니다.



노트

객체를 복사해서 다른 VI에 놓으면 객체에 대한 설명도 같이 복사됩니다. 새 객체에 대한 설명을 수정해야 할 수도 있습니다.

블록 다이어그램 터미널을 복제하거나 복사하는 경우, 새 터미널은 블록 다이어그램에 나타납니다. LabVIEW는 원래 터미널의 로컬 변수, 컨트롤 참조, 프로퍼티 노드, 인보크 노드는 복사하지 않습니다.

로컬 변수 또는 컨트롤 참조를 복사하고 붙여넣을 경우, LabVIEW는 터미널 또한 복사합니다. 로컬 변수나 컨트롤 참조를 복제할 경우, LabVIEW는 원래 터미널에 새로운 참조를 생성합니다.

블록 다이어그램에서 프런트패널 컨트롤을 복사하여 새 VI의 블록 다이어그램에 붙여넣으면 프런트패널의 배열은 원본 VI와 달라집니다. 프런트패널 객체는 오버래핑 또는 보이지 않는 영역에 나타남을 방지하기 위해 왼쪽 위 코너에 놓여집니다.

함축적으로 연결된 프로퍼티 노드 또는 인보크 노드를 복사하고 붙여넣기를 할 경우, LabVIEW는 터미널에서 노드의 연결을 끊습니다. 함축적으로 연결된 노드를 복제할 경우, LabVIEW는 새로운 함축적으로 연결된 노드를 생성합니다. 노드를 새 VI로 끌어서 노드를 복제할 경우, LabVIEW는 터미널에서 노드의 연결을 끊습니다. 터미널에 **노드를 다시 연결**할 수 있습니다.

객체를 한 VI에서 끌어서 다른 VI에 놓음으로 선택된 객체를 VI 사이에서 복사할 수 있습니다.



노트 SubVI를 복사하는 경우, 클립보드(📄)가 VI 계층구조 윈도우에 나타납니다. VI가 메모리에 있지 않을 때 복사한 SubVI는 클립보드의 메모리에 그대로 있습니다.

한 객체에서 다른 객체로 색 복사하기

다음 단계를 따라 색 선택기를 사용하지 않고 한 객체의 색을 복사하여 두번째 객체로 복사합니다.

1. 색 얻기 도구를 사용하여 복사하려는 전경색과 배경색을 갖는 객체를 클릭합니다. **자동 도구 선택이 활성화**된 경우, 도구 팔레트에서 색 얻기 도구를 선택하여 자동 도구 선택을 비활성화합니다.



팁

<Shift> 키를 누르고 마우스 오른쪽 버튼을 클릭하여 커서의 위치에서 도구 팔레트의 임시 버전을 디스플레이할 수 있습니다.

2. 색칠 도구를 사용하여 색을 적용하려는 객체를 클릭합니다. 객체는 첫번째 객체에서 복사한 색으로 바뀝니다.

투명한 객체 생성하기

사용자가 부분적인 옵션을 선택할 때 같은 장소에 여러 개의 컨트롤과 인디케이터가 나타나도록, 프런트패널 객체를 투명하게 만들 수 있습니다. 투명 객체의 모양에만 영향을 줍니다. 객체는 일반적으로 마우스와 키 동작에 응답합니다.



노트

시스템 컨트롤의 색은 변경할 수 없습니다. 왜냐하면 시스템 컨트롤은 VI를 실행하는 플랫폼에 따라 모양이 변하기 때문입니다. 이들 컨트롤의 색깔은 사용자 시스템 설정과 같아집니다.

다음 단계를 따라 객체를 투명하게 만듭니다:

1. 색칠 도구를 사용하여 객체에서 마우스 오른쪽 버튼을 클릭하여 색 선택기를 디스플레이합니다. **자동 도구 선택이 활성화**된 경우, 도구 팔레트에서 색칠 도구를 선택하여 자동 도구 선택을 비활성화합니다.

팁 <Shift> 키를 누른 상태에서 마우스 오른쪽 버튼을 클릭하여 커서의 위치에서 도구 팔레트의 임시 버전을 디스플레이합니다.

2. 색 선택기의 오른쪽 상단 코너에 위치한 T 상자를 선택합니다.

또한, 프런트패널 객체의 부분인 경계선을 포함하여 투명하게 만들 수 있습니다.

다음 단계를 따라 객체의 경계선을 투명하게 만듭니다:

1. 색칠 도구를 사용하여 문자열 컨트롤에서 색 선택기를 디스플레이할 때와 같이 객체의 경계에서 마우스 오른쪽 버튼을 클릭합니다. **자동 도구 선택이 활성화**된 경우, 도구 팔레트에서 색칠 도구를 선택하여 자동 도구 선택을 비활성화합니다.
2. <F> 키를 누른 상태에서 색 선택기의 오른쪽 상단 코너에 위치한 T 상자를 선택합니다. LabVIEW는 컨트롤 경계를 투명하게 만듭니다. 프런트패널, 또는 객체를 포함하는 컨트롤의 배경색은 컨트롤 경계선의 색을 결정합니다.

숨겨진 프런트패널 객체 디스플레이하기

다음 단계를 따라 숨겨진 프런트패널 컨트롤 또는 인디케이터를 디스플레이합니다. 또한, **프런트패널 컨트롤과 인디케이터를 숨길** 수 있습니다.

1. 객체의 블록다이어그램 터미널을 찾습니다. 같은 데이터 타입을 가진 여러 터미널이 있다면, 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 **아이템>>라벨**을 선택합니다. 숨겨진 프런트패널 객체와 일치하는 라벨을 가진 객체를 찾습니다.
2. 터미널에서 마우스 오른쪽 버튼을 클릭하고 객체가 컨트롤이면 바로 가기 메뉴에서 **컨트롤 보이기를 선택**하고 객체가 인디케이터이면 **인디케이터 보이기를 선택**합니다.

또한, 보이기 프로퍼티를 사용하여 프로그램적으로 컨트롤과 인디케이터를 디스플레이할 수 있습니다.

또한, **모든 프런트패널 컨트롤과 인디케이터를 디스플레이**할 수 있습니다.

객체 간격 조절하기

객체 간격 조절 풀다운 메뉴 를 사용하거나 **편집>>아이템 간격 조절**을 선택하여 객체의 간격을 균일하게 합니다. 선택한 간격 조절 옵션은 변경할 때까지 영향을 미칩니다. 풀다운 메뉴에서 팔레트 아이콘 주위의 두꺼운 경계선은 현재 간격 조절 옵션을 나타냅니다.

다음 단계를 따라 객체를 간격 조절합니다.

1. 간격 조절하려는 **객체를 선택**합니다.
2. 프런트패널과 블록다이어그램 윈도우에서 도구 모음의 **객체 정렬** 폴다운 메뉴 오른쪽에 위치한 객체 간격 조절 폴다운 메뉴를 선택합니다.
3. 다음 옵션 중 선택합니다.
 - 위쪽 경계—선택한 객체의 위쪽 경계를 기준잡아 같은 거리만큼 간격 조절합니다.
 - 수직 중심—선택한 객체들의 수직 센터가 균일한 간격이 되도록 조절합니다.
 - 아래쪽 경계—선택한 객체들의 아래쪽 경계가 균일한 간격이 되도록 조절합니다.
 - 수직 간격—객체 사이의 수직 공간이 같도록 선택된 객체를 간격 조절합니다.
 - 수직 압축—객체의 위 아래 경계 사이에 공간이 없도록 선택된 객체를 간격 조절합니다.
 - 왼쪽 경계—선택한 객체들의 왼쪽 경계가 균일한 간격이 되도록 조절합니다.
 - 수평 중심—선택한 객체들의 수평 센터가 균일한 간격이 되도록 조절합니다.
 - 오른쪽 경계—선택한 객체들의 오른쪽 경계가 균일한 간격이 되도록 조절합니다.
 - 수평 간격—객체 사이의 수평 공간이 같도록 선택된 객체를 간격 조절합니다.
 - 수평 압축—객체의 왼쪽과 오른쪽 경계 사이에 공간이 없도록 선택된 객체를 간격 조절합니다.

객체의 다른 그룹에 간격 조절 동작을 반복하려면, <Ctrl-D> 키를 누릅니다. (macOS) <Command-D> 키를 누릅니다. (Linux) <Alt-D> 키를 누릅니다.

또한, 편집>>아이템 간격 조절을 선택하여 다른 그룹의 객체에 간격 조절 작업을 반복할 수 있습니다.

객체 그룹화하기 및 잠그기

위치 도구를 사용하여 그룹화 및 잠금 설정을 하려는 프런트패널 객체를 선택합니다. 도구 모음의 순서 재설정 버튼  을 클릭한 후 폴다운 메뉴에서 그룹 또는 잠금을 선택합니다. 그룹화된 객체는 위치 도구를 이용하여 위치를 옮기거나 크기를 바꿀 때, 상호간의 상대적인 배열과 크기를 유지합니다. 잠겨진 객체는 프런트패널에서 움직이지 않습니다. 그룹화된 객체는 그룹화를 해제하기 전까지는 프런트패널이나 블록다이어그램에서 삭제할 수 없습니다. 잠근 객체는 잠금을 해제하기 전까지는 프런트패널이나 블록다이어그램에서 삭제할 수 없습니다. 객체에 대한 그룹화와 잠금 설정을 동시에 할 수 있습니다. 동시에 객체를 그룹화하고 잠금으로 설정하는 경우, 프런트패널이나 블록다이어그램에서 지우기 전에 그룹화와 잠금 설정을 해제해야 합니다. 위치 도구 이외의 다른 도구들도 그룹화되고 잠금 설정된 객체에서 사용할 수 있습니다.

다음 단계를 따라 프런트패널 객체를 그룹화하고, 그룹화를 해제하고, 잠그고, 잠금해제를 합니다.

1. 위치 도구를 가지고 객체 주변의 공간을 끌어오거나, 각 객체를 클릭하면서 <Shift> 키를 눌러서 객체를 선택합니다. <Shift> 키를 클릭하는 방법을 사용하여 잠겨 있는 객체를 선택할 수 없습니다.
2. 순서재설정 버튼을 클릭하고 풀다운 메뉴에서 그룹, 그룹 해제, 잠금, 잠금해제를 선택합니다. 객체에 대한 그룹화와 잠금 설정을 동시에 할 수 있습니다.



노트

객체를 겹치게 하거나 층으로 놓을 때, 그룹으로 선택한 객체는 그룹에서 가장 앞의 객체가 위치해 있는 층으로 같이 움직입니다.

그룹의 일부인 객체를 선택할 때, 그룹은 노란색과 검은색 점선으로 하이라이트됩니다. 잠겨 있는 객체를 선택할 때, 회색과 검은색 점선으로 하이라이트됩니다. 그룹으로 되어 있고 잠겨 있는 객체를 선택할 때, 그룹은 회색과 검은색 점선으로 하이라이트됩니다.

그룹 안에서 객체의 상대적인 배치를 변경하려면, 그룹해제를 선택하고, 객체를 다시 배치한 후에 그룹을 선택합니다.

객체 순서 재설정하기

라벨 또는 다른 객체를 컨트롤 또는 인디케이터 위에 또는 일부가 겹쳐지도록 놓으면 스크린 업데이트가 느려지고 컨트롤 또는 인디케이터가 깜박거릴 수 있습니다. 순서 재설정 풀다운 메뉴 를 사용하여 객체를 서로에 대해 상대적으로 이동합니다.

다음 단계를 따라 각 객체 위에 쌓인 객체의 순서를 재설정합니다.

1. 이동하려는 객체를 선택합니다.
2. 순서재설정 풀다운 메뉴를 선택합니다.
3. 다음 옵션 중 선택합니다.
 - 앞으로 이동—선택된 객체를 다층에서 한 위치 높은 곳으로 이동합니다.
 - 뒤로 이동—다층에서 선택된 객체를 다층에서 한 위치 뒤로 움직입니다.
 - 맨 앞으로 이동—다층에서 선택된 객체를 맨 위로 움직입니다.
 - 맨 뒤로 이동—다층에서 선택된 객체를 맨 뒤로 움직입니다.



노트

또한, 블록다이어그램에서 **자동으로 기존의 와이어를 모두 재정리하고 기존의 객체를 모두 재정렬**할 수 있습니다.

다음 키보드 바로 가기 키를 사용해서 다층 객체에서 객체의 순서를 재설정합니다.



노트 이 바로 가기 키에서 <Ctrl> 키는 (macOS) <Option> 또는 <Command> 키, (Linux) <Alt> 키에 상응합니다.

- 앞으로 이동-<Ctrl+K>
- 뒤로 이동-<Ctrl+J>
- 맨 앞으로 이동-<Ctrl+Shift+K>
- 맨 뒤로 이동-<Ctrl+Shift+J>

클립보드로부터 프런트패널 객체 대체하기

다음 단계를 따라 프런트패널 객체를 클립보드의 객체로 대체합니다.

1. 위치 도구를 사용하여 기존의 컨트롤 또는 인디케이터를 대체할 프런트패널 객체를 선택합니다.
2. 편집>>복사를 선택하여 클립보드에 객체를 복사합니다.
3. 위치 도구를 사용하여 대체하려는 객체를 선택합니다.
4. 편집>>붙여넣기를 선택합니다. LabVIEW는 이전 객체 대신 새 객체를 붙여넣습니다.



노트

컨트롤을 인디케이터로 대체하는 경우, LabVIEW는 원래 객체의 데이터흐름 방향을 보존하며 **컨트롤을 인디케이터로 변경**하지 않습니다. 사용자가 인디케이터를 컨트롤로 대체하는 경우, LabVIEW는 인디케이터를 컨트롤로 변경하지 않습니다.

이 방법은 이전 객체의 특성을 보존하지는 않으나, 와이어는 객체에 연결되어 있습니다. **객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 대체를 선택**하여 원래 객체에 대한 정보를 최대한 보존합니다.

프런트패널 객체 대체하기

프런트패널 객체를 라벨, 캡션, 객체에 설정한 값을 잃지 않고 다른 컨트롤 또는 인디케이터로 대체할 수 있습니다.

다음 단계를 따라 프런트패널 객체를 대체합니다.

1. 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 대체를 선택하여, 컨트롤 팔레트의 임시 버전을 디스플레이합니다.

3. 임시 컨트롤 팔레트에서 컨트롤 또는 인디케이터를 선택하여 현재 프런트패널 객체를 대체합니다.



노트

컨트롤을 인디케이터로 대체하는 경우, LabVIEW는 원래 객체의 데이터흐름 방향을 보존하며 **컨트롤을 인디케이터로 변경**하지 않습니다. 사용자가 인디케이터를 컨트롤로 대체하는 경우, LabVIEW는 인디케이터를 컨트롤로 변경하지 않습니다.

또한, **프런트패널 객체를 클립보드의 다른 객체로 대체**할 수 있습니다.

프런트패널 객체 스케일하기

구획의 크기를 조정할 때, 구획 크기에 따라서 스케일되거나 자동으로 크기 조정되도록 프런트패널 객체를 설정할 수 있습니다. 하나의 프런트패널 객체를 스케일하도록 설정하거나, 포함하는 구획의 모든 객체를 스케일하도록 설정할 수 있습니다.

또한, 특정 프런트패널 객체가 구획에 크기를 맞추도록 설정할 수 있습니다. LabVIEW는 구획을 채우도록 객체의 크기를 조정합니다. 구획의 크기 조정할 때, 객체는 구획과 함께 스케일 또는 크기가 조정됩니다.



관련 정보

[하나의 프런트패널 객체를 스케일에 맞게 설정하기](#) (1333 페이지 참조)

[프런트패널의 모든 객체를 스케일에 맞게 설정하기](#) (1333 페이지 참조)

[단일 프런트패널 객체를 구획에 맞게 피팅하기](#) (1334 페이지 참조)

프런트패널 객체의 탭 순서 설정하기

탭 순서는 VI가 실행되는 동안 사용자가 <Tab> 키를 누를 때 LabVIEW가 선택하는 컨트롤과 인디케이터의 순서를 결정합니다. 또한 탭 순서는 **프런트패널 데이터를 로그**할 때 생성한 데이터로그 파일의 레코드에 나타나는 컨트롤과 인디케이터의 순서를 결정합니다. LabVIEW는 객체를 프런트패널이나 탭 컨트롤에 추가하는 순서를 기본 탭 순서로 사용합니다.

다음 단계를 따라 탭 순서를 변경합니다.

1. 편집>> 탭 순서 설정을 선택하여 프런트패널의 탭 순서 편집 모드를 활성화합니다.



노트

탭 컨트롤 페이지에서 컨트롤과 인디케이터의 탭 순서를 설정하려면, 페이지에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 페이지의 컨트롤 순서 재설정을 선택합니다.

프런트패널 모양은 다음 그림에서와 같이 탭 순서 편집 모드로 바뀝니다. 컨트롤과 인디케이터의 흰 상자는 현재 탭 순서를 나타냅니다. 검은 상자는 새 탭 순서를

나타냅니다. 이 숫자는 변경하기 이전에 동일합니다. 탭 순서 편집 모드에서는, 모



든 숨겨진 컨트롤이 나타납니다.

2. 도구 모음의 클릭하여 설정합니다 필드에 숫자를 입력합니다. 프런트패널 또는 탭 컨트롤의 컨트롤과 인디케이터의 개수보다 큰 숫자를 입력할 수 없습니다.
3. 탭 순서 커서로 객체를 클릭하여 객체의 위치 번호를 도구 모음의 클릭하여 설정합니다에 디스플레이된 숫자로 설정합니다. LabVIEW는 원래 순서를 기반으로 다른 프런트패널 또는 탭 컨트롤 객체에 자동으로 번호를 다시 부여합니다.
4. 단계 2와 3을 모든 원소가 원하는 순서를 가질 때까지 반복합니다.



노트 프런트패널에서 탭을 사용할 때 LabVIEW는 숨겨진 컨트롤을 건너뛵니다. 숨겨진 컨트롤의 순서를 설정할 필요는 없습니다.

1. 검은 상자가 원하는 탭 순서를 디스플레이하면, 도구 모음의 확인 버튼을 클릭하여 새로운 탭 순서를 수락하고 탭 순서 편집 모드를 끝냅니다. 도구 모음의 취소 버튼을 클릭하여 이전 탭 순서로 되돌리고 탭 순서 편집 모드를 끝냅니다.
2. VI를 실행하고 <Tab> 키를 눌러 프런트패널 또는 탭 컨트롤의 컨트롤이나 인디케이터를 탐색합니다. 왼쪽 또는 오른쪽 화살표 키를 사용하여 탭 컨트롤의 페이지를 탐색합니다.



노트 분리자 막대를 프런트패널에 추가하고 팬을 생성할 경우, 프런트패널에서 객체의 탭 순서는 변하지 않습니다. 어떤 구획에 있는 컨트롤이든지 탭 순서에서의 다음 컨트롤이 될 수 있습니다.

스케일을 가지는 숫자형 객체의 값 변경하기

다음 단계를 따라 슬라이드, 스크롤 막대, 또는 회전식 객체의 값을 변경하는 것을 연습합니다.

1. 수행 도구를 사용하여 슬라이더, 스크롤 박스, 또는 지침을 클릭한 후 슬라이드, 스크롤 막대, 또는 회전식 객체의 내부로 끌어옵니다. 슬라이더, 스크롤 박스, 또는 지침을 끌면 값이 변경되며, LabVIEW는 컨트롤의 값을 팁 상자에 디스플레이합니다.
2. 슬라이더, 스크롤 박스, 또는 지침이 아닌 슬라이드, 스크롤 막대 또는 회전식 객체의 내부를 클릭합니다. 슬라이더 또는 지침은 클릭한 값으로 이동합니다. 마우스 버튼을 놓기 전에, LabVIEW는 팁 상자에 컨트롤 값을 디스플레이합니다. 스크

롤 박스는 한 페이지 크기를 보여줍니다. 스크롤 막대 프로퍼티 대화 상자의 데이터 입력 페이지에서 스크롤 막대 페이지 크기를 설정할 수 있습니다.



노트

노브 또는 다이얼의 안을 클릭해도 지침은 클릭한 값으로 고정되지 않습니다. 지침을 움직여서 값을 변경해야 합니다.

또한, 숫자형 **컨트롤과 인디케이터**에서 하는 것과 같이 **디지털 디스플레이**를 사용하여 슬라이드 또는 회전식 객체의 값을 스케일에 맞게 변경할 수 있습니다.

숫자형 객체에서 기수 디스플레이하기

숫자형 객체에서 기수를 디스플레이할 수 있고 기수를 사용하여 데이터를 10진수, 16진수, 8진수, 2진수 또는 SI 표기법으로 변경할 수 있습니다.

다음 단계를 따라 숫자형 객체에서 기수를 디스플레이하고 데이터의 포맷을 변경합니다.

1. 숫자형 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이콘 >> 기수를 선택합니다. 문자 d, x, o, b 또는 p가 숫자 디스플레이에 나타납니다.
2. 수행도구를 사용하여 문자를 클릭하고 바로 가기 메뉴에서 10진수, 16진수, 8진수, 2진수, 또는 SI 표기법을 선택합니다. 숫자형 객체의 형이 부동소수이면, 10진수 또는 SI 표기법만 사용할 수 있습니다.



노트

숫자 컨트롤의 포맷을 변경하면, LabVIEW는 새로운 포맷에 맞도록 기수를 숨기거나 변경합니다.

또한, 기수 보이기? 프로퍼티를 사용하여 기수를 프로그램적으로 디스플레이합니다.

컨트롤과 인디케이터

컨트롤과 인디케이터 설정하기

컨트롤 및 인디케이터의 프로퍼티를 설정하여 객체의 모양 또는 동작을 변경합니다. 프로퍼티 대화 상자 또는 **바로 가기 메뉴**를 사용하여 프로퍼티를 설정할 수 있습니다.

컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 프로퍼티 대화 상자를 디스플레이합니다. VI가 실행되는 동안에는 프로퍼티 대화 상자를 디스플레이할 수 없습니다.

바로 가기 메뉴에 접근하려면, 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭합니다.



팁

여러 개의 객체에서 한 개의 일반적인 작업을 동시에 수행하려면, 객체를 선택한 후 바로 가기 메뉴를 사용하거나 프로퍼티 대화 상자를 사용합니다.

다음 옵션은 모든 컨트롤과 인디케이터 바로 가기 메뉴에 공통입니다:

- 보이는 아이템—이름 라벨과 캡션과 같이 숨기거나 보여줄 수 있는 객체의 부분을 나열합니다.
- 터미널 찾기—컨트롤 또는 인디케이터의 **블록다이어그램 터미널을 하이라이트** 합니다. 이 메뉴 아이템은 복잡한 블록다이어그램에서 특정 객체를 식별할 때 유용합니다.
- 인디케이터로 변경과 컨트롤로 변경—**현재 객체를 컨트롤로 변경**하거나 또는 **인디케이터로 변경**합니다.
- 설명과 팁—객체 사용에 대한 **객체 설명과 팁을 입력하고 볼 수 있는** 대화 상자를 디스플레이합니다.
- 생성—**로컬 변수, 참조, 프로퍼티 노드** 또는 **인보크 노드**를 생성하여 객체의 다양한 프로퍼티를 프로그램적으로 컨트롤합니다.
- 대체—지금까지 객체에 적용한 모든 편집 옵션을 잃지 않고 다른 컨트롤 또는 인디케이터로 **프런트패널 객체를 대체**합니다.
- 데이터 처리—다음 데이터 편집 옵션의 서브메뉴를 포함합니다:
 - 기본값으로 다시 초기화—컨트롤을 기본값으로 설정합니다. 또한, 기본값으로 다시 초기화 메소드를 사용하여 프로그램적으로 컨트롤을 초기값으로 다시 초기화할 수 있습니다.
 - 현재값을 기본값으로—컨트롤의 현재값을 새로운 기본값으로 만듭니다. 또한, 기본값:현재값을 기본값으로 메소드를 사용하여 프로그램적으로 모든 프런트패널 컨트롤의 기본값을 설정할 수 있습니다.
 - 데이터 잘라내기, 데이터 복사, 데이터 붙여넣기—프런트패널 객체의 내용을 잘라내고 복사하고 붙여넣습니다. 잘라내거나 복사한 데이터를 LabVIEW가 클립보드에 저장하기 때문에, 데이터를 LabVIEW 외부에 붙여넣을 수 있습니다.
- 고급—다음 고급 편집 옵션의 서브메뉴를 포함합니다:
 - 동기화된 디스플레이—**모든 업데이트**를 디스플레이합니다. 애니메이션을 디스플레이할 때 이 특징을 사용합니다. 또한, 동기화된 디스플레이 프로퍼티를 사용하여 이 옵션을 프로그램적으로 설정할 수 있습니다.
 - 사용자 정의—**프런트패널 객체를 사용자 정의**하는데 사용하는 **컨트롤 편집기** 윈도우를 디스플레이합니다.
 - 컨트롤 숨기기 또는 인디케이터 숨기기—보기에서 **프런트패널 객체를 숨깁니다**. 숨겨진 객체에 접근하려면, 블록다이어그램 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 컨트롤 보이기 또는 인디케이터 보이기를 선택합니다.
 - 활성 상태—객체가 활성화, 비활성화, 비활성화 & 회색화인지를 결정합니다.

I/O 컨트롤과 인디케이터

I/O 및 클래식 I/O 팔레트의 I/O 컨트롤은 데이터를 수집하고, 어플리케이션 사이에 데이터를 공유하고, 인스트루먼트를 컨트롤하는데 사용하는 함수와 함께 사용됩니다. I/O 컨트롤을 사용하여 통신하려는 인스트루먼트나 디바이스 리소스를 지정할 수 있습니다.

- 웨이브폼 컨트롤을 사용하여 분석 및 디스플레이할 데이터를 얻습니다.
- DAQmx 이름 컨트롤 팔레트에 위치한 컨트롤을 사용하여 Measurement & Automation Explorer (MAX)와 DAQ 어시스턴트에서 설정한 태스크, 글로벌 채널, 물리적인 채널, 터미널, 스케일, 디바이스 그리고 스위치 리소스에 접근합니다.



노트

(Windows) LabVIEW는 NI-DAQmx와 DAQ 어시스턴트를 지원합니다. (Linux) LabVIEW는 NI-DAQmx는 지원하지만 DAQ 어시스턴트는 지원하지 않습니다. (macOS) LabVIEW는 NI-DAQmx Base는 지원하지만 DAQ 어시스턴트는 지원하지 않습니다.

- VISA 리소스 이름과 IVI 논리적 이름 컨트롤을 사용하여 MAX에서 설정한 VISA 가명이나 논리적 이름에 접근합니다.
- FieldPoint I/O 포인트 컨트롤을 사용하여 MAX에서 생성하고 설정한 FieldPoint 아 이템에 접근합니다.
- 이미지 수집 보드와 함께 IMAQ 세션을 사용합니다.
- 모션 리소스 I/O 컨트롤을 사용하여 MAX에서 설정한 모션 리소스에 접근합니다. MAX에서 설정한 모든 아이템이 I/O 컨트롤 풀다운 메뉴에서 옵션으로 나타납니다.
- 공유 변수 컨트롤을 사용하여 VI에서 사용하는 공유 변수를 선택하거나, 탐색하거나, 입력합니다.
- 팔레트의 다른 컨트롤을 사용하여 연관된 이름이나 리소스에 접근합니다.

I/O 컨트롤을 선택한 후에, 디바이스나 인스트루먼트와 통신하는 I/O VI와 함께 사용하기 위해 I/O 컨트롤을 **설정**합니다.



노트

모든 I/O 이름 컨트롤은 모든 플랫폼에서 사용 가능합니다. 임의의 플랫폼에서 작성한 I/O VI를 개발하여 특정 플랫폼에 국한된 디바이스와의 통신을 할 수 있습니다. 그러나, 해당 디바이스를 지원하지 않는 플랫폼에서 플랫폼에 특정한 I/O 컨트롤과 함께 VI를 실행하려는 경우, 에러가 발생합니다.

일부 컨트롤에서는 여러개의 정의된 이름을 선택할 수 있습니다. 수행 도구를 사용하여 컨트롤을 클릭하고, 풀다운 메뉴에서 탐색을 선택하고 <Ctrl> 키를 누릅니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

LabVIEW 애드온에 대한 자세한 정보는 **툴킷, 모듈 또는 드라이버** 문서를 참조하십시오.

배리언트 컨트롤과 인디케이터

배리언트 컨트롤과 인디케이터를 사용하여 데이터를 subVI로 전달하거나 전달받습니다. 또한, 이를 사용하여 데이터를 수정하지 않고 디스플레이할 수 있습니다. **배리언트 데이터 타입**을 사용하여 배리언트 데이터를 수정합니다.

볼리언 컨트롤

시스템, 클래식, 익스프레스, 일반 팔레트에 있는 볼리언 컨트롤을 사용하여 사용자 인터페이스를 생성합니다.

I/O 컨트롤 설정하기

Traditional NI-DAQ (Legacy) 버추얼 채널, NI-DAQmx 리소스 (태스크, 채널, 스케일 등), IVI 논리적 이름, VISA 리소스 이름, FieldPoint I/O 포인트, 또는 Motion 리소스 이름을 I/O VI에 전달하여 인스트루먼트 또는 측정 디바이스와 통신을 하기 위해서 I/O 컨트롤을 설정합니다.

1. I/O 팔레트의 I/O 컨트롤 중 하나를 프런트패널에 추가합니다.
2. I/O 컨트롤의 화살표를 클릭한 후 설정한 리소스를 선택합니다. I/O 컨트롤이 풀다운 메뉴에 사용 가능한 아이템을 가지지 않는 경우, 반드시 컨트롤에 연계된 **리소스를 설정**해야 합니다.
3. 블록 다이어그램을 디스플레이합니다.
4. I/O 터미널을 측정 디바이스, 인스트루먼트, 또는 다른 디바이스와 통신을 하는 I/O VI의 I/O 입력에 연결합니다.

또한, 공유 변수 컨트롤을 설정하여 VI에서 사용할 공유 변수를 선택할 수 있습니다.

1. I/O 팔레트에서 공유 변수 컨트롤을 프런트패널에 추가합니다.
2. 공유 변수 컨트롤의 오른쪽에 있는 아래 화살표를 클릭하여 공유 변수를 선택합니다. 현재 열린 프로젝트의 모든 배치된 공유 변수가 풀다운 메뉴에 나타납니다. 현재 열린 프로젝트에 없는 공유 변수를 사용하려는 경우, 아래 화살표를 클릭하고 탐색을 선택합니다. 다른 프로젝트 또는 다른 컴퓨터에서 배치된 공유 변수를 탐색합니다.
3. 블록 다이어그램을 디스플레이합니다.
4. 공유 변수 컨트롤을 [DataSocket 읽기]와 [DataSocket 쓰기] 함수의 연결 입력 입력에 연결합니다.

LabVIEW 애드온에 대한 자세한 정보는 툴킷, 모듈 또는 드라이버 문서를 참조하십시오.

모든 숨겨진 컨트롤과 인디케이터 디스플레이하기

편집>> 숨겨진 컨트롤과 인디케이터 보이기를 선택하여 **사용자 컨트롤** 및 글로벌 변수의 모든 숨겨진 프런트패널 컨트롤과 인디케이터를 디스플레이할 수 있습니다. 이 옵션은 사용자 컨트롤 또는 글로벌 변수 VI의 편집 메뉴에서만 사용 가능합니다. 다음 단계를 따라 사용자 컨트롤이나 글로벌 변수가 아닌 VI의 디스플레이 컨트롤과 인디케이터를 디스플레이합니다.



노트

잠금 혹은 실행 예약 상태의 VI에서 이 옵션을 사용할 수 없습니다.

1. labview\project_ShowHidden의 ShowHidden Core VI를 엽니다. 디스플레이하려는 컨트롤과 인디케이터를 포함하는 VI와 동일한 **어플리케이션 인스턴스**에서 ShowHidden Core VI를 열어야 합니다.
2. ShowHidden Core VI의 VI 이름 텍스트박스에 타겟 VI의 이름을 입력합니다.
3. 실행 버튼을 클릭합니다.

타겟 VI의 모든 숨겨진 프런트패널 컨트롤과 인디케이터가 나타나며 ShowHidden Core VI의 컨트롤 라벨 인디케이터가 보이는 것처럼 컨트롤과 인디케이터의 라벨을 디스플레이합니다.

컨트롤과 인디케이터의 스크롤 막대 디스플레이하기

컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 보이는 아이템>> 스크롤 막대, 보이는 아이템>> 수직 스크롤 막대 또는 보이는 아이템>> 수평 스크롤 막대를 바로 가기 메뉴에서 선택하여 수직 스크롤 막대나 수평 스크롤 막대를 디스플레이합니다. 위치 도구를 사용하여 객체의 크기를 조정하여 프런트패널에서 객체가 갖는 공간을 최소화할 수 있습니다.



노트

객체가 최소한 두 스크롤 막대의 높이와 폭을 가질 때에만 문자열이나 경로 컨트롤 또는 인디케이터에 스크롤 막대를 디스플레이할 수 있습니다. 수평 스크롤 막대를 디스플레이하기 위해서는 문자열 컨트롤 또는 인디케이터의 **자동 줄바꿈을 비활성화**해야만 합니다.

또한, 컨트롤 또는 인디케이터의 스크롤 막대 보이기, 보이는 아이템:수직 스크롤 막대 보이기, 또는 보이는 아이템:수평 스크롤 막대 보이기 프로퍼티를 사용하여 스크롤 막대를 프로그램적으로 디스플레이합니다.

한 차원 이상의 배열인 경우에만 수직과 수평 스크롤 막대 모두를 가질 수 있습니다. 1D 배열을 수직으로 크기 조정하여 수직 스크롤 막대를 추가하고, 수평으로 크기 조정하여 수평 스크롤 막대를 추가합니다.

프론트패널 컨트롤과 인디케이터 숨기기

다음 단계를 따라 프론트패널 컨트롤 또는 인디케이터를 숨깁니다. 또한, **숨겨진 프론트패널 객체를 디스플레이**할 수 있습니다.

1. 숨기려는 프론트패널 객체에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 객체가 컨트롤이면 고급>>컨트롤 숨기기를, 객체가 인디케이터이면 고급>>인디케이터 숨기기를 선택합니다.
3. (선택 사항) 편집 시 숨겨진 컨트롤의 동작을 지정합니다.

태스크

편집>> 편집 중 숨겨진 컨트롤 보이기를 선택합니다.

편집>> 편집 중 숨겨진 컨트롤 숨기기를 선택합니다.

또한, 보이기 프로퍼티를 사용하여 프로그램적으로 컨트롤과 인디케이터를 숨길 수 있습니다.

컨트롤과 인디케이터의 부분으로 컨트롤과 인디케이터 사용자 정의하기

컨트롤 또는 인디케이터는 다른 컨트롤과 인디케이터를 부분으로 포함할 수 있습니다. 이 토픽에서, 다른 컨트롤과 인디케이터의 부분으로 사용되는 컨트롤과 인디케이터는 구성요소 컨트롤 및 인디케이터로 불립니다. 예를 들면, 슬라이드 컨트롤은 디지털 숫자 컨트롤을 구성요소 컨트롤로 만들기 위해 디지털 디스플레이를 위해 디지털 숫자 컨트롤을 사용합니다. 프론트패널에서 부분 이동을 시도하여 일부 컨트롤 또는 인디케이터가 컨트롤 또는 인디케이터의 구성요소인지 찾아낼 수 있습니다. 상위 컨트롤 또는 인디케이터와 독립적인 구성요소 컨트롤과 인디케이터를 이동시킬 수 있습니다.

종종 구성요소 컨트롤 또는 인디케이터 또한 부분들로 만들어집니다. **컨트롤 편집기** 윈도우에서 원래의 컨트롤과 인디케이터를 편집할 때, 구성요소 컨트롤 또는 인디케이터는 단일 부분으로 동작합니다. 그것은, 구성요소 컨트롤 또는 인디케이터의 독립적인 부분을 변경하거나 이동시킬 수 없습니다. 그러나, 구성요소 컨트롤 또는 인디케이터만 포함한 두번째 컨트롤 편집기 윈도우를 디스플레이하여 구성요소 컨트롤과 인디케이터를 사용자 정의할 수 있습니다. 편집 모드에서 구성요소 컨트롤 또는 인디케이터를 선택할 수 있으면 두번째 컨트롤 편집기 윈도우를 디스플레이하기 위해 원래의 컨트롤 편집기 윈도우를 사용자 정의 모드로 놓을 필요가 없습니다.

다음 단계를 따라 구성요소 컨트롤 또는 인디케이터를 사용자 정의합니다.

1. 프론트패널 윈도우에서 사용자 정의하려는 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>사용자 정의를 선택하여 컨트롤 편집기 윈도우를 디스플레이합니다.
2. 컨트롤 편집기 윈도우의 구성요소 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 고급>>사용자 정의를 선택하여, 해당 구성요소 컨트롤 또는 인디케이터만을 포함한 컨트롤 편집기 윈도우를 디스플레이합니다.

3. 사용자 정의 모드로 변경 버튼을 클릭하여 새로운 컨트롤 편집기 윈도우를 사용자 정의 모드로 놓습니다.
4. 구성요소 컨트롤 또는 인디케이터의 독립적인 부분에 원하는 변화를 만듭니다. 컨트롤 또는 인디케이터의 크기, 색, 그리고 원소의 상대적인 위치 그리고 컨트롤 또는 인디케이터에 반입하는 이미지를 변경할 수 있습니다.
5. 파일»변경된 사항 적용을 선택하여 원래의 컨트롤 편집기 윈도우에서 구성요소 컨트롤과 인디케이터에 변경된 사항을 적용합니다. 변경된 사항 적용 메뉴 아이템은 원래의 구성요소 컨트롤 또는 인디케이터를 변경한 이후에만 이용 가능합니다. 구성요소 컨트롤이나 인디케이터를 변경하고 파일»변경된 사항 적용을 선택하지 않고 컨트롤 편집기 윈도우를 닫으려하면, LabVIEW는 원래의 구성요소 컨트롤이나 인디케이터를 사용자 구성요소 컨트롤이나 인디케이터와 대체하려는 지 물어보는 메시지를 디스플레이합니다.

컨트롤 편집기 윈도우는 계속 포개어 넣을 수 있습니다. 그러나, 대부분의 컨트롤 또는 인디케이터는 최상위 레벨의 일부로서 다른 컨트롤 또는 인디케이터를 사용합니다. 그래프의 경우 다른 컨트롤이 부분으로 사용되는 복잡한 컨트롤을 부분으로 사용하는데 이 경우는 예외입니다.

프런트패널 윈도우에서 구성요소 부분 보이기

구성요소 컨트롤 또는 인디케이터가 프런트패널 윈도우에서 보이면, 프런트패널 윈도우에서 직접 구성요소 컨트롤 또는 인디케이터만을 포함하는 컨트롤 편집기 윈도우를 디스플레이할 수 있습니다.

다음 단계를 따라 프런트패널 윈도우에서 직접 구성요소 컨트롤 또는 인디케이터를 사용자 정의합니다.

1. 사용자 정의하려는 구성요소 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급»사용자 정의를 선택하여 해당 구성요소 컨트롤 또는 인디케이터만을 포함한 컨트롤 편집기 윈도우를 디스플레이합니다. 또한, 위치 도구를 사용하여 사용자 정의하려는 구성요소 컨트롤 또는 인디케이터를 선택한 후 편집»컨트롤 사용자 정의를 선택하여 컨트롤 편집기 윈도우를 디스플레이합니다.
2. 원하는 대로 구성요소 컨트롤 또는 인디케이터를 변경합니다. 컨트롤 또는 인디케이터의 크기, 색, 그리고 원소의 상대적인 위치 그리고 컨트롤 또는 인디케이터에 반입하는 이미지를 변경할 수 있습니다.
3. 파일»변경된 사항 적용을 선택하여 구성요소 컨트롤이나 인디케이터에 변경된 사항을 적용합니다. 변경된 사항 적용 메뉴 아이템은 원래의 구성요소 컨트롤 또는 인디케이터를 변경한 이후에만 이용 가능합니다. 구성요소 컨트롤이나 인디케이터를 변경하고 파일»변경된 사항 적용을 선택하지 않고 컨트롤 편집기 윈도우를 닫으려하면, LabVIEW는 원래의 구성요소 컨트롤이나 인디케이터를 사용자 구성요소 컨트롤이나 인디케이터와 대체하려는 지 물어보는 메시지를 디스플레이합니다.

프론트패널과 블록다이어그램에 사용자 컨트롤과 인디케이터 추가하기

컨트롤 편집기 윈도우에서 끌거나 컨트롤 팔레트 또는 함수 팔레트에서 선택하여, **사용자 컨트롤과 인디케이터**를 프론트패널과 블록다이어그램에 추가할 수 있습니다. 사용자 컨트롤이나 인디케이터를 블록다이어그램에 추가하면 사용자 컨트롤이나 인디케이터와 동일한 데이터 타입을 가진 상수를 생성합니다. 또한, **컨트롤과 함수 팔레트에 사용자 컨트롤과 인디케이터를 추가**할 수 있습니다.

다음 단계를 따라 사용자 컨트롤이나 인디케이터를 선택하고 이를 컨트롤이나 함수 팔레트로부터 프론트패널이나 블록다이어그램에 추가합니다.

1. 컨트롤 팔레트에서 컨트롤 선택 아이콘을 클릭하거나 함수 팔레트에서 VI 선택 아이콘이나 텍스트를 선택하여, 파일 대화 상자를 디스플레이합니다.
2. 위치시킬 사용자 컨트롤이나 인디케이터를 탐색합니다.
3. 파일 대화 상자에서, 사용자 컨트롤이나 인디케이터를 더블 클릭하거나 확인 버튼을 클릭하여 컨트롤이나 인디케이터를 커서에 놓습니다.
4. 사용자 컨트롤이나 인디케이터를 프론트패널이나 블록다이어그램에 추가합니다.
5. 이 VI를 사용자 컨트롤이나 인디케이터와 함께 저장합니다.



노트

사용자 컨트롤 또는 인디케이터를 VI에 추가할 때, 저장한 사용자 컨트롤 또는 인디케이터와 사용자 컨트롤 또는 인디케이터의 인스턴스 사이에는 연결이 없습니다. 사용자 컨트롤 또는 인디케이터의 각 인스턴스는 개별적이고 독립적인 복사본입니다. 그렇기 때문에, 사용자 컨트롤 또는 인디케이터 파일을 변경해도 이미 사용자 컨트롤 또는 인디케이터를 사용한 VI에는 영향을 주지 않습니다. 사용자 컨트롤 또는 인디케이터의 인스턴스를 사용자 컨트롤 또는 인디케이터 파일에 연결하려는 경우, 사용자 컨트롤 또는 인디케이터를 **타입 정의 또는 엄격한 타입 정의**로 저장합니다. 타입 정의 또는 엄격한 타입 정의의 모든 인스턴스는 해당 인스턴스를 생성한 원래 파일에 링크되어 있습니다. 사용자 컨트롤 또는 인디케이터를 타입 정의 또는 엄격한 타입 정의로 저장할 때, 타입 정의 또는 엄격한 타입 정의에 데이터 타입을 변경하면 그것을 사용하는 모든 VI의 타입 정의 또는 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다. 또한, 엄격한 타입 정의로 변경하는 외관은 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다.

숫자형 컨트롤과 인디케이터의 데이터 범위 변경하기

숫자형 컨트롤 또는 인디케이터의 중간값을 특정 증가치로 제한할 수 있습니다. 예를 들어, 16 비트 정수의 증가치를 10으로 단정도 부동소수의 증가치를 0.25로 제한할 수 있습니다. 리미트 또는 증가치를 변경하는 경우, 사용자가 범위 밖의 값을 설정하거나 증가치를 벗어나도록 시도할 경우 어떻게 할지를 결정할 수도 있습니다. 또한 [범위내 확인과 강제 변환] 함수를 사용하여 값이 범위 내에 있는지 여부를 결정하고 값이 범위 내에 들어오도록 강제 변환할 수 있습니다.

다음 단계를 따라 **기본적인 최소와 최대 데이터 범위 리미트** 내에서 다른 리미트를 지정합니다.



노트

아래에 설명된 필드를 사용하여 값이 지정된 숫자 범위를 벗어난 경우 사용자가 입력한 숫자 값을 어떻게 처리할지를 설정하십시오. 이 필드는 변경하는 것은 인디케이터의 값을 포함하여 프로그램적으로 설정된 값에 영향을 미치지 않습니다.

1. 숫자형 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 입력을 선택하여 숫자형 프로퍼티 대화 상자의 데이터 입력 페이지를 디스플레이합니다.
2. 기본 리미트 사용 확인란에서 확인 표시를 제거합니다.
3. 최소값 필드에서 낮은 값 리미트를 설정하고 리미트 밖의 값에 응답 섹션의 풀다운 메뉴에서 무시 또는 강제 변환을 선택합니다.
4. 최대값 필드에서 높은 값 리미트를 설정하고 리미트 밖의 값에 응답 섹션의 풀다운 메뉴에서 무시 또는 강제 변환을 선택합니다.
5. 증가 필드에서 증가값을 설정하고 리미트 밖의 값에 응답 섹션의 풀다운 메뉴에서 무시, 최근접값으로 강제 변환, 위로 강제 변환, 또는 아래로 강제 변환을 선택합니다.

리미트 밖의 값에 응답

설명

LabVIEW는 유효하지 않은 값을 변경하거나 표시하지 않습니다. 증가와 감소 버튼을 클릭하면 설정한 증가분만큼 값이 바뀌지만, 이 값은 최소값 또는 최대값을 넘지 않습니다.

LabVIEW는 최근접값으로 증가합니다. 예를 들어, 최소값이 3, 최대값이 10, 증가가 2인 경우, 유효한 값은 3, 5, 7, 9, 10입니다. LabVIEW는 0을 3으로, 6를 7로, 10을 10으로 강제 변환합니다.

LabVIEW는 증가값 하나만큼 값을 증가시킵니다.

LabVIEW는 증가값 하나만큼 값을 감소시킵니다.

또한 숫자형 프로퍼티 또는 절대 시간 프로퍼티를 사용하여 프로그램식으로 데이터 범위와 리미트 밖의 값에 대한 응답을 설정할 수 있습니다.

슬라이드 컨트롤과 인디케이터의 채우기 옵션 변경하기

수직 슬라이드, 수평 슬라이드, 탱크, 온도계는 기본적으로 최소값에서부터 슬라이더 값을 채웁니다.

다음 단계를 따라 슬라이드 컨트롤 또는 인디케이터의 채우기 옵션을 변경합니다.

1. 슬라이더에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 채우기 옵션을 선택한 후 다음 옵션 중에서 선택합니다.

- 최소값까지 채우기—최소값에서 슬라이더 값까지 채웁니다. 이것은 기본입니다.
- 최대값까지 채우기—슬라이더 값에서 최대값까지 채웁니다.
- 아래의 값까지 채우기—**하나 이상의 슬라이더**를 갖는 슬라이드 객체에서 사용 가능합니다. 마우스 오른쪽 버튼으로 클릭한 슬라이더에서 아래의 슬라이더까지 채웁니다. 마우스 오른쪽 버튼으로 클릭한 슬라이더 아래에 슬라이더가 없는 경우, 이 옵션은 최소값을 채웁니다.
- 위의 값까지 채우기—**하나 이상의 슬라이더**를 갖는 슬라이드 객체에서 사용 가능합니다. 마우스 오른쪽 버튼으로 클릭한 슬라이더에서 위의 슬라이더까지 채웁니다. 마우스 오른쪽 버튼으로 클릭한 슬라이더 위에 슬라이더가 없는 경우, 이 옵션은 최대값을 채웁니다.
- 채우기 없음—채워진 것을 제거합니다.

또한 채우기 스타일 프로퍼티를 사용하여 프로그램적으로 채우기 옵션을 변경할 수 있습니다.

숫자형 컨트롤과 인디케이터의 포맷과 정밀도 변경하기

기본적으로, LabVIEW는 계산기와 같이 숫자를 디스플레이하고 저장합니다. 숫자형 컨트롤 또는 인디케이터는 6자리까지 숫자로 디스플레이한 후 자동으로 지수 표기로 바뀝니다. 숫자형 프로퍼티 대화 상자의 디스플레이 포맷 페이지에서 지수 표기로 바꾸기 이전에 LabVIEW가 디스플레이하는 컨트롤 또는 인디케이터, 디지털 디스플레이, 팁 상자의 자릿수를 설정할 수 있습니다.

다음 단계를 따라 숫자형 객체의 포맷과 정밀도를 변경합니다.

1. 숫자형 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 디스플레이 포맷을 선택하여, 숫자형 프로퍼티 대화 상자의 디스플레이 포맷 페이지를 디스플레이합니다.
2. 타입 리스트박스에서 포맷을 선택합니다. 숫자형 객체의 **형**이 부동소수인 경우, 숫자를 16진수, 8진수 또는 2진수로 디스플레이할 수 없습니다.
3. 자릿수와 정밀도 타입 옵션을 사용하여 필요한 경우 숫자의 정밀도를 변경합니다. 선택한 정밀도는 오직 값의 디스플레이에만 영향을 줍니다. 내부 정밀도는 여전히 형을 따릅니다.



노트

직접 **포맷 문자열을 입력**할 수 있도록 텍스트 옵션을 디스플레이하려면 고급 편집 모드를 선택합니다. 포맷 문자열을 입력하여 컨트롤의 숫자 정밀도를 사용자 정의합니다.

1. 디지털 디스플레이의 포맷과 정밀도를 변경하려면, 적용 풀다운 메뉴에서 디지털 디스플레이를 선택한 후 단계 2와 단계 3을 반복합니다. 팁 상자는 컨트롤의 디지털 디스플레이와 같은 포맷 및 정밀도를 사용합니다. 컨트롤에 하나 이상의 디지털 디스플레이가 있는 경우, 팁 상자는 현재 활성화된 지침 또는 슬라이더에 대응하는 디지털 디스플레이의 포맷과 정밀도를 사용합니다. 적용 풀다운 메뉴는 디지털 디스플레이를 포함하는 아이템에서만 나타납니다.
2. 확인 버튼을 클릭합니다.

또한 디지털 프로퍼티를 사용하여 프로그램적으로 포맷과 정밀도를 설정할 수 있습니다.

불리언 컨트롤과 인디케이터에 그래픽 반입하기

컨트롤 편집기 윈도우를 사용하여 참, 거짓, 컨트롤이나 인디케이터의 변이 상태를 나타내기 위해 불리언 컨트롤과 인디케이터로 그래픽을 반입할 수 있습니다.

편집 모드의 컨트롤 편집기 윈도우에서 그래픽을 불리언 컨트롤이나 인디케이터로 반입하기

다음 단계를 따라 편집 모드의 컨트롤 편집기 윈도우에서 그래픽을 불리언 컨트롤이나 인디케이터로 반입합니다.

1. 사용자 정의하려는 프런트패널 불리언 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 고급 >> 사용자 정의를 선택하여 컨트롤 편집기 윈도우를 디스플레이합니다. 컨트롤 편집기 윈도우는 편집 모드로 열립니다.
3. 편집 >> 클립보드에 그림 반입을 선택하여 파일 대화 상자를 디스플레이합니다.
4. 반입하려는 그래픽을 탐색합니다.
5. 그래픽을 선택하고 그래픽을 더블 클릭하거나 또는 클립보드의 그래픽을 놓기 위해 확인 버튼을 클릭합니다.
6. 불리언 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴로부터 클립보드로부터 그림 반입을 선택하고, 다음 아이템들 사이에서 선택하십시오:
 - 거짓—불리언 컨트롤이나 인디케이터의 거짓 상태에 반입된 그래픽을 적용합니다. 컨트롤이나 인디케이터가 거짓 상태에 있을 때만 반입된 그래픽이 나타납니다.
 - 참—불리언 컨트롤이나 인디케이터의 참 상태에 반입된 그래픽을 적용합니다. 컨트롤이나 인디케이터가 참 상태에 있을 때만 반입된 그래픽이 나타납니다.
 - 그림—불리언 컨트롤이나 인디케이터의 모든 상태에 반입된 그래픽을 적용합니다. 불리언 컨트롤이나 인디케이터의 상태에 상관 없이 그래픽을 볼 수 있도록 되어 있습니다.

LabVIEW는 그래픽을 정상적인 상태로 그리고 상응하는 전이 상태로 반입합니다.



팁

그래픽을 반입하고 VI를 저장한 후에 불리언 컨트롤이나 인디케이터에 대한 변경사항을 취소하려면 컨트롤이나 인디케이터를 팔레트의 새로운 버전으로 대체하면 됩니다. 프런트패널을 디스플레이하고, 불리언 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 대체를 선택합니다. 이후, 대체하기 원하는 불리언 컨트롤이나 인디케이터를 선택하십시오.

사용자 정의 모드의 컨트롤 편집기 윈도우에서 그래픽을 불리언 컨트롤이나 인디케이터로 반입하기

다음 단계를 따라 사용자 정의 모드의 컨트롤 편집기 윈도우에서 그래픽을 불리언 컨트롤이나 인디케이터로 반입합니다.

1. 사용자 정의하려는 프런트패널 불리언 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 고급 » 사용자 정의를 선택하여 컨트롤 편집기 윈도우를 디스플레이합니다.
3. 사용자 정의 모드로 변경 버튼을 클릭하여 컨트롤 편집기 윈도우를 사용자 정의 모드로 놓습니다.
4. 불리언 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴로부터 그림 아이템을 선택하고, 다음 아이템들 사이에서 선택하십시오:
 - 첫번째 선택은 그래픽을 거짓 상태에 적용합니다.
 - 두번째 선택은 그래픽을 참 상태에 적용합니다.
 - 세번째 선택은 그래픽을 **놓을 때 스위치** 상태로 그래픽을 적용합니다. 이것은 참에서 거짓으로 전이 상태입니다.
 - 네번째 선택은 그래픽을 **놓을 때 래치** 상태로 그래픽을 적용합니다. 이것은 거짓에서 참으로 전이 상태입니다.



노트

시스템 버튼이나 시스템 취소 버튼과 같은 일부 시스템 컨트롤은 추가로 선택 기능을 제공하며, 이는 추가로 가능한 상태를 나타냅니다.

1. 불리언 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 파일로부터 반입을 선택하여 파일 대화 상자를 디스플레이합니다. 또한, 바로 가기 메뉴에서 파일로부터 같은 크기로 반입을 선택하여 불리언 컨트롤이나 인디케이터의 선택된 부분과 같은 크기로 그래픽을 반입합니다.
2. 반입하려는 그래픽을 탐색합니다.
3. 그래픽을 선택하고 그래픽에서 더블 클릭하거나 확인 버튼을 클릭하여 불리언 컨트롤이나 인디케이터의 선택된 부분을 그래픽으로 대체합니다.

프런트패널에서 사용자 컨트롤과 인디케이터 생성하기

다음 단계를 따라 프런트패널에 사용자 컨트롤 또는 인디케이터를 사용자 정의합니다. 한 번에 오직 한 개의 컨트롤 또는 인디케이터만 편집할 수 있습니다. 사용자 정의 컨트롤과 인디케이터를 생성하기 전에 **유의점 및 권장사항**을 참조하십시오.

1. 위치 도구를 사용하여 사용자 정의하려고 하는 컨트롤 또는 인디케이터를 선택합니다.

2. 편집>>사용자 정의 컨트롤을 선택하여 컨트롤 편집기 윈도우를 디스플레이합니다.



노트

또한, 사용자 정의하려는 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급>>사용자 정의를 선택하여 **컨트롤 편집기** 윈도우를 디스플레이할 수 있습니다.

3. 컨트롤 또는 인디케이터에 원하는 변화를 만듭니다. 컨트롤 또는 인디케이터의 크기, 색, 그리고 원소의 상대적인 위치 그리고 컨트롤 또는 인디케이터에 반입하는 이미지를 변경할 수 있습니다.
4. 파일>>변경된 사항 적용을 선택하여 컨트롤이나 인디케이터에 변경된 사항을 적용합니다. 변경된 사항 적용 메뉴 아이템은 원래의 컨트롤 또는 인디케이터를 변경한 이후에만 가능합니다. 컨트롤 또는 인디케이터를 변경하고 파일>>변경된 사항 적용을 선택하지 않고 컨트롤 편집기 윈도우를 닫으려 하면, LabVIEW는 원래의 컨트롤 또는 인디케이터를 사용자 컨트롤 또는 인디케이터와 대체하려는지 물어보는 메시지를 디스플레이합니다.
5. (옵션) 다른 VI에서 사용할 수 있도록 파일>>저장을 선택하여 사용자 컨트롤 또는 인디케이터를 저장합니다. 사용자 컨트롤 또는 인디케이터를 디렉토리나 **LLB**에 저장할 수 있습니다. 다른 VI에서 사용하기 위해 사용자 컨트롤 또는 인디케이터를 저장하지 않으려면, 사용자 컨트롤 또는 인디케이터를 저장하지 않고 컨트롤 편집기 윈도우를 닫습니다.

또한, **새 대화 상자를 사용**하여 사용자 컨트롤 또는 인디케이터를 생성할 수 있습니다.



노트 원래의 컨트롤을 대체하기 위해 사용자 컨트롤을 저장할 필요는 없습니다. 사용자 정의한 컨트롤이 **타입 정의**이고 원래의 컨트롤을 대체하면, 호출 VI를 닫을 때 LabVIEW가 타입 정의를 저장할 것을 요청합니다.

저장된 사용자 컨트롤과 인디케이터 편집하기

다음 단계를 따라 저장된 사용자 컨트롤 또는 인디케이터를 편집합니다.

1. 파일>>열기를 선택하여 파일 대화 상자를 디스플레이합니다.
2. 편집하려는 사용자 컨트롤 또는 인디케이터를 탐색합니다.
3. 사용자 컨트롤 또는 인디케이터를 선택하고 사용자 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 더블 클릭하거나 열기 버튼을 클릭하여, **컨트롤 편집기** 윈도우에 사용자 컨트롤 또는 인디케이터를 디스플레이합니다.
4. 사용자 컨트롤 또는 인디케이터를 원하는 대로 변경합니다. 컨트롤 또는 인디케이터의 크기, 색, 그리고 원소의 상대적인 위치 그리고 컨트롤 또는 인디케이터에 반입하는 이미지를 변경할 수 있습니다.

5. 파일 »저장을 선택하여 사용자 컨트롤 또는 인디케이터를 저장합니다.



노트

사용자 컨트롤 또는 인디케이터를 VI에 놓을 때, 저장한 사용자 컨트롤 또는 인디케이터와 VI의 사용자 컨트롤 또는 인디케이터의 인스턴스는 서로 연결되어 있지 않습니다. 사용자 컨트롤 또는 인디케이터의 각 인스턴스는 개별적이고 독립적인 복사본입니다. 그렇기 때문에, 사용자 컨트롤 또는 인디케이터 파일을 변경해도 이미 사용자 컨트롤 또는 인디케이터를 사용한 VI에는 영향을 주지 않습니다. 사용자 컨트롤 또는 인디케이터의 인스턴스를 사용자 컨트롤 또는 인디케이터 파일에 연결하려는 경우, 사용자 컨트롤 또는 인디케이터를 **타입 정의 또는 엄격한 타입 정의**로 저장합니다. 타입 정의 또는 엄격한 타입 정의의 모든 인스턴스는 해당 인스턴스를 생성한 원래 파일에 링크되어 있습니다. 사용자 컨트롤 또는 인디케이터를 타입 정의 또는 엄격한 타입 정의로 저장할 때, 타입 정의 또는 엄격한 타입 정의에 데이터 타입을 변경하면 그것을 사용하는 모든 VI의 타입 정의 또는 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다. 또한, 엄격한 타입 정의로 변경하는 외관은 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다.

사용자 컨트롤과 인디케이터 만들 때의 유의점과 권장사항

다음 리스트는 사용자 컨트롤과 인디케이터 만들 때 알아야 할 유의점과 권장사항들을 설명합니다.

- 한 플랫폼에서 생성한 그래픽을 다른 플랫폼에서 로드하면 약간 다르게 보입니다. 예를 들면, 투명한 배경에 불규칙한 모양을 가진 그래픽이 **다른 플랫폼**에서는 하얀 배경을 가질 수 있습니다.
- **컨트롤 편집기** 윈도우에서 컨트롤이나 인디케이터를 사용자 정의하면 컨트롤 또는 인디케이터의 모양만 바꾸고, 그 동작은 바꿀 수 없습니다. 컨트롤이나 인디케이터가 데이터를 디스플레이하는 방법을 변경하는데 컨트롤 편집기 윈도우를 사용할 수 없습니다. 또한, 컨트롤 또는 인디케이터를 크기조정과 같은 편집을 할 때 동작을 변경하는데 컨트롤 편집기 윈도우를 사용할 수 없습니다. 예를 들면, 링 컨트롤을 크게 만들 때, 증가와 감소 화살표 또한 높게 커집니다. 링 컨트롤의 아래에 나란히 있는 증가와 감소 화살표를 이동하면, 링 컨트롤이 더 커질 때 그 버튼도 커지게 됩니다.
- 사용자 컨트롤 또는 인디케이터를 사용하여 불규칙한 편집 작동이 발생한 경우, **엄격한 타입 정의**를 사용하여 이 에러를 수정합니다.
- 컨트롤과 인디케이터는 컨트롤 또는 인디케이터의 스타일에 따라 다르게 사용자 정의될 수 있습니다. 예를 들어, 일반 스타일 숫자형 컨트롤의 색을 변경하면, 증가와 감소 버튼의 색도 변경됩니다. 실버 스타일 숫자형 컨트롤에서는, 증가와 감소 버튼의 색이 변경되지 않습니다.

컨트롤

컨트롤의 키보드 바로 가기 키 설정하기

사용자가 마우스 없이 프런트패널 윈도우를 탐색할 수 있도록 컨트롤에 키보드 바로 가기 키를 지정할 수 있습니다. <Shift>와 <Ctrl> 키와 같은 조합키를 일부 바로 가기 키로 할당할 수 있습니다. 하나 이상의 컨트롤에 같은 키 조합을 할당할 수 없습니다.

LabVIEW는 숨겨진 컨트롤의 키보드 바로 가기 키에는 반응하지 않습니다.

다음 단계에 따라 컨트롤에 대한 키 탐색을 설정하고 VI가 실행되는 동안 사용자가 컨트롤에서 컨트롤로 탭 이동을 할 때 이것이 포함되지 못하도록 컨트롤을 설정합니다.

1. 프런트패널 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 고급>>키 조작을 선택하여 **프로퍼티** 대화 상자의 키 조작 페이지를 디스플레이합니다. 또한, 컨트롤에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 프로퍼티를 선택한 후, 키 조작 탭을 클릭합니다. 배열 컨트롤에서 키 조작을 설정하기 위해 배열 경계에서 마우스 오른쪽 버튼을 클릭합니다. 배열 원소에서 키 조작을 설정하기 위해 원소에서 마우스 오른쪽 버튼을 클릭합니다.



노트

인디케이터에는 데이터를 입력할 수 없기 때문에 인디케이터의 고급>>키 조작 바로 가기 메뉴 아이템은 비활성화됩니다.

1. 포커스 섹션의 풀다운 메뉴에서, 컨트롤에 할당하려는 바로 가기 키를 선택합니다. 포커스 섹션에서 <PageUp>, <PageDown>, <Home>, 또는 <End>를 바로 가기 키로 할당한 경우, 리스트박스나 트리 컨트롤에서 아이템 선택을 위해 이 키를 사용할 수 없습니다.
2. <Shift> 키 및/또는 <Ctrl> 키를 키보드 바로 가기 키에 포함시키려면, <Shift> 키 및/또는 <Ctrl> 키 확인란에 확인 표시를 합니다. 포커스 풀다운 메뉴에서 <F1>, <F2> 등과 같은 함수 키를 선택한다면 이 확인란을 이용할 수 있습니다.



노트 (macOS) 기능 키를 키보드 바로 가기 키로 사용하는 것은 기능 키가 Exposé에서 바로 가기 키로 할당되지 않은 경우에만 가능합니다. LabVIEW는 macOS에서 <VolumeUp>, <VolumeDown> 및 <F15>부터 <F24>까지의 기능 키를 바로 가기 키로 지원하지 않습니다.

1. VI가 실행되고 있는 동안 <Tab> 키를 사용하여 사용자가 컨트롤에 접근하는 것을 막기 위해 탭으로 이동할 때 이 컨트롤 건너뛰기 확인란에 확인 표시를 놓습니다.



노트 키 다운 이벤트를 사용하여 사용자가 키보드에서 특정 키를 누를 때 이벤트를 생성하도록 할 수 있습니다.

키보드 바로 가기 키를 할당하여 키 조작 페이지의 토글 섹션을 사용해서 불리언 컨트롤을 토글하거나, 키 조작 페이지의 증가 및 감소 섹션을 사용하여 숫자, 슬라이드, 노브,

링, 열거형 컨트롤을 증가 및 감소시킬 수 있습니다. 토글 섹션은 불리언 컨트롤에서만 사용할 수 있고 증가 및 감소 섹션은 숫자, 슬라이드, 노브, 링, 열거형 컨트롤에서만 사용할 수 있습니다.

또한, 컨트롤 프로퍼티를 사용하여 키보드 바로 가기 키를 컨트롤에 할당하고 탭의 작동을 프로그램적으로 설정할 수 있습니다.

컨테이너 컨트롤

컨테이너 및 클래식 컨테이너 팔레트에 위치한 컨테이너 컨트롤을 사용하여 컨트롤과 인디케이터를 그룹화하거나, 현재 VI의 프런트패널에 다른 VI의 프런트패널을 디스플레이합니다. (Windows) 또한 컨테이너 컨트롤을 사용하여 프런트패널 윈도우에 .NET과 ActiveX 객체를 디스플레이할 수 있습니다.



관련 정보

[탭 컨트롤 생성하기](#) (1018 페이지 참조)

[서브패널 컨트롤에 프런트패널 로드하기](#) (1015 페이지 참조)

[.NET 컨테이너에서 .NET 컨트롤 생성하기](#) (1431 페이지 참조)

[ActiveX 컨테이너 개요](#) (1440 페이지 참조)

서브패널 컨트롤 설정하기

서브패널 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 항목을 선택하여 서브패널 컨트롤을 설정합니다:

- 구획에 따라 객체 스케일 조정—구획의 크기가 조정될 때 객체가 **구획**의 크기에 맞도록 스케일합니다. 이 옵션은 다른 객체가 현재 구획과 함께 스케일되도록 설정되었거나 구획이 모든 객체를 구획과 함께 스케일되도록 설정된 경우에는 사용할 수 없습니다. 또한, 스케일링 모드 설정 메소드를 사용하여 이 옵션을 프로그램적으로 설정할 수 있습니다.
- 패널을 투명하게 만들기 — 서브패널 컨트롤에서 프런트패널의 배경을 투명하게 그립니다. 또한, 패널을 투명하게 프로퍼티를 사용하여 프로그램적으로 이 옵션을 설정할 수 있습니다.
- 다이어그램 열기 허용—다음 영역에서 사용자가 마우스 오른쪽 버튼을 클릭할 때 바로 가기 메뉴에 블록다이어그램 열기 항목을 디스플레이합니다.
 - 실행 중인 VI에서 서브패널 컨트롤의 경계
 - 실행 중인 VI에서 서브패널 컨트롤에 로드된 프런트패널 컨트롤 또는 인디케이터가 아닌 서브패널 컨트롤의 경계 내부

또한, 다이어그램 열기 허용 프로퍼티를 사용하여 프로그램적으로 이 옵션을 설정할 수 있습니다.



노트

서브패널의 SubVI 구획을 사용하여 서브패널 컨트롤 스크롤 막대의 보이기를 설정하고 서브패널 컨트롤에서 프론트패널을 스케일합니다.

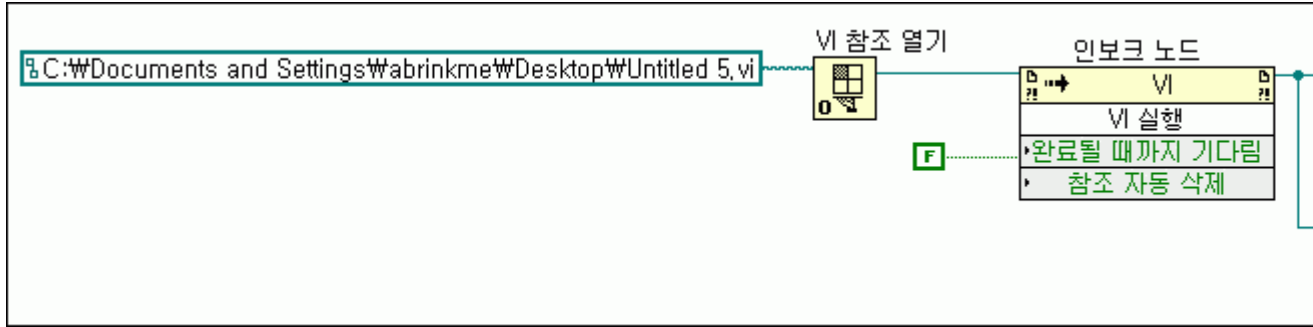
서브패널 컨트롤에 프론트패널 로드하기

서브패널 컨트롤을 사용하여 다른 VI의 프론트패널을 현재 VI의 서브패널에 디스플레이합니다. 예를 들어, 서브패널 컨트롤을 사용하여 사용자 인터페이스가 마법사처럼 작동하도록 디자인할 수 있습니다. 이전 또는 다음 버튼을 최상위 레벨 VI의 프론트패널에 추가하고 서브패널 컨트롤을 사용하여 마법사의 각 단계마다 다른 프론트패널을 로드할 수 있습니다.

다음 단계를 따라 서브패널 컨트롤에 프론트패널을 로드합니다.

1. 서브패널 컨트롤을 프론트패널에 추가합니다. LabVIEW는 VI 삽입 메소드가 선택된 [인보크 노드]를 블록다이어그램에 생성합니다.
2. 블록다이어그램의 [인보크 노드] 왼쪽에 [VI 참조 열기] 함수를 추가합니다.
3. 로드하려는 프론트패널을 가진 VI의 경로를 [VI 참조 열기] 함수의 VI 경로 입력에 연결합니다.
4. [VI 참조 열기] 함수의 vi 참조 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>메소드>>VI 실행을 선택하고 커서에 나타난 [인보크 노드]를 [VI 참조 열기] 함수의 오른쪽에 추가합니다.
5. [VI 참조 열기] 함수의 vi 참조 출력을 VI 실행 메소드의 vi 참조 입력에 연결합니다.
6. VI 실행 메소드의 완료까지 기다림 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택하여 거짓값을 갖는 불리언 상수를 생성합니다.
7. VI 실행 메소드의 vi 참조 출력을 VI 삽입 메소드의 VI 참조 입력에 연결합니다.
8. While 루프를 VI 삽입 메소드의 오른쪽에 추가합니다.
9. While 루프의 조건 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 컨트롤 생성을 선택하여 정지 버튼을 생성합니다.
10. [다음 ms 배수까지 기다림] 함수를 While 루프 안에 추가하고 숫자형 상수를 ms 배수 입력에 연결합니다. VI를 실행할 때 몇 밀리초를 경과시킬지 숫자로 지정합니다.
11. [참조 닫기] 함수를 While 루프의 오른쪽에 추가합니다.
12. VI 실행 메소드의 vi 참조 출력을 While 루프를 통과하여 [참조 닫기] 함수의 참조 입력에 연결합니다. 다음 블록다이어그램은 서브패널 컨트롤에서 프론트패널을

로드할 때의 예를 보여줍니다.



13. 블록다이어그램의 **모든 에러 입력과 에러 출력 터미널을 연결**합니다.

또한, 서브패널 프로퍼티와 메소드를 사용하여 서브패널 컨트롤을 프로그램적으로 수정할 수 있습니다.

서브패널 컨트롤 사용에 대한 예제는 labview\examples\Controls and Indicators\Containers\Containers.lvproj를 참조하십시오.

 예제 열기  관련 예제 찾기

탭 컨트롤

탭 컨트롤에 페이지 추가하기

탭 컨트롤에 다른 페이지를 추가하려면 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이전에 페이지 추가 또는 다음에 페이지 추가를 선택합니다.

탭 컨트롤 옵션 설정하기

탭 컨트롤의 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이템을 선택하여 탭 컨트롤을 설정합니다.

다음 단계를 따라 탭 컨트롤을 설정합니다.

1. 탭 컨트롤의 탭에서 마우스 오른쪽 버튼을 클릭합니다.
2. 다음 바로 가기 메뉴 아이템을 선택합니다:



노트

이 리스트는 탭 컨트롤의 탭에서 마우스 오른쪽 버튼을 클릭할 때 보이는 메뉴 아이템을 모두 포함하고 있지는 않습니다.

- 페이지의 컨트롤 순서 재설정—키 조작을 위해 활성 페이지에서 **객체의 순서를 재설정**합니다. 페이지에 있는 컨트롤이 두 개 이상이 아닌 경우, LabVIEW는 이 아이템을 비활성화합니다.
- 고급—다음 아이템 중에서 선택합니다:
 - 페이지 숨기기—현재 페이지를 숨깁니다. 기본 페이지를 숨기는 경우, 다음 페이지가 기본 페이지가 됩니다. 보이는 페이지가 하나인

경우, 이것이 기본 페이지이며 이를 숨길 수는 없습니다. 또한 페이지 보이기 프로퍼티를 사용하여 프로그램에서 탭 컨트롤의 페이지를 숨길 수 있습니다.

- 페이지 활성화 상태-탭 컨트롤 페이지의 활성화된 상태를 수정합니다. 활성화, 비활성화, 비활성화 & 회색화 중에서 선택합니다. 또한 페이지 활성화 상태 프로퍼티를 사용하여 프로그램에서 탭 컨트롤 페이지의 활성화 상태를 수정할 수 있습니다.
- 숨겨진 페이지 보이기-숨겨진 특정 페이지를 이름으로 보여줍니다. 또한 페이지 보이기 프로퍼티를 사용하여 프로그램적으로 탭 컨트롤의 숨겨진 페이지를 디스플레이할 수 있습니다.
- 페이지 캡션을 라벨과 일치시킴-이 옵션 옆에 있는 확인 표시를 제거하여 페이지에 연결된 페이지 라벨을 변경하지 않고 탭 텍스트를 변경합니다. 페이지 라벨을 디스플레이하려면, 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 페이지 라벨 디스플레이를 선택합니다. 페이지 라벨은 블록 다이어그램의 열거형 타입 컨트롤의 값에 대응합니다.
- 자동 크기 조정-페이지상에 놓인 객체가 탭 컨트롤의 경계를 벗어날 경우 탭 컨트롤의 크기를 증가시킵니다.
- 크기에 맞게 조정-탭 컨트롤이 내용에 맞도록 크기가 조정됩니다. 이 바로 가기 아이템을 활성화하면, LabVIEW는 자동으로 자동 크기 조정 아이템을 활성화합니다.
- 여러 색 허용-탭 컨트롤의 각 페이지에 서로 다른 색을 허용합니다.
- 여러 행 허용-탭 컨트롤의 폭을 초과해서 확장되는 모든 탭을 탭의 추가적인 행으로 배치합니다. 이 옵션 옆의 확인 표시를 제거하여 탭 컨트롤의 오른쪽 상단에 탐색 버튼을 디스플레이합니다. 이 탐색 버튼을 사용하여 보이지 않는 탭까지 스크롤합니다.
- 탭 크기-탭의 크기를 조정합니다. 탭을 내용에 맞춤, 모든 탭을 가장 큰 탭으로 크기변경, 고정 탭 크기 중에서 선택합니다. 고정 탭 크기를 선택한 경우, 위치 도구를 사용하여 개별 탭의 크기를 조정합니다. 탭 컨트롤의 전체 폭을 채우기 위해 각 탭의 행을 늘리도록 탭 자리맞춤에 확인 표시를 합니다. 탭 컨트롤이 여러 행의 탭을 가지는 경우, LabVIEW는 자동적으로 탭 크기를 맞춤니다.
- 탭 레이아웃-탭 컨트롤의 탭에 텍스트와 이미지 또는 텍스트와 이미지 중 하나를 사용할 지를 결정합니다. 또한 이 바로 가기 메뉴를 사용하여 탭에 이미지를 추가하거나 제거할 수 있습니다.
- 탭 위치-탭 컨트롤의 탭 위치를 변경합니다.

또한 탭 컨트롤과 페이지 프로퍼티를 사용하여 탭 컨트롤과 개별 탭 컨트롤 페이지를 프로그램식으로 설정할 수 있습니다. **개별 페이지를 설정**하기 이전에 반드시 탭 컨트롤 페이지의 참조를 얻어야 합니다.

프로그램적으로 탭 컨트롤 페이지 설정하기

다음 단계를 따라 프로그램적으로 탭 컨트롤 페이지를 설정합니다.

1. 탭 컨트롤의 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 프로퍼티 노드 >> 페이지를 선택합니다. LabVIEW는 블록 다이어그램에서 선택한 프로퍼티에 대해서 함축적으로 연결된 프로퍼티 노드를 생성합니다.

2. 블록다이어그램에 [배열 인덱스] 함수를 추가합니다.
3. 프로퍼티 노드의 페이지 출력을 [배열 인덱스] 함수의 n차원 배열 입력에 연결합니다.
4. [배열 인덱스] 함수의 인덱스 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 상수를 선택하여 숫자형 상수를 생성합니다.
5. 설정하려는 페이지의 인덱스를 입력한 후 <Enter> 키를 누릅니다. 인덱스는 탭 컨트롤에 나타날 때 0부터 n-1의 범위에서 페이지의 순서에 대응합니다. 이때 n은 탭 컨트롤의 페이지 개수입니다. 예를 들어, 탭 컨트롤의 첫번째 페이지를 설정하려면 0을 입력합니다.
6. 필요한 경우 위치 도구를 사용하여 [배열 인덱스] 함수를 **크기 조정**합니다.
7. 와이어링 도구를 사용하여 [배열 인덱스] 함수의 원소 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 페이지 클래스의 프로퍼티와 프로퍼티를 선택합니다. LabVIEW는 선택한 프로퍼티의 프로퍼티 노드를 생성합니다.
8. 블록다이어그램에 프로퍼티 노드를 추가하고, [배열 인덱스] 함수의 원소 출력을 단계 8에서 생성한 프로퍼티 노드의 참조 입력에 연결합니다.
9. 필요한 경우 프로퍼티 노드를 설정합니다.

탭 컨트롤 페이지에 대한 참조를 얻은 후, 페이지의 컨트롤 프로퍼티를 사용하여 프로그램적으로 탭 컨트롤 페이지 내의 컨트롤을 설정할 수도 있습니다.

프로그램적으로 탭 컨트롤 페이지 설정하기의 예제는 labview\examples\Controls and Indicators\Containers에 있는 Programmatically Manipulate a Tab Control VI를 참조하십시오.

 예제 열기  관련 예제 찾기

탭 컨트롤 생성하기

탭 컨트롤을 이용하여 좁은 영역에서 프런트패널의 컨트롤과 인디케이터를 중첩하여 사용합니다. 탭 컨트롤은 페이지와 탭으로 구성됩니다. 탭 컨트롤의 페이지 위에 프런트패널 객체를 추가하고, 구분자로 탭을 사용하여 각 페이지를 디스플레이합니다.

다음 단계를 따라 탭 컨트롤을 생성합니다.

1. 탭 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 페이지 사이를 전환하기 위해서, 수행 도구로 원하는 페이지의 탭을 클릭합니다. 또한, 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 페이지 맞바꾸기 또는 다른 페이지로 가기를 선택하여, 페이지 사이에서 전환할 수 있습니다.
3. 임의의 프런트패널 객체를 활성 페이지에 추가합니다. 페이지에 대한 탭이 페이지와 함께 선택되고 페이지의 객체가 보이면 페이지는 활성화됩니다. 사용자가 탭 컨트롤에 추가한 컨트롤과 인디케이터의 터미널은 일반적인 블록다이어그램 터미널처럼 보입니다.



팁

탭 컨트롤 페이지에 객체를 추가하는 경우, 자주 블록다이어그램 윈도우에 가서 새로 추가된 터미널을 정돈하십시오. 탭 컨트롤에 여러 객체를 추가하면, 블록다이어그램의 터미널은 혼란스러워지기 때문입니다.

블록다이어그램에서, 탭 컨트롤은 열거형의 컨트롤 또는 인디케이터입니다. 컨트롤로, 다른 블록다이어그램 노드에 활성 페이지의 값을 전달할 수 있습니다. 인디케이터로, 디스플레이된 페이지의 컨트롤에 노드를 연결할 수 있습니다. 탭 컨트롤을 동작시키기 위해 탭 컨트롤 터미널을 연결할 필요는 없습니다.



팁 케이스 구조의 선택자에 탭 컨트롤의 열거형 컨트롤 터미널을 연결하여 블록다이어그램을 더 깔끔하게 만들 수 있습니다. 이 방법으로, 탭 컨트롤의 각 페이지를 케이스 구조에서 서브다이어그램으로 만들 수 있습니다. 탭 컨트롤의 각 페이지에 있는 컨트롤과 인디케이터 터미널 (관련 터미널과 연계된 블록다이어그램과 와이어도 포함하여)을 케이스 구조의 서브다이어그램에 추가합니다. vi의 구성요소가 지속적으로 실행되어야 하는 경우 (vi의 컨트롤이 숨겨진 상태이더라도), 케이스 구조를 사용하지 마십시오. 케이스 구조는 한 번에 1개의 케이스밖에 실행하지 못하기 때문입니다.

탭 컨트롤에 이미지 디스플레이하기

다음 단계에 따라 탭 컨트롤이 이미지를 디스플레이하도록 설정합니다.

1. 편집>>클립보드에 그림 반입을 선택하여 LabVIEW에 이미지를 반입합니다.
2. 탭에서 마우스 오른쪽 버튼을 클릭하고 고급>> 탭 레이아웃>> 이미지만을 선택하여, 이미지를 디스플레이하도록 탭 레이아웃을 설정합니다.



노트

고급>> 탭 레이아웃>> 이미지 - 텍스트를 선택하여 이미지와 텍스트를 디스플레이하도록 탭 레이아웃을 설정할 수 있습니다.

3. 탭에서 마우스 오른쪽 버튼을 클릭하고 고급>> 탭 레이아웃>> 클립보드에서 이미지 반입을 선택하여, 클립보드에서 탭으로 이미지를 반입합니다.

탭에서 마우스 오른쪽 버튼을 클릭하고 고급>> 탭 레이아웃>> 탭으로부터 이미지 제거를 선택하여, 탭으로부터 이미지를 제거할 수 있습니다.

탭 컨트롤의 페이지 복제하기

페이지의 복제를 생성하려면, 복제하려는 페이지의 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 페이지 복제를 선택합니다.

컨트롤을 모든 탭 컨트롤 페이지에서 보이게 만들기

탭 컨트롤의 모든 페이지에서 볼 수 있는 컨트롤을 **탭 컨트롤**의 여러 페이지에 추가할 수 있습니다.

다음 단계를 따라 컨트롤을 탭 컨트롤의 여러 페이지에 추가합니다.

1. 프런트패널 윈도우에 **여러 페이지 탭 컨트롤**을 만듭니다.
2. 프런트패널의 빈 부분에 컨트롤을 만듭니다.

3. 생성한 컨트롤을 선택합니다. 화살표 키를 사용하여 선택한 컨트롤을 탭 컨트롤로 이동합니다.



노트

컨트롤은 VI가 편집 모드에 있는 동안 그림자 효과(Drop shadow)를 나타내어 실제로는 탭 컨트롤이 VI를 포함하지 않음을 나타냅니다. VI가 실행 중일 때에는 그림자 효과가 나타나지 않습니다.

탭 컨트롤에서 페이지 삭제하기

페이지와 그 내용을 삭제하려면, 삭제하려는 페이지의 탭에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 페이지 제거를 선택합니다. 페이지를 제거하면 페이지의 모든 프론트패널 객체도 삭제됩니다.

탭 컨트롤의 페이지 순서 재설정하기

페이지의 순서를 재설정하려면, 탭에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 페이지 재배치(또는 두 페이지뿐인 경우 페이지 맞바꾸기)를 선택하여 페이지 재배치 대화 상자에 접근합니다.

참조 번호 컨트롤

참조 번호 컨트롤

참조 번호와 클래식 참조 번호 팔레트에 위치한 참조 번호 컨트롤을 사용하여 파일, 디렉토리, 디바이스, 네트워크 연결과 관련된 작업을 수행합니다. 참조 번호를 VI로 또는 VI 밖으로 전달할 때 **다른 타입**의 참조 번호 컨트롤을 사용합니다.

참조 번호는 열어 놓은 객체의 임시 포인터이므로, 객체가 열려 있는 동안만 유효합니다. 객체를 닫으면, LabVIEW는 참조 번호와 객체의 연결을 끊어서 해당 참조 번호를 사용할 수 없게 됩니다. 객체를 다시 열면, LabVIEW는 처음 참조 번호와는 다른 새 참조 번호를 생성합니다. LabVIEW는 참조 번호와 관련된 객체에 메모리를 할당합니다. 참조 번호를 닫아 객체를 메모리에서 해제합니다.

LabVIEW는 객체에서 읽어 오거나 쓴 현재 위치와 사용자 접근 등급과 같은 각 참조 번호에 관련된 정보를 기억하고 있기 때문에, 한 객체에 대하여 서로 독립적인 작업을 동시에 수행할 수 있습니다. VI가 객체를 여러번 여는 경우, 열 때마다 매번 다른 참조 번호를 반환합니다. LabVIEW는 VI의 실행이 끝나면 자동으로 참조 번호를 닫습니다. 그러나 메모리와 기타 리소스를 가장 효율적으로 사용하려면 사용이 끝난 직후 참조 번호를 닫는 것이 좋은 프로그래밍 방법입니다. 참조 번호는 연 순서와 반대로 닫아야 합니다. 예를 들어 객체 A의 참조 번호를 얻고 객체 A에서 메소드를 실행시켜 객체 B의 참조 번호를 얻은 경우, 객체 B의 참조 번호를 먼저 닫고 객체 A의 참조 번호를 닫습니다.

엄격한 타입의 참조 번호 생성하기

어플리케이션, VI, 컨트롤 또는 인디케이터, 또는 엄격한 타입의 VI를 포함하여 모든 **참조 번호 컨트롤**의 데이터 타입을 지정할 수 있습니다. 엄격한 타입의 VI 참조 번호는 VI의 커넥터 팬을 포함하는 데이터 타입 정보를 가집니다. 엄격한 타입의 VI 참조 번호를 사용하여 다이내믹하게 로드된 VI를 참조에 의한 호출 노드 또는 비동기화 호출 시작 노드로 호출할 수 있습니다.

다음 단계를 따라 엄격한 타입의 VI 참조 번호 컨트롤의 타입을 지정합니다.

1. 참조 번호 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 VI 서버 클래스 선택 >> 탐색을 선택하여 원하는 커넥터 팬을 가진 VI를 디스크에서 선택합니다.

또한, VI 아이콘, 커넥터 팬, 또는 subVI 아이콘을 블록 다이어그램으로부터 끌거나 VI 계층 구조 윈도우를 VI 참조 번호 컨트롤로 끌 수 있습니다.

엄격한 타입의 컨트롤 참조 번호와 엄격하지 않은 타입의 컨트롤 참조 번호 사이에 전환하기

엄격한 타입의 컨트롤 참조 번호는 정확히 동일한 데이터 타입의 컨트롤 참조 번호를 수용합니다. **엄격하지 않은 타입의 컨트롤 참조 번호**는 더 유연하게 데이터 타입을 받아들입니다.

엄격한 타입의 컨트롤 참조 번호와 엄격하지 않은 타입의 컨트롤 참조 번호 사이에 전환하는 데는 5가지 방법이 있습니다.

- 엄격한 타입의 컨트롤 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 타입 포함 아이템으로부터 확인 표시를 제거하여 엄격하지 않은 타입의 컨트롤 참조 번호로 만듭니다.
- 엄격하지 않은 타입의 컨트롤 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 타입 포함 아이템을 선택하여 엄격한 타입의 참조 번호로 만듭니다. 이 방법은 데이터의 클래스가 너무 일반적인, 즉 G객체 또는 일반일 때 작동하지 않습니다.
- 엄격한 타입의 컨트롤 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 Ctrl 타입 선택을 선택하여 클래스를 변경하고 데이터 타입 포함 옵션을 비활성화합니다. 이것은 컨트롤 참조 번호를 엄격하지 않은 타입으로 만듭니다.
- 엄격한 타입의 컨트롤 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 VI 서버 클래스 선택을 선택한 후 일반, G객체 >> G객체, G객체 >> 컨트롤 >> 컨트롤과 같은 일반 컨트롤 클래스를 선택합니다. 일반 컨트롤 클래스는 엄격한 타입이 될 수 없습니다.
- 컨트롤을 프런트패널 컨트롤 참조 번호 컨트롤로 끌어 원래 컨트롤을 제거하고 엄격한 타입의 컨트롤 참조 번호를 생성합니다. 참조 번호 컨트롤의 왼쪽 하단의 코너에 있는 빨간 별표는 컨트롤 참조가 엄격한 타입임을 나타냅니다. <Ctrl> 키를 누르면서 컨트롤을 컨트롤 참조 번호 컨트롤로 끌어 원래 컨트롤을 유지하고 엄격한 타입의 컨트롤 참조 번호를 생성합니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

참조 번호 컨트롤의 타입

LabVIEW는 참조 번호 팔레트에 위치한 다음의 **참조 번호 컨트롤**을 포함합니다.



.NET 참조 번호—이 참조 번호를 사용하여 .NET 컨스트럭터 선택 대화 상자를 시작하고 Global Assembly Cache (GAC)에서 어셈블리를 선택하거나 프라이빗 어셈블리를 탐색합니다.

	어플리케이션 참조 번호—VI 서버와 함께 사용됩니다. LabVIEW 어플리케이션에 대한 참조를 열고 참조를 다른 VI에 파라미터로서 전달하려고 할 때 이 참조 번호 컨트롤을 사용합니다. 이 참조 번호를 VI 서버에 전달하여, 그 동작을 컨트롤하고 어플리케이션의 프로퍼티를 읽을 수 있습니다. 어플리케이션 참조 번호 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 VI 서버 클래스 선택을 선택하여, 어플리케이션, VI, 컨트롤, 인디케이터, 엄격한 타입의 VI와 같은 컨트롤의 데이터 타입을 변경할 수 있습니다. 컨트롤을 끌어서 어플리케이션 참조 번호에 놓고 연결하면 컨트롤이 참조하는 컨트롤 타입을 지정할 수 있습니다.
	오토메이션 참조 번호—ActiveX 함수와 함께 사용됩니다. ActiveX 서버 객체에 대한 참조를 열고 참조를 다른 VI에 파라미터로서 전달하려고 할 때 이러한 타입의 참조 번호를 사용합니다. 참조 번호의 타입을 지정하려면, 참조 번호의 타입에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 ActiveX 클래스 선택 >> 탐색을 선택하여, 타입 라이브러리로부터 객체 선택 대화 상자를 디스플레이하고 사용할 수 있는 ActiveX 서버 어플리케이션의 리스트를 탐색합니다.
	블루투스 네트워크 연결 참조 번호—블루투스 VI 및 함수와 함께 사용됩니다. 이 타입의 참조 번호를 사용하여 블루투스 디바이스에 연결할 수 있습니다.
	바이트 스트림 파일 참조 번호—텍스트나 2진 바이트 스트림 파일을 가지고 사용됩니다. 하나의 VI에서 파일을 열거나 생성하지만 다른 VI에서 I/O 동작을 수행하려고 할 때 이 타입의 참조 번호를 사용합니다. I/O 동작을 수행하는 VI의 프런트패널에서 참조 번호 컨트롤을 가져야 하고 파일을 열고 생성하는 VI의 프런트패널에서 참조 번호 인디케이터를 가져야 합니다.
	컨트롤 참조 번호—VI 서버와 함께 사용됩니다. 프런트패널 컨트롤이나 인디케이터에 대한 참조를 열고 참조를 다른 VI에 파라미터로서 전달하려고 할 때 이 참조 번호 컨트롤을 사용합니다. 이 참조 번호를 VI 서버에 전달하여, 컨트롤의 동작과 프로퍼티를 컨트롤할 수 있습니다. 컨트롤 참조 번호 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 VI 서버 클래스 선택을 선택하여, 어플리케이션, VI, 컨트롤, 인디케이터, 엄격한 타입의 VI와 같은 컨트롤의 데이터 타입을 지정할 수 있습니다. 프런트패널의 컨트롤 참조 번호 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이콘 보이기를 선택하여 참조 번호를 아이콘으로 보여 줍니다. 프런트패널의 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 보이기를 선택하여 참조 번호를 컨트롤로 보여줍니다. 여기서 컨트롤은 디스플레이를 위한 것입니다. 컨트롤을 끌어서 컨트롤 참조 번호에 놓고 연결하면 컨트롤이 참조하는 컨트롤 타입을 지정할 수 있습니다.
	데이터 로그 파일 참조 번호—데이터로그 파일과 함께 사용됩니다. 데이터 로그 파일은 고유한 구조를 가지고 있기 때문에, 데이터 로그 파일 참조 번호는 호출하는 VI로 그리고 호출하는 VI로부터 참조 번호와 파일 타입의 설명을 전달합니다. 파일의 구조를 정의하려면 참조 번호 안에 컨트롤을 놓습니다. 숫자를 가지고 있는 파일에 대해, 숫자를 가지고 있는 데이터 로그 참조 번호를 생성합니다. 파일에서 각 레코드가 숫자들의 집합을 가지고 있다면, 참조 번호 안에 클러스터를 놓고 그 클러스터 안에 두 개의 숫자 컨트롤을 놓습니다. 컨트롤을 끌어서 데이터 로그 파일 참조 번호에 놓고 연결하면 데이터 로그 파일의 데이터 타입을 지정할 수 있습니다.
	DataSocket 참조 번호—DataSocket 함수와 함께 사용됩니다. 데이터 연결에 대한 참조를 열고 참조를 다른 VI에 파라미터로서 전달하려고 할 때 이 타입의 참조 번호를 사용합니다. 참조 번호의 타입을 지정하기 위해, 그 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 DataSocket 클래스 선택을 선택하여 DataSocket 클래스를 탐색합니다.

	이벤트 콜백 참조 번호—이 참조 번호를 사용하여 이벤트 콜백을 등록 해제하거나 다시 등록합니다. 또한, 이 참조 번호를 사용하여 subVI에 이벤트 콜백 등록을 전달할 수 있습니다.
	IrDA 네트워크 연결 참조 번호—IrDA 함수와 함께 사용됩니다. IrDA 디바이스와의 연결을 열려고 할 때 이러한 타입의 참조 번호를 사용합니다.
	메뉴 참조 번호—메뉴 함수와 함께 사용됩니다. SubVI에 대한 VI 메뉴 참조를 전달하려고 할 때 이 타입의 참조 번호를 사용합니다.
	어러런스 참조 번호—어러런스 함수와 함께 사용됩니다. 한 VI에서 어러런스를 생성하지만 다른 VI에서 어러런스를 설정하거나 기다리려고 할 때 이러한 타입의 참조 번호를 사용합니다.
	TCP 네트워크 연결 참조 번호—TCP VI 및 함수와 함께 사용됩니다. 하나의 VI에서 네트워크 연결을 열지만 다른 VI의 네트워크 연결에서 I/O 동작을 수행하려고 할 때 이 타입의 참조 번호를 사용합니다.
	UDP 네트워크 연결 참조 번호—UDP VI 및 함수와 함께 사용됩니다. 특히 UDP 소켓을 열려고 할 때 이러한 타입의 참조 번호를 사용합니다. UDP 소켓 참조를 subVI로 전달할 수 있고 그래서 subVI는 소켓을 사용할 수 있습니다.
	VI 참조 번호—VI 서버와 함께 사용됩니다. VI에 대한 참조를 열고 그 참조를 다른 VI에 파라미터로서 전달하려고 할 때 이 참조 번호 컨트롤을 사용합니다. 이 참조 번호를 VI 서버에 전달하여, VI의 동작과 프로퍼티를 컨트롤할 수 있습니다. VI 참조 번호 컨트롤을 사용하여 VI 참조 번호를 하나의 VI에서 다른 VI로 전달합니다. VI에 엄격한 타입의 참조를 원할 때 VI 참조 열기 함수는 엄격한 타입의 VI 참조 번호 컨트롤 을 타입 지정자 입력으로 요구합니다. 이 경우에, 참조 번호 컨트롤의 값은 중요하지 않습니다. 이 함수는 이 타입만을 사용합니다. 컨트롤을 끌어서 VI 참조 번호에 놓고 연결하면, 컨트롤이 참조하는 컨트롤 타입을 지정할 수 있습니다.

다음의 참조 번호는 **참조 번호** 팔레트에 위치해 있지 않습니다.

- 데이터 참조 번호 설정—하나의 VI에서 설정 파일을 열거나 그것을 다른 VI에 쓰려고 할 때 설정 파일 VI와 함께 사용됩니다. 설정 파일 VI의 참조 번호 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다. 설정 데이터 참조 번호 컨트롤을 생성하면 설정 데이터 참조를 생성하지 않습니다. [설정 데이터 열기] VI를 사용하여 설정 데이터 참조를 생성해야 합니다.
- IMAQ 세션 참조 번호—이미지 수집 VI와 함께 사용됩니다.
- 모션 참조 번호—**NI SoftMotion** VI 및 함수와 함께 사용됩니다.
- 알림자 참조 번호—블록 다이어그램의 다른 섹션 또는 동일한 컴퓨터에서 실행되고 있는 다른 VI로부터 데이터를 받을 때까지 블록 다이어그램의 실행을 중지하려고 할 때 알림 동작 함수와 함께 사용됩니다. 알림 동작 함수의 참조 번호 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다. 알림자 참조 번호 컨트롤을 생성하면 알림자 참조를 생성하지 않습니다. [알림자 얻기] 함수를 사용하여 알림자 참조를 얻습니다. 프런트패널의 알림자 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이콘 보이기를 선택하여 참조 번호를 아이콘으로 보여줍니다. 프런트패널의 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 보이기를 선택하여 참조 번호를 컨트롤로 보여줍니다. 여기서 컨트롤은 디스플레이를 위한 것입니다. 컨트롤을 끌어서 알림자 출력 참조 번호에 놓고 연결하면, 알림자의 데이터 타입을 지정할 수 있습니다.

- **큐 참조 번호**—나중에 개별적인 원소 또는 전체 데이터 원소의 배열로 가져올 수 있도록 큐에 데이터를 저장하고자 할 때 큐 동작 함수와 함께 사용됩니다. 큐 동작 함수의 참조 번호 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다. 큐 참조 번호 컨트롤을 생성하면 큐 참조를 생성하지 않습니다. [큐 얻기] 함수를 사용하여 큐 참조를 얻습니다. 프런트패널의 큐 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이콘 보이기를 선택하여 참조 번호를 아이콘으로 보여줍니다. 프런트패널의 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 보이기를 선택하여 참조 번호를 컨트롤로 보여줍니다. 여기서 컨트롤은 디스플레이를 위한 것입니다. 컨트롤을 끌어서 큐 참조 번호에 놓고 연결하면, 큐의 데이터 타입을 지정할 수 있습니다.
- **랑데부 참조 번호**—하나의 VI에서 랑데부를 생성하고 다른 VI에서 랑데부를 기다릴 때 랑데부 VI와 함께 사용됩니다. 랑데부 VI의 참조 번호 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다. 랑데부 참조 번호 컨트롤을 생성하면 유효한 랑데부 참조를 생성하지 않습니다. [랑데부 생성] VI를 사용하여 랑데부 참조를 얻습니다.
- **세마포어 참조 번호**—두 VI 또는 같은 VI의 두 섹션이 병렬로 실행되는 것일 방지하고자 할 때 세마포어 VI와 함께 사용됩니다. 세마포어 VI의 참조 번호 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다. 세마포어 참조 번호 컨트롤을 생성하면 세마포어에 대한 유효한 참조를 생성하지 않습니다. [세마포어 참조 얻기] VI를 사용하여 세마포어에 대한 참조를 생성해야 합니다.
- **공유 변수 참조 번호**—변수 프로퍼티와 함께 사용하여 프로그램적으로 공유 변수를 설정합니다. 변수 참조 프로퍼티의 변수 참조 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다.
- **TDMS 파일 참조 번호**—TDMS 스트리밍 VI와 함께 사용되어 2진 측정 파일(.tdms)에 웨이브폼과 웨이브폼 프로퍼티를 읽거나 씁니다. [TDMS 읽기] 함수의 TDMS 파일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 이 참조 번호 컨트롤을 생성합니다. TDMS 파일 컨트롤을 생성한다고 TDMS 파일 참조가 생성되지는 않습니다. TDMS 파일 참조를 생성하려면 TDMS 열기 함수를 사용해야 합니다.
- **데이터 값 참조 번호**—LabVIEW 컴파일러로 하여금 데이터 값을 복사하고 메모리에서 관련 값을 관리하도록 요청하지 않으면서 데이터 값을 사용하려는 경우, In Place 원소 구조와 함께 사용됩니다. In Place 원소 구조의 데이터 값 참조 출력에서 마우스 오른쪽 버튼을 클릭하여, 이 참조 번호 컨트롤을 생성합니다. 데이터 값 참조 번호 컨트롤을 생성해도 **데이터 값 참조**가 생성되지는 않습니다. 데이터 값 참조를 생성하려면, [새 데이터 값 참조] 함수를 사용해야 합니다. 프런트패널의 데이터 값 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이콘 보이기를 선택하여, 참조 번호를 아이콘으로 보여줍니다. 프런트패널의 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 보이기를 선택하여 참조 번호를 컨트롤로 보여줍니다. 여기서 컨트롤은 디스플레이를 위한 것입니다.
- **외부 데이터 값 참조 번호**—LabVIEW에 데이터를 불러올 때 복사본을 만들지 않고 외부 데이터 값에서 작업하려는 경우, In Place 원소 구조와 함께 사용됩니다. 외부 데이터 값 참조 번호 컨트롤을 생성해도 **외부 데이터 값 참조**가 생성되지는 않습니다. 외부 데이터 값 참조를 지원하는 디바이스 드라이버로부터 외부 데이터 값 참조를 얻어야 합니다. 데이터 값 참조를 외부 데이터 값 참조로 바꾸려면, 데이터 값 참조에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 외부 선택합니다. 또한 외부? 프로퍼티를 사용하여 데이터 값 참조를 프로그램적으로 외부 데이터 값 참조로 변경할 수 있습니다. 프런트패널의 외부 데이터 값 참조 번호에서

마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이콘 보이기를 선택하여, 참조 번호를 아이콘으로 보여 줍니다. 프런트패널의 참조 번호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 보이기를 선택하여 참조 번호를 컨트롤로 보여줍니다. 여기서 컨트롤은 디스플레이를 위한 것입니다.

- 변수 참조 번호—공유 변수 함수와 변수 프로퍼티 및 메소드와 함께 사용하여 공유 변수를 프로그램적으로 찾고, 읽고, 씁니다. 공유 변수 함수의 공유 변수 참조 번호 입력 입력 터미널에서 마우스 오른쪽 버튼을 클릭하여 이 참조 번호 컨트롤을 생성합니다.

링 및 열거형 컨트롤

링 & 열거형 컨트롤

링 & 열거형과 클래식 링 & 열거형 팔레트에 있는 **링 및 열거형 타입 컨트롤과 인디케이터**를 사용하여 선택이 가능한 문자열의 리스트를 생성합니다.

링 컨트롤에 그림 아이템 추가하기

새로운 그림 링 컨트롤 또는 텍스트와 그림 링 컨트롤에는 빈 그림 디스플레이를 갖는 하나의 아이템이 있습니다. 그림을 링 컨트롤에 반입하기 전에 반드시 그림을 클립보드에 복사해야만 합니다. LabVIEW는 사용자의 그래픽 소프트웨어가 지원하는 모든 그래픽 파일을 반입할 수 있습니다.

다음 단계를 따라 그림을 링 컨트롤로 반입합니다.

1. 그림을 클립보드로 복사합니다.
2. 그림 링 컨트롤 또는 텍스트와 그림 링 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 클립보드로부터 그림 반입을 선택합니다.
3. 다른 그림을 클립보드에 복사합니다.
4. 그림 링 컨트롤 또는 텍스트와 그림 링 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 전에 그림 반입 또는 후에 그림 반입을 선택합니다.

수행 도구를 사용하여 그림 상자를 클릭하고 모든 반입된 그림을 봅니다.

링 컨트롤 아이템에 특정 숫자값 지정하기

링 컨트롤에 아이템의 리스트를 설정할 때, 각 아이템에 특정 숫자값을 지정할 수 있습니다. 아이템에 특정 숫자값을 지정하지 않는 경우, LabVIEW는 첫번째 아이템에 대해서 0의 값을 시작으로 리스트의 아이템 순서에 따라서 순차적인 값을 지정합니다.

다음 단계를 따라 링 컨트롤의 아이템에 특정 숫자값을 지정합니다.

1. 링 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이템 편집을 선택하여 링 프로퍼티 대화 상자의 아이템 편집 페이지를 디스플레이합니다.
2. 순차적인 값 확인란에서 확인 표시를 제거합니다.
3. 변경하려는 숫자값을 포함하는 값 열에 있는 셀에서 더블 클릭합니다.

4. 사용자가 컨트롤의 대응하는 라벨을 선택할 때 블록다이어그램에 사용하려는 숫자값을 입력합니다. 링 컨트롤의 각 아이템은 유일한 숫자값을 가집니다.



노트

입력한 값은 반드시 링 컨트롤의 **숫자형**에 유효한 값이어야 합니다. 예를 들어, 링 컨트롤의 형이 16 비트 부호없는 정수인 경우, 음수값을 입력할 수 없습니다.

1. 변경하려는 각 숫자값에 대해서 3-4 단계를 반복합니다.
2. 확인 버튼을 클릭합니다.

또한 문자열과 값 [] 프로퍼티를 사용하여 프로그램에서 특정 숫자값을 지정할 수 있습니다.

정의되지 않은 숫자값을 허용하기 위해서 링 컨트롤 설정하기

사용자가 컨트롤에 대해 정의된 아이템 리스트에 아직 어떠한 엔트리와도 연계되지 않은 숫자값을 입력할 수 있도록 링 컨트롤을 설정할 수 있습니다. 링 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 정의되지 않은 값 허용을 선택하여 사용자가 컨트롤의 디지털 디스플레이에 임의의 숫자값을 입력할 수 있도록 허용합니다.

또한, 정의되지 않은 값 허용 프로퍼티를 사용하여 프로그램적으로 정의되지 않은 숫자값을 허용하도록 할 수 있습니다.



관련 정보

[링과 열거형 타입 컨트롤에서 런타임으로 아이템 선택하기](#) (1027 페이지 참조)
[라벨과 디지털 디스플레이 보이기와 숨기기](#) (1137 페이지 참조)

링 컨트롤과 상수의 숫자값 디스플레이하기

링 컨트롤 또는 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 디지털 디스플레이를 선택하여 현재 아이템과 연관있는 숫자값을 디스플레이합니다.

링 컨트롤과 열거형 타입 컨트롤에서 아이템 편집하기

다음 단계를 완료하여 링 컨트롤 또는 열거형 타입 컨트롤에서 아이템을 편집합니다:

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이템 편집을 선택하여 프로퍼티 대화 상자의 아이템 편집 페이지를 디스플레이합니다.
2. 다음 동작 중 하나 또는 이상을 완료합니다:
 - <Insert> 키를 눌러서 아이템을 추가합니다. (macOS) Insert 버튼을 클릭합니다.
 - 아이템을 선택하고 <Delete> 키를 눌러 아이템을 삭제합니다.
 - 아이템을 삭제하고 비활성화 버튼을 눌러 아이템을 비활성화합니다. VI가 실행 중인 경우에는 비활성화된 아이템을 선택할 수 없습니다. 아이템이

이미 비활성화된 경우, 활성화 버튼을 눌러 활성화할 수 있습니다. 또한 비활성화 항목의 프로퍼티를 사용하여 링 컨트롤 항목을 프로그램적으로 비활성화할 수 있습니다.

- 항목을 더블 클릭하여 항목의 이름을 편집합니다.
- 항목을 끌어서 놓아 항목의 순서를 변경합니다.
- 비어있는 셀에 하이픈 - 을 입력하여 컨트롤의 항목 리스트에 분리자를 생성합니다. 분리자는 링 또는 열거형 타입 컨트롤의 메뉴에서 항목을 분리하는 가는 막대입니다.



노트

<Ctrl> 키를 누른 상태에서 항목을 선택하고, 원하는 동작을 수행하여 여러 개의 항목을 동시에 끌어서 놓거나, 활성화 또는 비활성화하고, 삭제할 수 있습니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

3. 확인을 클릭합니다.

또한, 문자열의 프로퍼티를 사용하여 링 컨트롤에 프로그램적으로 항목을 추가하거나 삭제할 수 있습니다.

링과 열거형 타입 컨트롤에서 런타임으로 항목 선택하기

다음 방법을 사용하여 VI가 실행 모드에 있는 동안 링이나 열거형 타입 컨트롤에서 항목을 선택합니다:

- 증가나 감소 버튼을 클릭합니다.
- <Tab> 키를 눌러서 **컨트롤을 탐색**하고 위와 아래 방향 화살표 키를 누릅니다.

다음의 방법을 사용하여 VI가 실행 모드에 있는 동안 링이나 열거형 타입의 컨트롤에서 항목을 선택하거나 또는 VI가 편집 모드에 있는 동안 링이나 열거형 상수의 항목을 선택합니다.

- 컨트롤을 클릭하여 바로 가기 메뉴를 디스플레이하고 위 아래 방향 화살표 키를 사용하여 선택하려는 항목을 탐색합니다. 바로 가기 메뉴에서 항목을 하이라이트한 후에, <Enter> 키를 눌러서 항목을 선택합니다.
- 컨트롤을 클릭하여 바로 가기 메뉴를 디스플레이하고 선택하려는 문자열의 시작 문자를 타입합니다. 타입하는 문자로 시작하는 컨트롤에서 LabVIEW는 처음의 항목을 하이라이트합니다. <Enter> 키를 눌러서 항목을 선택합니다.
- **디지털 디스플레이**를 볼 수 있다면, 디지털 디스플레이에 커서를 놓고 숫자값을 입력하고 <Enter> 키를 누릅니다. 컨트롤에 존재하지 않는 값을 입력하면, LabVIEW는 입력한 값을 가장 가까운 유효한 값으로 변경합니다.
- 링 컨트롤이 **정의되지 않은 숫자값**을 허용하는 경우, 컨트롤을 클릭하고, 바로 가기 메뉴에서 <그외>를 선택하고, 나타나는 디지털 디스플레이에 숫자값을 입력하고, <Enter> 키를 누릅니다. 정의되지 않은 값이 꺾쇠 괄호(<,>)로 링 컨트롤에 나타납니다. 디지털 디스플레이를 볼 수 있는 경우, 디지털 디스플레이에 커서를 놓고 숫자값을 입력하고 <Enter> 키를 누를 수도 있습니다.

열거형 타입 컨트롤 사용하기

열거형 타입 컨트롤을 사용하여 사용자에게 선택할 수 있는 아이템 리스트를 제공합니다. 열거형 타입 컨트롤 또는 열거형은 텍스트 또는 메뉴 **링 컨트롤**과 비슷합니다. 둘 다 사이클할 수 있는 문자열 목록을 포함합니다. 하지만, 열거형 타입 컨트롤의 데이터 타입은 컨트롤의 숫자값과 문자열 라벨에 대한 정보를 포함합니다. 링 컨트롤의 데이터 타입은 숫자형입니다.



노트

사용자가 열거형 타입 컨트롤에 정의되지 않은 값을 입력하도록 허용할 수 없으며, 열거형 타입 컨트롤의 아이템에 특정 숫자값을 할당할 수 없습니다. 이 기능이 필요한 경우, **링 컨트롤**을 사용하십시오.

열거형 타입 컨트롤을 사용하여 케이스 구조의 케이스를 선택할 수 있습니다.

열거형 타입 컨트롤의 숫자형은 8, 16, 또는 32비트 부호없는 정수입니다. 열거형 타입 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 형을 선택하여 컨트롤의 **형을 변경**합니다.

링 컨트롤 사용하기

링 컨트롤을 사용하여 어떤 값이 아이템을 나타내는지를 알 필요없이 리스트에서 아이템을 선택합니다. LabVIEW는 선택된 아이템과 연계된 값을 블록다이어그램에 전달합니다. 선택된 아이템을 실행하는 케이스 구조(조건적인 코드)에서 케이스를 선택합니다.

다음 단계를 따라 링 컨트롤을 생성합니다.

1. 링 컨트롤을 프런트패널에 추가합니다.
2. 링 컨트롤에 텍스트 또는 그림 아이템을 추가합니다.
3. (옵션) 링 컨트롤에서 아이템을 제거할 수 있습니다.



관련 정보

[링 컨트롤과 열거형 타입 컨트롤에서 아이템 편집하기](#) (1026 페이지 참조)

[링 컨트롤에 그림 아이템 추가하기](#) (1025 페이지 참조)

[링 컨트롤과 열거형 타입 컨트롤에서 아이템 편집하기](#) (1026 페이지 참조)

경로 컨트롤의 값 변경하기

경로 컨트롤 또는 인디케이터의 경로 기호()가 경로 없음 기호()로 바뀌면, 경로가 유효하지 않음을 나타내기 위해 텍스트 필드에 <경로 없음>이 나타납니다. <경로 없음> 값을 경로 컨트롤의 기본값으로 사용하여, 사용자가 경로를 제공하지 않은 경우 이를 감지하고 경로 선택 옵션이 있는 파일 대화 상자를 디스플레이합니다. [파일 대화 상자] 익스프레스 VI를 사용하여 파일 대화 상자를 디스플레이합니다.

다음 단계를 따라 경로 컨트롤의 값을 변경합니다.

1. 경로 기호를 클릭합니다.

2. 바로 가기 메뉴에서 유효한 경로 또는 경로 없음을 선택합니다.

경로 컨트롤의 탐색 옵션 설정하기

경로 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이콘 >> 탐색 버튼을 선택해서 그 옆에 확인 표시를 하여 경로 컨트롤의 탐색 버튼을 디스플레이할 수 있습니다. 또한, 탐색 버튼:보이기 프로퍼티를 사용하여 프로그램적으로 탐색 버튼을 디스플레이할 수 있습니다.

다음 단계를 따라 선택할 수 있는 경로의 타입, 시작 경로, 일치 패턴 등 파일 경로 컨트롤의 탐색 버튼을 클릭할 때 나타나는 파일 대화 상자의 셋팅을 사용자 정의합니다.

1. 경로 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 탐색 옵션을 선택하여 경로 프로퍼티 대화 상자의 탐색 옵션 페이지를 디스플레이합니다.
2. 선택 모드 섹션에서 적당한 옵션을 선택합니다. 이 섹션의 옵션은 [파일 대화 상자] 익스프레스 VI의 설정 대화 상자에 있는 선택 모드 옵션과 유사합니다.



노트

경로 컨트롤을 사용하여 LLB 또는 묶음 라이브러리를 선택하려면 LLB 및 묶음 프로젝트 라이브러리에서 파일 선택 허용 확인란에서 확인 표시를 제거하고 파일 대화 상자에서 LLB 또는 묶음 라이브러리를 선택합니다. 또는 LLB 및 묶음 프로젝트 라이브러리에서 파일 선택 허용 확인란에 확인 표시를 하고, 파일 또는 폴더 옵션 버튼을 선택하고, 파일 대화 상자에서 LLB 또는 묶음 라이브러리를 선택한 후, 나타나는 두 번째 파일 대화 상자의 파일 리스트에서 두 번째 폴더 아이콘을 선택할 수 있습니다.

1. 확인 버튼을 클릭합니다.



팁 [파일 대화 상자] 익스프레스 VI 또는 [파일 열기/생성/대체] 함수를 사용하여 파일 대화 상자에 여러 파일 타입을 나열하려면 패턴 터미널에 문자열 상수 또는 컨트롤을 연결하고 `fx*.bin;*.c`처럼 세미콜론을 사용하여 패턴을 구분합니다.

정의되지 않은 문자열 값을 허용하기 위해서 콤보 박스 컨트롤 설정하기

기본적으로, 콤보 박스 컨트롤은 컨트롤에 대해서 미리 정의된 문자열의 리스트에 있지 않더라도 사용자가 **문자열 값을 입력**할 수 있도록 허용합니다. 콤보 박스 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 정의되지 않은 문자열 허용을 선택하여 확인 표시를 제거하고 사용자가 컨트롤에 정의되지 않은 문자열 값을 입력하는 것을 막습니다.

또한, 정의되지 않은 문자열 허용 프로퍼티를 사용하여 프로그램적으로 정의되지 않은 문자열 값을 허용하도록 할 수 있습니다.

콤보 박스 컨트롤에서 아이템 편집하기

다음 단계를 완료하여 콤보 박스 컨트롤에서 아이템을 편집합니다:

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이템 편집을 선택하여 프로퍼티 대화 상자의 아이템 편집 페이지를 디스플레이합니다.
2. 다음 동작 중 하나 또는 이상을 완료합니다:
 - <Insert> 키를 눌러서 아이템을 추가합니다. (macOS) Insert 버튼을 클릭합니다.
 - 아이템을 선택하고 <Delete> 키를 눌러 아이템을 삭제합니다.
 - 아이템을 더블 클릭하여 아이템의 이름을 편집합니다.
 - 아이템을 끌어서 놓아 아이템의 순서를 변경합니다.
 - 비어있는 셀에 하이픈 -을 입력하여 컨트롤의 아이템 리스트에 분리자를 생성합니다. 분리자는 콤보 박스 컨트롤의 메뉴에서 아이템을 분리하는 가는 막대입니다.



노트

<Ctrl> 키를 누른 상태에서 아이템을 선택하고, 원하는 동작을 수행하여 여러 개의 아이템을 동시에 끌어서 놓거나, 삭제할 수 있습니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

3. 확인을 클릭합니다.

문자열[] 프로퍼티 또는 문자열과 값[] 프로퍼티를 사용하여 콤보 박스 안의 문자열을 프로그램적으로 편집할 수 있습니다.

X컨트롤

X컨트롤 생성하기

X컨트롤을 사용하여 LabVIEW에서 복잡한 컨트롤과 인디케이터를 설계하고 생성할 수 있습니다.



노트

LabVIEW Professional Development System에서만 X컨트롤을 생성하고 편집할 수 있습니다. VI에 X컨트롤이 있을 경우, 모든 LabVIEW 패키지에서 그 VI를 실행할 수 있습니다.

다음 단계를 따라 X컨트롤을 생성합니다:

1. X컨트롤을 사용하려는 프로젝트를 엽니다. 새 프로젝트를 생성하여 프로젝트 탐색기 윈도우를 디스플레이할 수 있습니다.
2. 프로젝트 탐색기 윈도우에서, 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기>>X컨트롤을 선택합니다. LabVIEW는 프로젝트에 새 X컨트롤 라이브러리를 생성합니다. X컨트롤 라이브러리 내에, LabVIEW는 자동으로 필요한 기능을 생성합니다.



팁 또한 새로 만들기 대화 상자의 X컨트롤 템플릿을 사용하여 X컨트롤을 생성할 수 있습니다.

1. X컨트롤 라이브러리에서 데이터 기능을 더블 클릭하여 데이터 타입 정의를 열고 편집합니다. 숫자 컨트롤을 새 X컨트롤의 데이터 타입을 나타내는 컨트롤로 대체합니다. 파일»변경된 사항 적용을 선택하여 데이터 기능의 변경을 전달합니다.
2. X컨트롤 라이브러리에서 상태 기능을 더블 클릭하여 상태 타입 정의를 열고 편집합니다. X컨트롤의 모양이나 상태에 영향을 주는 모든 정보를 클러스터 컨트롤의 일부로 포함합니다. 파일»변경된 사항 적용을 선택하여 상태 기능의 변경을 전달합니다.
3. X컨트롤 라이브러리의 외관 기능을 더블 클릭하여 **외관 VI를 편집**하고 엽니다.
4. X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택하여 X컨트롤 프로퍼티 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 X컨트롤 라이브러리를 설정하고, X컨트롤을 문서화하고, X컨트롤의 기능, 프로퍼티 및 메소드 및 VI에 대한 셋팅을 설정합니다.
5. X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 저장을 선택하여 X컨트롤을 저장합니다.

또한 **옵션 기능**, **프로퍼티**, **메소드** 및 VI를 XControl 라이브러리에 추가할 수 있습니다. X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기 선택하여 X컨트롤 라이브러리에 아이템을 추가합니다.

X컨트롤 생성하기예 대한 예제는 labview\examples\Controls and Indicators\XControls\Dual Mode Thermometer 디렉토리의 Simple Dual Mode Thermometer XControl.lvproj를 참조하십시오.



예제 열기  관련 예제 찾기

X컨트롤 기능 생성하기

X컨트롤 기능은 X컨트롤이 올바르게 작동하는데 요구되는 X컨트롤의 기본 요소입니다. X컨트롤을 생성할 때, LabVIEW는 자동으로 필요한 기능을 생성합니다. 다음 단계를 따라 X컨트롤 라이브러리에 옵션적 기능을 추가합니다.

1. 프로젝트 탐색기 윈도우의 X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기»기능을 선택하여 기능 선택 대화 상자를 디스플레이합니다. 이 대화 상자는 X컨트롤에 적용 가능한 모든 필수 기능과 옵션 기능을 나열합니다. 기능 선택 대화 상자를 사용하여 X컨트롤 라이브러리에 옵션 기능을 추가합니다.
2. 기능 섹션에서, 기능을 선택하고 추가 버튼을 클릭하여 X컨트롤 라이브러리에 기능을 추가합니다.

- 기능을 편집하려면, X컨트롤 라이브러리의 기능을 더블 클릭하여 VI 또는 기능을 나타내는 타입 정의를 엽니다.



관련 정보

[X컨트롤 기능 사용하기](#) (101 페이지 참조)

[X컨트롤 생성하기](#) (1030 페이지 참조)

[X컨트롤 기능 사용하기](#) (101 페이지 참조)

[X컨트롤 사용하기](#) (101 페이지 참조)

[LabVIEW에서 프로젝트 관리하기](#) (290 페이지 참조)

X컨트롤 프로퍼티 생성하기

X컨트롤 프로퍼티를 사용하여 X컨트롤 사용자가 프로그램적으로 X컨트롤을 설정할 수 있습니다. X컨트롤 사용자는 프로퍼티 노드를 사용하여 X컨트롤 프로퍼티를 얻거나 설정합니다. LabVIEW는 사용자가 사용자 X컨트롤 프로퍼티를 얻거나 설정한 후 **[외관] VI**를 호출합니다. 프로퍼티 사용이 디스플레이 상태를 변경하는 경우, **[외관] VI**는 디스플레이 상태를 업데이트하여 X컨트롤이 모양을 업데이트할 수 있도록 합니다. 이 경우 디스플레이 상태 변경 이벤트가 **[외관] VI**에 생성됩니다.

X컨트롤 라이브러리에서, 각 X컨트롤 프로퍼티는 프로퍼티가 읽기 전용인지, 쓰기 전용인지, 또는 읽기/쓰기인지에 따라 하나 또는 그 이상의 VI로 나타냅니다. 각 VI는 프로퍼티의 읽기 또는 쓰기 기능을 나타냅니다.

다음 단계를 따라 **X컨트롤 라이브러리**에 프로퍼티를 추가합니다.

- X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기 >> 프로퍼티를 선택하여 프로퍼티 생성 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 X컨트롤 라이브러리에 프로퍼티를 추가합니다. LabVIEW는 X컨트롤 라이브러리에 프로퍼티와 같은 이름을 가진 폴더를 생성합니다. 읽기/쓰기 프로퍼티를 생성하는 경우, 프로퍼티 폴더는 프로퍼티로부터 읽는 VI와 프로퍼티에 쓰는 VI를 포함합니다. 읽기 전용이나 쓰기 전용 프로퍼티를 생성하는 경우, 프로퍼티 폴더는 각각 프로퍼티로부터 읽는 VI만, 혹은 프로퍼티에 쓰는 VI만을 포함합니다.



노트

프로퍼티 폴더는 VI를 두 개 이상 포함할 수 없습니다.

- 프로퍼티를 편집하려면, 프로퍼티 폴더의 각 VI를 더블 클릭하여 프로퍼티의 읽기 또는 쓰기 기능을 나타내는 VI를 엽니다.

읽기 VI 편집하기

[읽기] VI는 X컨트롤 프로퍼티의 읽기 기능을 나타냅니다. **[읽기] VI**는 디스플레이 상태와 컨테이너 상태 컨트롤 및 디스플레이 상태, 값, 에러 인디케이터를 포함합니다. LabVIEW가 이 컨트롤과 인디케이터를 사용하여 디스플레이 상태와 컨테이너 상태를 프로퍼티에 전달하고 에러 정보를 가져오므로 이 컨트롤이나 인디케이터를 삭제하거나 변경하

지 않도록 하십시오. 기본으로, [읽기] VI는 값의 LED 인디케이터를 포함합니다. LED 인디케이터를 적절한 데이터 타입의 인디케이터로 대체하여 프로퍼티가 읽는 데이터의 타입을 설정합니다.



노트 X컨트롤 프로퍼티가 읽기/쓰기인 경우, [읽기] VI의 값 인디케이터의 데이터 타입은 [쓰기] VI의 값 컨트롤의 데이터 타입과 일치해야 합니다.

VI를 삭제하려면, VI에서 마우스 오른쪽 버튼을 클릭하고 프로젝트에서 제거를 선택합니다. 프로퍼티 폴더에서 마우스 오른쪽 버튼을 클릭하고 새로 만들기>>[읽기] VI를 선택하여 [읽기] VI를 쓰기 전용 프로퍼티에 추가합니다.

쓰기 VI 편집하기

[쓰기] VI는 [읽기] VI와 유사합니다. [읽기] VI와 같은 디스플레이 상태와 컨테이너 상태, 에러 컨트롤 및 디스플레이를 포함합니다. LabVIEW가 이 컨트롤과 인디케이터를 사용하여 디스플레이 상태와 컨테이너 상태를 프로퍼티에 전달하고 에러 정보와 업데이트된 디스플레이 상태를 가져오므로 이 컨트롤이나 인디케이터를 삭제하거나 변경하지 않도록 하십시오. 기본으로, [쓰기] VI는 값의 LED 컨트롤을 포함합니다. LED 컨트롤을 적절한 데이터 타입의 컨트롤로 대체하여 프로퍼티가 쓰는 데이터의 타입을 설정합니다.



노트 X컨트롤 프로퍼티가 읽기/쓰기인 경우, [읽기] VI의 값 인디케이터의 데이터 타입은 [쓰기] VI의 값 컨트롤의 데이터 타입과 일치해야 합니다.

VI를 삭제하려면, VI에서 마우스 오른쪽 버튼을 클릭하고 프로젝트에서 제거를 선택합니다. 프로퍼티 폴더에서 마우스 오른쪽 버튼을 클릭하고 새로 만들기>>[쓰기] VI를 선택하여 [쓰기] VI를 읽기 전용 프로퍼티에 추가합니다.

X컨트롤 메소드 생성하기

X컨트롤 메소드를 사용하여 X컨트롤 사용자가 프로그램적으로 X컨트롤을 설정할 수 있습니다. X컨트롤 사용자는 인보크 노드를 사용하여 X컨트롤 메소드를 수행합니다. LabVIEW는 사용자가 사용자 X컨트롤 메소드를 수행한 후 **[외관] VI**를 호출합니다. 메소드 수행이 디스플레이 상태를 변경하는 경우, 외관 VI는 디스플레이 상태를 업데이트하여 X컨트롤이 모양을 업데이트할 수 있도록 합니다. 이 경우 디스플레이 상태 변경 이벤트가 [외관] VI에 생성됩니다.

X컨트롤 라이브러리에서, 각 X컨트롤 메소드는 VI로 나타냅니다.

다음 단계를 따라 **X컨트롤 라이브러리**에 메소드를 추가합니다.

1. X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기>>메소드를 선택하여 X컨트롤 라이브러리에 새 메소드를 추가합니다. LabVIEW는 X컨트롤 라이브러리에 새 메소드를 생성합니다.
2. X컨트롤 라이브러리의 메소드를 더블 클릭하여 메소드의 파라미터를 나타내는 컨트롤과 인디케이터를 추가합니다.

메소드 VI 편집하기

메소드 VI는 X컨트롤 메소드를 나타냅니다. 메소드 VI는 디스플레이 상태 컨트롤 및 디스플레이 상태, 컨테이너 상태, 에러 인디케이터를 포함합니다. LabVIEW가 이 컨트롤과 인디케이터를 사용하여 디스플레이 상태를 메소드에 전달하고 에러 정보를 가져오므로 이 컨트롤이나 인디케이터를 삭제하거나 변경하지 않도록 하십시오.

컨트롤이나 인디케이터를 추가하여 메소드의 파라미터를 나타냅니다. 파라미터는 입력, 출력, 또는 입력/출력 파라미터일 수 있습니다. VI에 추가한 각 컨트롤 또는 인디케이터는 하나의 파라미터의 부분만이 될 수 있습니다. LabVIEW는 컨트롤이나 인디케이터의 이름을 파라미터의 이름으로 사용합니다.



노트

파라미터가 입력/출력 파라미터인 경우, 컨트롤/인디케이터 짝을 사용하여 파라미터를 나타내야 합니다. 컨트롤/인디케이터 짝은 반드시 같은 데이터 타입이어야 합니다.

컨트롤이나 인디케이터를 커넥터 팬에 연결합니다. 메소드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 메소드 설정을 선택하여 메소드 설정 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 메소드에 추가한 파라미터를 설정합니다.

VI를 삭제하려면, VI에서 마우스 오른쪽 버튼을 클릭하고 프로젝트에서 제거를 선택합니다.

X컨트롤의 모양 변경하기

[외관] VI의 윈도우 모양 프로퍼티를 설정하여 **X컨트롤**의 모양을 조절합니다.

X컨트롤 스크롤 막대

X컨트롤의 스크롤 막대를 보이거나 숨기려면 [외관] VI의 프런트패널 윈도우에서 스크롤 막대를 보이거나 숨깁니다. 윈도우 모양 사용자 정의 대화 상자의 수직 스크롤 막대 보이기와 수평 스크롤 막대 보이기 확인란을 사용하여 스크롤 막대를 보이거나 숨깁니다. X컨트롤이 여러 팬을 가진 경우, [외관] VI의 각 팬의 스크롤 막대를 보이거나 숨겨야 합니다. 분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후 각 구획의 바로 가기 메뉴에서 보이는 아이tem >> 수평 스크롤 막대 또는 보이는 아이tem >> 수직 스크롤 막대를 선택하여 스크롤 막대를 디스플레이합니다.

X컨트롤 경계

X컨트롤의 경계를 보이거나 숨기려면 [외관] VI의 프런트패널의 제목 표시줄을 보이거나 숨깁니다. 윈도우 모양 사용자 정의 대화 상자의 제목 표시줄 보이기 확인란을 사용하여 제목 표시줄을 보이거나 숨깁니다.

X컨트롤 배경 (Windows 및 macOS)

X컨트롤의 배경을 투명하게 하려면, [외관] VI의 프런트패널을 투명하게 합니다. 윈도우 모양 사용자 정의 대화 상자의 윈도우를 투명하게 실행 확인란에 확인 표시를 하여 X컨트롤을 투명하게 합니다.

X컨트롤 크기 조정

외관 VI에서 프런트패널 스케일링을 활성화시키거나 외관 VI의 이벤트 구조에서 패널 크기 조정 이벤트를 처리하여 **X컨트롤**이 크기 조정에 응답하도록 설정할 수 있습니다.

외관 VI에서 프런트패널 스케일링을 활성화하는 경우, X컨트롤의 모양을 구성하는 컨트롤은 X컨트롤의 크기가 조정될 때 자동으로 크기 조정합니다. 다음 단계를 따라 외관 VI의 프런트패널 스케일링을 활성화합니다.



노트

외관 VI가 여러 팬을 가진 경우, 각 팬에 대해 다음 단계를 완료해야 합니다.

1. X컨트롤 라이브러리의 외관 기능을 더블 클릭하여 **외관 VI를 편집**하고 엽니다.
2. 프런트패널의 크기를 재조정할 때 모든 프런트패널 객체를 **자동으로 스케일**하도록 설정합니다.

혹은, 외관 VI의 이벤트 구조에서 패널 크기 조정 이벤트를 처리할 수 있습니다. X컨트롤이 크기 조정될 때, LabVIEW는 외관 VI를 패널 크기 조정 이벤트와 함께 호출합니다. 이벤트에 대한 응답으로, X컨트롤의 모양을 구성하는 각 컨트롤을 개별적으로 이동하고 크기 조정할 수 있습니다. 패널 크기 조정 이벤트를 사용하면 크기 조정에서 더 많은 유연성을 가질 수 있기 때문에 유용합니다. 그러나 더 많은 코딩이 필요합니다.

다음 단계를 따라 외관 VI의 이벤트 구조를 설정하여 패널 크기 조정 이벤트를 처리합니다.

1. X컨트롤 라이브러리의 외관 기능을 더블 클릭하여 외관 VI를 편집하고 엽니다.
2. 블록 다이어그램의 이벤트 구조에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 이벤트 케이스 추가를 선택하여 이벤트 편집 대화 상자를 디스플레이합니다.
3. 대화 상자의 이벤트 소스 리스트에서 <이 VI>를 선택합니다.
4. 이벤트 리스트에서 패널 크기 조정을 선택합니다.
5. 확인 버튼을 클릭하여 설정을 저장하고 대화 상자를 닫습니다.



노트 X컨트롤의 크기가 일정하게 고정되어야 하는 경우, 외관 VI의 프런트패널에서 크기 조정을 비활성화합니다. 윈도우 모양 사용자 정의 대화 상자의 사용자 정의 윈도우 크기 조정 허용 확인란에서 확인 표시를 제거하여 크기 조정을 비활성화합니다.

X컨트롤이 최소 크기를 가져야 하는 경우, 외관 VI에 최소 크기를 지정합니다. VI 프로퍼티 대화 상자의 윈도우 크기 페이지에 있는 최소 패널 크기 옵션에 길이와 높이 값을 입력합니다.

X컨트롤 디버깅하기

[외관] VI를 디버깅하려면, VI의 프런트패널을 열고, X컨트롤에서 마우스 오른쪽 버튼을 클릭하여 바로 가기 메뉴에서 고급 >> 블록 다이어그램 보이기를 선택합니다. 브레이크포인트, 단계별 실행, 프로브를 사용하여 다른 VI를 디버깅하는 것과 마찬가지로 [외관] VI를 디버깅합니다.

기능, 프로퍼티, 또는 메소드를 디버깅하려면, 기능, 프로퍼티 또는 메소드 VI에 브레이크포인트를 설정합니다. LabVIEW가 VI를 호출할 때, LabVIEW는 브레이크포인트에서 실행을 멈추고 VI의 블록 다이어그램을 디스플레이합니다. 단계별 실행과 프로브를 사용하여 다른 VI를 디버깅하는 것과 마찬가지로 VI를 디버깅합니다.



관련 정보

[X컨트롤 외관 기능 사용하기](#) (102 페이지 참조)

[브레이크포인트 관리하기](#) (1379 페이지 참조)

[VI의 단계별 실행](#) (909 페이지 참조)

[프로브 도구 사용하기](#) (1377 페이지 참조)

[X컨트롤 기능 사용하기](#) (101 페이지 참조)

[X컨트롤 프로퍼티 생성하기](#) (1032 페이지 참조)

[X컨트롤 메소드 생성하기](#) (1033 페이지 참조)

사용 중인 X컨트롤 편집하기

VI의 프런트패널에 **X컨트롤**을 추가할 때, LabVIEW는 X컨트롤 라이브러리를 로드합니다. LabVIEW는 VI가 X컨트롤을 사용하고 있는 동안 X컨트롤 라이브러리를 잠급니다. X컨트롤 라이브러리가 잠긴 동안에는 편집할 수 없습니다. 다음 단계를 따라 VI가 사용하고 있는 X컨트롤을 잠금해제하고 편집합니다.

1. 프로젝트 탐색기 윈도우에서 X컨트롤 라이브러리의 프로젝트 파일 .lvproj를 엽니다.
2. X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 편집을 위해 라이브러리 잠금해제를 선택합니다.



노트

LabVIEW가 편집을 위해 X컨트롤 라이브러리를 잠금해제하기 전에 X컨트롤을 사용하는 모든 VI의 실행을 정지해야 합니다.

1. 편집을 위해 라이브러리 잠금해제 아이템을 사용하면, LabVIEW는 라이브러리를 잠금해제합니다. 그러나 X컨트롤을 사용하는 모든 VI는 깨지게 됩니다. X컨트롤 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 변경된 사항을 인스턴스에 적용을 선택하여 변경 사항을 X컨트롤을 사용하는 모든 VI에 적용합니다. VI는 더 이상 깨지지 않아야 합니다.

외관 VI에서 참조 얻기

다음 단계를 완료하여 특정 X컨트롤을 소유하는 VI에 대한 참조를 얻습니다.

1. X컨트롤 라이브러리의 외관 기능을 더블 클릭하여 **[외관] VI**를 열고 편집합니다.
2. 블록 다이어그램에 있는 컨테이너 상태 클러스터의 출력에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 클러스터, 클래스, & 배리언트 팔레트를 선택합니다.
3. 팔레트에서 **[이름으로 풀기]** 함수를 선택하고 이를 컨테이너 상태 클러스터에 연결합니다.
4. **[이름으로 풀기]** 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **아이템 선택** > 참조 번호를 선택합니다.
5. 참조 번호에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **생성** > 컨트롤 클래스 프로퍼티 > 소유자를 선택합니다. **[이름으로 풀기]** 함수 옆에 프로퍼티 노드를 놓습니다.
6. **[이름으로 풀기]** 함수의 출력을 프로퍼티 노드의 참조 입력에 연결합니다.
7. 참조 번호에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **생성** > 컨트롤 클래스 프로퍼티 > 객체 소유 VI를 선택하여 두번째 프로퍼티 노드를 생성합니다. 단계 5에서 생성한 소유자 프로퍼티 노드의 오른쪽에 프로퍼티 노드를 놓습니다.
8. 소유자 프로퍼티 노드의 참조 출력 출력을 객체소유 VI 프로퍼티 노드의 참조 입력에 연결합니다.
9. VI를 실행합니다.

텍스트 기반의 컨트롤에서 실행 중 끊어서 놓기 동작 설정하기

VI의 문자열, 경로, 리스트박스, 여러 열 리스트 박스, 트리 컨트롤과 같은 텍스트 기반 컨트롤 사이에서 데이터를 끊어서 놓기를 활성화하려면, 이러한 컨트롤에서 끊어서 놓기 동작을 설정해야 합니다. 다음과 같은 방법으로 텍스트 기반의 컨트롤에서 끊어서 놓기 동작을 설정할 수 있습니다:

- 실행하는 도중 컨트롤에서 끊어오기 허용.
- 실행하는 도중 컨트롤에 놓기 허용.
- 컨트롤에서 데이터 이동하기 및 복사하기 중 선택.
- 끌기 소스가 아닌 컨트롤에 데이터 놓기 허용.
- 트리 컨트롤의 상위 아이템의 끌기 허용.
- 트리 컨트롤에 놓은 아이템을 상위 아이템으로 놓기 허용.

여러 끊어서 놓기 셋팅을 동시에 활성화하여 텍스트 기반의 컨트롤에서 특정한 동작을 설정합니다. 예를 들어, 리스트박스에서 트리 컨트롤로 끌기를 활성화하려면, 리스트박스에서 끌기를 허용하고 리스트박스에서 끌어온 데이터를 리스트박스가 아닌 컨트롤에 놓는 것을 허용하며, 트리 컨트롤에 놓는 것을 허용해야 합니다.

실행하는 도중 컨트롤에서 끌어오기 허용

컨트롤	바로 가기 메뉴 아이템	프로퍼티
문자열	고급»끌기 허용	문자열 프로퍼티: 끌기 허용
리스트박스	끌어서 놓기»끌기 허용	리스트박스 프로퍼티: 끌기/놓기:끌기 허용
여러 열 리스트박스	끌어서 놓기»아이템 끌기»끌기 허용	여러 열 리스트박스 프로퍼티: 끌기/놓기:끌기 허용
트리	끌어서 놓기»끌기 허용	트리 프로퍼티: 끌기/놓기:끌기 허용

실행하는 도중 컨트롤에 놓기 허용

컨트롤	바로 가기 메뉴 아이템	프로퍼티
문자열	고급»놓기 허용	문자열 프로퍼티: 놓기 허용
경로	고급»놓기 허용	경로 프로퍼티: 놓기 허용
리스트박스	끌어서 놓기»놓기 허용	리스트박스 프로퍼티: 끌기/놓기:놓기 허용
여러 열 리스트박스	끌어서 놓기»놓기 허용	여러 열 리스트박스 프로퍼티: 끌기/놓기:놓기 허용
트리	끌어서 놓기»아이템 끌기»놓기 허용	트리 프로퍼티: 끌기/놓기:놓기 허용

컨트롤에서 데이터를 이동 또는 복사 중 선택

컨트롤	바로 가기 메뉴 아이템	프로퍼티
리스트박스	• 끌어서 놓기»끌기 모드»이동만 • 끌어서 놓기»끌기 모드»복사만 • 끌어서 놓기»끌기 모드»복사 또는 이동	리스트박스 프로퍼티: 끌기/놓기:끌기 모드
여러 열 리스트박스	• 끌어서 놓기»끌기 모드»이동만 • 끌어서 놓기»끌기 모드»복사만 • 끌어서 놓기»끌기 모드»복사 또는 이동	여러 열 리스트박스 프로퍼티: 끌기/놓기:끌기 모드
트리	• 끌어서 놓기»끌기 모드»이동만 • 끌어서 놓기»끌기 모드»복사만 • 끌어서 놓기»끌기 모드»복사 또는 이동	트리 프로퍼티: 끌기/놓기:끌기 모드

끌기 소스가 아닌 컨트롤에 놓기 허용

컨트롤	바로 가기 메뉴 아이템	프로퍼티
리스트 박스	끌어서 놓기»아이템 끌기»컨트롤 밖에서 끌어서 놓기 허용	리스트박스 프로퍼티: 끌기/놓기:컨트롤 밖에서 끌어서 놓기 허용
여러 열 리스트 박스	끌어서 놓기»아이템 끌기»컨트롤 밖에서 끌어서 놓기 허용	여러 열 리스트박스 프로퍼티: 끌기/놓기:컨트롤 밖에서 끌어서 놓기 허용
트리	끌어서 놓기»아이템 끌기»컨트롤 밖에서 끌어서 놓기 허용	트리 프로퍼티: 끌기/놓기:컨트롤 밖에서 끌어서 놓기 허용

트리 컨트롤의 상위 아이템 끌기 허용

컨트롤	바로 가기 메뉴 아이템	프로퍼티
트리	끌어서 놓기»아이템 끌기»상위 아이템의 끌기 허용	트리 프로퍼티: 끌기/놓기:상위 아이템의 끌기 허용



노트

트리 컨트롤에서 상위 아이템을 하위 아이템과 함께 다른 텍스트 기반 컨트롤에 끌어놓으면, 상위 아이템만 놓아집니다. 상위 및 하위 아이템을 다른 트리 컨트롤에 놓을 때만 상, 하위 아이템이 모두 놓아집니다.

트리 컨트롤에 놓은 아이템을 상위 아이템으로 놓기 허용

컨트롤	바로 가기 메뉴 아이템	프로퍼티
트리	끌어서 놓기»아이템 놓기»아이템 사이에 놓기 허용	트리 프로퍼티: 끌기/놓기:아이템 사이에 놓기 허용

사용자 정의된 끌어서 놓기 동작 구현하기

LabVIEW에는 텍스트 기반의 컨트롤에서의 끌어서 놓기 동작 기능이 내장되어 있지만, 텍스트 기반이 아닌 컨트롤을 포함하여 모든 컨트롤에서 사용자 정의된 끌어서 놓기 동작을 프로그램적으로 구현할 수 있습니다. 끌어서 놓기 이벤트로 끌어서 놓기 동작을 프로그램적으로 구현하여 다음과 같은 태스크를 수행합니다:

- 텍스트 기반이 아닌 컨트롤 사이에서 데이터를 끌어서 놓을 수 있도록 합니다.

- 텍스트 기반의 컨트롤에서 사용자 정의된 끌어서 놓기 동작을 제공합니다.



노트

이러한 토픽은 이벤트 구조, 프로퍼티 및 메소드 사용에 익숙하다고 가정합니다.

다음 단계를 완료하여 사용자 정의된 끌어서 놓기 동작을 구현합니다:

1. 끌기 소스와 놓기 타겟으로 사용할 컨트롤 식별.
2. 사용자가 끌어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비.
3. 사용자가 끌어서 놓기 작업을 중단하는 것을 감지.
4. 끌어진 데이터를 제공하여 끌어서 놓기 작업 시작.
5. 놓기 타겟이 끌어진 정보를 받을 수 있는지 확인.
6. 끌어진 정보로 놓기 타겟 업데이트.
7. VI를 언제 정지할지 결정.



노트 이 토픽에서는 블록 다이어그램을 보기 쉽게 하고 사용자 정의된 끌어서 놓기 동작을 구현하는데 필수적인 단계에 집중하기 위해 에러 핸들링 과정을 생략했습니다. 에러 핸들링 매커니즘을 제공하여 에러가 발생한 원인 및 지점을 설명하도록 하십시오.

추가 예제

끌어서 놓기 이벤트를 사용하여 끌어서 놓기 동작을 구현하는 예제는 labview \examples\Structures\Event Structure 디렉토리에 있는 Drag and Drop - Passing Custom Drag Data VI를 참조합니다.



예제 열기



관련 예제 찾기

그래프의 플롯에서 끌어서 놓기 동작을 구현하는 예제는 labview\examples\Structures\Event Structure 디렉토리에 있는 Drag and Drop - Initiating a Custom Drag VI를 참조하십시오.

 예제 열기  관련 예제 찾기



관련 정보

[텍스트 기반의 컨트롤에서 실행 중 끊어서 놓기 동작 설정하기](#) (1037 페이지 참조)

[이벤트 구조에서 이벤트를 모니터하는 방법 선택하기](#) (408 페이지 참조)

[프로퍼티와 메소드 생성하기](#) (1427 페이지 참조)

[프로퍼티와 메소드 생성하기](#) (1427 페이지 참조)

[단계 1: 끌기 소스와 놓기 타겟으로 사용할 컨트롤 식별](#) (1041 페이지 참조)

[단계 2: 사용자가 끊어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비](#) (1042 페이지 참조)

[단계 3: 사용자가 끊어서 놓기 작업을 강제 종료하는 것을 감지](#) (1043 페이지 참조)

[단계 4: 끊어진 데이터를 제공하여 끊어서 놓기 작업 시작](#) (1044 페이지 참조)

[단계 5: 놓기 타겟이 끊어진 데이터를 받을 수 있는지 확인](#) (1045 페이지 참조)

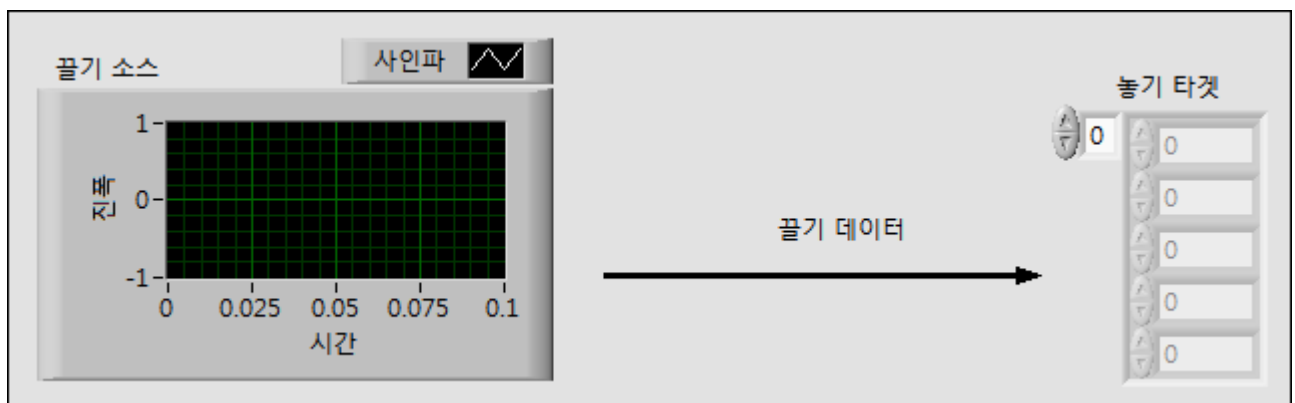
[단계 6: 끊어진 정보로 놓기 타겟을 업데이트](#) (1047 페이지 참조)

[이벤트 구조에서 래치된 정지 불리언 컨트롤 핸들링](#) (1274 페이지 참조)

[에러 핸들링](#) (263 페이지 참조)

단계 1: 끌기 소스와 놓기 타겟으로 사용할 컨트롤 식별

끊어서 놓기 이벤트는 끌기 소스 및 놓기 타겟과 연관되어 있습니다. 끌기 소스는 데이터를 제공하는 컨트롤이며, 놓기 타겟은 끊어진 데이터를 받아들이는 컨트롤입니다. 이 토픽에서는 웨이브폼 그래프에서 숫자형 컨트롤에 정보를 끊어서 놓는 방법을 설명합니다. 그러므로, 다음과 같이 끌기 소스는 웨이브폼 그래프이며, 놓기 타겟은 배열입니다:



사용자가 끊어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비하는 경우, 컨트롤을 관련 이벤트의 이벤트 소스로 설정하여 끌기 소스 및 놓기 타겟으로 식별합니다.

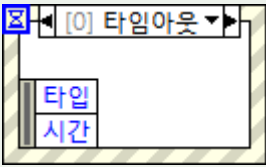
어떤 컨트롤을 끌기 소스와 놓기 타겟으로 사용할지 식별한 후, [사용자가 끊어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비](#)합니다.

단계 2: 사용자가 끌어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비

사용자가 컨트롤 사이에서 데이터 끌어서 놓기를 시도하는 경우, 특정한 이벤트가 발생하게 됩니다. 사용자가 끌기 소스에서 놓기 타겟으로 성공적으로 데이터를 복사 또는 이동하도록 하려면, 관련 이벤트가 사용자 동작을 끌어서 놓기 동작으로 해석하도록 설정해야 합니다.

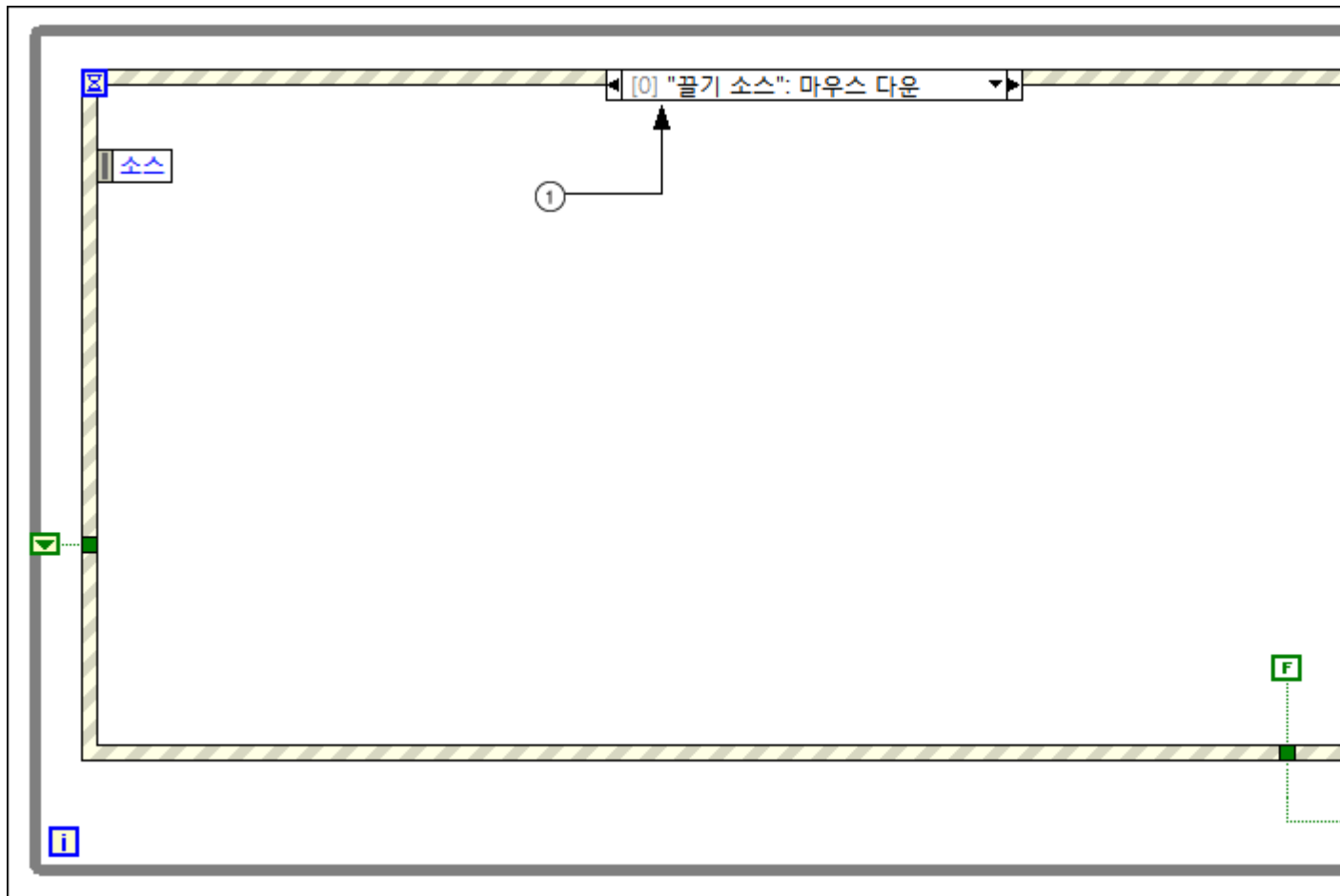
사용 객체

다음 블록다이어그램 객체를 사용하여 사용자가 끌어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비합니다.

이벤트 구조	While 루프	시프트 레지스터
		
생성		

실행 단계

다음 블록다이어그램을 생성하여 사용자가 끌어서 놓기 이벤트를 생성하는 것을 감지하도록 VI를 준비합니다.



다음 리스트에서는 이전 블록다이어그램에 대한 중요한 상세 정보를 설명합니다:



노트

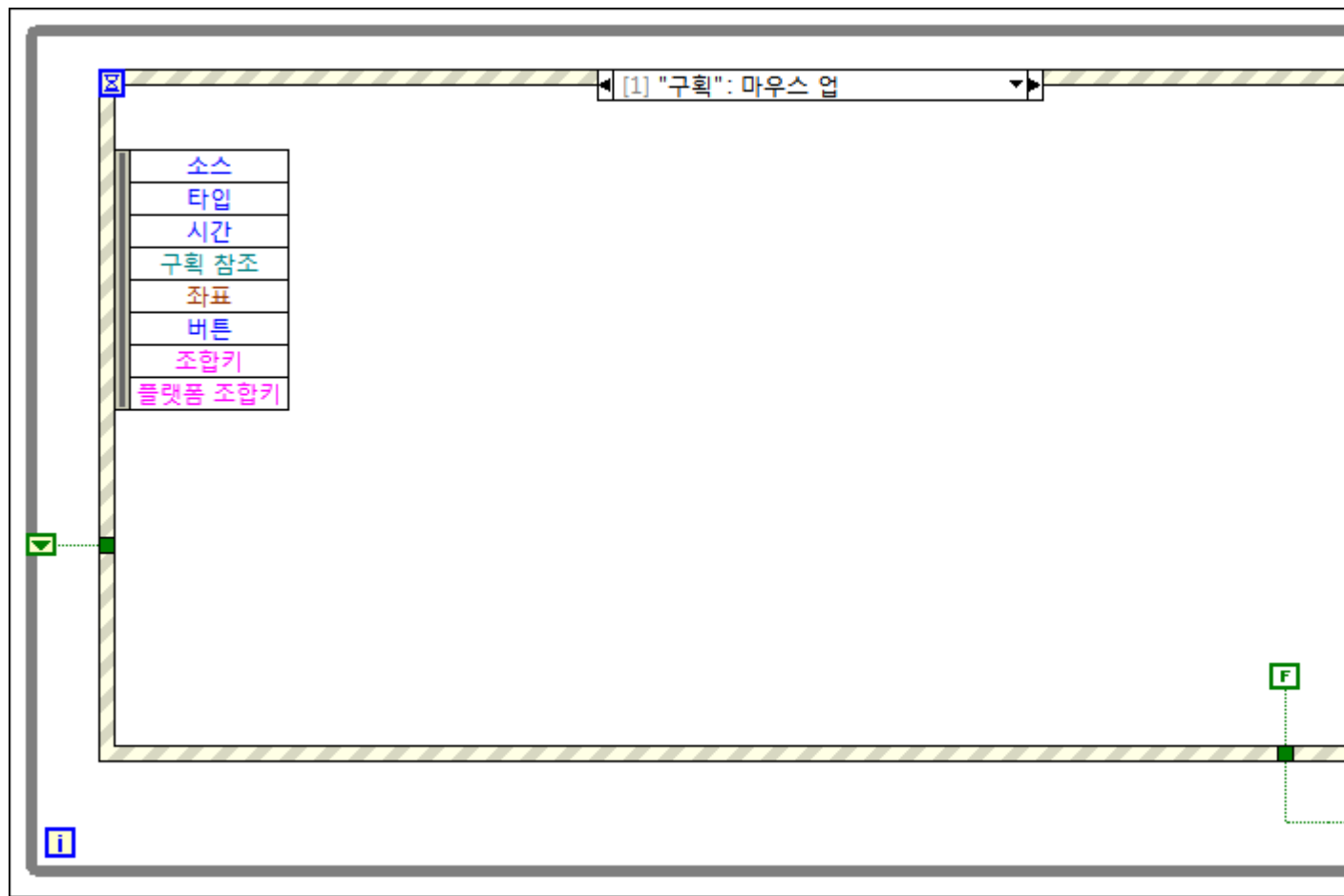
위 이벤트의 컨트롤에 붙이는 라벨은 끌기 소스와 놓기 타겟입니다. 끌어서 놓기 동작을 구현하려면, 끌어오려는 소스를 마우스 다운 및 마우스 커서 이탈 이벤트의 이벤트 소스로 설정하고, 놓기 타겟을 끌기 입력 및 놓기 이벤트의 이벤트 소스로 설정합니다.

VI가 사용자가 끌어서 놓기 이벤트를 시작하는 것을 감지하도록 준비한 후, **사용자가 끌어서 놓기 작업을 중단하는 것을 감지**합니다.

단계 3: 사용자가 끌어서 놓기 작업을 강제 종료하는 것을 감지

사용자가 프런트패널에서 놓기 타겟이 아닌 다른 곳에 마우스 버튼을 놓으면 끌어서 놓기 작업이 강제 종료됩니다. 사용자가 마우스 버튼을 놓을 때 끌어서 놓기 작업을 종료하려면, 마우스 업 이벤트를 감지하여 이 이벤트가 발생할 때 끌기 동작이 시작하는 것을 방지해야 합니다. 시프트 레지스터의 값을 거짓으로 하면 마우스 업 이벤트가 발생할 때 VI가 끌기를 시작하는 것을 방지할 수 있습니다.

다음 블록다이어그램을 생성하여 사용자가 끌어서 놓기 작업을 강제 종료하는 것을 감지합니다.



사용자가 끌어서 놓기 작업을 강제 종료한 것을 감지한 후, **끌어온 데이터를 제공하여 끌어서 놓기 작업을 시작**합니다.

단계 4: 끌어온 데이터를 제공하여 끝어서 놓기 작업 시작

끝기 소스로부터 놓기 타겟으로 데이터를 전송하려면, 먼저 사용자가 마우스 버튼을 누르고 커서를 끝기 소스의 경계 밖으로 움직일 때 끌어온 데이터를 저장해야 합니다. LabVIEW는 끌어온 데이터를 저장한 후, 사용자가 놓기 작업을 시작할 때 이 데이터를 불러와 놓기 타겟을 업데이트할 수 있습니다.

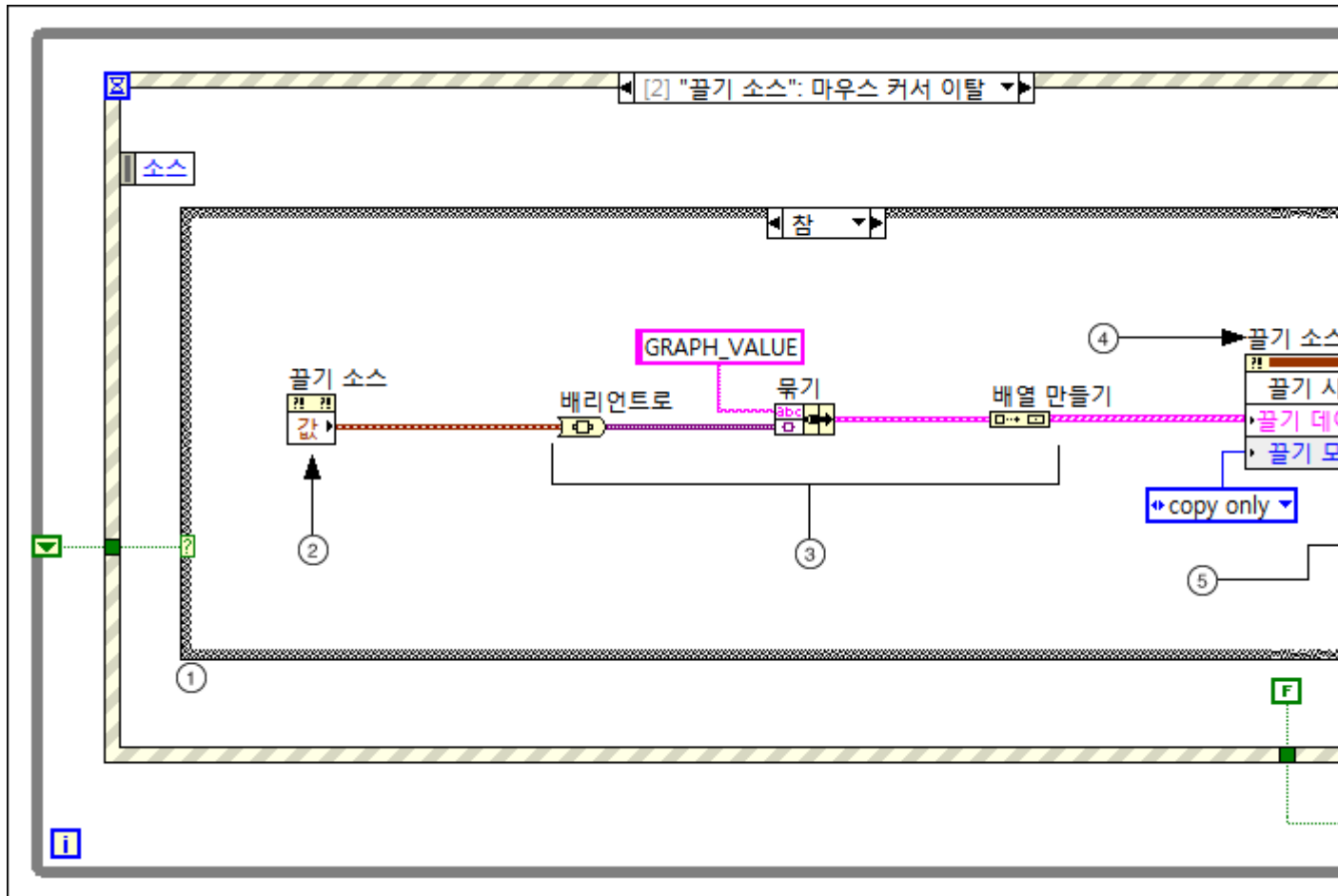
사용 객체

다음 블록 다이어그램 객체를 사용해서 끌어온 데이터를 제공하여 끌기를 시작합니다:

케이스 구조	값	배리언트로	묶기	배열 만들기	끝기 시작

다음 단계

다음 블록다이어그램 객체를 생성해서 끌어온 데이터를 제공하여 끌기를 시작합니다:



다음 리스트에서는 이전 블록다이어그램에 대한 중요한 상세 정보를 설명합니다.



노트

사용자 데이터 이름을 선택할 때 데이터 이름을 접두어 LV_로 시작하지 마십시오. 접두어 LV_는 LabVIEW 내장 데이터 타입을 위해 남겨둡니다.

끌어온 데이터를 제공하여 끌어서 놓기 작업을 시작한 후, **놓기 타겟이 끌어온 데이터를 받을 수 있는지 확인**하십시오.

단계 5: 놓기 타겟이 끌어온 데이터를 받을 수 있는지 확인

놓기 작업의 예상 결과를 결정하고 이 결과를 사용자에게 알려려면, 놓기 타겟이 끌어온 데이터를 받을 수 있는지 확인합니다. LabVIEW는 놓기 동작이 발생할 때, 놓기 타겟이 끌어온 데이터로 업데이트 되는 것을 커서를 변경하여 나타내도록 기본 설정되어 있습니다.

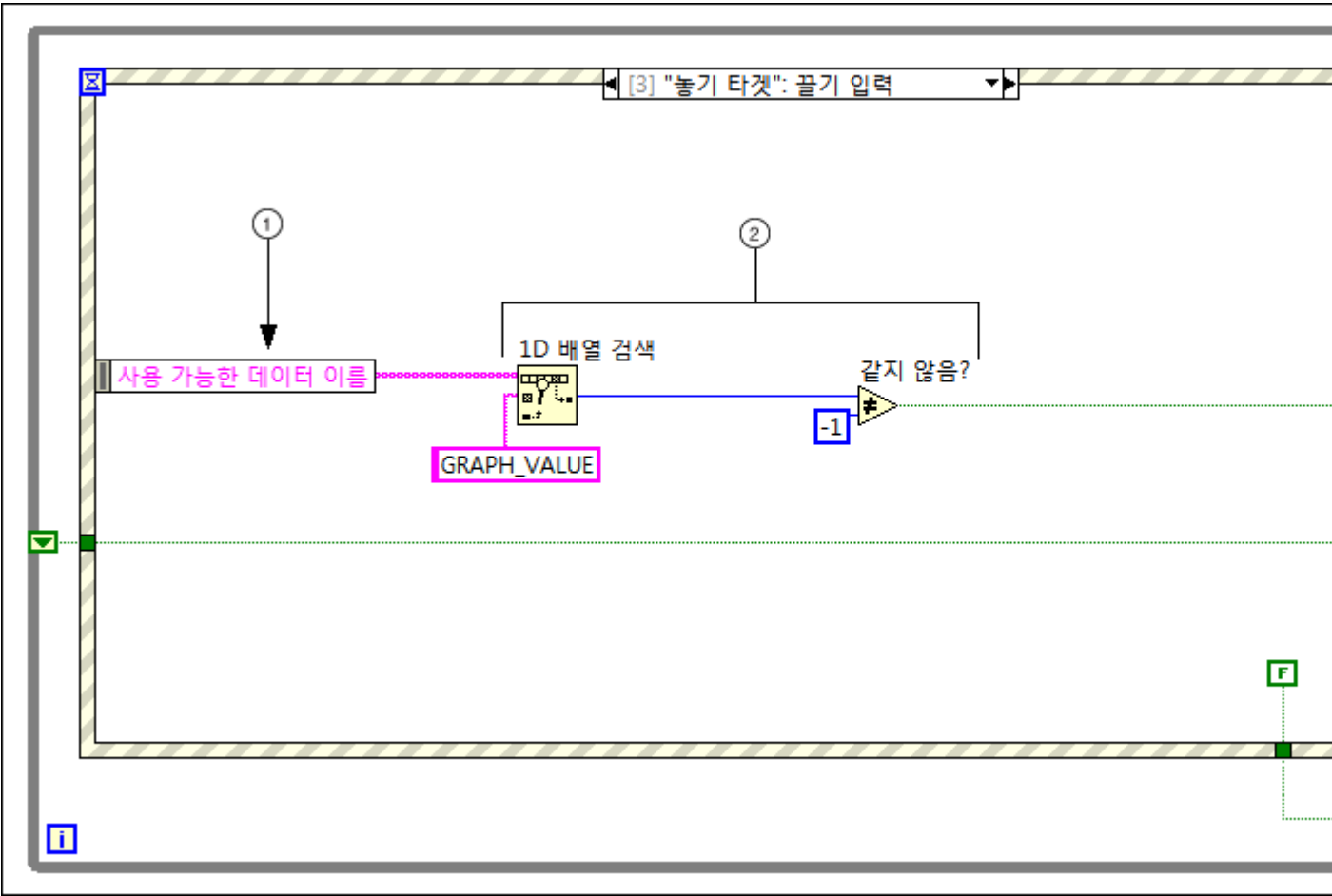
사용 객체

다음 블록다이어그램 객체를 사용하여 놓기 타겟이 끌어진 데이터를 받을 수 있는지 확인합니다:

1D 배열 검색	같지 않음?
	

다음 단계

다음 블록다이어그램을 생성하여 놓기 타겟이 끌어진 데이터를 받을 수 있는지 확인합니다.



다음 리스트에서는 이전 블록다이어그램에 대한 중요한 상세 정보를 설명합니다.

①	사용 가능한 데이터 이름 파라미터는 마우스 커서 이탈 이벤트에서 끌기 시작 메소드에 전달하는 사용자 데이터 이름을 포함하는 문자열 배열을 제공합니다.

②

사용 가능한 데이터 이름을 검색하여 놓기 타겟이 받아들이기를 원하는 사용자 데이터 이름을 찾습니다. [1D 배열 검색] 함수가 데이터 이름을 찾는 경우, 놓기 타겟이 끌어온 데이터를 받을 수 있습니다.

놓기 타겟이 끌어온 데이터를 받을 수 있는지 확인한 후 **끌어온 데이터로 놓기 타겟을 업데이트**합니다.

단계 6: 끌어온 정보로 놓기 타겟을 업데이트

놓기 타겟을 끌어온 데이터로 업데이트하려면, 놓기 타겟이 사용자 데이터 이름을 받는 지 알아야 합니다. 놓기 타겟이 사용자 데이터 이름을 받는 경우, 저장해놓은 끌어온 데이터를 불러와 놓기 타겟에 제공합니다.

사용 객체

다음 블록 다이어그램 객체를 사용해서 끌어온 데이터를 놓기 타겟에 제공합니다:

1D 배열 검색	같지 않음?	케이스 구조	끌기 놓기 데이터 얻기	값
				

다음 단계

다음 블록 다이어그램 객체를 생성하여 끌어온 데이터로 놓기 타겟을 업데이트합니다.

6. 다른 VI에서 사용할 수 있도록 파일 >> 저장을 선택하여 사용자 컨트롤 또는 인디케이터를 저장합니다. 사용자 컨트롤 또는 인디케이터를 디렉토리나 **LLB**에 저장할 수 있습니다. 다른 VI에서 사용하기 위해 사용자 컨트롤 또는 인디케이터를 저장하지 않으려면, 사용자 컨트롤 또는 인디케이터를 저장하지 않고 컨트롤 편집기 윈도우를 닫습니다.



노트

프론트패널에 사용자 컨트롤 또는 인디케이터를 추가할 때, 사용자 컨트롤은 .ctl 파일을 요구하지 않습니다. 프론트패널에 **타입 정의**를 추가할 때, 타입 정의는 .ctl 파일을 요구합니다.

또한, **편집 메뉴**를 사용하여 사용자 컨트롤 또는 인디케이터를 생성할 수 있습니다.

배열과 클러스터

배열

배열, 행렬 & 클러스터 컨트롤과 인디케이터

배열, 행렬 & 클러스터 및 클래식 배열, 행렬 & 클러스터 팔레트에 위치한 배열, 행렬 및 클러스터 컨트롤과 인디케이터를 사용하여 다른 컨트롤과 인디케이터의 배열 및 클러스터와 실수 또는 복소수 데이터 타입의 행렬을 생성합니다.

이 팔레트는 또한 표준 에러 클러스터 컨트롤과 인디케이터를 포함합니다. 배리언트 컨트롤은 클래식 배열, 행렬 & 클러스터 팔레트에 나타납니다.



관련 정보

클러스터 (1061 페이지 참조)

에러 클러스터 사용하기 (1389 페이지 참조)

배리언트 컨트롤과 인디케이터 (1002 페이지 참조)

배열에 원소 추가하기

다음 단계를 따라 원소를 배열에 추가합니다.

1. 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급 >> 마지막 원소 보이기를 선택합니다. 인덱스 디스플레이는 배열에서 마지막 원소의 인덱스 값으로 변경됩니다.
2. 인덱스 디스플레이에서 증가 버튼을 클릭하여 크기를 증가시키려는 차원의 인덱스 수를 증가시킵니다. 각 차원을 증가시킬 필요는 없습니다. 예를 들어, 2×2×10 배열을 3×3×30으로 증가시키려면, 인덱스에 2, 2, 29를 입력하여 새로운 최종 위치인 (2, 2, 29)로 이동합니다.

3. 값을 새로운 최종 원소에 지정합니다. LabVIEW는 이전 최종 원소와 새로운 최종 원소 사이의 원소를 스칼라 **기본값**으로 채웁니다.

배열을 축소하고 기존 데이터 값의 일부 또는 모두를 보존하려는 경우, **배열 셀을 자르고 복사하고 붙여넣기**하거나 **배열을 비웁니다**.

배열 기본값 변경하기

다음 단계를 따라 배열의 기본값을 조작합니다.

1. 프런트패널 윈도우에 1D 배열 셀을 **생성**합니다.
2. 프런트패널 윈도우의 배열 셀에 디지털 숫자형 컨트롤을 추가합니다.
3. 최소 6개의 원소가 보이도록 배열 셀을 **크기 조정**합니다.
4. 배열의 3번째 원소의 값을 1.0으로 설정합니다. LabVIEW는 처음 2개 원소의 값을 0.0으로 설정합니다. 이것은 배열의 초기 스칼라 기본값입니다.
5. 배열의 3번째 원소에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리>현재값을 기본값으로 선택하여 원래 0.0이었던 3번째 원소의 스칼라 기본값을 1.0으로 설정합니다. 배열 인덱스 디스플레이 또는 배열 셀이 아닌 원소에서 마우스 오른쪽 버튼을 클릭합니다.
6. 배열의 6번째 원소의 값을 2.0으로 설정합니다. LabVIEW는 4번째와 5번째 원소를 배열의 부분으로 만들며 그 값을 단계 5에서 배열의 새 스칼라 기본값으로 설정한 1.0으로 설정합니다. LabVIEW는 처음 두 원소의 값을 변경하지 않습니다. 왜냐하면 스칼라 기본값이 0.0이었을 때 배열에 추가되었기 때문입니다. 값을 입력하여 배열에 원소를 추가할 때, LabVIEW는 현재의 끝 원소와 추가한 원소 사이의 원소의 값을 스칼라 기본값으로 설정합니다.
7. 배열 인덱스 디스플레이 또는 셀 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리>현재값을 기본값으로 선택하여 전체 배열에 대한 기본값을 설정합니다. LabVIEW는 각 원소의 현재값을 원소의 기본값으로 설정합니다. 예를 들어, 2.0은 현재 6번째 원소의 기본값입니다.
8. 배열의 첫번째 원소에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리>기본값으로 다시 초기화를 선택하여 첫번째 원소의 값을 0.0에서 1.0으로 변경합니다. 이것은 전체 배열의 스칼라 기본값입니다.
9. 배열 인덱스 디스플레이 또는 셀 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리>기본값으로 다시 초기화를 선택합니다. 첫번째 원소의 값은 0.0으로 다시 돌아가는 것을 명심하십시오. 이것은 전체 배열의 기본값을 설정할 때 이 원소의 값이었습니다.



노트

스칼라 기본값을 변경하는 경우는 실제로는 드뭅니다. 일반적으로 전체 배열의 기본값을 설정하거나 복구합니다.

배열 차원 변경하기

새 배열은 하나의 차원과 하나의 인덱스 디스플레이를 가집니다. 다음 방법 중 하나를 사용하여 더 많은 차원을 추가합니다:

- 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차원 추가를 선택하여 한 번에 하나의 차원을 추가합니다.
- 위치 도구를 사용하여 인덱스 디스플레이를 원하는 차원이 될 때까지 크기 조정합니다.

다음 방법 중 하나를 사용하여 차원을 제거합니다:

- 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차원 제거를 선택하여 한 번에 하나의 차원을 제거합니다.
- 위치 도구를 사용하여 차원을 제거하기 위해서 인덱스 디스플레이를 크기 조정합니다.

배열 지우기

다음 단계를 따라 배열 컨트롤, 인디케이터, 또는 데이터의 상수를 비웁니다.

1. 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 데이터 처리»빈 배열을 선택합니다.

배열 컨트롤과 인디케이터 생성하기

배열 셀을 위치시키고 숫자, 불리언, 문자열, 경로, 참조 번호, 또는 클러스터 컨트롤 또는 인디케이터와 같은 데이터 객체를 셀로 끌어서 배열 프런트패널 컨트롤이나 인디케이터를 생성합니다. 다른 배열이나 차트는 그 원소가 될 수 없습니다. 원소가 그래프인 경우, 이 데이터 타입은 배열이 될 수 없지만 배열을 포함하는 클러스터는 될 수 있습니다.

다음 단계를 따라 배열 컨트롤 또는 인디케이터를 생성합니다.

1. 프런트패널에 배열 셀을 추가합니다. 배열 셀은 왼쪽에 인덱스 디스플레이를, 오른쪽에 원소 디스플레이를, 옵션으로 라벨을 포함합니다.
2. 컨트롤 팔레트가 보이지 않을 경우, 원소 디스플레이 또는 프런트패널 윈도우에서 마우스 오른쪽 버튼을 클릭하여 컨트롤 팔레트를 디스플레이합니다.
3. 컨트롤 또는 인디케이터를 배열 셀로 끌어옵니다. 예를 들어, 숫자 컨트롤을 배열 셀로 끕니다. 유효하지 않은 컨트롤 또는 인디케이터를 배열 셀로 끌려고 시도할 경우, 배열 셀에 컨트롤 또는 인디케이터를 추가할 수 없습니다. 유효하지 않은 컨트롤 또는 인디케이터의 예는 XY 그래프입니다. 프런트패널 윈도우에서 블록 다이어그램 윈도우로 기존 배열 컨트롤을 복사하거나 끌 수 있습니다.

4. 배열을 크기 조정하여 배열에 원소를 추가합니다.



팁

배열에서 Microsoft Excel과 같은 곳으로 데이터를 반출할 수 있습니다. 배열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반출을 선택하여 사용 가능한 반출 옵션을 봅니다.

배열의 기본 크기와 값

고정된 수의 원소로 배열의 크기를 제한할 수 없습니다. 그러나 배열의 기본값 설정 컨트롤을 사용하면, 기본 크기는 설정할 수 있습니다. 배열 컨트롤을 생성할 때 기본값과 배열 크기를 설정합니다.



노트

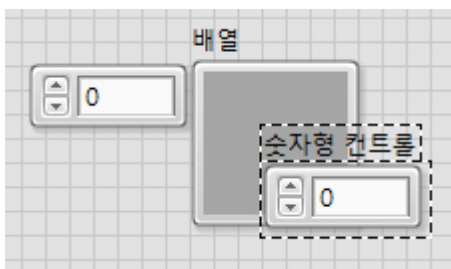
필요 이상 크게 배열의 기본 크기를 만들지 않습니다. 배열의 기본 크기를 크게 설정했다면, 배열의 각 원소를 위한 기본 데이터를 VI에 저장시 디스크에서 VI의 크기가 증가합니다.

VI에서 사용하기 위해 배열 컨트롤 설정하기

데이터 객체가 없는 배열 셀은 정의되어 있지 않으며 어떠한 데이터 타입도 가지고 있지 않습니다. VI는 배열 셀에 컨트롤이나 인디케이터를 놓아 데이터 타입을 할당할 때까지 깨진 상태로 있게 됩니다.

배열 셀에 컨트롤 또는 인디케이터를 놓게 되면, 빈 배열은 컨트롤 또는 인디케이터의 데이터 타입을 갖게 됩니다. 배열에 데이터 타입을 할당한 후에, 배열이 비어 있는 것에 상관없이 VI에서 이 데이터 타입을 사용할 수 있습니다. 빈 배열의 길이는 0이며, 배열 원소는 비활성화됩니다. 빈 배열을 SubVI를 호출하는 최상위 VI에 놓아 배열을 생성할 수 있습니다.

예를 들어, 다음 이미지는 숫자형 데이터 타입을 받게 될 빈 배열 셀을 디스플레이합니다.



배열의 기본값 타입

프론트패널 배열 컨트롤은 다음 2개의 기본값을 갖습니다:

- 정의된 원소에 대한 기본값 - 이 값은 VI가 로드될 때의 배열값이며 다른 프론트패널 컨트롤의 기본 값과 유사합니다. 배열의 원소의 값이 기본값을 구성합니다. 기

본 배열값은 또한 호출 VI가 배열의 값을 지정하지 않은 경우 SubVI에 값을 제공합니다.

- 정의되지 않은 원소에 대한 기본값 - 이 값은 배열을 확장시킬 때 배열을 채우기 위해 사용됩니다. 예를 들면, 배열의 정의된 부분 위에 배열 인덱스를 설정하고 정의되지 않은 원소에 값을 입력하면, 배열의 정의된 부분과 새로 정의한 원소 사이의 원소는 정의되지 않은 원소에 대한 기본값으로 설정됩니다.

배열의 크기 결정하기

다음 단계를 따라 배열의 크기를 결정합니다.

1. 배열 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 고급 >> 마지막 원소 보이기를 선택합니다. 인덱스 디스플레이는 배열의 차원을 보여주므로, 배열의 크기를 알 수 있습니다. 예를 들면, 15행과 15열의 배열을 가지고 있고 위 단계를 수행하면, 인덱스 디스플레이는 위쪽 디스플레이에 14와 아래쪽 디스플레이에 14를 보여줍니다.

또한, **배열의 데이터 크기를 변경**할 수 있습니다.

테이블에 배열 디스플레이하기

1D 배열의 인덱스 디스플레이는 가장 왼쪽에 보이는 원소를 식별합니다.

기본적으로, 새로운 배열은 1D 배열입니다. 원소 디스플레이는 인덱스 디스플레이를 포함하는 단일 원소입니다. 그러나, 배열 셀의 크기조절을 통해 원소의 테이블로서 또한 배열을 디스플레이할 수 있습니다.

다음 단계를 따라 2D 숫자 배열을 생성하고 테이블에 그것을 디스플레이한 후 인덱스 디스플레이를 이용하여 디스플레이를 변경합니다.

1. 프런트패널 윈도우에 배열 셀을 추가합니다. 배열 셀은 왼쪽에 인덱스 디스플레이를, 오른쪽에 원소 디스플레이를, 옵션으로 라벨을 포함합니다.
2. 컨트롤 팔레트가 보이지 않는 경우, 원소 디스플레이에서 마우스 오른쪽 버튼을 클릭하여 컨트롤 팔레트를 디스플레이합니다.
3. 배열 셀에 숫자형 컨트롤을 추가합니다.
4. 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 차원 추가를 선택하여 단일 원소 형태에 2D 숫자 배열을 생성합니다. 단일 원소 배열은 하나의 원소만 디스플레이합니다. 양쪽 인덱스 값을 현재 0으로 유지합니다. 위쪽 인덱스는 행을 참조하고, 아래쪽 인덱스는 열을 참조합니다.
5. 단일 원소 배열에서 원소를 둘러싸고 있는 4개의 코너 중 1개 위로 위치 도구를 움직입니다. 위치 도구로 모양을 변경하면 크기 재조정 핸들이 나타납니다.
6. 위치 도구를 사용하여 배열에 원소를 추가 또는 삭제하기 위해 원소 디스플레이를 크기 조정합니다. 배열을 두 개의 행과 세 개의 열을 가지도록 크기 조정합니다. 또한, 각각의 원소 디스플레이를 크기조정할 수 있습니다. 원소 디스플레이를 크기 조정할 때, 모든 다른 원소는 변경한 것과 똑같이 크기 조정됩니다.

7. 라벨링 도구 또는 수행 도구를 사용하여 배열에서 각 숫자 컨트롤의 값을 지정합니다. 첫번째 행의 값을 1에서 3까지 지정하고, 두번째 행에 4에서 6까지 지정하며, 이후도 이와 같은 방식으로 지정합니다.
8. 인덱스 디스플레이는 0에서 시작합니다. 인덱스 디스플레이를 사용하여 왼쪽 위에 보이는 원소의 좌표를 지정합니다. 인덱스 디스플레이를 조작하여 같은 방법으로 **숫자형 컨트롤**을 수행합니다. 행과 열 인덱스 컨트롤의 증가하면 배열에서 특정한 원소가 어떻게 지정되는지 볼 수 있습니다. 지정된 원소는 항상 왼쪽 위의 원소 디스플레이에 나타납니다.

열의 행으로 2D 배열을 생각합니다. 왼쪽에 있는 두 개 박스의 위쪽 디스플레이는 행 인덱스이고 아래쪽 디스플레이는 열 인덱스입니다. 행과 열의 표시를 합하여 지정된 위치의 값을 보여줍니다.

데이터의 아이템을 위치시키기 위해 요청된 인덱스 숫자는 배열의 차원을 결정합니다.

같은 타입의 컨트롤 또는 인디케이터를 작동하는 것과 같은 방식으로 배열의 원소를 작동합니다.

배열에서 원소 디스플레이하기

배열은 실제로 배열에 있는 것보다 많거나 적은 수의 원소를 디스플레이할 수 있습니다. 배열의 크기를 조정하여 배열이 디스플레이하는 원소의 수를 변경할 수 있습니다.

다음 단계를 따라 배열의 크기를 조정합니다:

1. 커서를 배열 셀 경계 위로 이동합니다. 크기 조정 핸들이 배열의 크기를 조정할 수 있는 포인트에 나타납니다.
2. 커서를 크기 조정 핸들 위로 이동하여 커서를 크기 조정 커서로 변경합니다.
3. 크기 조정 커서를 사용하여 점선 경계가 배열에 디스플레이하려는 원소 수를 둘러쌀 때까지 수직 또는 수평으로 크기 조정 핸들을 끕니다.



노트

배열이 디스플레이하는 원소를 변경하려면 인덱스 디스플레이의 증가와 감소 버튼을 사용합니다. 또한 배열이 디스플레이하는 첫번째 원소의 인덱스를 입력할 수도 있습니다.

크기 조정 작업을 취소하려면, 점선 경계가 사라질 때까지 크기 조정 핸들을 윈도우 밖으로 계속 끌거나 마우스 버튼을 놓기 전에 <Esc> 키를 누릅니다.

인덱스 디스플레이를 보여주거나 숨기기

다음 단계를 따라 인덱스 디스플레이를 보여주거나 숨깁니다.

1. 인덱스를 보이려는 인덱스 디스플레이 또는 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭합니다.

- 배열 바로 가기 메뉴로부터 보이는 아이템»인덱스 디스플레이 아이템을 선택하여 인덱스 디스플레이를 보이거나 숨깁니다.

또한, 보이는 아이템:인덱스 보이기 프로퍼티를 사용하여 프로그램적으로 인덱스 디스플레이를 보이거나 숨길 수 있습니다.

배열의 셀을 선택하고, 잘라내고, 복사하고, 붙여넣기

다음 단계를 따라 배열 인덱스를 사용하여 선택을 위한 셀을 정의합니다.

- 프론트패널에 배열 셀을 추가합니다.
- 배열 셀에 숫자형 컨트롤을 추가합니다.
- 배열의 크기를 조정**하여 4개의 행과 4개의 열을 가집니다.
- 배열을 초기화하기 위해 4, 4 셀에 값을 추가합니다.
- 배열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급»선택 보이기를 선택합니다. 이 옵션을 활성화하여 기존의 선택된 셀을 보여줍니다.
- 배열 셀의 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 원소 갭 추가를 선택하여 얇게 열린 공간에 의해 셀을 분리합니다. 원소 갭을 추가하기에는 필요 없을지라도, 이 열린 경계선은 셀을 선택하기에 더 쉽게 만듭니다. 셀을 선택할 때, 선택한 셀을 나타내는 두꺼운 경계선이 이 열린 공간을 채웁니다.
- 배열 인덱스 디스플레이를 1, 1로 지정합니다. 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급»선택 시작을 선택합니다.
- 배열 인덱스 디스플레이를 3, 3으로 지정합니다. 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급»선택 종료를 선택합니다. 선택은 더 높은 인덱스 번호에 의해 나타난 셀이 아닌 더 낮은 인덱스 번호에 의해 나타나는 셀을 포함합니다. 단계 7과 8은 셀을 1, 1에서 2, 2까지 선택합니다. 1, 1에서 3, 3까지 셀을 포함하려면 배열 인덱스를 4, 4로 설정하고, 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급»선택 종료를 선택합니다.



노트

선택된 셀을 잘라내면 선택된 셀의 전체 행과 열이 모두 제거되지만 오직 선택된 셀만 클립보드에 저장됩니다. 선택에 의해 둘러싸인 행과 열에 있는 정보가 사라집니다.

- 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 데이터 처리»데이터 잘라내기를 선택하여 데이터를 잘라내거나, 데이터 처리»데이터 복사를 선택하여 데이터를 복사합니다.
- 적절한 인덱스 디스플레이 값을 사용하여 단계 7과 8을 반복하여 데이터를 붙여넣으려는 배열에 같은 크기의 영역을 지정합니다.
- 배열 셀 경계선에서 마우스 오른쪽 버튼을 클릭하고 데이터를 붙여 넣도록 바로 가기 메뉴에서 데이터 처리»데이터 붙여넣기를 선택합니다. 선택한 내용을 붙여 넣은 후에, 하나의 경계선이 붙여 넣은 데이터의 주변에 하이라이트되어 있다는 것을 확인하십시오. 이것은 삽입 포인트이고 배열에서 유지됩니다. 이 라인을

숨기기 위해, 배열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급
 >>선택 보이기를 선택하여 이 메뉴 아이템에서 확인 표시를 제거합니다.

다음 단계를 따라 선택한 셀의 표시를 없앱니다.

1. 인덱스 디스플레이를 모든 차원에서 0으로 설정합니다.
2. 인덱스 디스플레이에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급
 >>선택 시작을 선택합니다.
3. 인덱스 디스플레이에서 다시 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서
 고급>>선택 종료를 선택합니다.

실행 모드에 있고, 배열에서 마우스 오른쪽 버튼을 클릭할 때 이 메뉴 아이템에 접근할 수 있습니다.



노트 실행 모드에서 인디케이터의 데이터를 **다시 초기화하거나 잘라내기, 붙여넣기**할 수 없습니다. 기본값으로 다시 초기화, 데이터 잘라내기, 데이터 붙여넣기 바로 가기 옵션은 실행 모드의 컨트롤에서만 사용 가능합니다.

배열 기본값 설정하기

LabVIEW 배열에는 현재값을 기본값으로 및 기본값으로 다시 초기화라고 하는 바로 가기 메뉴 아이템이 있습니다. 이 아이템에는 민감한 내용이 있습니다. 두 개의 다른 환경에서 이들을 사용할 수 있는데, 그것은 전체 배열에 각 원소의 기본값을 셋팅하거나 하나의 배열 원소에 기본값을 셋팅하기입니다.

다음 단계를 따라 배열의 기본값을 설정합니다.

1. 프런트패널에 배열 셀을 추가합니다.
2. 인덱스 디스플레이나 배열 셀 경계에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 데이터 처리>>현재값을 기본값으로 선택하여 현재 각 원소에 가지고 있는 값을 배열의 기본값으로 설정합니다.
4. 바로 가기 메뉴에서 데이터 처리>>기본값으로 다시 초기화를 선택하고 배열의 기본값을 원래 기본값으로 설정합니다.

다음 단계를 따라 배열 원소를 위한 기본값을 설정합니다.

1. 프런트패널에 배열 셀을 추가합니다.
2. 배열의 원소에 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리
 >>현재값을 기본값으로 또는 데이터 처리>>기본값으로 다시 초기화를 선택합니다. 이것은 원소의 값을 배열이 확장될 때 채우기 위해 사용되는 스칼라 값으로 설정합니다.

배열 상수 생성하기

배열을 숫자, 불리언, 문자열, 경로, 참조 번호, 또는 클러스터 상수와 같은 유효한 상수와 결합하여 블록다이어그램에 배열 상수를 생성합니다. 다른 배열은 원소가 될 수 없습니다.

1. 블록다이어그램에 배열 상수를 추가합니다. 왼쪽에 인덱스 디스플레이, 오른쪽에 빈 원소 디스플레이, 그리고 라벨 옵션이 있는 배열 셸이 나타납니다.
2. 배열 셸에 상수를 추가합니다. 예를 들어, 배열 셸에 숫자형 상수를 놓을 수 있습니다. 프런트패널 윈도우에 있는 기존의 배열을 블록다이어그램 윈도우로 복사하거나 끌어와서 같은 데이터 타입의 상수를 생성할 수 있습니다. 배열 셸은 자동적으로 배열 셸 안에 위치시킨 객체에 따라 크기 조정됩니다. 배열 셸 안에 객체를 놓을 때, 배열 상수의 데이터 타입을 정의합니다.

배열의 기본 데이터

기본값은 배열이 확장할 때 배열을 채우는데 사용되는 값입니다. 예를 들면, 배열 인덱스를 배열의 정의된 부분 밖의 값으로 설정하고 원소에 배열의 정의된 부분을 지난 한 행보다 큰 값을 입력하는 경우, 배열의 이전 끝과 추가한 원소 사이의 원소는 기본값으로 설정됩니다.

[배열 크기] 함수를 사용하여 배열의 크기를 결정할 수 있습니다. While 루프를 사용하며 마지막 원소를 지나 배열을 인덱스하거나, [배열 인덱스] 함수의 인덱스 입력에 너무 큰 값을 입력하거나, 배열 인덱스 함수에 빈 배열을 제공하여 의도치 않게 배열의 경계를 넘어 인덱스할 수 있습니다.

배열 내의 원소, 행, 열 삭제하기

1D 배열 내의 원소와 2D 배열 내의 행 또는 열을 삭제할 수 있습니다. 1D 배열의 원소를 삭제하려면, 프런트패널로 전환하고, 배열 원소에서 마우스 오른쪽 버튼을 클릭하여 바로 가기 메뉴에서 데이터 처리 >> 원소 삭제를 선택합니다. 2D 배열의 행 또는 열을 삭제하려면, 배열의 행 또는 열에서 마우스 오른쪽 버튼을 클릭하고 데이터 처리 >> 행 삭제 또는 열 삭제를 선택합니다.

또한, 배열에서 **원소, 행, 열, 페이지**를 **프로그램적으로 삭제**할 수 있습니다.

배열 내에 원소, 행, 열을 삽입하기

1D 배열에 원소를 삽입하고 2D 배열에 행이나 열을 삽입할 수 있습니다. 1D 배열에 원소를 추가하기 위해, 프런트패널로 전환하여, 배열에 마우스 오른쪽 버튼을 클릭하고 데이터 처리 >> 앞에 원소 삽입을 선택합니다. 2D 배열에 행이나 열을 추가하려면, 배열에서 마우스 오른쪽 버튼을 클릭하고 데이터 처리 >> 앞에 행 삽입이나 앞에 열 삽입을 선택합니다.

또한, 배열에 **원소, 행, 열, 페이지**를 **프로그램적으로 삽입**할 수 있습니다.

배열 내의 원소, 행, 열, 페이지를 프로그램적으로 삭제하기

배열 내의 원소, 행, 열, 페이지를 프로그램적으로 삭제할 수 있습니다. 무엇을 삭제할 수 있는지는 배열이 얼마나 많은 차원을 가지고 있는가에 따라서 달라집니다. 예를 들면, 2차원 이상의 배열로부터 행이나 열을 삭제할 수 있습니다. 3차원 이상의 배열로부터 페이지를 삭제할 수 있습니다.

다음 단계를 따라 배열의 원소, 행, 열, 또는 페이지를 대체합니다.

1. 블록 다이어그램에 [배열로부터 삭제] 함수를 추가합니다.
2. 임의의 차원의 [배열로부터 삭제] 함수의 n 차원 배열 입력에 연결합니다. 함수는 배열의 차원에 기초해 자동으로 크기를 조정합니다.
3. 다음 테이블에서 어떤 작업을 수행하려는지 결정하고 관련된 단계를 완성합니다.



노트
인덱스 입력은 0을 처음으로 어떤 원소, 행, 열, 또는 페이지에서 삭제를 시작할 지 지정합니다. 길이 입력은 삭제하려는 원소, 행, 열, 또는 페이지의 개수를 지정합니다. 다음 테이블에서 n 은 배열에서 원소, 열, 행, 페이지 또는 차원의 총 개수입니다.

n 차원 배열에 연결된 배열	삭제하기	이 단계를 마칩니다.
1D 배열	원소	1부터 n 까지의 값을 길이에 연결합니다. 0부터 n 까지의 값을 인덱스에 연결합니다.
2D 배열	행	1부터 n 까지의 값을 길이에 연결합니다. 0부터 n 까지의 값을 인덱스(행)에 연결합니다.
열	1부터 n 까지의 값을 길이에 연결합니다. 0부터 n 까지의 값을 인덱스(열)에 연결합니다.	
3D ~ n D 배열	페이지	1부터 n 까지의 값을 길이에 연결합니다. 0부터 n 까지의 값을 인덱스(페이지)에 연결합니다.

1. VI를 실행합니다.

배열 내에 원소, 행, 열, 페이지를 프로그램적으로 삽입하기

배열 내에 원소, 행, 열, 페이지를 프로그램적으로 삽입할 수 있습니다. 무엇을 삽입할 수 있는지는 배열이 얼마나 많은 차원을 가지고 있는지에 따라 달라집니다. 예를 들면, 2차원 이상의 배열 내에 행이나 열을 삽입할 수 있지만, 원소를 삽입할 수는 없습니다. 3차원 이상의 배열 내에 페이지를 삽입할 수 있지만, 원소, 행, 열을 삽입할 수는 없습니다.

다음 단계를 따라 배열의 원소, 행, 열, 또는 페이지를 삽입합니다.

1. 블록 다이어그램에 [배열에 삽입] 함수를 추가합니다.
2. 임의의 차원을 가진 배열을 [배열에 삽입] 함수의 n 차원 배열 입력에 연결합니다. 함수는 배열의 차원에 기초해 자동으로 크기를 조정합니다.
3. 다음 테이블에서 어떤 작업을 수행하려는지 결정하고 관련된 단계를 완성합니다.



노트

인덱스 입력은 0을 처음으로 원소나 배열을 삽입할 원소, 행, 열 또는 페이지를 지정합니다. 원소와 배열은 인덱스에 연결하는 값 앞에 추가됩니다. 다음 테이블에서 n 은 배열에서 원소, 열, 행, 페이지 또는 차원의 총 개수입니다.

n 차원 배열에 연결된 배열	삽입하기	이 단계를 마칩니다.
1D 배열	원소	0부터 n 까지의 값을 인덱스에 연결합니다. 값을 새 원소/부분배열에 연결합니다.
행이나 열에 대한 1D 배열	0부터 n 까지의 값을 인덱스에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.	
2D 배열	행	0부터 n 까지의 값을 인덱스(행)에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.
열	0부터 n 까지의 값을 인덱스(열)에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.	
2D 배열(행과 열)	0부터 n 까지의 값을 인덱스(행) 또는 인덱스(열)에 연결합니다. 2D 배열을 새 원소/부분배열에 연결합니다.	
3D ~ n D 배열	페이지	0부터 n 까지의 값을 인덱스(페이지)에 연결합니다. 2D 배열을 새 원소/부분배열에 연결합니다.
3D 배열(페이지)	0부터 n 까지의 값을 인덱스(페이지)에 연결합니다. 3D 배열을 새 원소/부분배열에 연결합니다.	

1. [배열에 삽입] 함수의 **크기를 조정**하여 배열 내에 다른 원소, 행, 열, 또는 페이지를 삽입하고 2와 3 단계를 반복합니다.
2. VI를 실행합니다.

삽입한 배열의 차원 크기가 [배열에 삽입] 함수의 n 차원 배열 입력에 연결된 배열의 차원 크기보다 더 작으면, 이 함수는 숫자값 배열의 경우에 제로와 같이 기본 데이터를 삽입하는 배열로 채웁니다.

삽입한 배열의 차원 크기는 [배열에 삽입] 함수의 n 차원 배열 입력에 연결된 배열의 차원 크기보다 더 크면, 함수는 끝에서부터 배열을 잘라냅니다.

배열 내의 원소, 행, 열, 페이지 대체하기

배열의 원소, 행, 열, 또는 페이지를 대체할 수 있습니다. 무엇을 대체할 수 있는지는 배열이 얼마나 많은 차원을 가지는지에 따라 달라집니다. 예를 들어, 2 또는 그 이상 차원의 배열에서, 행 또는 열을 1D 배열로 대체할 수 있습니다. 3 또는 그 이상 차원의 배열에서, 페이지를 2D 배열로 대체할 수 있습니다.

다음 단계를 따라 배열의 원소, 행, 열, 또는 페이지를 대체합니다.

- 1. 블록 다이어그램에 [배열 부분 대체] 함수를 추가합니다.
- 2. 임의의 차원의 배열을 [배열 부분 대체] 함수의 n 차원 배열 입력에 연결합니다. 함수는 배열의 차원에 기초해 자동으로 크기를 조정합니다.
- 3. 다음 테이블에서 어떤 작업을 수행하려는 지 결정하고 관련된 단계를 완성합니다.



노트
인덱스 입력은 0을 처음으로 어떤 원소, 행, 열, 또는 페이지를 대체할 지 지정합니다. 새 원소/부분배열 입력은 원소를 대체하려는 값 또는 행, 열, 또는 페이지를 대체하려는 배열을 지정합니다.

n차원 배열에 연결된 배열	대체하기	이 단계를 마칩니다.
1D 배열	원소	0부터 n 까지의 값을 인덱스에 연결합니다. 값을 새 원소/부분배열에 연결합니다.
2D 배열	행	0부터 n 까지의 값을 인덱스(행)에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.
열	0부터 n 까지의 값을 인덱스(열)에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.	
원소	0부터 n 까지의 값을 인덱스(행)에 연결합니다. 0부터 n 까지의 값을 인덱스(열)에 연결합니다. 값을 새 원소/부분배열에 연결합니다.	

n차원 배열에 연결된 배열	대체하기	이 단계를 마칩니다.
3D ~ nD 배열	페이지	0부터 n까지의 값을 인덱스(페이지)에 연결합니다. 2D 배열을 새 원소/부분배열에 연결합니다.
행	0부터 n까지의 값을 인덱스(페이지)에 연결합니다. 0부터 n까지의 값을 인덱스(행)에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.	
열	0부터 n까지의 값을 인덱스(페이지)에 연결합니다. 0부터 n까지의 값을 인덱스(열)에 연결합니다. 1D 배열을 새 원소/부분배열에 연결합니다.	
원소	0부터 n까지의 값을 인덱스(페이지)에 연결합니다. 0부터 n까지의 값을 인덱스(행)에 연결합니다. 0부터 n까지의 값을 인덱스(열)에 연결합니다. 값을 새 원소/부분배열에 연결합니다.	

1. [배열 부분 대체] 함수의 크기를 조정하여 배열 내의 다른 원소, 행, 열, 또는 페이지를 대체하고 2와 3 단계를 반복합니다.
2. VI를 실행합니다.

클러스터

클러스터

클러스터는 여러 타입의 데이터 원소를 하나의 그룹으로 묶습니다. 클러스터의 예는 불리언 값, 숫자값, 문자열이 결합된 LabVIEW 에러 클러스터를 들 수 있습니다. 클러스터는 텍스트 기반 프로그래밍 언어의 레코드(record) 또는 구조(struct)와 유사합니다.



관련 정보

[원소를 클러스터로 묶기](#) (1064 페이지 참조)

[클러스터를 배열로 변환하기](#) (1067 페이지 참조)

[클러스터 상수 생성하기](#) (1064 페이지 참조)

[클러스터 원소 대체하기](#) (1065 페이지 참조)

[클러스터에서 원소 풀기](#) (1065 페이지 참조)

클러스터 컨트롤을 인디케이터로 클러스터 인디케이터를 컨트롤로 변경하기

원소 디스플레이에 놓는 처음 원소가 클러스터가 컨트롤이 될지 인디케이터가 될지를 결정합니다. 다음번에 추가하는 원소는 첫번째 원소와 일치하도록 변환됩니다. 클러스터는 컨트롤과 인디케이터를 포함할 수 있지만, 둘을 결합할 수는 없습니다. 컨트롤과 인디케이터를 혼합하려고 시도할 경우, LabVIEW는 클러스터의 첫번째 원소에 일치하도록 변환합니다.

다음 단계를 따라 클러스터 원소를 컨트롤에서 인디케이터로, 또 그 반대로 변경합니다.

1. 임의의 클러스터 원소에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 컨트롤로 변경 또는 인디케이터로 변경을 선택하여 클러스터와 모든 원소를 인디케이터에서 컨트롤로 또는 컨트롤에서 인디케이터로 변경합니다.

클러스터 컨트롤과 인디케이터 생성하기

블리언, 문자열, 그래프, 배열, 숫자형 등과 같은 유효한 데이터 객체 또는 원소와 클러스터 셀과 결합하여 프런트패널에 클러스터 컨트롤 또는 인디케이터를 생성합니다.

다음 단계를 따라 클러스터 컨트롤 또는 인디케이터를 생성합니다.

1. 프런트패널 윈도우에 **클러스터** 셀을 추가합니다.
2. 컨트롤 팔레트가 보이지 않을 경우, 원소 디스플레이 또는 프런트패널 윈도우에서 마우스 오른쪽 버튼을 클릭하여 컨트롤 팔레트를 디스플레이합니다.
3. 컨트롤 또는 인디케이터를 클러스터 셀로 끌어옵니다. 예를 들어, 숫자 컨트롤을 클러스터 셀로 끌어옵니다.

또한, 블록 다이어그램 윈도우에서 기존의 **클러스터 상수**를 프런트패널 윈도우로 복사하거나 끌어서 클러스터 컨트롤을 생성할 수 있습니다.

클러스터 원소 순서 변경하기

클러스터 원소는 셀에서의 그 위치에 관계 없이 논리적인 순서를 가지고 있습니다. 클러스터에 첫 번째로 놓인 객체는 원소 0이고, 두 번째 객체는 원소 1이 되는 방식으로 순서가 정해집니다. 원소를 삭제하면 순서가 자동으로 조절됩니다. 블록 다이어그램에 있는 [뭉치] 및 [풀기] 함수에서 원소가 터미널로 표시되는 순서는 클러스터 순서에 의해 결정됩니다.

다음 단계를 따라 클러스터 순서를 보고 변경합니다.

1. 클러스터 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 클러스터 내의 컨트롤 순서 재설정을 선택합니다. 도구 모음과 클러스터가 다음 변경을 디스플레이합니다:

- 클러스터는 각 원소에서 숫자가 있는 일련의 흑백 상자를 보여줍니다. 원소의 흰 상자는 클러스터 순서에서의 현재 위치를 나타냅니다.
 - 검은 상자의 숫자는 흰 상자의 숫자와 일치합니다. 순서를 변경하면, 검은 상자는 원소에 대한 순서에서 새로운 위치를 보여줍니다.
 - 또한, 도구 모음은 검은 상자에서 숫자를 디스플레이합니다. 도구 모음에서 숫자는 변경하는 다음 원소에 영향을 줍니다.
 - 커서는 클러스터 순서 커서로 변경됩니다.
2. 다음의 방법을 사용하여 순서를 변경합니다:
 - 클러스터 순서에서 원소를 클릭합니다. 검은 상자에서 원소의 숫자는 도구 모음의 숫자와 일치하도록 바뀝니다.
 - 클릭하여 설정합니다 필드에서 새로운 순서를 위한 숫자를 타입하고 원소를 클릭합니다.
 - 설정하려는 순서로 원소를 클릭하고 순서 숫자를 자동으로 증가시킵니다.
 3. 클러스터 순서를 설정하기 위해 도구 모음의 확인 버튼을 클릭한 후 클러스터 순서 편집 모드를 끝냅니다. 취소 버튼을 클릭하여 원래 순서로 되돌아갑니다.

클러스터 크기 조정하기

다음 단계를 따라 내용에 맞게 클러스터의 크기를 줄입니다.

1. 클러스터 셀 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 자동 크기 조정»크기에 맞게 조정을 선택합니다.
2. 다른 원소를 클러스터 셀에 추가합니다. 자동 크기 조정»크기에 맞게 조정이 선택되면, 클러스터 셀에 다른 원소를 추가할 때 필요한 경우 클러스터는 스스로 크기를 조정합니다.
3. 클러스터 셀 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 자동 크기 조정»수평으로 배치 또는 자동 크기 조정»수직으로 배치를 선택하여 클러스터의 원소를 수평이나 수직으로 정렬합니다.

클러스터 기본값 설정하기

다음 단계를 따라 전체 클러스터의 기본값을 각 개별 원소의 현재값으로 설정합니다.

1. 프런트패널 윈도우의 클러스터 셀 경계에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 데이터 처리»현재값을 기본값으로를 선택합니다.

다음 단계를 따라 모든 원소를 그 개별적인 기본값으로 다시 설정합니다.

1. 프런트패널 윈도우의 클러스터 셀 경계에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 데이터 처리»기본값으로 다시 초기화를 선택합니다.

다음 단계를 따라 클러스터에서 하나의 원소의 기본값을 변경합니다.

1. 프런트패널 윈도우에서, 기본값을 설정하려는 원소를 선택합니다.
2. 원소에 대해 새로운 기본값을 입력합니다.
3. 원소에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 데이터 처리»현재값을 기본값으로 선택합니다.

원소를 클러스터로 묶기

원소를 클러스터로 묶어 프로그램에서 클러스터를 생성합니다.

다음 단계를 따라 원소를 클러스터로 묶습니다.

1. [묶기] 함수를 블록다이어그램에 추가합니다.
2. 필요한 경우, 클러스터에 원소로 사용하려는 입력의 개수를 포함하도록 [묶기] 함수를 **크기 조정**합니다. 입력을 연결하지 않은 채로 둘 수 없습니다.
3. 프런트패널 컨트롤 터미널 또는 VI로부터의 출력 및 함수를 [묶기] 함수의 원소 입력에 연결합니다. 입력한 연결한 순서는 클러스터의 원소 순서를 결정합니다.
4. 출력 클러스터 터미널에서 마우스 오른쪽 버튼을 클릭한 후 생성»인디케이터를 선택합니다.

LabVIEW는 출력 클러스터 출력에 묶인 클러스터를 반환합니다.



팁

[이름으로 묶기] 함수는 판독하기 쉬운 블록다이어그램을 생성하는데 효과적인 도구입니다.

클러스터 상수 생성하기

클러스터를 배열, 숫자, 불리언, 문자열, 경로, 참조 번호, 또는 클러스터 상수와 같이 유효한 상수와 결합하여 블록다이어그램에 클러스터 상수를 생성합니다.

1. 블록다이어그램에 클러스터 상수를 추가합니다.
2. 클러스터 셀에 상수를 추가합니다. 예를 들어, 클러스터 셀에 숫자형 상수를 추가할 수 있습니다.

객체를 상수 셀에 추가할 때, 사용자가 클러스터 상수의 데이터 타입을 정의합니다.

또한, 프런트패널 윈도우의 기존 **클러스터 컨트롤**을 블록다이어그램에 복사하거나 끌어어서 동일한 원소의 데이터 타입을 갖는 상수를 쉽게 생성할 수 있습니다.



노트

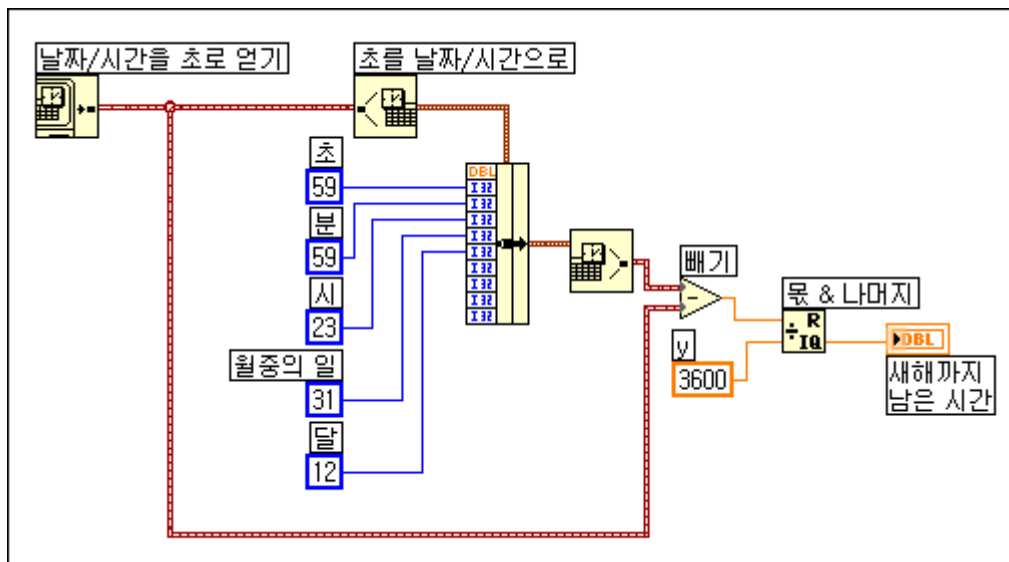
블록다이어그램에 있는 클러스터 상수의 크기를 줄이려면 클러스터 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 클러스터를 아이콘으로 보기를 선택합니다. 이 변경을 실행 취소하려면 바로 가기 메뉴에서 클러스터를 아이콘으로 보기를 다시 선택합니다. 또한 클러스터 상수의 경계를 더블 클릭하여 아이콘으로 볼 수도 있습니다. 이 변경을 실행 취소하려면 클러스터 상수 아이콘을 더블 클릭하십시오.

클러스터 원소 대체하기

다음 단계를 따라 다른 원소에 영향을 미치지 않고 클러스터의 하나 또는 두 원소의 값을 대체하거나 변경합니다.

1. [뮬기] 함수를 블록다이어그램에 추가합니다.
2. 기존의 클러스터를 [뮬기] 함수의 클러스터 입력에 연결합니다. [뮬기] 함수가 확장하여 연결한 클러스터의 원소를 디스플레이합니다.
3. 대체 값을 [뮬기] 함수에 연결합니다.

다음 블록다이어그램은 날짜-시간 클러스터를 사용하여 1월 1일까지의 시간 수를 계산합니다. [뮬기] 함수의 중간 터미널이 연결되어 있으므로, 대체 값을 가진 원소 입력 터미널만이 연결되어야 합니다. 연결되지 않은 원소의 값은 변경되지 않습니다.



클러스터에서 원소 풀기

클러스터에서 원소 풀기는 클러스터 원소 순서대로 클러스터의 모든 원소에 접근하고 배치합니다. 이 원소를 풀려면, 먼저 클러스터를 채워야 합니다. 빈 클러스터 컨트롤을 [풀기] 함수에 연결한 경우, 연결한 클러스터를 채울 때까지 와이어는 유효하지 않습니다.

다. 클러스터에서 원소를 푼 후, 각 원소를 VI, 함수, 인디케이터에 연결할 수 있습니다. 이 클러스터 풀기 방법은 클러스터의 모든 원소에 접근해야 할 경우 유용합니다. 또한, **이름으로** 클러스터에서 모든 원소를 풀 수 있습니다.

다음 단계를 따라 클러스터에서 원소를 풉니다.

1. [풀기] 함수를 블록다이어그램에 추가합니다.
2. 채운 클러스터를 [풀기] 함수에 연결합니다. 각 원소의 데이터 타입 표현이 원소 출력으로 나타납니다.



팁

[이름으로 풀기] 함수는 자체적으로 기술하는 블록다이어그램을 생성하는데 효과적인 도구입니다.

이름으로 클러스터에서 원소 풀기

이름으로 클러스터에서 원소 풀기는 클러스터 원소 순서대로 이름으로 클러스터의 원소에 접근하고 배치합니다. 이름으로 클러스터 원소를 풀려면, 원소가 라벨되어 있어야 합니다. 이름으로 클러스터에서 원소를 푼 후, 원소를 VI, 함수 또는 인디케이터에 연결할 수 있습니다. 이 클러스터 풀기 방법은 같은 데이터 타입의 원소를 포함하는 클러스터에서 한 원소에 접근해야 할 경우 유용합니다. 또한, **이름을 사용하지 않고** 클러스터에서 모든 원소를 풀 수 있습니다.

다음 단계를 따라 이름으로 클러스터에서 원소를 풉니다.

1. 블록다이어그램에 [이름으로 풀기] 함수를 추가합니다.
2. 클러스터를 [이름으로 풀기] 함수에 연결합니다. 클러스터 원소 순서에서 첫번째 원소가 원소 출력에 나타납니다.
3. 다음 방법 중 하나의 방법으로 함수에서 개별 원소를 선택합니다:
 - 원하는 클러스터의 이름이 나타날 때까지 **함수의 크기를 조정**합니다.
 - 원소 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 아이템 선택을 선택한 후, 원하는 원소를 선택합니다.
 - 수행도구를 사용하여 원소 출력을 클릭하고 바로 가기 메뉴에서 원하는 원소를 선택하십시오.



노트

하위 클러스터에서 [이름으로 풀기] 함수를 사용하면, 함수에서 마우스 오른쪽 버튼을 클릭하고 전체 이름 보이기를 선택하여 원소 이름과 하위 클러스터의 원소에 대한 상위 클러스터 이름을 디스플레이할 수 있습니다. 하위 클러스터의 원소 이름이 같거나 비슷할 때 유용합니다.

배열을 클러스터로 변환하기

배열을 클러스터로 변환하고 [클러스터] 함수를 사용하여 배열의 내용을 디스플레이할 수 있습니다.

다음 단계를 따라 배열을 클러스터로 변환합니다.

1. 숫자 컨트롤의 1D 배열을 생성하고 5개의 원소를 추가합니다. 각 원소에 제로가 아닌 값을 지정합니다.
2. 블록다이어그램에 [배열을 클러스터로] 함수를 추가합니다.
3. 배열을 [배열을 클러스터로] 함수에 연결합니다.
4. [배열을 클러스터로] 함수를 더블 클릭하여 클러스터 크기 대화 상자를 엽니다.
 1. 값을 5로 변경하여 클러스터 크기가 배열의 크기와 일치하도록 합니다.
 2. 확인을 클릭합니다.
5. 배열을 클러스터로 노드의 오른쪽 측면에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»인디케이터를 선택하여 클러스터 인디케이터를 생성합니다.
6. 프런트패널에서 VI를 실행합니다. 클러스터 인디케이터는 배열의 내용을 디스플레이합니다.
7. 배열의 값을 변경한 후 다시 VI를 실행합니다. 클러스터 인디케이터는 새 값을 디스플레이합니다. 인덱스 디스플레이를 변경하여 큰 배열 또는 다차원을 가진 배열의 다른 섹션을 봅니다. 배열에서 원소의 값을 설정하려면, 인덱스 디스플레이로 원소를 볼 수 있도록 한 후 원소의 값을 설정합니다.

또한, **클러스터를 배열로 변환**할 수도 있습니다.

클러스터를 배열로 변환하기

같은 타입의 원소를 가지는 클러스터를 배열로 변환하고 배열 함수를 사용하여 내용을 조작할 수 있습니다.

다음 단계를 따라 클러스터를 배열로 변환합니다.

1. 프런트패널에 **클러스터**를 추가합니다.
2. 블록다이어그램에 [클러스터를 배열로] 함수를 추가합니다.
3. 클러스터를 [클러스터를 배열로] 함수에 연결합니다.
4. [클러스터를 배열로] 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»인디케이터를 선택하여 배열 인디케이터를 생성합니다.
5. 프런트패널에서 VI를 실행합니다. 배열 인디케이터는 클러스터의 값을 디스플레이합니다.

또한, **배열을 클러스터로 변환**할 수도 있습니다.

배열과 클러스터 움직이기

다음 단계를 따라 배열이나 클러스터를 움직입니다.

1. 원소 디스플레이만이 아닌 전체 배열이나 클러스터를 선택하기 위해, 위치 도구를 사용하여 배열 셀 경계, 인덱스 디스플레이, 클러스터 셀 경계를 클릭합니다.
2. 새 위치로 배열이나 클러스터를 끕니다.



노트

또한, 배열이나 클러스터를 가로질러 대각선으로 위치 도구를 끌어서 배열이나 클러스터를 선택할 수 있습니다. 이런 식으로 배열이나 클러스터를 선택한 후에, 위치 도구를 사용하여 그것을 움직일 수 있습니다.

전체 배열이나 클러스터 대신에 배열이나 클러스터의 원소만을 움직이려고 할 때, 배열 셀/클러스터 셀과 그 원소는 함께 묶여 있지 않기 때문에 원소는 배열이나 클러스터로부터 분리됩니다. 편집» 실행 취소를 선택하여 그 동작을 취소하거나 마우스 버튼을 놓기 전에 <Esc> 키를 누릅니다.

배열이나 클러스터의 원소 사이의 탭 이동하기

<Tab> 키를 사용하여 VI를 실행할 때 하나의 배열이나 클러스터 안에서 원소를 이동시키고 할 때 다음의 단계를 완료합니다.

1. 프런트패널을 디스플레이합니다. VI가 실행되는 동안, 배열과 클러스터가 하이라이트될 때까지 <Tab> 키를 누릅니다.
2. <Ctrl> 키와 아래 방향 화살표 키를 눌러서 배열이나 클러스터 안에서 키 포커스를 이동합니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.
3. <Tab> 키를 눌러서 배열이나 클러스터 원소와 배열 인덱스를 통해 순차적으로 이동합니다.
4. 프런트패널 컨트롤 사이의 탭 이동을 반환하기 위해 <Ctrl> 키와 위 방향 화살표 키를 누릅니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

행렬

행렬

행렬은 선형 대수 연산과 같은 일부 수학 연산에서 실수 또는 복소수 스칼라 데이터의 행이나 열을 그룹화합니다. 실수 행렬은 배정도 원소를 포함하며 복소수 행렬은 배정도 구성요소를 가진 복소수 원소를 포함합니다. 행렬은 두 차원만 가질 수 있습니다. 행렬의 배열은 생성할 수 없습니다. [묵기] 함수를 사용하여 둘 이상의 행렬을 결합하여 클러스터를 생성합니다. 행렬은 배열과 같은 제한을 갖습니다.



관련 정보

[배열을 행렬로 변환하기](#) (1069 페이지 참조)

[행렬을 배열로 변환하기](#) (1070 페이지 참조)

[배열과 클러스터를 이용한 데이터의 그룹화](#) (417 페이지 참조)

행렬 컨트롤, 인디케이터, 상수 생성하기

행렬을 가진 프런트패널 컨트롤, 인디케이터, 행렬이 있는 상수를 사용하여 선형 대수 연산과 같은 일부 수학 연산에서 실수 또는 복소수 스칼라 데이터의 행이나 열을 그룹화합니다.

다음 단계를 따라 행렬 컨트롤, 인디케이터 또는 상수를 생성합니다.

1. 다음 컨트롤 중에서 하나를 프런트패널 윈도우에 추가합니다:
 - 실수 행렬
 - 복소수 행렬



노트

프런트패널 실수 행렬 컨트롤 또는 복소수 행렬을 생성할 때, LabVIEW는 컨트롤의 데이터 타입을 설정하여 선택한 행렬 컨트롤에 맞춥니다. 행렬 컨트롤을 대체하여 행렬의 타입을 변경할 수 있습니다.

1. (옵션) 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 인디케이터로 변경을 선택하여 행렬 컨트롤을 행렬 인디케이터로 변환할 수 있습니다. 행렬 컨트롤과 인디케이터는 배열 컨트롤과 인디케이터와 같이 작동합니다.



노트 기본적으로, 행렬 컨트롤과 인디케이터는 하나 이상의 원소를 보여주고 두 차원 모두에 대해 **스칼라 막대를 디스플레이**합니다. 원소 타입은 행렬 컨트롤로 설정되므로, 행렬 컨트롤을 대체해야만 행렬 타입을 변경할 수 있습니다.

1. (옵션) 또한 행렬 상수를 생성할 수도 있습니다. 블록 다이어그램에서 행렬 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 상수로 변경을 선택합니다.

배열을 행렬로 변환하기

배정도 또는 복소수의 1D 또는 2D 배열을 **행렬**로 변환하고 [배열] 함수를 사용하여 행렬 값을 수정할 수 있습니다. 다음 단계를 따라 배열을 행렬로 변환합니다.

1. 부동소수의 1D 또는 2D 배열을 프런트패널에 추가합니다.
2. 블록 다이어그램에 [배열을 행렬로] 함수를 추가합니다.
3. 배열을 [배열을 행렬로] 함수에 연결합니다.
4. [배열을 행렬로] 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 인디케이터를 선택하여 행렬 인디케이터를 생성합니다.
5. VI를 실행합니다. 행렬 인디케이터는 원래 배열의 내용을 디스플레이합니다.

1D 배열을 [배열을 행렬로] 함수에 연결하는 경우, 함수는 배열 원소를 행렬의 첫번째 열에 복사합니다. 함수는 복소수 원소를 포함하는 배열을 복소수 행렬로 변환하고 모든 다

큰 배열은 실수 행렬로 변환합니다. LabVIEW는 행렬 원소를 배정도 숫자 값으로 저장합니다. 또한, **행렬을 배열로 변환**할 수도 있습니다.

행렬을 배열로 변환하기

행렬을 2D 배열로 변환하고 [배열] 함수를 사용하여 행렬 값을 수정할 수 있습니다. 다음 단계를 따라 행렬을 2D 배열로 변환합니다.

1. 프런트패널에 실수 행렬 또는 복소수 행렬 컨트롤을 추가합니다.
 - 실수 행렬
 - 복소수 행렬
2. 블록 다이어그램에 [행렬을 배열로] 함수를 추가합니다.
3. 행렬을 [행렬을 배열로] 함수에 연결합니다.
4. [행렬을 배열로] 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 2D 배열 인디케이터를 생성합니다.
5. 프런트패널에서 VI를 실행합니다. 배열 인디케이터는 배열의 내용을 디스플레이합니다.

단일 행 또는 열의 행렬을 1D 배열에 저장할 수 있습니다. 그러나, LabVIEW는 실행될 때까지 행렬의 차원을 결정할 수 없습니다. [행렬을 배열로] 함수가 반환한 2D 배열은 [행렬을 배열로] 함수에 연결한 행렬과 같은 실수 또는 복소수 원소 타입을 가집니다. 또한, **배열을 행렬로 변환**할 수도 있습니다.

수식 노드, 식 노드 및 스크립트 노드

수식 노드와 식 노드 함수

다음 리스트에는 수식 노드와 식 노드 함수의 이름과 그 설명이 포함되어 있습니다. 2진 함수를 제외한 나머지 함수는 수식 분석 VI 및 [비선형 커브 피팅] VI와 함께 사용할 수도 있습니다. 모든 함수의 이름은 반드시 소문자여야 합니다. 또한, [수식 노드]와 [식 노드]에서 **연산자**와 pi 상수는 사용할 수 있습니다.



노트

수식 노드와 식 노드 함수는 Microsoft Excel, MATLAB® 소프트웨어, 또는 다른 텍스트 기반 수학 환경에서 같은 이름을 가진 함수와 같은 값을 반환하지 않을 수도 있습니다.



노트 C 프로그래밍 언어에 익숙한 경우 수식 노드를 매우 효율적으로 사용할 수 있습니다. LabVIEW 도움말은 사용자가 C에 익숙하다고 가정하며, C 프로그래밍과 관련된 토픽은 다루지 않습니다. 더 자세한 정보는 Brian W. Kernighan과 Dennis M. Ritchie의 The C Programming Language를 참고하시거나 Herbert Schildt의 Teach Yourself C를 참조하십시오.

함수	상응하는 LabVIEW 함수	설명
absx	절대값	절대값 x 를 반환합니다.
acosx	아크 코사인	라디안으로 표시되는 x 의 아크코사인을 계산합니다.
acoshx	역 쌍곡형 코사인	x 의 역 쌍곡형 코사인을 계산합니다.
asinx	아크 사인	라디안으로 표시되는 x 의 아크 사인을 계산합니다.
asinhx	역 쌍곡형 사인	x 의 역 쌍곡형 사인을 계산합니다.
atanx	아크 탄젠트	라디안으로 표시되는 x 의 아크 탄젠트를 계산합니다.
atan2y, x	y/x 의 아크 탄젠트	라디안으로 표시되는 y/x 의 아크 탄젠트를 계산합니다.
atanhx	역 쌍곡형 탄젠트	x 의 역 쌍곡형 탄젠트를 계산합니다.
ceilx	올림	다음 높은 정수로 x 를 올림합니다(최소 정수 $\geq x$).
cosx	코사인	x 의 코사인을 계산하며, 이때 x 는 라디안으로 표시됩니다.
coshx	쌍곡형 코사인	x 의 쌍곡형 코사인을 계산합니다.
cotx	코탄젠트	x ($1/\tan x$)의 코탄젠트를 계산하며, 이 때 x 는 라디안으로 표시됩니다.
cscx	코시컨트	x ($1/\sin x$)의 코시컨트를 계산하며, 이 때 x 는 라디안으로 표시됩니다.
expx	지수	e 의 x 거듭제곱을 계산합니다.
expm1x	지수-1	e x 거듭제곱에서 1을 뺀 값을 계산합니다($e^x - 1$).
floorx	내림	다음 낮은 정수로 x 를 내림합니다(최대 정수 $\leq x$).
getexpx	가수 및 지수	x 의 지수를 반환합니다.
getmanx	가수 및 지수	x 의 가수를 반환합니다.
intx	반올림	x 를 가장 가까운 정수로 반올림합니다.
intrzx	—	x 를 x 와 제로 사이에서 가장 가까운 정수로 반올림합니다.
lnx	자연로그	x (밑이 e 인)의 자연로그를 계산합니다.
lnp1x	자연로그(인수 +1)	$(x + 1)$ 의 자연로그를 계산합니다.
logx	상용로그	x (밑이 10인)의 로그를 계산합니다.

함수	상응하는 LabVIEW 함수	설명
log2x	밑이 2인 로그	x(밑이 2인)의 로그를 계산합니다.
maxx, y	최대 및 최소	x와 y를 비교하고 큰 값을 반환합니다.
minx, y	최대 및 최소	x와 y를 비교하고 작은 값을 반환합니다.
modx, y	몫과 나머지	몫이 음의 무한대로 반올림될 때 x/y의 나머지를 계산합니다.
powx, y	x의 거듭제곱	x의 y 거듭제곱을 계산합니다.
rand	난수(0 –1)	0과 1 사이의 부동소수를 생성합니다.
remx, y	몫과 나머지	몫이 음의 무한대로 반올림될 때 x/y의 나머지를 계산합니다.
secx	시컨트	x의 시컨트를 계산하며, 이 때 x는 라디안 (1/cosx)으로 표시됩니다.
signx	부호	x가 0보다 크면 1, x가 0과 같으면 0, x가 0보다 작으면 -1을 반환합니다.
sinx	사인	x의 사인을 계산하며, 이때 x는 라디안으로 표시됩니다.
sincx	싱크(Sinc)	x (sinx/x)로 나눈 x의 사인을 계산하며 이 때 x는 라디안으로 표시됩니다.
sinhx	쌍곡형 사인	x의 쌍곡형 사인을 계산합니다.
sizeOfDimary, di	—	배열 ary를 위해 지정된 차원의 크기 di를 반환합니다.
sqrtx	제곱근	x의 제곱근을 계산합니다.
tanx	탄젠트	x의 탄젠트를 계산하며, 이때 x는 라디안으로 표시됩니다.
tanhx	쌍곡형 탄젠트	x의 쌍곡형 탄젠트를 계산합니다.

수식 노드와 식 노드에서 연산자의 우선 순위

연산자의 우선 순위는 가장 높은 것으로부터 작은 것까지 다음과 같습니다. 같은 라인에 있는 연산자는 같은 우선 순위를 가집니다.

**	지수
+, -, !, ~, ++, --	1진법 플러스, 1진법 마이너스, 논리적 부정, 비트 보수, 선증가와 후증가, 선감소와 후감소 ++와 --는 식 노드에서 이용할 수 없습니다.

<code>*</code> , <code>/</code> , <code>%</code>	곱하기, 나누기, 나눴자(나머지)
<code>+</code> 와 <code>-</code>	더하기와 빼기
<code>>></code> 와 <code><<</code>	오른쪽과 왼쪽 연산 이동
<code>></code> , <code><</code> , <code>>=</code> , <code><=</code>	더 큰, 더 작은, 더 크거나 같은, 더 작거나 같은
<code>!=</code> , <code>==</code>	같지 않음과 같음
<code>&</code>	비트 and
<code>^</code>	비트 exclusive or
<code> </code>	비트 or
<code>&&</code>	논리적 and
<code> </code>	논리적 or
<code>?</code> :	조건적인 평가
<code>= op =</code>	값 할당, 바로 가기 수행, <code>op</code> 할당은 <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>>></code> , <code><<</code> , <code>&</code> , <code>^</code> , <code> </code> , <code>%</code> 또는 <code>**</code> 가 될 수 있습니다. <code>= op=</code> 은 식 노드에서 이용할 수 없습니다.

할당 연산자 `=`은 지수 연산자 `**`와 마찬가지로 오른쪽으로(오른쪽에서 왼쪽으로 그룹을 지어서) 연산합니다. 다른 모든 이항 연산자는 왼쪽으로 연산합니다.

출력에서 참의 숫자값은 1이고, 거짓은 0입니다. 0의 논리값은 거짓이고 모든 다른 숫자는 참입니다. 조건식의 논리값

`<lexpr> ? <texpr>: <fexpr>`

`<lexpr>`의 논리값이 참이면 `<texpr>`이고 그외의 경우에 `<fexpr>`입니다.



노트

C 프로그래밍 언어에 익숙한 경우, 수식 노드를 사용하는 것이 가장 효율적입니다. LabVIEW 도움말은 사용자가 C에 익숙하다고 가정하며, C 프로그래밍과 관련된 토픽은 다루지 않습니다. 더 자세한 정보는 Brian W. Kernighan과 Dennis M. Ritchie의 *The C Programming Language*를 참고하시거나 Herbert Schildt의 *Teach Yourself C*를 참조하십시오.

수식 노드에서 배열 조작

입력 배열과 입출력 배열은 연결된 배열로부터 타입을 얻습니다. 이 배열은 사용자가 수식 노드 내에서 배열을 선언할 것을 요구하지 않습니다. 하지만, 수식 노드에 로컬 배열과 출력 배열은 반드시 선언해야 합니다.

배열은 C에서와 같이 제로를 기본으로 합니다. 하지만 C와 달리 LabVIEW는 범위를 벗어난 배열 원소에 대한 할당을 동작이 없는 것으로 취급하며 아무런 할당도 발생하지 않습니다.

니다. 또한 C와는 달리, 범위를 벗어난 배열 원소에 대해서 참조를 만들 경우, LabVIEW는 제로값을 반환합니다. 배열 출력이 배열 입력에 대응하지 않을 경우 반드시 수식 노드에 배열 출력을 선언해야 하며, 이 경우 두 터미널은 반드시 이름을 공유해야 합니다.

입력 터미널이 배열인 경우, 다음 함수를 사용하여 배열의 모양을 결정할 수 있습니다.

```
numDim = numberOfDims(inputArr);

firstrowsize = sizeOfDim(inputArr, 0);
```

배열 조작 예제

다음과 같이 일반적인 배열 선언을 합니다.

```
int16 a[4][7]
```

배열을 선언한 후, 다음과 같이 할당할 수 있습니다.

```
a[3][2]=455;
```

또한 다음과 같이 배열 원소를 식에 사용할 수 있습니다.

```
a[0][5]=a[1][6]*3+a[5][0];
```

수식 노드 터미널 타입 변경하기

다음 단계를 따라 수식 노드 터미널을 출력에서 입력으로 또는 그 반대로 변경합니다.

1. 출력 또는 입력 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 입력으로 변경 또는 출력으로 변경을 선택합니다.

수식 노드 생성하기

다음 단계를 따라 [수식 노드]를 생성합니다.

1. 블록 다이어그램에 [수식 노드]를 추가합니다.
2. 사용할 수 있는 **함수**와 **연산자**를 검토합니다.
3. 라벨링 도구 또는 수행 도구를 사용하여 계산하려는 방정식을 [수식 노드]에 입력합니다. 할당하려는 각 대상에 대해 반드시 하나의 변수를 할당 기호 =의 왼쪽에 놓습니다. 할당하는 각 명령문은 반드시 세미콜론 ;으로 끝내야 합니다. 올바른 **수식 노드 구문**을 사용하고 있는지를 확인합니다. 구문 에러가 발생하는 경우, 깨

진 실행 버튼을 클릭하여 에러 리스트 윈도우를 디스플레이합니다. LabVIEW는 구문 에러를 # 기호로 표시합니다.



팁

슬래쉬-별표 쌍 /*주석*/으로 에워싸거나 두 개의 슬래쉬 //주석 다음에 주석을 추가하여 [수식 노드]에 텍스트를 추가합니다.

1. [수식 노드] 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력 추가를 선택하여 각 입력 변수에 대한 입력 터미널을 생성합니다. 나타나는 터미널에 변수 이름을 입력합니다. VI가 실행 중일 때를 제외하고는 라벨링 도구 또는 수행 도구를 사용하여 언제든지 변수 이름을 편집할 수 있습니다. 변수 터미널은 대소문자를 구분합니다. [수식 노드]에서 터미널 또는 방정식의 개수는 제한이 없습니다. **터미널 타입을 변경**하거나 **터미널을 제외**할 수 있습니다.
2. [수식 노드] 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 출력 추가를 선택하여 각 출력 변수에 대한 출력 터미널을 생성합니다. 나타나는 터미널에 변수 이름을 입력합니다. VI가 실행 중일 때를 제외하고는 라벨링 도구 또는 수행 도구를 사용하여 언제든지 변수 이름을 편집할 수 있습니다. 출력 변수는 입력 변수보다 두꺼운 경계를 가집니다.



노트

두 입력과 두 출력은 각각 같은 이름을 가질 수 없습니다. 하지만, 출력은 입력과 같은 이름을 가질 수 있습니다.

1. (옵션) 출력 터미널의 기본 데이터 타입은 배정도 부동소수점 실수입니다. 데이터 타입을 변경하기 위해서는, 출력 터미널과 정확히 같은 이름을 가지는 입력 터미널을 생성하고 데이터 타입을 입력 터미널에 연결합니다. 이렇게 하면 터미널에 기본값을 제공합니다. 또한, **수식 노드 구문**을 사용하여 [수식 노드] 안에 있는 변수를 정의할 수 있습니다. 예를 들어, `int32 y;`는 출력 터미널 y의 데이터 타입을 32비트 정수로 변경합니다.
2. [수식 노드]의 입력과 출력 터미널을 블록 다이어그램의 대응하는 터미널에 연결합니다. 모든 입력 터미널은 반드시 연결되어야만 합니다. 출력 터미널은 연결되지 않아도 됩니다.

수식 노드 구문

수식 노드 구문은 텍스트 기반의 프로그래밍 언어에서 사용되는 구문과 유사합니다. C에서처럼 끝 지정은 세미콜론(;)임을 기억합니다. **스코프 규칙**을 사용하여 수식 노드에서 변수를 선언합니다. 또한, 수식 노드에서 **허용되는 함수**와 **허용되는 연산자 및 우선 순위**를 유념하십시오.



노트

C 프로그래밍 언어에 익숙한 경우, 수식 노드를 사용하는 것이 가장 효율적입니다. LabVIEW 도움말은 사용자가 C에 익숙하다고 가정하며, C 프로그래밍과 관련된 토픽은 다루지 않습니다. 더 자세한 정보는 Brian W. Kernighan과 Dennis M. Ritchie의 *The C Programming Language*를 참고하시거나 Herbert Schildt의 *Teach Yourself C*를 참조하십시오.

수식 노드 구문은 배커스-나우어 형식 (BNF 표기법)을 사용하여 아래와 같이 요약됩니다. 개요는 비터미널 기호를 포함합니다: 복합 명령, 식별자, 조건 식, 숫자, 배열 크기, 부동소수 타입, 정수 타입, 왼쪽, 할당 연산자, 함수 빨간색 고정 폭 굵은체로 표기되는 기호는 실제 사용시에 주어진대로 정확하게 사용해야 하는 터미널 기호입니다. 기호 #은 그것을 따르는 임의 숫자를 나타냅니다.

명령 리스트:

명령; 명령; 명령 리스트

명령:

할당 복합 명령 컨트롤 명령 조건 명령 반복 명령 스위치 명령 변수 선언

할당:

식 왼쪽 할당 연산자 할당

식:

식 2진 연산자 식 1진 연산자 식 식 1진 연산자 식? 식: 식 식 식별자 상수 함수 이름 인수 리스트

왼쪽:

식별자 식별자 배열 예약

배열 예약:

[할당] [할당]배열 예약

할당 연산자: 다음 중 하나

= += -= *= /= >>= <<= &= ^= |= %= **=

2진 연산자: 다음 중 하나

+ - * / ^ != == > < >= <= && || &% **

1진 연산자: 다음 중 하나

+ - ! ++ -- ~

인수 리스트:

식 식, 인수 리스트

상수:

pi 숫자

복합 명령:

{ 명령 리스트 }:

다음 테이블은 컨트롤, 조건, 반복, 스위치 명령을 위한 수식 노드 구문을 제공합니다.

명령 타입	구조	문법	설명/예제
컨트롤 명령	Break 명령	break	Break 키워드를 사용하여 가장 가까운 Do, For, While 루프를 나오거나 수식 노드에서 구문을 바꿉니다.
연속 명령문	continue	Continue 키워드를 사용하여 수식 노드에서 가장 가까운 Do, For, While 루프의 다음 반복에 컨트롤을 전달합니다.	
조건 명령	조건 명령: if 명령 if-else 명령	<pre> if (y==x && a[2][3]<a[0][1]) { int32 temp; temp = a[2][3]; a[2][3] = y; y=temp; } else x=y; </pre>	
If 명령	if 명령: if 할당 명령		
If-Else 명령	if-else 명령: if 명령 else 명령		
반복 명령	반복 명령: do 명령 for 명령 while 명령		
Do 루프	do 명령 while 할당	<pre> do { int32 temp; temp = --a[2]+y; y=y-1; } while (y!=x && a[2]>=a[0]); </pre>	
For 루프	for [할당]; [할당]; [할당] 명령	<pre> for (y=5; y<x; y*=2) { int32 temp; temp = --a[2]+y; x-=temp; } </pre>	
While 루프	while 할당 명령	<pre> while (y!=x && a[2]>=a[0]) { int32 temp; temp = --a[2]+y; y=y-1; } </pre>	

명령 타입	구조	문법	설명/예제
Switch 명령	switch 할당 {케이스 명령 리스트}	switch(month){ case 2: days = evenyear? 29: 28; break; case 4:case 6:case9: days = 30; break; default: days = 31; break; }	
Case 명령 리스트	case 명령 리스트: case 명령 case 명령 리스트 case 명령		
Case 명령	case 명령: case 번호 : 명령 리스트 default : 명령 리스트		

변수 선언:

타입 지정자 식별자 타입 지정자 식별자 배열 인덱스 리스트 타입 지정 식별자 = 할당

배열 인덱스 리스트:

[정수 상수] [정수 상수] 배열 인덱스 리스트

타입 지정자:

부동소수 타입 정수 타입

부동소수 타입:

float float32 float64

정수 타입:

int int8 int16 int32 uint8 uint16 uint32

숫자 아님: 다음 중 하나

_ a~z A~Z

숫자: 다음 중 하나

0 1 2 3 4 5 6 7 8 9

제로가 아닌 숫자: 다음 중 하나

1 2 3 4 5 6 7 8 9

2진 숫자: 다음 중 하나

0 1

8진 숫자: 다음 중 하나

0 1 2 3 4 5 6 7

16진 숫자: 다음 중 하나

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

식별자:

숫자 없음 [처음 아닌 문자]

처음 아닌 문자:

숫자 없음 [처음 아닌 문자] 숫자 [처음 아닌 문자]

숫자:

정수 상수 부동 상수

정수 상수:

10진 상수 2진 상수 8진 상수 16진 상수

10진 상수:

0진수가 아닌 #진수

2진 상수:

0b #2진 숫자 0B #2진 숫자

8진 상수:

0 #8진 숫자

16진 상수:

0x #16진 숫자 0X #16진 숫자

부동 상수:

소수 지수 부분 10진 상수 지수 부분

소수:

#숫자. 숫자 #숫자

지수 부분:

e [부호] #숫자 E [부호] #숫자

부호: 다음 중 하나

+ -

주석:

// 주석 /* 주석 */

수식 노드에서 터미널 삭제하기

다음 단계를 따라 수식 노드에서 터미널을 삭제합니다.

1. 삭제하려는 입력 또는 출력 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 제거를 선택합니다.

또한, 터미널을 선택하고 <Delete> 또는 <Backspace> 키를 누릅니다.

수식 노드에서 변수를 선언하기 위한 스코프 규칙

수식 노드는 C처럼 변수를 선언하기 위해 같은 스코프 규칙을 사용합니다.

- 괄호가 쳐진 블록에서 선언하는 모든 변수는 괄호가 쳐진 블록 안에서 단지 접근할 수 있습니다.
- 모든 입력 터미널은 가장 상위의 블록에서(괄호로 둘러싸여 있지 않은) 변수로서 고려되고, 가장 상위의 블록에서 다시 선언될 수 없습니다.
- LabVIEW는 가장 상위의 블록에서(괄호로 둘러싸여 있지 않은) 선언하는 변수를 출력 터미널에 같은 이름으로 일치시키려고 합니다.
- 입력 터미널에 상응하지 않고 가장 상위의 블록에서 선언되지 않는 출력 터미널을 선언합니다.

변수를 선언할 때 정확한 수식 노드 구문을 사용합니다.



노트

C 프로그래밍 언어에 익숙한 경우, 수식 노드를 사용하는 것이 가장 효율적입니다. LabVIEW 도움말은 사용자가 C에 익숙하다고 가정하며, C 프로그래밍과 관련된 토픽은 다루지 않습니다. 더 자세한 정보는 Brian W. Kernighan과 Dennis M. Ritchie의 The C Programming Language를 참고하시거나 Herbert Schildt의 Teach Yourself C를 참조하십시오.

관련 정보

수식노드 구문

식노드 생성하기

[식노드]를 사용하여 단일 변수를 갖는 수식이나 방정식을 계산합니다. [식노드]는 변수가 한 개인 경우 유용하지만 변수가 여러 개인 경우는 복잡해집니다.

[식노드]는 입력 터미널에 전달한 값을 변수의 값으로 사용합니다. 출력 터미널은 계산한 값을 반환합니다.

다음 단계를 따라 [식노드]를 생성합니다.

1. 블록 다이어그램에 [식노드]를 추가합니다.
2. 라벨링 도구 또는 수행 도구를 사용하여 [식노드] 안에 방정식을 입력합니다. 방정식에 하나 이상의 변수를 가질 수 없습니다. 산술식에 대해서 대부분의 텍스트 기반 프로그래밍 언어에서 사용되는 구문과 비슷한 [구문](#)을 사용합니다. 구문 에러가 발생하는 경우, 깨진 실행 버튼을 클릭하여 에러 리스트 윈도우를 디스플레이합니다. LabVIEW는 구문 에러를 # 기호로 표시합니다.
3. 변수에 대해 값을 제공하는 VI, 함수, 컨트롤, 또는 상수 터미널을 [식노드]의 입력 터미널에 연결합니다. 또한 [식노드]의 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>컨트롤 또는 생성>>상수를 선택하여 컨트롤 또는 상수를 생성할 수 있습니다.
4. [식노드]의 출력 터미널을 방정식의 값을 받으려는 VI, 함수, 또는 인디케이터 터미널에 연결합니다. 또한, [식노드]의 출력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 인디케이터를 생성할 수 있습니다.

식노드 구문

다음의 식노드 구문의 요약은 배커스-나우어 폼(BNF 표기법)을 사용합니다.

<식> : = <식> <2진 연산자> <식>

↓1진 연산자> <식>

↓식> ? <식> : <식>

| (<식>)

↓식별자>

↓상수>

↓함수> <인수 리스트>

<2진 연산자>:= +|-|*|/|^|!=|==↑↓↑↓=|&&|||&|||^|**

<1진 연산자>:= +|-|!|~



노트

식 노드 구문에 옵션 아이টে를 포함시키려면, 대괄호를 사용하여 아이টে를 둘러쌉니다.

스크립트 노드

스크립트 노드를 사용하여 LabVIEW에서 수학 스크립트를 실행합니다. LabVIEW는 MATLAB® 스크립트 노드라는 스크립트 노드를 제공합니다. 이 노드는 MATLAB 언어 구문으로 작성된 스크립트의 처리를 용이하게 합니다.



노트

MATLAB 스크립트 노드는 MATLAB 소프트웨어 스크립트 서버를 호출하여 스크립트를 실행하기 때문에 MATLAB 스크립트 노드를 사용하기 위해서는 반드시 컴퓨터에 정품 인증된 MATLAB 소프트웨어 6.5 또는 이후 버전이 설치되어 있어야만 합니다. MATLAB 스크립트 노드는 Windows에서만 사용할 수 있습니다.

스크롤 막대를 스크립트 노드에 추가하기

스크립트 노드 안에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이টে>>스크롤 막대를 선택합니다. 스크롤 막대는 스크립트 노드의 오른쪽 측면에 나타납니다.

LabVIEW에서 MATLAB® 소프트웨어 호출하기

MATLAB® 소프트웨어 버전 6.5 또는 이후 버전의 정품이 설치되어 있어야만 MATLAB 스크립트 노드를 사용할 수 있습니다. 스크립트 노드가 MATLAB 소프트웨어 스크립트 서버를 호출하여 MATLAB 언어 구문으로 작성된 스크립트를 실행하기 때문입니다. MATLAB 스크립트 노드는 Windows에서만 사용할 수 있습니다.

다음 단계를 따라 MATLAB 언어 구문으로 작성된 스크립트를 사용하는 VI를 생성하고 실행합니다.

1. 블록 다이어그램에 MATLAB 스크립트 노드를 추가합니다.



노트

LabVIEW Full 및 Professional Development System에서만 MATLAB 스크립트 노드를 생성할 수 있습니다. VI가 MATLAB 스크립트 노드를 포함할 경우, 모든 LabVIEW 패키지에서 그 VI를 실행할 수 있습니다.

1. 수행 도구 또는 라벨링 도구를 사용하여 다음 스크립트를 MATLAB 스크립트 노드에 입력합니다. `a=rand(50) surf(a)`
2. 출력을 MATLAB 스크립트 노드에 추가하고 출력에 대한 인디케이터를 생성합니다.
 1. MATLAB 스크립트 노드 프레임의 오른쪽 부분에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 출력 추가를 선택합니다.
 2. 출력 터미널에 `a`를 입력하여 스크립트에서 `a` 변수의 출력을 추가합니다.
 3. 출력 터미널의 데이터 타입을 검증합니다. MATLAB 스크립트 노드에서, 새로운 입력 또는 출력의 기본 데이터 타입은 실수값입니다. `a` 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 타입 선택 >> 실수의 2D 배열을 선택합니다.
 4. `a` 출력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 인디케이터를 선택하여 실수의 2D 배열로 라벨이 붙여진 숫자형 인디케이터의 2D 배열을 생성합니다.
3. 에러 출력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 인디케이터를 선택하여 에러 출력이라는 라벨이 붙여진 에러 출력 인디케이터를 생성합니다.
4. VI를 실행할 때 스크립트가 생성하는 숫자를 보기 위해서 실수의 2D 배열 프런트 패널 인디케이터를 크기 조정합니다.
5. VI를 실행합니다. LabVIEW는 **MATLAB 소프트웨어 스크립트 서버**를 호출하며, 임의의 값의 행렬을 생성하고, MATLAB 소프트웨어에 행렬을 디스플레이하고, 해당 정보를 그래프에 플롯하고, 행렬을 구성하는 값을 실수의 2D 배열 프런트 패널 인디케이터에 디스플레이합니다.
6. (옵션) MATLAB 언어 구문으로 작성된 스크립트를 **디버그**합니다.

MATLAB 스크립트 노드에서 **스크립트 막대를 디스플레이**할 수 있습니다.



노트 MATLAB 스크립트 노드 내의 스크립트는 MATLAB 소프트웨어 환경에서 작동하는 것처럼 동일하게 작동합니다.

스크립트 노드에서 스크립트 서버 변경하기

MATLAB® 소프트웨어는 MATLAB 언어 구문으로 작성된 스크립트를 실행하는 스크립트 서버 엔진을 설치합니다. 타사 소프트웨어를 통해 스크립트를 실행하는 스크립트 서버 엔진이 설치될 수도 있습니다. LabVIEW는 이 대응하는 스크립트 노드를 통해 해당 스크립트 서버 엔진과 통신합니다. 스크립트 노드를 변환하여 LabVIEW와 통신하는 스크립트 서버 엔진을 변경할 수 있습니다.

다음 단계를 완료하여 LabVIEW와 통신하는 스크립트 서버 엔진을 변경합니다.

1. MATLAB 스크립트 노드 또는 타사 스크립트 노드의 경계에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 스크립트 서버 선택을 선택하고 LabVIEW가 통신해야 할 스크립트 서버 엔진을 선택합니다.

스크립트 지우기

다음 단계를 따라 스크립트 노드의 내용을 지웁니다.

1. 스크립트 노드에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 스크립트 지우기를 선택합니다.

스크립트 노드 터미널의 데이터 타입 설정하기

MATLAB 스크립트 노드 터미널의 데이터 타입을 설정할 수 있습니다.

MATLAB 스크립트 노드

MATLAB 언어 구문은 느슨하게 타입된 스크립트 언어이며 스크립트가 실행될 때까지는 변수의 데이터 타입을 결정하지 않습니다. 따라서, LabVIEW는 편집 모드에서 변수의 타입을 결정할 수 없습니다. LabVIEW는 가능한 데이터 타입을 찾기 위해서 스크립트 서버에 쿼리합니다. 각 터미널에 대한 LabVIEW 데이터 타입을 선택할 수 있습니다.



노트

변수 데이터 타입을 정확하게 설정하지 않는 경우, LabVIEW는 실행 시 에러를 만들거나, 부정확한 정보를 반환합니다.

다음 단계를 완료하여 MATLAB 스크립트 노드의 입력 또는 출력 터미널의 데이터 타입을 변경합니다.

1. 입력 또는 출력의 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 타입 선택을 선택합니다. 사용 가능한 데이터 타입의 리스트가 나타납니다.
2. 사용하려는 데이터 타입을 선택합니다.

디버깅 스크립트

스크립트 노드를 사용하여 LabVIEW에서 수학 스크립트를 실행합니다. LabVIEW는 MATLAB® 스크립트 노드라는 스크립트 노드를 제공합니다. 이 노드는 MATLAB 언어 구문으로 작성된 스크립트의 처리를 용이하게 합니다.

보다 쉽게 스크립트를 디버깅하려면, 다음 프로그래밍 기술을 이용하십시오:

- LabVIEW로 반입하기 전에 디버깅과 테스트를 위한 목적으로 원래 프로그램에서 스크립트를 작성하고 실행합니다.
- **데이터 타입을 확인합니다.** 새 입력과 출력을 만들 때, 터미널의 데이터 타입이 옳은지 확인합니다. 또한, LabVIEW와 스크립트 서버 엔진 사이에 통과하는 값을 모니터링하기 위해 입력과 출력에 해당하는 컨트롤과 인디케이터를 생성합니다. 이것

은 필요하다면 어디에서 스크립트 노드가 잘못된 값을 계산하는지 지적하도록 합니다.

- 디버깅 정보에 대해 에러 확인 파라미터를 활용합니다. 에러 출력 터미널에 대한 인디케이터를 MATLAB 스크립트 노드에 먼저 생성한 후에 VI를 실행해야 하며 이렇게 하면 실행 시 생성되는 에러 정보를 볼 수 있습니다.

스크립트 반입 또는 반출하기

다른 .m 파일 언어 구문과 같은 다른 텍스트 기반의 구문으로 이미 작성한 스크립트가 있는 경우, 이를 LabVIEW에 반입할 수 있습니다. 또한, LabVIEW에 있는 스크립트를 텍스트 파일에 저장할 수 있습니다.

다음 단계에 따라 스크립트를 LabVIEW의 다른 **스크립트 노드**로 반입합니다.

1. MathScript 스크립트 노드를 선택한 후 블록다이어그램에 스크립트 노드를 추가할 영역을 끌어서 늘립니다.



노트

LabVIEW Full 및 Professional Development System에서만 MATLAB 스크립트 노드를 생성할 수 있습니다. VI가 MATLAB 스크립트 노드를 포함할 경우, 모든 LabVIEW 패키지에서 그 VI를 실행할 수 있습니다. 컴퓨터에 MATLAB® 소프트웨어 6.5 또는 이후 버전의 정품이 설치되어 있어야만 MATLAB 스크립트 노드를 사용할 수 있습니다.

1. 스크립트 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 반입을 선택하여 파일 대화 상자를 디스플레이합니다.
2. 반입하려는 파일을 선택하고 확인 버튼을 클릭합니다. 스크립트 텍스트가 노드에 나타납니다.

다음 단계를 따라 스크립트를 텍스트 파일에 저장합니다.

1. 스크립트 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 반출을 선택하여 파일 대화 상자를 디스플레이합니다.
2. 텍스트 파일을 위해 사용할 이름을 입력하거나 덮어쓰려는 파일을 선택합니다.
3. 확인 버튼을 클릭합니다.

스크립트 노드의 라인 번호 보이기 및 숨기기

수식 노드와 MATLAB 스크립트 노드에서 라인 번호를 표시하고 숨길 수 있습니다.

다음 단계를 따라 스크립트 노드에서 라인 번호를 보이거나 숨깁니다.

1. 노드 안에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 보이는 아이템 >> 라인 번호를 선택합니다.

변수

변수 읽기와 쓰기

로컬 또는 글로벌 변수를 생성한 후, 변수에서 데이터를 읽거나 변수에 데이터를 쓸 수 있습니다. 기본으로, 새 변수는 데이터를 받습니다. 이 종류의 변수는 인디케이터로 작동하며 쓰기 로컬 또는 글로벌 변수입니다. 또한, 변수를 설정하여 데이터 소스 또는 읽기 로컬이나 글로벌 변수로서 작동하도록 할 수 있습니다.

블록다이어그램에서 컨트롤과 인디케이터를 구별하는 것과 같은 방법으로 읽기 로컬 또는 글로벌 변수를 쓰기 로컬 또는 글로벌 변수와 구별할 수 있습니다. 읽기 로컬 또는 글로벌 변수는 컨트롤과 유사한 굵은 경계선이 있습니다. 쓰기 로컬 또는 글로벌 변수는 인디케이터와 유사한 얇은 경계를 가집니다.



팁

글로벌 변수에서 값을 읽기 전에 값을 써서 그 값에 접근할 때 초기화되어 있는 값을 가지게 됩니다.



관련 정보

[로컬 변수 생성하기](#) (1087 페이지 참조)

[글로벌 변수 생성하기](#) (1088 페이지 참조)

[로컬 변수의 데이터 방향 변경하기](#) (1086 페이지 참조)

[글로벌 변수의 데이터 방향 변경하기](#) (1088 페이지 참조)

로컬 변수

로컬 변수의 데이터 방향 변경하기

읽기로 변경과 쓰기로 변경 바로 가기 메뉴 항목을 사용하여 로컬 변수의 데이터 방향을 변경하도록 읽기 로컬을 쓰기 로컬로 또는 그 반대로 변경합니다.

다음 단계를 따라 로컬 변수의 설정을 변경하여 읽기 로컬로 동작하도록 한 후 쓰기 로컬로 다시 변경합니다.

1. 문자열 컨트롤을 프런트패널에 추가합니다.
2. 라벨링 도구를 사용하여 컨트롤 이름에 라벨을 붙입니다.
3. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>로컬 변수를 선택합니다. 문자열 컨트롤에 대한 로컬 변수가 블록다이어그램에 나타납니다. 기본적으로, 로컬 변수는 쓰기 로컬이며 얇은 경계를 가집니다.
4. 블록다이어그램의 로컬 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 읽기로 변경을 선택하여 로컬 변수를 컨트롤처럼 동작하도록 설정합니다. 로컬 변수는 읽기 로컬로 바뀌며 두꺼운 경계를 가집니다.

- 로컬 변수 노드에서 다시 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 쓰기 변경을 선택하여 변수를 쓰기 로컬로 만듭니다.

로컬 변수 생성하기

블록다이어그램의 두 위치에서 **프론트패널의 한 객체에 쓰기** 또는 블록다이어그램의 두 위치에서 프론트패널의 한 객체를 읽을 때 로컬 변수를 사용할 필요가 있습니다.

기존 프론트패널 객체나 블록다이어그램 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성»로컬 변수를 선택하여, 블록다이어그램에 추가할 수 있는 로컬 변수를 생성합니다.

다음 단계를 따라 로컬 변수를 생성하고 이 로컬 변수는 프론트패널 객체와 연관되어 있습니다.

- 블록다이어그램에 로컬 변수를 추가합니다. 이 경우, 로컬 변수 노드는 아직 컨트롤이나 인디케이터에 연결되지 않은 상태입니다.
- 로컬 변수 노드에서 마우스 오른쪽 버튼을 클릭하고 아이템 선택 바로 가기 메뉴를 이용하여 프론트패널의 객체를 선택합니다. 이 바로 가기 메뉴는 고유 라벨을 가진 모든 프론트패널 객체를 열거합니다. 또한, 수행 도구 또는 라벨링 도구를 사용하여 로컬 변수 노드를 클릭하고 바로 가기 메뉴에서 프론트패널 객체를 선택합니다.



노트

LabVIEW는 로컬 변수를 프론트패널의 객체에 연결할 때 고유 라벨을 참고하기 때문에, 프론트패널의 컨트롤과 인디케이터에 충분히 설명하는 고유 라벨을 할당해야 합니다.



데이터 통신 방법

로컬 변수 사용 시 메모리 관리

로컬 변수는 데이터 버퍼의 복사본을 생성합니다. 로컬 변수에서 데이터를 읽어올 때, 이와 연결된 컨트롤의 데이터에 대한 새로운 버퍼를 생성하게 됩니다.

로컬 변수를 사용하여 대량의 데이터를 블록다이어그램의 한 위치에서 다른 위치로 이동하는 경우, 일반적으로 와이어를 사용한 데이터 전송보다 더 많은 메모리를 사용하게 되어 결국 실행 속도가 느려지게 됩니다. 실행시 데이터를 저장해야 하는 경우, **시프트 레지스터** 사용을 고려하십시오.

글로벌 변수

글로벌 변수의 데이터 방향 변경하기

읽기로 변경과 쓰기로 변경 바로 가기 메뉴 항목을 사용하여 글로벌 변수의 데이터 방향을 변경하도록 읽기 글로벌을 쓰기 글로벌로 또는 그 반대로 변경합니다.

다음 단계를 따라 글로벌 변수의 설정을 변경하여 읽기 글로벌로 동작하도록 한 후 쓰기 글로벌로 다시 변경합니다.

1. 블록다이어그램에 글로벌 변수를 추가합니다. 기본적으로, 글로벌 변수는 쓰기 글로벌이며 경계가 얇습니다.
2. 글로벌 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 읽기로 변경을 선택하여 글로벌 변수를 컨트롤처럼 동작하도록 설정합니다. 글로벌 변수는 읽기 글로벌로 바뀌며 두꺼운 경계를 가집니다.
3. 글로벌 변수 노드에서 다시 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 쓰기로 변경을 선택하여 변수를 쓰기 글로벌로 만듭니다.

글로벌 변수 생성하기

각각이 하나의 프런트패널 객체를 가진 여러 개의 단일 글로벌 VI를 만들거나, 비슷한 변수를 함께 그룹으로 만들기 위해 여러 개의 프런트패널 객체를 가진 하나의 글로벌 VI를 만들 수 있습니다.

다음 단계를 따라 여러 개의 프런트패널 객체를 가지는 글로벌 VI를 만들 수 있습니다.

1. 블록다이어그램에 글로벌 변수를 추가합니다.
2. 수행 도구 또는 위치 도구를 사용해서 글로벌 변수 노드를 더블 클릭하여 글로벌 VI의 프런트패널 윈도우를 디스플레이합니다.
3. 일반 프런트패널 윈도우에서 했던 방법과 동일하게 컨트롤과 인디케이터를 프런트패널에 추가합니다.



노트

LabVIEW는 **고유 라벨**을 사용하여 글로벌 변수를 구분하기 때문에, 설명적인 고유 라벨로 프런트패널의 컨트롤과 인디케이터를 라벨링하십시오.

1. 글로벌 VI를 저장하고 닫습니다.
2. 원래 VI의 블록다이어그램으로 돌아가서 글로벌 변수 노드를 마우스 오른쪽 버튼을 클릭하고 항목 선택 바로 가기 메뉴에서 프런트패널 객체를 선택합니다. 이 바로 가기 메뉴는 고유 라벨을 가진 모든 프런트패널 객체를 열거합니다. 또한, 수행 도구 또는 위치 도구를 사용하여 글로벌 변수 노드를 클릭하고 바로 가기 메뉴에서 프런트패널 객체를 선택합니다.

다음 단계를 따라 다른 VI에서 글로벌 변수를 사용합니다.

1. 함수 팔레트의 VI 선택 아이콘이나 텍스트를 클릭하고, 글로벌 VI를 저장한 디렉토리를 찾아서, VI를 더블 클릭합니다.
2. 블록다이어그램에 글로벌 변수를 추가합니다. 기본적으로, 글로벌 변수는 사용자가 글로벌 VI 안에 추가한 고유 라벨과 연계된 첫번째 프런트패널 객체입니다.
3. 블록다이어그램에 사용자가 추가한 글로벌 변수 노드에서 마우스 오른쪽 버튼을 클릭한 후, 아이템 선택 바로 가기 메뉴에서 프런트패널 객체를 선택하여, 글로벌 변수를 다른 프런트패널 객체의 데이터와 연계시킵니다.



노트 여러 VI가 데이터를 공유할 때처럼 와이어로 연결할 수 없을 때만 글로벌 변수를 사용합니다. 예를 들어, 한 VI에서 파일 경로가 변하고 다른 VI가 그 경로를 열어야 한다면, 여러 호출 VI를 통해 이 경로를 논리적인 방법으로 연결할 수는 없습니다. 글로벌 변수를 이용하여 경로 데이터를 가지고 있는 것이 가장 좋은 선택일 것입니다.

데이터 통신 방법 홈

템플릿 글로벌 변수 생성하기

다음 단계를 따라 글로벌 변수를 만들 때 사용할 수 있는 템플릿을 생성합니다.

1. 재사용하려는 프런트패널 객체만을 포함하기 위해서 **글로벌 변수를 생성**합니다.
2. 파일»다른 이름으로 저장을 선택하여 파일 대화 상자를 디스플레이합니다.



노트

이미 글로벌 변수를 저장한 경우, 대신 파일»다른 이름으로 저장을 선택하여 다른 이름으로 저장 대화 상자를 디스플레이합니다. 복사와 복사본 열기 라디오 버튼을 선택하고 계속 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.

1. (Windows) 파일 대화 상자의 저장할 타입 풀다운 메뉴에서 템플릿 VI를 선택합니다. (macOS와 Linux) 템플릿 VI 파일 확장자 .vit를 포함합니다.
2. 템플릿 글로벌 변수의 이름을 입력한 후 확인 버튼을 클릭합니다.



주의 사용자 VI 및 다른 파일을 **LabVIEW와 함께 설치되는 .llb 파일**에 저장하지 마십시오.

글로벌 변수 사용 시 메모리 관리

글로벌 변수에서 데이터를 읽어올 때, LabVIEW는 그 글로벌 변수에 저장된 데이터의 복사본을 생성합니다.

대용량의 배열이나 문자열을 다루는 경우, 글로벌 변수를 관리하려면 상당한 시간과 메모리가 요구됩니다. 배열의 글로벌 변수를 다루는 경우, LabVIEW는 한 개의 배열 원소를

수정하더라도 배열 원소 전체를 저장하고 편집하므로 비효율적입니다. 어플리케이션의 여러 장소에서 글로벌 변수를 읽으면, 여러개의 메모리 버퍼가 만들어지므로 비효율적이며 시스템의 성능을 저하시킵니다.

공유 변수

공유 변수에 대한 방화벽과 네트워크 주소 전송(NAT) 라우터 설정하기

방화벽이나 라우터를 통해 네트워크 공개된 **공유 변수**를 전송하려면 방화벽과 네트워크 주소 전송(NAT) 라우터를 설정해야 합니다. 방화벽이나 라우터의 설정이 잘못된 경우, 네트워크 공개된 공유 변수는 작동하지 않을 수 있습니다.

네트워크 공유 변수 서버는 사용자가 실행하는 각 어플리케이션 하나당 한 개씩 대응하는 TCP 포트 외에도 추가적으로 TCP 포트 2343과 UDP 포트 2343을 사용합니다. **NI Publish-Subscribe Protocol (NI-PSP)** 프로토콜이 포트 59110에서부터 사용 가능한 TCP 포트를 찾기 시작하여 포트를 찾을 때까지 하나씩 증가합니다. 설정해야 하는 포트 중에서 사용 가능한 포트의 개수는 사용자가 실행하는 어플리케이션의 수에 따라 다릅니다. 직접 LogosXT.ini 파일을 생성하고 LogosXT_BasePort 및 LogosXT_NumPortsToCheck 파라미터에 새 값을 입력하여 수동으로 **NI-PSP 프로토콜이 사용하는 TCP 포트의 범위를 설정**할 수 있습니다. (Phar Lap ETS, VxWorks) Phar Lap ETS 및 VxWorks 타겟의 경우, ni-rt.ini 설정 파일에서 TCP 포트의 범위를 설정할 수 있습니다.

가능한 경우, TCP를 사용하여 네트워크 공유 변수를 전송하십시오.



노트

(옵션) 네트워크 공유 변수 서버에 대해 UDP 포트도 사용 가능합니다. 필수인 UDP 포트 2343 외에, 서버는 포트 6000에서부터 시작하는 UDP 포트 범위를 사용합니다. 네트워크에 공개된 공유 변수 서버가 사용하는 6000 이상인 포트의 개수는 컴퓨터에서 실행되는 서버의 개수에 따라 달라집니다. 네트워크에 공개된 공유 변수 클라이언트는 포트 5000으로 시작하는 UDP 포트 범위를 사용합니다. 네트워크에 공개된 공유 변수 클라이언트가 사용하는 5000 이상인 포트의 개수는 컴퓨터에서 실행되는 클라이언트의 개수에 따라 달라집니다. 서버나 클라이언트가 사용하는 포트의 수를 알고 싶으면, 먼저 방화벽을 구성하지 않은 채로 **TCP**를 사용하여 필요한 포트 수를 파악합니다.

다음 단계를 따라 방화벽에 있지 않는 클라이언트를 방화벽내의 서버에 연결합니다. 방화벽 구성 단계는 사용하는 방화벽에 따라 다릅니다. 포트 열기에 대한 정보는 해당 방화벽에 대한 문서를 참조하십시오.

1. 서버 발견 알림을 허용한 상태로, 들어오는 패킷을 위해 TCP 포트 2343과 UDP 포트 2343을 엽니다.
2. TCP 포트 59110에서 시작하여, 실행하는 각 어플리케이션에 대해 하나씩 TCP 포트를 엽니다.
3. (선택 옵션) 들어오는 패킷을 위해 6000에서 6010사이의 범위에 있는 UDP 포트를 엽니다.

방화벽 내의 클라이언트를 방화벽에 있지 않은 서버에 연결할 수도 있습니다. 일부 방화벽은 별도의 설정없이 들어오는 패킷을 자동으로 전송합니다. (선택 옵션) 방화벽이 자동으로 포트를 열지 않는 경우, 들어오는 패킷을 위해 5000에서 5010 사이의 범위에 있는 UDP 포트를 엽니다.

다음 단계를 따라 NAT에 있지 않는 클라이언트를 방화벽내의 서버에 연결합니다. 라우터 구성 단계는 사용하는 라우터에 따라 다릅니다. 포트 포워딩에 대한 정보는 해당 라우터에 대한 문서를 참조하십시오. 라우터 내의 컴퓨터 한대 만이 서버로 작동할 수 있습니다.

1. TCP 포트 2343과 UDP 포트 2343을 서버를 실행하는 컴퓨터에 전달합니다.
2. TCP 포트 59110에서 시작하여, 실행하는 각 어플리케이션에 대해 하나씩 TCP 포트를 전달합니다.
3. (선택 옵션) 6000에서 6010 사이의 범위에 있는 UDP 포트를 서버를 실행 중인 컴퓨터에 전달합니다.

NAT 라우터내의 클라이언트를 라우터에 있지 않은 서버에 연결할 수도 있습니다. 일부 라우터는 별도의 설정없이 포트를 자동으로 포워드합니다 (옵션) 라우터가 포트를 자동으로 전달하지 않는 경우, 5000에서 5010 사이의 범위에 있는 UDP 포트를 클라이언트를 실행하는 컴퓨터에 전달합니다.



노트 Windows 방화벽이 활성화 상태인 Windows 컴퓨터에 네트워크 변수를 배포하는 경우, \National Instruments\Shared\Tagger\tagsrv.exe와 \windows\SysWOW64\lkads.exe를 Windows 방화벽 예외 리스트에 추가해야 컴퓨터에 호스트된 변수를 읽거나 쓸 수 있습니다.

공유 변수 스케일링하기

다음과 같은 데이터 타입을 가진 공유 변수의 값을 스케일링할 수 있습니다.

- 숫자형
- 불리언
- U32 비트 필드



노트

네트워크에 공개된 공유 변수, I/O 변수 또는 I/O 가명에 대해서만 스케일링을 사용할 수 있습니다. 스케일링하고자 하는 공유 변수가 네트워크 공유 변수일 경우, 반드시 데이터 소스를 가지고 있어야 합니다. 공유 변수 프로퍼티 대화 상자의 변수 페이지에 있는 가명 활성화 확인란에 확인 표시를 하여 네트워크 공유 변수가 데이터 소스를 갖도록 설정합니다.

숫자형 공유 변수 스케일링하기

다음 단계에 따라 숫자형 공유 변수를 스케일링합니다.

1. **프로젝트 탐색기** 윈도우의 공유 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 공유 변수 프로퍼티 대화 상자를 디스플레이합니다.
2. 스케일링을 선택하여 스케일링 페이지를 디스플레이합니다.
3. 스케일링 활성화 확인란에 확인 표시를 합니다.
4. 스케일 타입 풀다운 메뉴에서 다음 스케일링 옵션 중 하나를 선택합니다.
 - 선형—원시 단위 범위와 공학 단위 범위 간의 선형 $mx + b$ 변환을 활성화합니다.
 - 제곱근—원시 단위(기본 최소 스케일 및 기본 최대 스케일) 범위와 공학 단위(공학 최소 스케일 및 공학 최대 스케일) 범위간의 제곱근 변환을 활성화합니다. 이는 $b + m * \sqrt{x - o}$ 이고, 이때 b = 공학 최소 스케일, m = (공학 최대 스케일 - 공학 최소 스케일) / $\sqrt{\text{기본 최대 스케일} - \text{기본 최소 스케일}}$, o = 기본 최소 스케일입니다.
5. (옵션) 공학 단위 텍스트 박스에서, 섭씨, 파운드, 그램 등과 같은 데이터 측정의 단위를 지정합니다.
6. 기본 최대 스케일, 기본 최소 스케일, 공학 최대 스케일, 공학 최소 스케일 숫자형 컨트롤에서 원시 단위 범위 및 공학 단위 범위를 지정합니다.
7. (옵션) 데이터를 지정된 범위로 강제 변환하려면 범위 내로 강제 변환 확인란에 확인 표시를 합니다.
8. 확인 버튼을 클릭하여 공유 변수 프로퍼티 대화 상자를 닫습니다.

불리언 공유 변수 스케일링하기

불리언 데이터 타입을 갖는 공유 변수의 경우, **반전 스케일링**이 가능합니다. 다음 단계에 따라 불리언 공유 변수를 스케일링합니다.

1. **프로젝트 탐색기** 윈도우의 공유 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 공유 변수 프로퍼티 대화 상자를 디스플레이합니다.
2. 스케일링을 선택하여 스케일링 페이지를 디스플레이합니다.
3. 스케일링 활성화 확인란에 확인 표시를 합니다.
4. **공유 변수 엔진** (SVE)의 반전 확인란에 확인 표시를 하여, SVE가 디바이스 서버와 통신할 때 불리언 값을 반전합니다.
5. 확인 버튼을 클릭하여 공유 변수 프로퍼티 대화 상자를 닫습니다.

U32 비트 필드 공유 변수 스케일링하기

U32 비트 필드의 데이터 타입을 갖는 공유 변수의 경우 마스크 반전 스케일링 및/또는 마스크 선택 스케일링이 가능합니다. 다음 단계에 따라 비트 배열 공유 변수를 스케일링합니다.

1. **프로젝트 탐색기** 윈도우의 공유 변수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 공유 변수 프로퍼티 대화 상자를 디스플레이합니다.

2. 이 대화 상자의 리스트에서 스케일링을 선택하여 스케일링 페이지를 디스플레이합니다.
3. 스케일링 활성화 확인란에 확인 표시를 합니다.
4. 마스크 반전 스케일 텍스트 박스에서, 스케일링이 필요한 비트를 설정합니다. LabVIEW는 사용자가 설정하지 않는 비트를 마스크하고 이 마스크된 비트를 0으로 된 배열로 보고합니다.
5. 마스크 선택 스케일 텍스트 박스에서, 어떤 비트가 디바이스 서버와 **공유 변수 엔진** 사이에서 반전될 것인지 설정합니다. 이 텍스트 박스에서 지정한 비트만 반전할 수 있습니다.
6. 확인 버튼을 클릭하여 공유 변수 프로퍼티 대화 상자를 닫습니다.

(DSC Module) 공유 변수를 프로그램식으로 스케일링하려면 공유 변수 IO 프로퍼티를 사용하십시오.

변수 웹 서비스가 찾은 리소스

변수 웹 서비스는 다음과 같은 리소스를 공개합니다:

VariableObject – 다음 프로퍼티를 포함하는 VariableObject 모음:

- Url [문자열] – VariableObject의 다이나믹 API URL. Url 프로퍼티는 VariableObject를 고유한 방법으로 식별합니다.
- Type [문자열] – VariableObject의 타입. 사용 가능한 타입은 "container"와 "variable"입니다.
- Name [문자열] – VariableObject의 이름.
- Children [VariableObject []] – 현재 VariableObject의 하위 아이টে를 표현하는 VariableObject의 배열. 이 프로퍼티는 연기되도록 기본 설정되어 있습니다. \$expand 파라미터를 사용하여 이 프로퍼티 인라인을 나타내십시오.

Variable – VariableObject를 상속하고 다음 프로퍼티를 추가합니다:

- Access [문자열] – Variable의 Access 권한. 사용 가능한 권한은 "Read", "Write", "Read/Write"입니다.
- DataType [문자열] – Variable의 데이터 타입. 변수 웹 서비스는 원형 타입 및 원형 타입의 배열을 지원합니다. 복잡한 데이터 타입에서는 직렬화된 XML 문자열 또는 JSON 문자열을 Value 프로퍼티의 값으로 사용하십시오.
- VariableValue [문자열] – 현재 Variable의 추가적인 프로퍼티. 이 프로퍼티는 연기되도록 기본 설정되어 있습니다. \$expand 파라미터를 사용하여 이 프로퍼티 인라인을 나타내십시오.

VariableValue – 다음의 Variable 프로퍼티를 포함합니다:

- `Url` [문자열] — 상응하는 `Variable`의 다이나믹 API URL. `Url` 프로퍼티는 `VariableValue`를 고유한 방법으로 식별합니다.
- `Access` [문자열] — `Variable`의 `Access` 권한. 사용 가능한 권한은 "Read", "Write", "Read/Write"입니다.
- `Quality` [int64] — `Variable`의 실제 품질 비트.
- `ErrorStatus` [int32] — `Variable`의 에러 상태 코드.
- `DataType` [문자열] — `Variable`의 데이터 타입. 변수 웹 서비스는 원형 타입 및 원형 타입의 배열을 지원합니다. 복잡한 데이터 타입에서는 직렬화된 XML 문자열 또는 JSON 문자열을 `Value` 프로퍼티의 값으로 사용하십시오.
- `Value` [문자열] — `Variable`의 `Value`의 문자열 버전.
- `Timestamp` [문자열] — `Variable` 타임스탬프를 Date Time 포맷으로 나타낸 문자열.
 - JSON — 세계시 1970년 1월 1일 오전 12시 이후 경과한 밀리초를 나타내는 직렬화된 문자열.
 - XML — RFC 3339에 따라 포맷된 문자열. 예: 2011-05-25T17:04:30

`VariableValueGroup` — 변수 업데이트 및 배치 읽기 변수를 얻는데 사용되며, 사용자가 생성하는 이 그룹은 다음 프로퍼티를 포함합니다:

- `Id` [문자열] — 그룹의 ID. `Id` 프로퍼티는 `VariableValueGroup`을 고유하게 식별합니다.
- `Timeout` [int32] — 비활성화로 인해 변수 연결이 닫히기까지 소요되는 시간(밀리초)입니다. 이 프로퍼티의 값은 -1로 기본 설정되어 있으며, 타임아웃 없음을 나타냅니다.
- `VariableValues` [`VariableValue` []] — `VariableValueGroup`에 포함된 `VariableValues`의 리스트. 이 프로퍼티는 연기도도록 기본 설정되어 있습니다. `$expand` 파라미터를 사용하여 이 프로퍼티 인라인을 나타내십시오.

공유 변수에 대한 기본적인 정보 찾기

필요한 권한: `VariableBrowse`

상응하는 `VariableObject`를 탐색하여 공유 변수의 이름, 타입 및 URL을 알 수 있습니다. `VariableObject`를 탐색하려면 다음 URI 포맷을 사용합니다: `http://[host]:[port]/nivvariable/VariableObjects('[dynamic api url]')`

예: GET `http://my_server:3580/nivvariable/VariableObjects('ni.var.psp://localhost')?format=json`



노트

명령 예는 JSON 응답만 반환합니다. XML 응답을 반환하려면 명령에서 `$format=json`을 삭제하십시오.

VariableObject의 Children 프로퍼티를 보려면 연기된 하위 URI를 따릅니다.

```
예: GET http://my_server:3580/nivariable/
VariableObjects('ni.var.psp://localhost')/Children?
$format=json
```

\$expand 파라미터를 사용하여 하위 아이টে을 얻기 위해 별도의 요청을 하지 않고도 Children 프로퍼티로 나타나는 인라인을 얻습니다. 예: GET http://my_server:3580/nivariable/VariableObjects('ni.var.psp://localhost')?\$format=json&\$expand=children

단 한 개의 VariableObject 프로퍼티만 얻으려면 다음 URI 포맷을 사용합니다:

```
GET http://[host]:[port]/nivariable/VariableObjects('[dynamic
api url]')/[specific property]
```

사용 가능한 프로퍼티는 Url, Type, Name입니다. 예를 들어, ni.var.psp://localhost의 type 프로퍼티를 얻으려면, 다음 URI를 사용합니다:

```
GET http://my_server:3580/nivariable/
VariableObjects('ni.var.psp://localhost')/type?$format=json
```

변수값 읽기

필요한 권한: VariableRead

공유 변수의 값을 읽으려면 VariableValue를 읽어야 합니다. 다음 URI 포맷을 사용하여 VariableValue를 읽습니다:

```
GET http://[host]:[port]/nivariable/VariableValues('[dynamic
api url]')
```

예를 들어 ni.var.psp://localhost/hvac/t01의 값을 읽으려면 다음 URI를 사용합니다.

```
GET http://my_server:3580/nivariable/
VariableValues('ni.var.psp://localhost/hvac/t01')?
$format=json
```



노트

JSON 응답에서 타임스탬프는 세계시 1970년 1월 1일 오전 12시 이후 경과한 밀리초로 나타납니다.

타임스탬프를 사용하지 않고 값 프로퍼티를 읽으려면 다음 URI를 사용합니다:

```
GET http://my_server:3580/nivariable/
VariableValues('ni.var.psp://localhost/hvac/t01')/Value?
$format=json
```

변수에 쓰기

필요한 권한: VariableWrite

변수값을 변경하려면, 변수에 다음 데이터 포맷을 사용하는 요청 본문을 사용하여 PUT 요청을 합니다:

```
{ "Value" : "[new value]" }
```

예를 들어 `ni.var.psp://localhost/hvac/t01`의 값을 변경하려면 다음 URI를 사용합니다:

```
PUT http://my_server:3580/nivariable/  
VariableValues('ni.var.psp://localhost/hvac/t01')/Value
```



노트

변수의 값 프로퍼티는 변수가 문자열 또는 다른 타입인 경우에도 반드시 직렬화된 문자열 값이어야 합니다. 문자열 타입 변수에 쓸 때는 다음 포맷을 사용합니다: `"\"test\""`

헤더 상태 코드 204인 빈 응답 본문은 변경이 성공적으로 완료되었음을 나타냅니다. PUT 요청 데이터가 유효하지 않거나 변수에 쓸 수 없는 경우, 500 내부 에러 또는 400 유효하지 않은 요청 코드가 반환됩니다. 에러가 발생한 경우, 응답 본문은 요청에서 지정된 포맷으로 직렬화된 표준 OData 에러 객체를 포함합니다.

동시에 여러 변수에 쓰기

필요한 권한: VariableWrite

다음 포맷을 사용하여 `process1`의 `variable1`과 `variable2`를 업데이트합니다.

```
POST http://my_server:3580/nivariable/BatchWrite
```

POST JSON 데이터

```
[ {  
  "Url"  
  : "ni.var.psp://localhost/process1/variable1", "Value"  
  : "false"  
}, {  
  "Url"  
  : "ni.var.psp://localhost/process1/variable2", "Value"  
  : "\"this is a new string value\""  
}]
```

POST XML 데이터

```
<?xml version="1.0" encoding="utf-8"?>
<BatchWriteArguments>
<BatchWriteArgument>
<Url>ni.var.psp://localhost/process1/variable1</Url>
<Value>>false</Value>
</BatchWriteArgument>
<BatchWriteArgument>
<Url>ni.var.psp://localhost/process1/variable2</Url>
<Value>this is a new string value</Value>
</BatchWriteArgument>
</BatchWriteArguments>
```

변수에 쓰기가 모두 성공하면 서버는 다음 응답을 반환하며 이 응답의 응답 본문은 빈 배열로 이루어져 있습니다:

HTTP/1.1 200 OK

응답 본문(JSON)

```
[]
```

응답 본문(XML)

```
<?xml version="1.0" encoding="utf-8"?>
<BatchWriteErrors>
</BatchWriteErrors>
```

한 개 이상의 변수 쓰기에 실패하는 경우, 응답은 다음과 같은 에러 설명을 포함합니다. 예를 들어, variable2에 쓰지 못한 경우, 서버는 다음을 반환합니다:

HTTP/1.1 200 OK

응답 본문(JSON)

```
[ {
  "Url"
  : "ni.var.psp://localhost/process1/variable2", "Error"
  : {
    "code"
    : "", "message"
    : {
      "lang"
      : "en-us", "value"
```

```

        : "ni.var.psp://localhost/process1/variable2 is not
writable"
    }
}]]

```

응답 본문(XML)

```

<?xml version="1.0" encoding="utf-8"?>
<BatchWriteErrors>
<BatchWriteError>
<Url><ni.var.psp://localhost/process1/variable2</Url>
<error xmlns="http://schemas.microsoft.com/ado/
2007/08/dataservices/metadata">
<code>0</code>
<message xml:lang="en-US">"ni.var.psp://localhost/
process1/variable2 is not writable"</message>
</error>
</BatchWriteError>
</BatchWriteErrors>

```



노트

각 배치는 선택한 모든 쓰기 가능한 변수에 쓰고, 모든 쓰기 시도 실패에 대한 에러를 반환합니다. 쓰기 시도에 한번 실패해도 배치 쓰기 전체가 정지되지는 않습니다.

변수 그룹 생성하기

필요한 권한: VariableCreateDeleteGroup

변수값 그룹은 변수값의 사용자 정의된 리스트를 참조합니다. 한 번의 요청으로 이러한 변수의 값을 읽을 수 있습니다. \$delatoken 기능을 사용하여 업데이트된 변수값만 얻을 수 있습니다.

변수 그룹을 사용하려면 먼저 그룹을 생성해야 합니다. 그룹 생성 예는 다음과 같습니다:
POST http://my_server:3580/nivariable/VariableValueGroups

POST JSON 데이터:

```

HTTP Header - Content-Type : application/jsonHTTP Body -{
  "VariableValues"
: [{ "__metadata" : {
    "uri"
: "http://my_server:3580/nivariable/

```

```
VariableValues('ni.var.psp://localhost/process1/
variable1')"} }], { "_metadata" : {

    "uri"

    : "http://my_server:3580/nivariable/
VariableValues('ni.var.psp://localhost/process2/
variable2')"} } ] }
```

POST XML 데이터:

```
HTTP Header - Content-Type : application/xmlHTTP Body -<?
xml version="1.0" encoding="utf-8"?><entry
xmlns:d="http://my_server:3580/nivariable/"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/
dataservices/metadata" xmlns="http://www.w3.org/2005/
Atom"> <title type="text"></title>
<updated>2010-02-27T21:36:47Z</updated> <author> <name />
</author> <category term="nivariable.VariableValueGroup"
scheme="http://schemas.microsoft.com/ado/2007/08/
dataservices/scheme" /> <link rel="http://
schemas.microsoft.com/ado/2007/08/dataservices/related/
VariableValues" href="VariableValues('ni.var.psp://
localhost/process1/variable1')"/> <link rel="http://
schemas.microsoft.com/ado/2007/08/dataservices/related/
VariableValues" href="VariableValues('ni.var.psp://
localhost/process1/variable2')"/> <content
type="application/xml"> <m:properties> <d:Timeout>15000</
d:Timeout> </m:properties> </content></entry>
```



노트

변수값 그룹의 리소스를 생성할 때 요구되는 프로퍼티는 포함하지 않아도 됩니다. 변수값 그룹의 ID 프로퍼티는 일반적으로 요구되는 프로퍼티이고, OData에서 생성되는 리소스에는 요구되는 프로퍼티가 반드시 포함되어야 합니다. 그러나 어떤 ID가 사용 가능한지 알 수 있는 방법은 없습니다. OData 클라이언트가 요청을 보내기 위해 ID 프로퍼티의 값을 요구하는 경우, 임의의 값을 할당할 수 있습니다. 변수 웹 서비스는 사용자가 제공한 ID 프로퍼티 값을 고유한 값으로 대체합니다.

VariableValueGroup이 생성되면 서버는 201 응답을 반환하고 새로 생성한 리소스의 URI를 나타내는 위치 헤더를 추가합니다. 응답 본문에는 새로 생성된 그룹의 ID가 포함되어 있습니다.

변수값 그룹이 생성되지 않는 경우, 서버는 다음 응답 중 하나를 반환합니다:

- 500 내부 서버 에러—서버에서 에러가 발생했습니다.
- 400 유효하지 않은 요청—요청된 POST 데이터가 유효하지 않습니다.

응답 본문은 표준 OData 에러 객체를 사용하여 실패의 원인을 나타냅니다.

변수 그룹 삭제하기

필요한 권한: VariableCreateDeleteGroup

그룹을 삭제하려면 변수 그룹의 ID에 DELETE 요청을 보냅니다. 예는 다음과 같습니다:

```
DELETE http://my_server:3580/nivariable/
VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')
```

DELETE 요청이 성공하면, 204 상태 코드가 본문이 없는 응답과 함께 반환됩니다. DELETE 요청이 실패하면, 에러를 상세히 설명하는 OData 에러 객체가 요청된 포맷(XML 또는 JSON)으로 반환됩니다.

변수 그룹 읽기

필요한 권한: VariableReadGroup

VariableValueGroup을 생성한 후, 그룹 ID를 사용하여 다른 리소스와 마찬가지로 그룹을 읽을 수 있습니다. 예: GET http://my_server:3580/nivariable/VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')? \$format=json

그룹에 포함되어 있는 VariableValues를 읽으려면, 연기된 VariableValues URI를 요청합니다:

```
GET http://my_server:3580/nivariable/
VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')/
VariableValues?$format=json
```

\$expand 옵션을 사용하여 그룹 응답에 VariableValues를 포함할 수 있습니다. 예: GET http://my_server:3580/nivariable/VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')? \$expand=VariableValues&\$format=json

변수 업데이트 얻기

업데이트를 받으려면 HTTP 클라이언트는 반드시 수동으로 풀해야 합니다. HTTP는 이벤트 발생 또는 변경을 HTTP 클라이언트에게 알리는데 사용하는 서버 Push 메소드를 제공하지 않습니다. 풀 수행 시 불필요한 트래픽을 감소하기 위해 OData는 변경 또는 추가된 아이템만 반환하는 \$deltatoken 파라미터를 제공합니다.

VariableValueGroups 그룹에서 \$deltatoken 파라미터는 마지막 요청 이후 변경되지 않은 변수값을 건너뛰고 업데이트된 변수값만 제공합니다.

이 과정은 다음과 같은 방법으로 작동합니다:

우선, HTTP 클라이언트는 그룹의 변수값 프로퍼티를 읽습니다.


```
GET http://my_server:3580/nivariable/
VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')/
VariableValues?$format=json
```

HTTP 클라이언트가 초기 데이터 세트를 받습니다. HTTP 클라이언트는 `_delta` 프로퍼티가 제공한 다음 URI를 사용하여 델타 변경을 요청하며, 이 때 변수값의 타임스탬프는 `$deltatoken`입니다.

```
GET http://my_server:3580/nivariable/
VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')/
VariableValues?$format=json&$deltatoken=1305692428170
```

이전 GET에서 아무 것도 변경되지 않은 경우, 결과는 비어있습니다. 변경사항을 얻으려면 마지막 응답에서 지정된 델타 URI를 따릅니다.

```
GET http://my_server:3580/nivariable/
VariableValueGroups('1587dcd4a87d118ae3ddd0fce09246f6')/
VariableValues?$format=json&$deltatoken=1305692428170
```



노트

변경되지 않은 델타값을 요청해도 HTTP 클라이언트에 에러가 발생하지 않습니다.

외부 I/O 또는 다른 공유에 연결된 변수를 정해진 주기 안에 풀하기

공유 변수가 외부 I/O 또는 다른 공유 변수에 연결된 경우, 값은 `deltatoken` 토큰보다 오래된 타임스탬프를 기준으로 업데이트됩니다. 예를 들어, 시간 `t`에서 델타 변경을 요청하고 값이 `t+1`에서 새로 변경되지만 이때 기준이 되는 타임스탬프는 `t-1`입니다. 다음에 또 델타 변경을 얻을 때 서버는 `t-1` 타임스탬프를 새 값으로 간주하지 않습니다. 이 문제를 방지하려면 `$delta` 링크를 사용하는 대신 정해진 주기 안에 변수값을 풀하십시오.

여러 공유 변수 편집하기

여러 변수 편집기 윈도우를 사용하여 한 번에 많은 수의 공유 변수를 설정할 수 있습니다. 예를 들어, 여러 공유 변수의 데이터 타입을 한꺼번에 변경할 수 있습니다. 다음의 단계를 따라 여러 배정도 변수를 단정도 변수로 변경합니다.

1. 프로젝트 라이브러리 또는 공유 변수에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 여러 변수 편집기를 선택하여 여러 변수 편집기 윈도우를 디스플레이합니다.
2. 데이터 타입 열에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 오름차순으로 정렬을 선택합니다.
3. 데이터 타입이 배정도인 셀을 모두 선택합니다.
4. 배정도에서 클릭한 후 메뉴에서 단정도를 선택합니다.
5. 완료 버튼을 클릭하여 여러 변수 편집기 윈도우를 닫습니다.
6. 프로젝트 탐색기 윈도우에서, 공유 변수가 속하는 라이브러리를 저장하여 설정사항을 저장합니다.

연결 변수 생성하기

다음 단계에 따라 여러 연결 변수를 동시에 생성합니다.

1. 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 연결 변수 생성을 선택하여 연결 변수 생성 대화 상자를 디스플레이합니다.
2. 소스 탐색 풀다운 메뉴를 사용하여, 공유 변수의 데이터 소스를 구분합니다.
3. 프로젝트 아이템 트리 또는 네트워크 아이템 트리에 있는 **NI Publish-Subscribe Protocol (NI-PSP)** 데이터 아이템을 탐색합니다.
4. 현재 프로젝트 라이브러리에 추가하려는 공유 변수를 선택한 후, 추가 버튼을 클릭합니다. LabVIEW는 사용자가 선택하는 공유 변수를 기반으로 새 연결 공유 변수를 생성합니다.
5. 확인 버튼을 클릭하여, 연결 변수 생성 대화 상자를 닫고 여러 변수 편집기 윈도우를 디스플레이합니다.
6. (옵션) 여러 변수 편집기 윈도우에서 공유 변수를 편집합니다.
7. 완료 버튼을 클릭합니다. 연결 공유 변수는 프로젝트 탐색기 윈도우에서 사용자가 선택한 프로젝트 라이브러리 아래에 나타납니다.
8. 파일 >> 저장을 선택하여 프로젝트 라이브러리의 변경사항을 저장합니다.

공유 변수 설정을 스프레드시트에서 반입하기

공유 변수는 데이터 타입 및 변수 타입과 같은 설정 정보를 포함합니다. 다음 단계에 따라 스프레드시트로부터 공유 변수 설정 정보를 반입합니다.



노트

문자열 공유 변수의 초기값을 활성화하는 경우, 스프레드시트에서 이 값을 지정하도록 하십시오. 초기값이 활성화되었지만 설정되지 않았거나 비어있는 문자열 공유 변수를 반입하는 경우, LabVIEW는 에러를 반환합니다.

1. 공유 변수 설정을 .csv 파일에 **반출**합니다.
2. 스프레드시트 어플리케이션에서, 생성한 .csv 파일을 열고 프로퍼티를 수정합니다.
3. LabVIEW에서, 설정을 반입하려는 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 변수 반입을 선택합니다.
4. 반입하려는 CSV 대화 상자에서 .csv 파일을 선택하고 확인 버튼을 클릭합니다.



노트(DSC Module) 공유 변수에 대한 설명이 라인 브레이크를 포함하는 상태에서 해당 공유 변수에 대한 설정 정보를 스프레드시트로 반출하면 라인 브레이크는 스페이스로 변환됩니다. 이 공유 변수 설정 정보를 다시 프로젝트 라이브러리로 반입하면, 그 결과로 나타나는 설명은 라인 브레이크가 아닌 스페이스를 포함합니다.

공유 변수 설정을 스프레드시트에 반출하기

공유 변수는 데이터 타입 및 변수 타입과 같은 설정 정보를 포함합니다. 다음 단계에 따라 공유 변수 설정 정보를 스프레드시트 파일에 반출합니다.

1. 프로젝트 라이브러리에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 변수 반출을 선택합니다.
2. 공유 변수 설정을 반출할 때 사용하려는 .csv 스프레드시트 파일의 이름을 입력합니다.
3. 스프레드시트 어플리케이션에서, 생성한 .csv 파일을 엽니다.



노트

(DSC Module) 공유 변수에 대한 설명이 라인 브레이크를 포함하는 상태에서 해당 공유 변수에 대한 설정 정보를 스프레드시트로 반출하면 라인 브레이크는 스페이스로 변환됩니다. 이 공유 변수 설정 정보를 다시 프로젝트 라이브러리로 반입하면, 그 결과로 나타나는 설명은 라인 브레이크가 아닌 스페이스를 포함합니다.

숫자형 데이터

숫자형 변환

출력을 다른 숫자형을 가진 대상에 연결한 경우, LabVIEW는 다음 원칙에 따라서 데이터를 변환합니다:

- 부호있는 또는 부호없는 정수를 부동소수로—LabVIEW는 데이터를 가장 가까운 값으로 변환합니다. 이 변환은 부동소수가 정수보다 더 정밀할 때 정확합니다. 정수가 부동소수보다 더 정밀할 경우, LabVIEW는 변환에서 최대한 정밀도를 유지합니다.
- 부동소수를 부호있는 또는 부호없는 정수에 연결—LabVIEW는 범위를 벗어나는 값을 정수의 최대 또는 최소값으로 강제변환합니다. 예를 들어, 음의 부동소수를 부호없는 정수로 변환하는 경우, 결과는 0입니다.
- LabVIEW는 열거형을 부호없는 정수로 취급합니다—예를 들어, 부동소수 -1을 열거형으로 변환하는 경우, LabVIEW는 이 값을 열거형 범위 안으로 강제 변환합니다. 열거형의 범위가 0 ~ 25인 경우, LabVIEW는 이 값을 0으로 변환합니다.
- 정수를 정수에 연결—LabVIEW는 범위를 벗어나는 값을 정수의 최대 또는 최소값으로 강제 변환하지 않습니다. 소스가 대상보다 더 작은 경우, LabVIEW는 부호있는 소스의 부호를 확장하고 부호없는 소스의 나머지 비트를 제로로 채웁니다. 소스가 대상보다 더 큰 경우, LabVIEW는 값의 최하위 비트만 복사합니다.
- 정수, 부동소수, 또는 고정 소수점 수를 고정 소수점 수에 연결—LabVIEW는 고정 소수점 수에서 원하는 범위 밖의 값을 최대값과 최소값 범위내로 강제 변환시킵니다.

동일한 데이터 타입을 입력으로 요구하는 숫자형 함수에 서로 다른 숫자 데이터 타입을 연결할 경우, LabVIEW는 터미널 중 하나를 다른 터미널과 같은 형으로 변환합니다.

LabVIEW는 더 많은 비트를 사용하는 형을 선택합니다. 비트 수가 같은 경우, LabVIEW는 부호있는 쪽보다 부호없는 쪽을 선택합니다.



노트

(FPGA Module) FPGA VI에서 발생하는 강제변환은 상당한 FPGA 리소스를 소모할 수 있으며 특히 터미널이 SGL 데이터 타입으로 또는 SGL 데이터 타입에서 변환될 때 소모가 큼니다.

LabVIEW는 블록다이어그램의 노드에 **강제 변환점**을 놓아, LabVIEW가 해당 노드로 전달된 값을 다른 형으로 강제 변환하였음을 나타냅니다. **일부 강제 변환의 경우**, 강제 변환점은 메모리 사용 및 실행 시간의 증가를 가져오고, 결과의 정확도를 떨어뜨릴 수 있습니다. 강제 변환점을 제거하려면, 컨트롤 또는 상수의 출력 데이터 타입을 데이터를 받는 터미널의 데이터 타입에 일치시킵니다. 블록다이어그램의 컨트롤 또는 상수에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 형을 선택하여 입력값의 형을 터미널의 형으로 변경합니다.

숫자형 변경하기

LabVIEW가 데이터를 강제 변환하는 횟수를 제한하여 메모리를 절약하기 위해서 숫자형 객체의 형을 변경합니다.

기본적으로, 숫자형 상수의 형은 입력하는 상수의 값을 자동으로 채택합니다. 예를 들어, 1의 값을 갖는 숫자형 상수의 기본 형은 32 비트 정수입니다. 상수를 1.1로 변경할 경우, 상수의 형은 배정도 부동소수로 바뀝니다.

다음 단계를 따라 숫자형 객체의 형을 변경합니다.

1. 숫자형 객체에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 형을 선택합니다.
3. 데이터를 가장 잘 표현하는 **데이터 타입**을 선택합니다.



노트

슬라이드와 회전식 컨트롤 및 인디케이터는 복소수를 표현할 수 없습니다.

바로 가기 메뉴를 사용하여 객체의 형을 변경한 경우, 객체는 입력한 값에 관계없이 지정된 형을 유지합니다.

나누기, 사인, 코사인과 같은 일부 함수는 항상 부동소수 출력을 만듭니다. 정수를 이 함수의 입력에 연결할 경우, 함수는 계산을 수행하기 이전에 정수를 배정도 부동소수로 변환합니다.

값에 따라 타입을 결정하기 위해서 상수를 리셋하려면, 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 입력한 데이터에 적용을 선택합니다.

컨트롤 참조 번호의 숫자형 변경하기

다음 단계를 따라 컨트롤 참조 번호의 숫자 형을 변경합니다.

1. 컨트롤 참조 번호 프런트패널 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 컨트롤 보이기를 선택하여 참조 번호 컨트롤을 참조된 객체로 디스플레이합니다.
2. 프런트패널 컨트롤 참조 번호에 나타난 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 형을 선택합니다.
3. 사용하려는 **데이터 타입**을 선택합니다.

컨트롤 참조 번호 아이콘으로 되돌아가기 위해서, 컨트롤 참조 번호 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이콘 보이기를 선택합니다.

숫자형 함수의 출력 셋팅 설정하기

숫자형 함수는 **다형성 함수**입니다. 다른 형의 둘 이상의 숫자 입력을 하나의 함수에 연결할 때, 일반적으로 함수는 더 크고 넓은 형으로 데이터를 반환합니다. 함수는 실행하기 전에 작은 형을 가장 넓은 형으로 강제 변환하며, LabVIEW는 변환이 일어난 터미널에 강제 변환점을 놓습니다.

그러나 숫자형 함수를 설정하여 지정한 데이터 타입을 반환할 수 있습니다. **고정 소수점 형**을 출력으로 선택하고 데이터 타입 페이지의 소스에 적용 확인란에서 확인 표시를 제거하는 경우, 출력의 인코딩 그리고 함수의 오버플로우 및/또는 반올림 조건 처리 방식을 수동으로 설정할 수 있습니다. 이 확인란에서 확인 표시를 제거하면, 함수의 출력 터미널에 **파란색 강제 변환점**이 나타납니다. 이런 경우, 출력 타입의 폭과 범위에 따라 데이터 손실 발생 여부가 결정됩니다.

출력 형 설정하기

다음 단계를 따라 숫자형 함수의 출력에 대한 형을 설정합니다.

1. 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여, 객체 프로퍼티 대화 상자를 디스플레이합니다.
2. 출력 설정 페이지에서, 형 섹션에 있는 데이터 타입 아이콘을 클릭한 후 바로 가기 메뉴에서 출력으로 사용할 데이터 타입을 선택합니다. 선택한 데이터 타입이 입력값의 데이터 타입과 일치하지 않으면, LabVIEW는 소스에 적용 확인란에서 확인 표시를 제거합니다.
3. 형을 FXP(고정 소수점)로 설정한 경우, 다음 섹션으로 계속 진행하십시오. 그렇지 않은 경우에는 확인 버튼을 클릭하여 대화 상자를 닫고 설정 셋팅을 적용합니다.

고정 소수점 출력의 셋팅 설정하기

다음 단계를 따라 고정 소수점 출력에 대한 셋팅을 설정합니다.

1. 숫자형 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여, 객체 프로퍼티 대화 상자를 디스플레이합니다.
2. 출력 설정 페이지에서, 형 섹션에 있는 데이터 타입 아이콘을 클릭한 후 바로 가기 메뉴에서 FXP(고정 소수점) 아이콘을 선택합니다. 고정 소수점 데이터 타입이 입력 값의 데이터 타입과 일치하지 않기 때문에, LabVIEW는 소스에 적용 확인란에서 확인 표시를 제거합니다.
3. 반올림 모드를 선택하여 함수의 **반올림 조건** 처리 방식을 지정합니다.
4. 오버플로우 모드를 선택하여 함수의 **오버플로우 조건** 처리 방식을 지정합니다.
5. (옵션) 다음 단계를 따라 출력의 인코딩을 설정합니다.
 1. 부호 있음 또는 부호 없음을 선택하여 부호 있는 정수 또는 부호 없는 정수를 나타낼지를 지정합니다.
 2. 워드 길이 필드에서, 출력의 값을 나타내는데 사용할 총 비트 수를 지정합니다.
 3. 정수부분 워드 길이 필드에서, 출력의 값을 나타내는데 사용할 정수 비트 수를 지정합니다.



노트

인코딩 지정은 범위 지정에 비해 성능에 미치는 영향이 작습니다. 범위 지정에 따른 성능의 영향은 최소입니다. 그러나 값을 FPGA와 같은 타겟에서 원하는 범위로 강제변환하는 VI를 실행하면 성능에 대한 영향이 클 수 있습니다.

1. (옵션) 다음 단계를 따라 출력의 범위를 설정합니다. 유효자리 줄이기를 오버플로우 모드로 선택하면, LabVIEW가 자동적으로 인코딩에 맞도록 범위를 설정하기 때문에 사용자가 출력의 범위를 설정할 수 없습니다.
 1. 최소값 필드에, 출력의 최소값을 입력합니다.
 2. 최대값 필드에, 출력의 최대값을 입력합니다.
 3. 증분 필드에, 범위 내의 숫자간 증가분을 입력합니다.



노트 두 셋팅이 서로에 대해 의존적이므로 사용자가 인코딩과 범위를 모두 설정할 수는 없습니다. 마지막으로 설정한 셋팅이 LabVIEW가 고정 소수점 수에서 사용하는 셋팅입니다.

1. (옵션) 오버플로우 상태 포함 확인란에 확인 표시를 하여 고정 소수점 출력에 **오버플로우 상태**를 포함시킬 수 있습니다. 숫자형 함수의 입력 중 하나에 오버플로우 상태가 포함되어 있으면, 이 확인란에 확인 표시가 되어 있는지 여부에 관계없이 고정 소수점 출력도 오버플로우 상태를 포함할 수 있습니다.
2. 확인 버튼을 클릭하여 대화 상자를 닫고 설정 셋팅을 적용합니다.

숫자형 데이터 타입

LabVIEW는 숫자형 데이터 타입을 부동소수, 고정 소수점 수, 정수, 부호없는 정수, 복소수로 나타냅니다. 이러한 숫자형 데이터 타입 사이의 차이는 데이터를 저장하는데 사용하는 비트 수와 나타내는 데이터 값입니다.

서로 얼마나 가까운지를 비교할 수 있습니다. 예를 들어, 두 부동소수의 차의 절대값이 정의된 허용오차보다 작은 경우, 두 숫자는 같은 것으로 가정합니다.

숫자 데이터 타입 비트, 자릿수, 범위에 대한 추가적인 정보는 [숫자형 데이터 타입 테이블](#)을 참조하십시오. 부동소수에는 3가지 타입이 있습니다.

	단정도(SGL)—단정도 부동소수는 32비트의 IEEE 단정도 포맷을 가집니다. 메모리 절감이 중요하고 숫자의 범위가 오버플로우될 것 같지 않은 경우 단정도 부동소수를 사용합니다.
	배정도(DBL)—배정도 부동소수는 64비트의 IEEE 배정도 포맷을 가집니다. 배정도는 숫자형 객체의 기본 포맷입니다. 대부분의 경우 배정도 부동소수를 사용합니다.
	확장형(EXT)—확장형 숫자를 디스크에 저장하면 LabVIEW는 이 숫자를 플랫폼에 독립적인 128비트 포맷으로 저장 합니다. 메모리에서, 크기와 정밀도는 플랫폼에 따라 다릅니다. 꼭 필요한 경우에만 확장형 부동소수를 사용합니다. 확장형 연산의 속도는 플랫폼에 따라 다릅니다.

데이터 수집 VI는 가끔 부동소수의 배열을 반환합니다.

정수

정수는 전체 숫자를 나타내며 양 또는 음의 값이 될 수 있습니다. 숫자 데이터 타입 비트, 자릿수, 범위에 대한 추가적인 정보는 [숫자형 데이터 타입 테이블](#)을 참조하십시오. 4가지 타입의 정수가 있습니다.

	바이트(I8)—바이트 정수는 8 비트의 스토리지를 가집니다.
	워드(I16)—워드 정수는 16비트의 스토리지를 가집니다.
	롱(I32)—롱 정수는 32비트의 스토리지를 가집니다. 대부분의 경우에, 32비트 정수를 사용하는 것이 가장 좋습니다.
	쿼드(I32)—쿼드 정수는 64비트의 스토리지를 가집니다.

LabVIEW가 부동소수를 정수로 변경할 때, VI는 가장 가까운 짝수로 반올림합니다. 예를 들어, LabVIEW는 2.5를 3으로, 3.5를 4로 반올림합니다. LabVIEW의 숫자 형 변환 방법에 대한 더 자세한 정보는 [숫자 변환](#)을 참조하십시오.

숫자형 데이터 타입 테이블

다음 테이블은 LabVIEW에서 사용 가능한 [숫자형 데이터 타입](#)을 디스플레이합니다. LabVIEW는 각 데이터 타입을 다른 방법으로 **저장**합니다.

터미널	숫자 데이터 타입	디스크의 스토리지 비트	대략적인 10진수의 자릿수	대략적인 범위
	단정도 부동소수	32	6	최소 양수: 1.40e-45 최대 양수: 3.40e+38 최소 음수: -1.40e-45 최대 음수: -3.40e+38
	배정도 부동소수	64	15	최소 양수: 4.94e-324 최대 양수: 1.79e+308 최소 음수: -4.94e-324 최대 음수: -1.79e+308
	확장형 부동소수	128	플랫폼에 따라 15에서 20로 다양합니다.	최소 양수: 6.48e-4966 최대 양수: 1.19e+4932 최소 음수: -e-324 최대 음수: -1.79e+308
	복소수 단정도 부동소수	64	6	각 부분(실수와 허수)이 단정도 부동소수와 같음
	복소수 배정도 부동소수	128	15	각 부분(실수와 허수)이 배정도 부동소수와 같음
	복소수 확장형 부동소수	256	플랫폼에 따라 15에서 20로 다양합니다.	각 부분(실수와 허수)이 확장형 부동소수와 같음
	고정 소수점	오버플로우 상태를 포함하는 경우 64 또는 72입니다.	사용자 설정에 따라 다양합니다.	사용자 설정에 따라 다양합니다.
	바이트 부호있는 정수	8	2	-128 ~ 127
	워드 부호있는 정수	16	4	-32,768 ~ 32,767
	롱 부호있는 정수	32	9	-2,147,483,648 ~ 2,147,483,647
	쿼드 부호있는 정수	64	18	-1e19 에서 1e19
	바이트 부호없는 정수	8	2	0 ~ 255
	워드 부호없는 정수	16	4	0 ~ 65,535
	롱 부호없는 정수	32	9	0 ~ 4,294,967,295

터미널	숫자 데이터 타입	디스크의 스토리지 비트	대략적인 10진수의 자릿수	대략적인 범위
	워드 부호없는 정수	64	19	0 ~ 2e19
	128 비트 타임스탬프	128	19	최소 시간: 01/01/1600 00:00:00 UTC 최대 시간: 01/01/3001 00:00:00 UTC

부호없는 정수

부호없는 정수는 음수 아닌 정수만을 나타내며 두 형 모두 비트 수가 같으므로 부호있는 정수보다 더 큰 범위의 양수를 가집니다. 숫자 데이터 타입 비트, 자릿수, 범위에 대한 추가적인 정보는 [숫자형 데이터 타입 테이블](#)을 참조하십시오. 부호없는 숫자에는 4가지 타입이 있습니다.

	바이트(U8)—바이트 부호없는 정수는 8비트의 스토리지를 가집니다.
	워드(U16)—워드 부호없는 정수는 16비트의 스토리지를 가집니다.
	롱(U32)—롱 부호없는 정수는 32비트의 스토리지를 가집니다.
	워드(U64)—워드 정수는 64비트의 스토리지를 가집니다.

고정 소수점 데이터

고정 소수점 수

고정 소수점 수는 사용자 정의한 범위와 정밀도에서의 유리수를 나타냅니다. 고정 소수점 수에 대해 1부터 64비트 사이의 모든 크기를 지정할 수 있습니다. 부호있음 또는 부호없음으로 고정 소수점 수를 설정할 수 있습니다.

숫자형 데이터 타입의 비트, 자릿수, 범위에 대한 더 자세한 정보는 [숫자형 데이터 타입 테이블](#)을 참조하십시오.

	고정 소수점—고정 소수점 수는 최대 64비트의 스토리지를 가집니다.



- 관련 정보
 - [숫자형 데이터](#) (61 페이지 참조)
 - [고정 소수점 수 설정하기](#) (1111 페이지 참조)
 - [숫자형 데이터 타입 테이블](#) (1108 페이지 참조)

고정 소수점 수 설정하기

숫자형 컨트롤, 상수, 또는 인디케이터의 **고정 소수점 수**에 대한 **인코딩**을 설정할 수 있습니다.

고정 소수점 컨트롤, 상수 또는 인디케이터를 설정하면, 객체는 지정한 셋팅에 맞지 않는 숫자를 디스플레이할 수 없습니다. 설정한 객체가 인디케이터인 경우, LabVIEW는 모든 입력 값을 인디케이터의 고정 소수점 설정 셋팅에 부합하는 인디케이터로 강제 변환합니다.

컨트롤 설정하기

실행 단계에 따라 숫자형 컨트롤의 고정 소수점 수를 설정합니다.

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 숫자형 프로퍼티 대화 상자를 디스플레이합니다.
2. 데이터 타입 페이지에서 형 섹션에 있는 데이터 타입 아이콘을 클릭한 후 바로 가기 메뉴에서 FXP(고정 소수점)를 선택합니다. 고정 소수점 설정 섹션은 인코딩과 범위 옵션에 대한 기본값을 디스플레이합니다.
3. (옵션) 다음 단계를 따라 고정 소수점 수의 인코딩을 설정합니다.
 1. 부호있음 또는 부호없음을 선택하여 부호있는 정수 또는 부호없는 정수를 나타낼 지를 지정합니다.
 2. 워드 길이 필드에서, 고정 소수점 수의 값을 나타내는데 사용할 총 비트 수를 지정합니다.
 3. 정수부분 워드 길이 필드에서, 고정 소수점 수의 값을 나타내는데 사용할 정수 비트의 수를 지정합니다.



노트

LabVIEW는 고정 소수점 수의 인코딩에서 지정한 값에 따라 **범위**를 계산합니다.

1. (옵션) 오버플로우 상태 포함 확인란에 확인 표시를 하여 고정 소수점 수에 **오버플로우 상태**를 포함시킬 수 있습니다.
2. 확인 버튼을 클릭하여 대화 상자를 닫고 설정 셋팅을 적용합니다.

상수 설정하기

실행 단계에 따라 숫자형 상수의 고정 소수점 수를 설정하십시오.

1. 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 숫자형 상수 프로퍼티 대화 상자를 디스플레이합니다.
2. 데이터 타입 페이지에서, 입력한 데이터에 적용 확인란에서 확인 표시를 제거합니다.



노트 입력한 데이터에 적용을 선택하면 LabVIEW는 가능한 가장 짧은 워드 길이와 정수부분 워드 길이에서 입력한 모든 값을 디스플레이합니다. LabVIEW가 고정 소수점 설정 셋팅에서의 변경사항을 유지하게하려면, 이 확인란에서 확인 표시를 제거해야 합니다.

1. 형 섹션의 데이터 타입 아이콘을 클릭한 후 바로 가기 메뉴에서 FXP(고정 소수점)를 선택합니다. 고정 소수점 설정 섹션은 범위와 인코딩 옵션에 대한 기본값을 디스플레이합니다.
2. (옵션) 다음 단계를 따라 고정 소수점 수의 인코딩을 설정합니다.
 1. 부호있음 또는 부호없음을 선택하여 부호있는 정수 또는 부호없는 정수를 나타낼 지를 지정합니다.
 2. 워드 길이 필드에서, 고정 소수점 수의 값을 나타내는데 사용할 총 비트 수를 지정합니다.
 3. 정수부분 워드 길이 필드에서, 고정 소수점 수의 값을 나타내는데 사용할 정수 비트의 수를 지정합니다.



노트 LabVIEW는 고정 소수점 수의 인코딩에서 지정한 값에 따라 고정 소수점 수의 **범위**를 계산합니다.

1. (옵션) 오버플로우 상태 포함 확인란에 확인 표시를 하여 고정 소수점 수에 **오버플로우 상태**를 포함시킬 수 있습니다.
2. 확인 버튼을 클릭하여 대화 상자를 닫고 설정 셋팅을 적용합니다.

인디케이터 설정하기

실행 단계에 따라 숫자형 인디케이터의 고정 소수점 수를 설정하십시오.

1. 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 숫자형 프로퍼티 대화 상자를 디스플레이합니다.
2. 데이터 타입 페이지에서 형 섹션에 있는 데이터 타입 아이콘을 클릭한 후 바로 가기 메뉴에서 FXP(고정 소수점)를 선택합니다. 고정 소수점 설정 섹션은 범위와 인코딩 옵션에 대한 기본값을 디스플레이합니다.
3. (옵션) 값이 입력 고정 소수점 값의 고정 소수점 설정 셋팅을 상속받게 하려면 소스에 적용 확인란에 확인 표시를 합니다. 이 옵션을 선택한 경우, 단계 7으로 건너뛰십시오.
4. (옵션) 다음 단계를 따라 고정 소수점 수의 인코딩을 설정합니다.
 1. 부호있음 또는 부호없음을 선택하여 부호있는 정수 또는 부호없는 정수를 나타낼 지를 지정합니다.
 2. 워드 길이 필드에서, 고정 소수점 수의 값을 나타내는데 사용할 총 비트 수를 지정합니다.
 3. 정수부분 워드 길이 필드에서, 고정 소수점 수의 값을 나타내는데 사용할 정수 비트의 수를 지정합니다.



노트 LabVIEW는 고정 소수점 수의 인코딩에서 지정한 값에 따라 고정 소수점 수의 범위를 계산합니다.

1. (옵션) 오버플로우 상태 포함 확인란에 확인 표시를 하여 고정 소수점 수에 **오버플로우 상태**를 포함시킬 수 있습니다.
2. 확인 버튼을 클릭하여 대화 상자를 닫고 설정 셋팅을 적용합니다.

고정 소수점 수 사용에 대한 유의점과 권장사항

다음 리스트는 고정 소수점 수를 사용할 때 알아야 할 유의점과 권장사항들을 설명합니다.

- 시프트 레지스터를 고정 소수점 데이터 크기를 증가시키는 숫자형 함수가 있는 루프에 추가하면 타입 정보에 사이클을 생성할 수 있습니다. 데이터가 증가하면서 LabVIEW가 허용하는 최대 워드 길이 또는 사용자가 지정하는 최대 워드 길이에 가까워지면 오버플로우가 발생할 수 있습니다. 오버플로우를 처리하는 방식을 숫자형 함수가 설정하지 않은 경우, LabVIEW는 데이터를 부동소수로 변환하여 가능한 데이터 손실과 VI 깨짐을 방지합니다.
- 지정한 워드 길이가 타겟이 허용하는 최대 워드 길이를 초과할 수 있습니다. 이런 경우가 발생하면, LabVIEW는 워드 길이를 가능한 최대 길이로 설정하고 오버플로우 및 반올림에 대해 기본 셋팅을 사용합니다. 기본적으로, 워드 길이는 최상위 비트를 유지합니다. 이러한 워드 길이 감소로 값이 반올림되지만 오버플로우는 방지됩니다. 타겟이 허용하는 최대 워드 길이에 대한 정보는 각 타겟에 대한 문서를 참조하십시오.
- LabVIEW 및 다른 어플리케이션은 고정 소수점 수를 밑이 2인 시스템인 2진수 포맷으로 인코딩합니다. 십진수 시스템은 밑이 10이므로, 일부 십진수 값은 고정 소수점에서 나타낼 수 없습니다. 예를 들어, 0.1은 십진수 포맷에서 정확하게 표현할 수 있지만, 32 비트 워드 길이와 16 비트 정수부분 워드 길이에서 0.1을 고정 소수점 수로 변환하면, LabVIEW는 유리 함수를 밑이 2인 시스템에서 나타낼 수 있도록 0.100006으로 변경합니다.



관련 정보

[숫자형 데이터](#) (61 페이지 참조)

[루프 반복 사이에서 값 전달하기](#) (369 페이지 참조)

[시프트 레지스터가 있는 루프에서 고정 소수점 데이터의 사이클에 적용하기](#) (1114 페이지 참조)

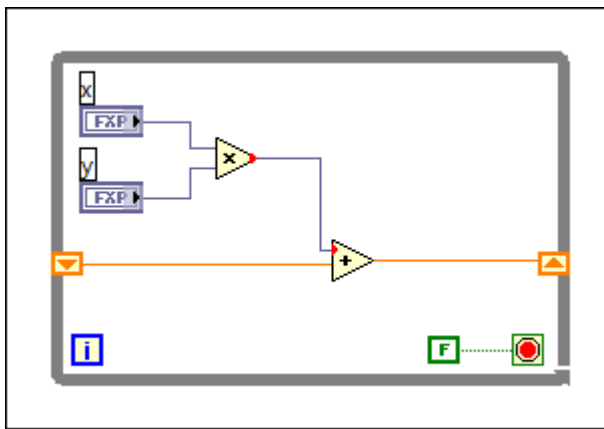
[숫자형 함수의 출력 셋팅 설정하기](#) (1105 페이지 참조)

[숫자형 데이터](#) (61 페이지 참조)

시프트 레지스터가 있는 루프에서 고정 소수점 데이터의 사이클에 적용하기

곱셈 또는 증가처럼 **고정 소수점 데이터**의 크기를 증가시키는 작업을 루프내에서 수행하고, 작업의 결과를 시프트 레지스터에 전달하는 경우, LabVIEW는 출력 데이터를 고정 소수점 형으로 변환하여 **고정 소수점 데이터의 오버플로우**로 인해 발생가능한 데이터 손실을 방지합니다. 루프내에서 수행하는 작업에 따라, 매번 루프가 반복될 때마다 고정 소수점 값의 워드 길이가 지수적으로 증가할 수 있기 때문에 오버플로우가 발생할 수 있습니다.

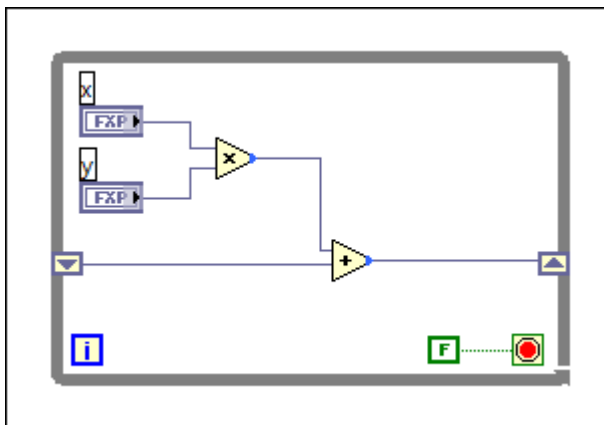
다음 블록 다이어그램에서 나타나는 오렌지 와이어는 LabVIEW가 자동으로 고정 소수점 데이터를 유동소수로 변환하는 부분입니다. 강제 변환점은 LabVIEW가 숫자형 함수의 출력값 비트를 제거하는 중임을 나타냅니다.



LabVIEW가 자동으로 데이터를 유동소수로 변환하지 않게 하려면, 시프트 레지스터로부터 데이터를 처리하는 함수의 **출력 셋팅을 설정**하여 출력 데이터 타입과 함수가 오버플로우 발생시 이를 처리하는 방식을 지정해야 합니다.

예를 들어, 이전의 블록 다이어그램에서, 더하기 함수를 설정하여 고정 소수점 값만을 출력하도록 설정할 수 있습니다. 출력 데이터 타입을 고정 소수점으로 설정한 후, 블록 다이어그램의 오렌지색 와이어의 색이 변경되어 고정 소수점 데이터가 지나가고 있음을 나타냅니다.

다음의 이미지는 고정 소수점 값을 출력토록 추가 함수를 설정한 이후의 블록 다이어그램의 모양을 나타냅니다. 숫자형 함수의 강제 변환점은 이 함수에서 출력 셋팅을 설정했음을 나타냅니다.



고정 소수점 수의 오버플로우 상태 LED 보이기 및 숨기기

오버플로우 상태를 포함하도록 **고정 소수점 컨트롤, 상수, 또는 인디케이터를 설정**한 이후, 컨트롤, 상수, 인디케이터의 오버플로우 상태 LED를 디스플레이하고 숨길 수 있습니다. 이 LED는 고정 소수점 수가 **오버플로우**된 결과인지를 나타냅니다. 다음 단계를 따라 오버플로우 상태 LED를 보여주거나 숨깁니다.

1. 오버플로우 상태를 포함하도록 **고정 소수점 컨트롤, 상수 또는 인디케이터를 설정**합니다.
2. 프론트패널의 고정 소수점 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하거나, 블록 다이어그램의 고정 소수점 상수에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 보이는 아이템 >> 오버플로우 상태 LED를 선택합니다.



노트

컨트롤, 상수 또는 인디케이터에 오버플로우 상태가 포함되도록 설정하지 않으면, 오버플로우 상태 LED 바로 가기 메뉴가 나타나지 않습니다.

고정 소수점 데이터 타입을 지원하는 VI와 함수

LabVIEW에 있는 모든 VI와 함수가 고정 소수점 데이터 타입을 지원하는 것은 아닙니다. 고정 소수점 데이터 타입을 지원하지 않는 VI 또는 함수에 고정 소수점 수를 와이어로 연결하는 경우, VI는 실행 버튼이 깨진 채로 보입니다. 다음의 VI와 함수는 고정 소수점 데이터 타입을 지원합니다.

배열 VI와 함수

다음 배열 VI는 고정 소수점 데이터 타입을 지원합니다.

- 배열 원소 감소
- 배열 원소 증가

타입 지정 VI 및 함수

다음 타입 지정 VI 및 함수는 고정 소수점 데이터 타입을 지원합니다.

- 고정 소수점 숫자 타입 지정
- 소수 타입 지정
- 실수 또는 웨이브폼 타입 지정
- 실수 타입 지정
- 스칼라 숫자 또는 웨이브폼 타입 지정
- 스칼라 숫자 타입 지정
- 구조 타입 일치 지정

- 구조 타입 불일치 지정

비교 VI와 함수

다음 비교 VI와 함수는 고정 소수점 데이터 타입을 지원합니다.

- 같음?
- 0과 같음?
- 크거나 같음?
- 0보다 크거나 같음?
- 보다 큼?
- 0보다 큼?
- 범위내 확인과 강제변환
- 값 변경?
- 작거나 같음?
- 0보다 작거나 같음?
- 보다 작음?
- 0보다 작음?
- 최대 및 최소
- 같지 않음?
- 0과 같지 않음?

변환 함수

다음의 변환 함수는 고정 소수점 데이터 타입을 지원합니다:

- 불리언 배열을 숫자로
- 타입으로 강제 변환
- 숫자를 불리언 배열로
- 바이트 정수로
- 배정도 부동소수로
- 확장형 정밀도 부동소수로
- 고정 소수점으로
- 롱 정수로
- 쿼드 정수로
- 단정도 부동소수로
- 부호없는 바이트 정수로
- 부호없는 롱 정수로
- 부호없는 쿼드 정수로

- 부호없는 워드 정수로
- 워드 정수로

데이터 다루기 함수

다음의 데이터 다루기 함수는 고정 소수점 데이터 타입을 지원합니다:

- 패턴화된 문자열로
- 논리적 이동
- 캐리와 왼쪽으로 회전
- 캐리와 오른쪽으로 회전
- 타입 캐스트
- 문자열로부터 패턴화 해제

LabVIEW 스키마 함수

다음의 LabVIEW 스키마 함수는 고정 소수점 데이터 타입을 지원합니다:

- 패턴화된 XML로
- XML로부터 패턴화 해제

숫자형 함수

다음의 숫자형 함수는 고정 소수점 데이터 타입을 지원합니다:

- 절대값
- 더하기
- 감소
- 나누기
- 증가
- 곱하기
- 음수화
- 역수
- 반올림
- 올림
- 버림
- 2의 거듭제곱으로 스케일
- 부호
- 제곱
- 제곱근

- 빼기

문자열/숫자 변환 함수

다음의 문자열/숫자 변환 함수는 고정 소수점 데이터 타입을 지원합니다:

- 10진수 문자열을 숫자로
- 소수점/지수 문자열을 숫자로
- 16진수 문자열을 숫자로
- 숫자를 10진수 문자열로
- 숫자를 공학 문자열로
- 숫자를 지수형 문자열로
- 숫자를 소수점형 문자열로
- 숫자를 16진수 문자열로
- 숫자를 8진수 문자열로
- 8진수 문자열을 숫자로

타이밍 VI와 함수

다음 타이밍 VI는 고정 소수점 데이터 타입을 지원합니다.

- 데이터 흐름 지연



관련 정보

[고정 소수점 수](#) (1110 페이지 참조)

배리언트 데이터

배리언트 데이터

때로는 VI가 다양한 타입의 데이터를 보편적인 방법으로 처리해야 하는 경우도 있습니다. 특정 데이터 타입에 대해 각각 서로 다른 VI를 작성할 수 있지만, 한 VI를 변경하면 나머지 VI도 모두 변경할 필요가 있기에 여러 개의 사용자 VI의 복사본을 보유하면 관리하기 어렵습니다. 이러한 상황에 대한 해결책으로 LabVIEW에는 배리언트 데이터 타입이 있습니다. 배리언트 데이터 타입은 LabVIEW의 모든 데이터 타입을 위한 일반적인 컨테이너입니다. 다른 데이터를 배리언트로 변환할 때 배리언트는 데이터 및 원래 데이터 타입을 저장하여 LabVIEW가 차후에 배리언트 데이터를 원래 데이터로 올바르게 변환시킬 수 있습니다. 예를 들어, 문자열 데이터를 배리언트로 변환하는 경우에는 배리언트가 문자열의 텍스트와 함께 해당 데이터가 원래 문자열이었음(경로, 바이트 배열, 기타 LabVIEW 데이터 타입이 아니었음)을 알리는 인디케이터를 저장하게 됩니다.

[배리언트] 함수를 사용하여 배리언트 데이터를 생성하고 조작합니다. 배리언트 데이터로 패턴화된 데이터를 변환하고 LabVIEW 데이터를 변환할 수 있습니다. LabVIEW 데이터로 배리언트 데이터를 변환할 수 있습니다. 데이터 타입 분석 VI를 사용하여 배리언트에 저장된 데이터 타입을 불러오고 다른 데이터 타입과 비교합니다.

배리언트 데이터 타입을 사용하여 데이터의 속성을 저장할 수 있습니다. 속성은 사용자가 정의하고 배리언트 데이터 타입이 저장하는 데이터에 대한 정보입니다. 예를 들어, 일정 데이터가 언제 생성되었는지를 알려는 경우, 데이터를 배리언트 데이터로 저장하고 시간 문자열을 저장하기 위해서 시간으로 불리는 속성을 추가할 수 있습니다. 속성 데이터는 어떠한 타입도 될 수 있습니다. 또한, 배리언트 데이터에서 속성을 삭제하거나 가져올 수 있습니다.



관련 정보

[패턴화된 데이터를 배리언트 데이터로 변환하기](#) (1121 페이지 참조)

[LabVIEW 데이터를 배리언트 데이터로 변환하기](#) (1122 페이지 참조)

[배리언트 데이터를 LabVIEW 데이터로 변환하기](#) (1122 페이지 참조)

[배리언트 데이터에 속성 추가하기](#) (1119 페이지 참조)

[배리언트 데이터에서 속성 삭제하기](#) (1123 페이지 참조)

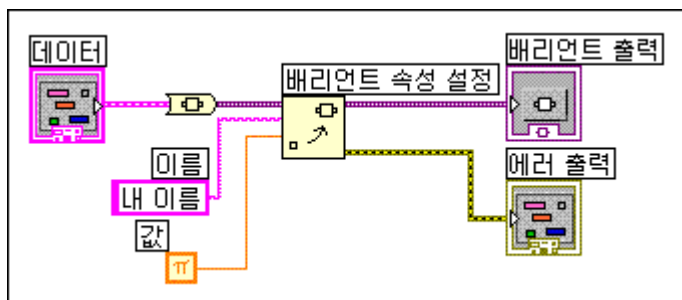
[배리언트 데이터에서 속성 가져오기](#) (1124 페이지 참조)

[배리언트 데이터 처리하기](#) (125 페이지 참조)

배리언트 데이터에 속성 추가하기

다음 단계를 따라 **배리언트 데이터**에 속성을 추가합니다.

1. 블록 다이어그램에 [배리언트 속성 설정] 함수를 추가합니다.
2. 배리언트 데이터를 [배리언트 속성 설정] 함수의 배리언트 입력에 연결합니다.
3. [배리언트 속성 설정] 함수의 이름 및 값 입력을 연결합니다.
4. [배리언트 속성 설정] 함수의 배리언트 출력 출력을 배리언트 데이터를 입력으로 받아들이는 VI 또는 함수에 연결합니다.
5. VI를 실행합니다. 블록 다이어그램의 모양이 다음 블록 다이어그램과 유사할 것입니다.



배리언트에서 특정 데이터 타입 포함 여부 확인하기

배리언트가 특정 데이터 타입을 포함하는지 확인하려면, 데이터 타입 분석 VI를 사용하여 데이터 타입 정보를 가져옵니다. [배리언트를 데이터로] 함수와 달리, 데이터 타입 분석 VI는 배리언트에 저장된 실제 데이터를 가져오지 않습니다. 대신 이 VI는 데이터 타입에 대한 정보를 반환하기 때문에, 데이터 타입에 대한 정보를 얻기 위해 저장된 데이터를 조작할 필요가 없습니다.

배리언트에 저장된 데이터 타입과 비교하여 확인하고 싶은 데이터 타입이 무엇인지 알면, 데이터 타입 분석 VI를 사용하여 이 데이터 타입을 비교하면 됩니다. 예를 들어, 어플리케이션이 임의의 프런트패널을 분석하여 컨트롤의 값을 가져오는 경우, 이 토픽의 예를 사용하여 에러 클러스터를 포함하는 배리언트 컨트롤을 식별할 수 있습니다.

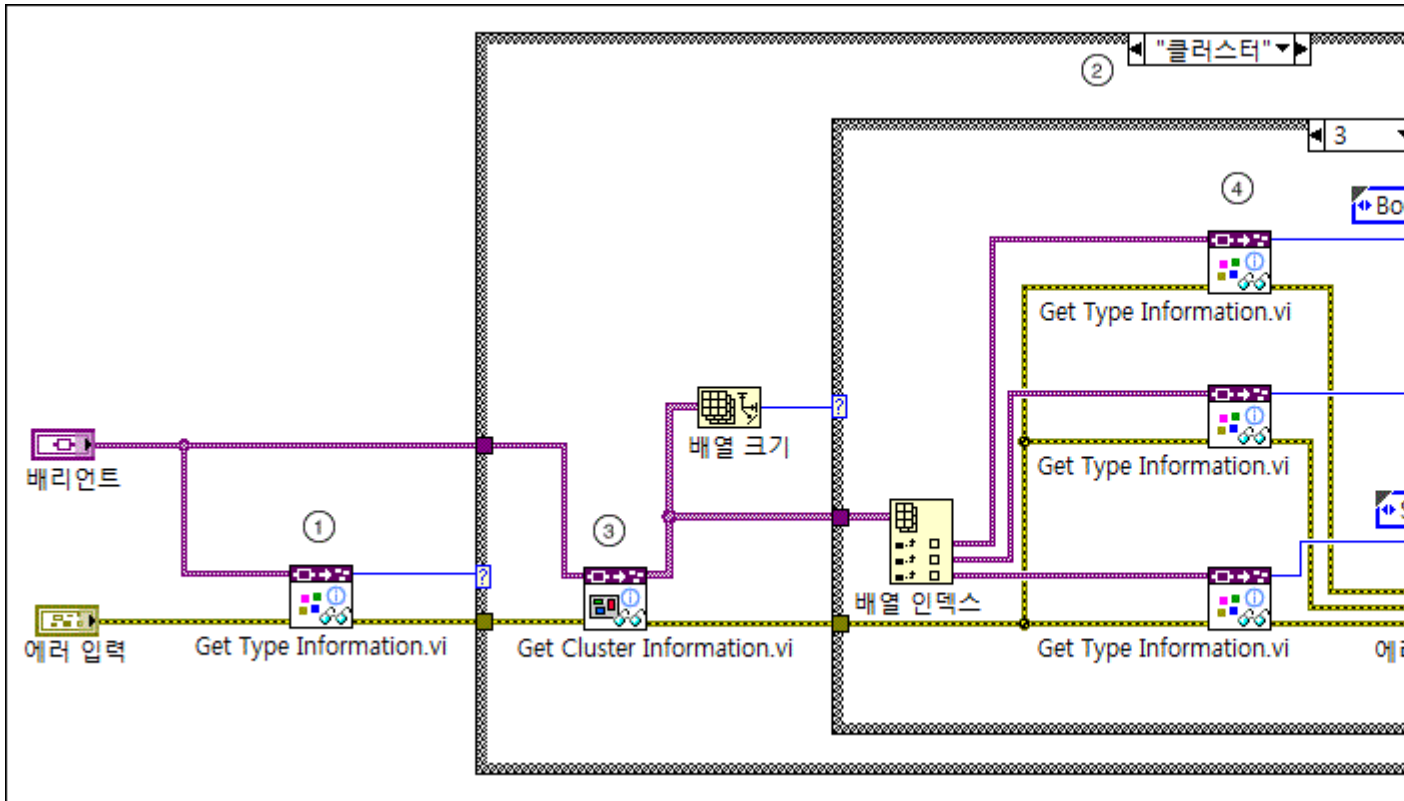
사용 객체

다음 블록 다이어그램 객체를 사용하여 배리언트에 저장된 데이터 타입 정보를 가져옵니다.

타입 정보 얻기	케이스 구조	[클러스터 정보 얻기] VI
		

다음 단계

다음 블록 다이어그램을 생성하여 알고 있는 데이터 타입을 배리언트가 포함하는지 확인합니다. 이 예에서는 배리언트에 저장된 데이터 타입을 에러 클러스터 원소와 비교하여 확인합니다.



위의 블록다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.

①	[타입 정보 얻기] VI를 사용하여 배리언트에 저장된 데이터 타입을 지정하는 열거형을 반환합니다. 이 예에서, 데이터 타입은 클러스터입니다.
②	비교하려는 데이터 타입과 대응되도록 케이스 구조의 선택자 라벨을 편집합니다. 이 예에서, 라벨과 데이터 타입은 모두 "클러스터"입니다.
③	[클러스터 정보 얻기] VI를 사용하여 클러스터 원소 데이터 타입을 포함하는 배열을 반환합니다. 배리언트에 포함된 데이터 타입이 무엇인지 확인되면, 데이터 타입 분석 팔레트에서 데이터 타입 특정 VI를 사용하여(예를 들어, [숫자형 정보 얻기] 또는 [고정 소수점 정보 얻기] VI) 보다 특정한 데이터 타입 정보를 반환할 수 있습니다.
④	확인된 에러 클러스터 데이터 타입의 원소 개수에 맞게 내부 케이스 구조의 선택자 라벨을 편집합니다.
⑤	[타입 정보 얻기] VI를 다시 사용하여 에러 클러스터의 각 세 원소에 대해 열거형을 반환합니다. 그런 다음 이 개별 원소의 데이터 타입 및 이 원소의 표준 순서를 확인하려는 데이터 타입 및 LabVIEW 에러 클러스터의 순서와 비교하면 됩니다.

패턴화된 데이터를 배리언트 데이터로 변환하기

다음 단계를 따라 [패턴화된 문자열로] 함수를 사용하여 패턴화한 데이터를 배리언트 데이터로 변환합니다.

1. 블록다이어그램에 [패턴화된 문자열을 배리언트로] 함수를 추가합니다.

- 2. 패턴화된 데이터의 데이터 타입을 나타내는 숫자 정수의 배열을 [패턴화된 문자열을 배리언트로] 함수의 타입 문자열 입력에 연결합니다.
- 3. 패턴화된 데이터 문자열을 [패턴화된 문자열을 배리언트로] 함수의 데이터 문자열 입력에 연결합니다.
- 4. [패턴화된 문자열을 배리언트로] 함수의 배리언트 출력을 배리언트 데이터를 입력으로 받아들이는 VI 또는 함수에 연결합니다.
- 5. VI를 실행합니다.

LabVIEW 데이터를 배리언트 데이터로 변환하기

배리언트 데이터 타입은 LabVIEW의 모든 데이터 타입을 위한 일반적인 컨테이너입니다. 다른 데이터를 배리언트로 변환할 때, 배리언트는 데이터 및 원래 데이터 타입을 저장합니다. 데이터 타입에 관계없이 데이터를 조작하는 것이 중요할 때 배리언트 데이터를 사용하십시오.

다음 단계를 따라 LabVIEW 데이터를 배리언트 데이터로 변환합니다.

- 1. 블록다이어그램에 [배리언트로] 함수를 추가합니다.
- 2. 데이터를 임의의 데이터 입력에 연결합니다.
- 3. [배리언트로] 함수의 배리언트 출력을 배리언트 데이터를 입력으로 받는 VI 또는 함수에 연결합니다.
- 4. VI를 실행합니다.

LabVIEW 데이터를 배리언트 데이터로 변환한 후, 데이터에 속성을 추가할 수 있습니다.

또한 배리언트 데이터를 LabVIEW 데이터로 변환할 수 있습니다.

배리언트 데이터를 LabVIEW 데이터로 변환하기

[배리언트를 데이터로] 함수를 사용하여 배리언트 데이터를 LabVIEW 데이터로 변환합니다. 배리언트를 변환하기 전에 배리언트에 저장된 데이터 타입을 확인해야 합니다.

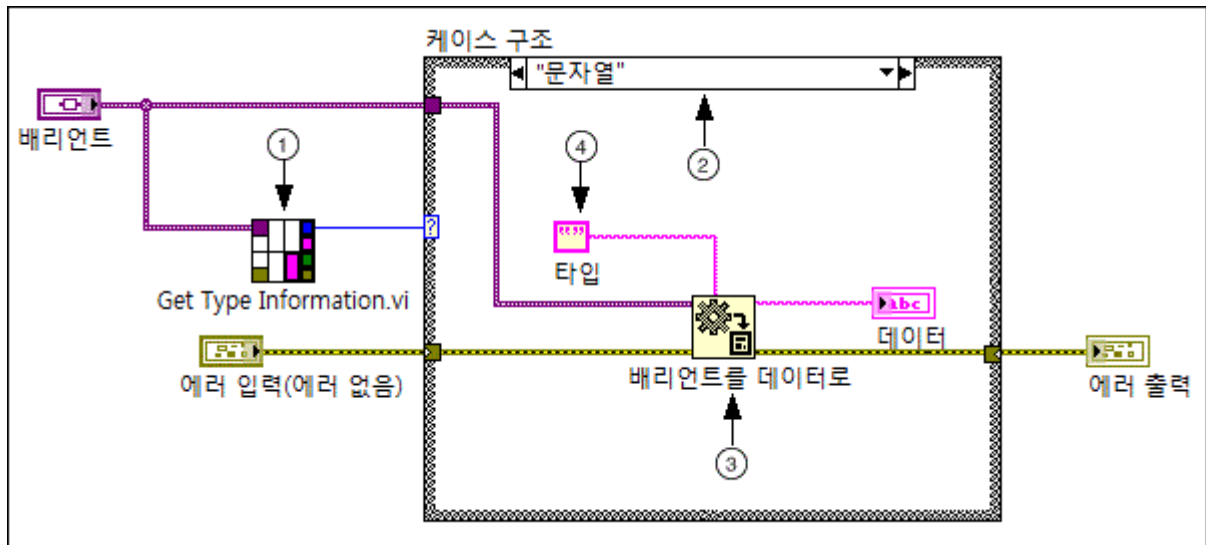
사용 객체

다음 블록다이어그램 객체를 사용하여 배리언트에 저장된 데이터 타입을 확인하고 배리언트를 해당 데이터 타입으로 변환합니다.

타입 정보 얻기	케이스 구조	[배리언트를 데이터로] 함수
		

실행 단계

다음 블록다이어그램을 생성하여 배리언트에 저장된 데이터 타입을 확인하고 배리언트를 해당 데이터 타입으로 변환합니다.



위의 블록다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.

①	[타입 정보 얻기] vi는 배리언트에 저장된 데이터 타입을 불러옵니다.
②	가능한 각 데이터 타입에 대해 서브다이어그램을 추가 합니다. 서브다이어그램은 배리언트 데이터 타입이 선택자 라벨에 나타나는 데이터 타입과 일치하는 경우에만 실행됩니다.
③	각 서브다이어그램에서, [배리언트를 데이터로] 함수는 배리언트를 변환합니다.
④	각 서브다이어그램에서, [배리언트를 데이터로] 함수의 타입 입력에 연결된 데이터 타입은 선택자 라벨의 데이터 타입과 일치해야 합니다. 타입 입력의 데이터 타입과 배리언트 입력의 데이터 타입이 호환되지 않는 경우, LabVIEW는 에러를 반환합니다. LabVIEW는 타입 입력의 값을 무시합니다.

관련 정보

배리언트 데이터 처리하기

LabVIEW 데이터를 배리언트 데이터로 변환하기

배리언트 데이터에서 속성 삭제하기

다음 단계를 따라 **배리언트 데이터**에서 특정한 속성을 삭제합니다.

1. 블록다이어그램에 [배리언트 속성 삭제] 함수를 추가합니다.

2. 배리언트 데이터의 속성을 [배리언트 속성 삭제] 함수의 배리언트 입력에 연결합니다.
3. 모든 속성을 삭제하려면, [배리언트 속성 삭제] 함수의 이름 (모든 속성) 입력을 연결하지 않습니다. 지정된 속성을 삭제하려면, 이름 (모든 속성) 입력안에 속성의 이름을 입력합니다.
4. VI를 실행합니다.

배리언트 데이터에서 속성 가져오기

다음 단계를 따라 **배리언트 데이터**에서 특정한 속성을 가져옵니다.

1. 블록다이어그램에 [배리언트 속성 얻기] 함수를 추가합니다.
2. 배리언트 데이터를 [배리언트 속성 얻기] 함수의 배리언트 입력에 연결합니다.
3. 속성의 이름을 [배리언트 속성 얻기] 함수의 이름 입력에 연결합니다.
4. [배리언트 속성 얻기] 함수의 기본값 입력을 연결합니다. 기본값에 무엇을 연결했는지에 따라 값 출력 연결 시 생성되는 인디케이터의 타입이 결정됩니다.
5. [배리언트 속성 얻기] 함수의 발견됨 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다.
6. [배리언트 속성 얻기] 함수의 값 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 속성과 관련된 값을 가져옵니다.
7. VI를 실행합니다.

발견됨이 참인 경우, 값 인디케이터는 속성의 값을 디스플레이합니다. 발견됨이 거짓인 경우, 값 인디케이터는 기본값 입력에 연결한 값을 디스플레이합니다.

다음 단계를 따라 배리언트 데이터에서 모든 속성 이름을 가져옵니다.

1. 블록다이어그램에 [배리언트 속성 얻기] 함수를 추가합니다.
2. 배리언트 데이터를 [배리언트 속성 얻기] 함수의 배리언트 입력에 연결합니다.
3. [배리언트 속성 설정] 함수의 이름 입력과 기본값 입력은 연결하지 않습니다.
4. 이름 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다.
5. [배리언트 속성 얻기] 함수의 값 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 각 속성과 관련된 값을 가져옵니다.
6. VI를 실행합니다.

속성의 이름이 프런트패널의 이름 배열 인디케이터에 나타납니다. 관련된 값은 값 배열 인디케이터에 배리언트의 리스트로 나타나며, 이를 **변환**하여 VI와 함수에서 사용할 수 있습니다.

단위

단위를 숫자형 컨트롤 또는 인디케이터와 관련짓기

화씨 또는 킬로미터/초와 같은 물리적 측정 단위를 부동소수로 표현된 숫자형 컨트롤 또는 인디케이터에 연관시킬 수 있습니다.

다음 단계를 따라 단위를 숫자형 컨트롤 또는 인디케이터와 연계시킵니다.

1. 프론트패널 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목»단위 라벨을 선택합니다. 단위 라벨이 나타납니다.



노트

National Instruments는 절대 시간으로 포맷되어 있는 숫자형 객체에 단위를 사용하지 않도록 권장합니다.

1. 올바른 **약자**와 **구문**을 사용하여 단위를 입력합니다. 예를 들면, 스페이스를 사용하여 곱셈을 나타내고 포워드 슬래쉬(/)를 사용하여 나눗셈을 나타냅니다. 단위 구문이 적절치 않은 경우, 물음표 ?가 라벨에 나타납니다.
2. 숫자형 객체에 **값을 입력**합니다.



노트 단위 라벨을 생성할 때 LabVIEW는 단위 라벨의 타입에 관계없이 자동으로 블록다이어그램의 모든 데이터를 SI 단위로 변환합니다.

단위를 변경하려면, 수행 도구 또는 라벨링 도구를 사용하여 단위 라벨의 안쪽을 클릭하고 단위 문자열을 변경합니다. 다음 단계를 따라 단위를 변경하고 단위 구문 에러를 피합니다.

1. 단위 라벨에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 단위 문자열 만들기를 선택하여 단위 문자열 만들기 대화 상자를 디스플레이합니다.
2. 단위 리스트박스에서 적당한 단위를 선택합니다.
3. 확인 버튼을 클릭합니다.



노트 단위를 그래프 또는 차트와 연관지으려면 관련된 단위를 가진 객체를 그래프 또는 차트에 연결하십시오. 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 항목»단위 라벨을 선택하여 연결을 통하여 수집된 단위를 디스플레이합니다. 기본으로, LabVIEW는 그래프나 차트에 연결한 객체와 연관된 단위에 기반하여 SI 단위를 그래프 또는 차트에 연관시킵니다. 단위는 같은 현상을 측정하는 호환되는 단위로만 변경할 수 있습니다. 예를 들어, 차트의 입력이 마일을 나타내는 mi 단위를 가지고 있는 경우, 차트 단위를 ft, in, 또는 m으로는 편집할 수 있으나 길이 단위가 아닌 N, Hz, min, acre, 또는 A로는 편집할 수 없습니다.

또한, 텍스트 프로퍼티를 사용하여 프로그램적으로 단위를 변경할 수 있습니다. 단위 라벨 프로퍼티는 텍스트 프로퍼티에서 사용할 수 있는 참조를 생성합니다.

LabVIEW에서 사용 가능한 단위

다음 테이블은 LabVIEW에서 사용 가능한 단위를 디스플레이합니다. LabVIEW는 SI 단위와 다른 측정 시스템의 단위 모두를 인지합니다. 기호 *는 LabVIEW가 사용하는 기본 단위를 나타냅니다.

SI 기본 단위

LabVIEW의 기본 단위는 SI 기본 단위입니다.

물리량 이름	단위	약자
길이	미터*	m
질량	킬로그램*	kg
시간	초*	s
전류	암페어*	A
열역학적인 온도	켈빈*	K
물질의 양	몰*	mol
광도	칸델라*	cd

SI 유도 단위

LabVIEW는 또한 SI 기본 단위에서 나온 다른 단위도 인지합니다. 다음 테이블은 LabVIEW가 인식하는 SI 유도 단위를 나타냅니다.

물리량 이름	단위	약자
영역	평방 미터	m ²
볼륨	입방 미터	m ³
질량	그램	g
속도	초당 미터	m/s
가속도	제곱초당 미터	m/s ²
파수	역 미터	m ⁻¹
밀도	입방 미터당 킬로그램	kg/m ³

물리량 이름	단위	약자
부피	킬로그램당 입방 미터	m ³ /kg
전류 밀도	평방 미터당 암페어	A/m ²
자기장 세기	미터당 암페어	A/m
농도	입방 미터당 몰 수	mol/m ³
휘도	평방 미터당 칸델라	cd/m ²
질량비	킬로그램당 킬로그램	kg/kg
평면 각도	라디안*	rad
입체 각도	입체호도법*	sr
주파수	헤르츠	Hz
힘	뉴턴	N
압력, 응력	파스칼	Pa
에너지, 열량, 일의 양	줄	J
힘, 복사속	와트	W
전하	쿨롱	C
전위차	볼트	V
커패시턴스	패러드	F
전기 저항	옴	Ohm
전기 컨덕턴스	지멘스	S
자속	웨버	Wb
자속 밀도	테슬라	T
인덕턴스	헨리	H
온도 (절대)	섭씨	degC
빛을내는 유량	루멘	lm
조도	럭스	lx
활성도 (방사성 핵종)	베크렐	Bq
흡수선량, 커마(Kerma), (나누어진) 특정 에너지	회색	Gy
선량당량	시버트	Sv

기타 단위

LabVIEW는 또한 SI가 아닌 측정 시스템의 다양한 단위도 인지합니다. 다음 테이블은 이 같은 단위를 나열합니다.

물리량 이름	단위	약자
활성도	퀴리	Ci
영역	에이커	에이커
반	b	
헥타르	ha	
다이나믹 점도	포이즈	P
전력	마력(전기)	hp
에너지	영국 온도 단위(평균)	Btu
칼로리(열)	교정	
전자 볼트	eV	
에르그	에르그	
힘	다인	dyn
파운드	lbf	
조도	푸트칸들(조명도)	fc
길이	옹스트롬	옹스트롬
천문학 단위	AU	
체인	체인	
큐빗	큐빗	
길	길	
페르미	페르미	
푸트	ft	
펄롱	펄롱	
허블	허블	
인치	in	
광년	ly	
마일	mi	

물리량 이름	단위	약자
로드(길이)	로드(길이)	
야드	yd	
렌즈 파워	디옵터	dpt
자력	전자기 단위	전자기 단위
가우스	Gs	
맥스웰	Mx	
에르스텟	Oe	
질량	캐럿	CM
DRAM	dr	
그레인	gr	
톤	t	
온스	oz	
파운드	lb	
슬러그(질량)	슬러그(질량)	
스톤	스톤	
톤 (영국)	Ton	
톤 (미국)	ton	
합쳐진 원자 질량 단위	u	
평면 각도	도	°
분	'	
초	"	
압력	기압	atm
바	bar	
수은주	m.Hg	
평방 인치당 파운드	psi	
토르	토르	
방사선량	라드	라드
방사선 노출	뢴트겐	r

물리량 이름	단위	약자
방사선 노출: man	렘(roentgen equivalent man)	rem
온도 (절대)	화씨	degF
온도차 (상대)	섭씨	Cdeg
화씨	Fdeg	
시간	일	d
시	h	
분	min	
년	a	
속도	노트	kn
볼륨	배럴	배럴
부셀	부셀	
유체 드램	fl.dr	
유체 온스	fl.oz	
갤런 (영국)	UKgal	
갤런 (US)	gal	
질	질	
리터	l	
미닛	미닛	
팩	팩	
포인트	포인트	
쿼트	쿼트	

다형성 단위 생성하기

프론트패널 숫자형 컨트롤 또는 인디케이터의 단위 라벨에 \$x를 입력하여 **다형성** 단위를 생성합니다. 이 때 x는 \$1의 예와 같이 숫자입니다. \$x를 실제 단위의 자리 표시자로 생각할 수 있습니다. VI가 호출될 때, LabVIEW는 VI에서 \$x의 모든 어커런스를 전달한 단위로 대체합니다.

하나 이상의 다형성 단위를 사용할 때, \$2, \$3 등과 같이 약자를 사용할 수 있습니다.

다음 단계를 따라 VI에 다형성 단위를 생성합니다.

1. 프런트패널 숫자형 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목 >> 단위 라벨을 선택합니다.
2. \$x를 입력합니다. 이 때, x는 1부터 9의 임의의 숫자입니다.
3. 프런트패널 숫자형 객체의 밖을 클릭하여 편집 세션을 끝냅니다.

단위 구문

단위를 숫자형 컨트롤 또는 인디케이터와 관련시킬 때 다음 구문을 사용합니다.

- 어떤 단위도 모든 SI 접두어를 가질 수 있습니다. 접두어는 그 바로 뒤의 단위에만 적용됩니다.
- 스페이스를 사용하여 곱셈을 나타냅니다. LabVIEW는 여러 스페이스를 하나의 스페이스로 해석합니다.
- 포워드 슬래쉬 /를 사용하여 나눗셈을 나타냅니다. 포워드 슬래쉬 주변에 스페이스를 사용하는 경우, LabVIEW는 스페이스를 무시합니다. 포워드 슬래쉬는 그 바로 뒤의 단위에만 적용됩니다.
- 캐럿 ^을 사용하여 지수를 나타냅니다. 지수는 영(0)을 제외한 양 또는 음의 정수가 될 수 있습니다. 캐럿은 그 바로 앞의 단위에만 적용됩니다.

다음 테이블은 올바른 구문과 동등한 수학 공식을 가지는 단위 문자열의 예제를 디스플레이합니다.

단위 문자열	동등한 수학 공식
ft lb	ft * lb
m/s	m / s (초로 나눈 미터)
m /s/s	m / (s * s)
s^2	s^2
kg m/s	kg * m * s^-1

디지털 데이터 잘라내기, 복사하기, 붙여넣기

다음 단계를 따라 디지털 데이터를 잘라냅니다.

1. 잘라내기를 원하는 디지털 데이터 컨트롤 또는 디지털 웨이브폼 컨트롤의 행 또는 열을 선택합니다.
2. 행 또는 열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 >> 데이터 잘라내기를 선택합니다.

다음 단계를 따라 디지털 데이터를 복사합니다.

1. 복사하려는 디지털 데이터 컨트롤 또는 디지털 웨이브폼 컨트롤의 행 또는 열을 선택합니다.
2. 행 또는 열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 >> 데이터 복사를 선택합니다.

다음 단계를 따라 디지털 데이터를 붙여넣습니다.

1. 복사한 행 또는 열과 같은 숫자로 구성되어 있는 디지털 데이터 컨트롤 또는 디지털 웨이브폼 컨트롤의 영역을 선택합니다.
2. 행 또는 열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 >> 데이터 붙여넣기를 선택합니다.

데이터의 전체 행이나 전체 열만 잘라낼 수 있습니다. 잘라낸 디지털 데이터로는 새 행이나 열을 생성할 수 없습니다. 자르거나 복사한 영역과 같은 차원의 영역에 디지털 데이터를 붙여넣기해야 됩니다. 예를 들어, 한 개 행에서 4비트의 데이터를 복사했다면, 같은 행이나 다른 행의 4비트의 데이터를 선택하여 붙여 넣기해야 합니다. 두 개의 행과 두 개의 열에서 4 비트의 데이터를 복사한다면, 2행 2열의 영역에 이 데이터를 붙여넣어야 됩니다.

데이터와 이미지 반출하기

그래프, 차트, 테이블, 배열에서 데이터를 반출할 수 있습니다. 또한 그래프, 차트, 테이블의 이미지를 반출할 수도 있습니다.

그래프, 차트, 테이블, 배열에서 데이터 반출하기

다음 단계를 따라 그래프, 차트, 테이블, 배열에서 데이터를 반출합니다.

1. 그래프, 차트, 테이블, 배열에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 반출을 선택하고 다음 옵션 중 하나를 선택합니다.

- 클립보드로 데이터 반출
- Excel로 데이터 반출
- DIAdem으로 데이터 반출

또한 Excel로 데이터 반출 메소드를 사용하여 그래프, 차트, 테이블, 배열, 디지털 데이터 컨트롤에서 데이터를 프로그램적으로 반출할 수 있습니다.

또한 클립보드로 데이터 반출 메소드를 사용하여 그래프, 차트, 테이블, 배열, 디지털 데이터 컨트롤에서 데이터를 프로그램적으로 반출할 수 있습니다.

DIAdem으로 데이터 반출 메소드를 사용하여 그래프, 차트에서 프로그램적으로 데이터를 반출할 수 있습니다.

그래프, 차트, 테이블의 이미지 반출하기

다음 단계를 따라 그래프, 차트, 테이블, 그림 컨트롤, 디지털 데이터, 또는 디지털 웨이브폼 컨트롤이나 인디케이터의 이미지를 반출합니다.

1. 그래프, 차트, 테이블, 그림 컨트롤, 디지털 데이터, 디지털 웨이브폼 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 단순화된 이미지로 반출을 선택하여 단순화된 이미지로 반출 대화 상자를 디스플레이합니다.



노트

단순화된 이미지 반출 메뉴 아이템은 강도 그래프, 강도 차트 또는 3D 그래프, 또는 3D 그림 컨트롤에서는 사용 가능하지 않습니다.

1. 단순화된 이미지로 반출 대화 상자에서, 파일의 포맷을 선택합니다.
2. 클립보드에 저장할 것인지 또는 디스크에 저장할 것인지를 선택합니다.
3. (옵션) 눈금 숨기기 확인란에 확인 표시를 해서 그래프 또는 차트의 눈금을 숨깁니다.
4. 저장 버튼을 클릭하여 이미지를 디스크에 저장합니다.

또한, 단순화된 이미지 반출 메소드를 사용하여 그래프, 차트, 테이블, 디지털 데이터 컨트롤의 이미지를 프로그램적으로 반출할 수 있습니다.

이미지 얻기 메소드를 사용하여 컨트롤과 인디케이터의 이미지를 프로그램적으로 반출할 수 있습니다.



관련 정보

[그래프와 차트 사용자 정의](#) (449 페이지 참조)

[그래프와 차트 사용자 정의](#) (449 페이지 참조)

[그래프와 차트 사용자 정의](#) (449 페이지 참조)

디지털 데이터 컨트롤에 행과 열을 삽입하고 삭제하기

행을 추가하기 위해서, 샘플 열의 한 샘플에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 앞에 행 삽입을 선택합니다.

행을 삭제하려면, 샘플 열의 한 샘플에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 행 삭제를 선택합니다.

열을 삽입하려면, 신호 열의 한 신호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 앞에 열 삽입을 선택합니다.

열을 삭제하기 위해서, 신호 열의 한 신호에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 열 삭제를 선택합니다.

LabVIEW에서 함수 비교하기

배열 비교하기

일부 비교 함수에는 배열 비교에 사용할 수 있는 두 가지 모드가 있습니다. 집합 비교 모드에서, 함수는 단일 불리언 값을 반환합니다. 원소 비교 모드에서, 함수는 개별 원소를 비교한 후 불리언 값의 배열을 반환합니다.

집합 비교 모드에서, 문자열 비교와 배열 비교 작업은 정확히 같은 방식으로 진행됩니다. —비교 함수는 문자열을 ASCII 문자의 배열로 취급합니다.



노트

[최대 & 최소] 함수는 다른 비교 함수와는 다르게 동작합니다. [최대 & 최소] 함수는 사용자가 지정하는 모드와 출력에 따라 배열의 최대 원소나 배열의 최소 원소를 반환합니다.



관련 정보

[원소나 집합을 비교하기 위한 비교 함수 설정하기](#) (1137 페이지 참조)

[비교 함수 사용하기](#) (111 페이지 참조)

불리언 값 비교하기

[비교] 함수는 참의 불리언 값을 거짓의 불리언 값보다 큰 것으로 취급합니다.

문자 비교하기

문자 비교 함수 [10진수?], [16진수?], [8진수?], [인쇄가능?], [화이트 스페이스?], [문자의 클래스]를 사용하여 문자를 비교한 후 문자의 타입을 결정합니다.

비교 함수의 입력이 문자열인 경우, 함수는 첫번째 문자를 테스트합니다. 입력이 빈 문자열인 경우, 결과는 거짓입니다. 입력이 숫자인 경우, 함수는 ASCII 문자 코드로 해석합니다. 입력이 부동소수이면 함수는 값을 가장 가까운 정수로 반올림합니다.

클러스터 비교하기

비교할 클러스터는 반드시 같은 수의 원소를 포함해야만 하고, 클러스터의 각 원소는 반드시 호환 가능한 타입이어야 하며, 원소는 반드시 클러스터에서 같은 순서이어야 합니다. 예를 들어, DBL과 문자열의 클러스터를 I32와 문자열의 클러스터에 비교할 수 있습니다.

원소 비교 모드에서, [비교] 함수는 각 원소가 입력 클러스터의 원소에 대응하는 불리언 원소의 클러스터를 반환합니다.

집합 비교 모드에서, 비교 함수는 클러스터의 원소를 비교한 후 단일 불리언 결과를 반환합니다. 비교 함수는 결정이 날 때까지 순차적으로 원소를 비교하며, 영어 단어의 알파벳 순서에서 서로 다른 문자가 나올 때까지 각 단어의 철자를 비교하는 경우와 유사합니다. 그렇기 때문에, 함수는 다음 단계를 수행하여 비교의 결과를 찾습니다.

- 함수는 클러스터의 처음부터 시작하여 각 입력 클러스터의 대응하는 원소를 비교합니다.
- 대응하는 원소가 같지 않은 경우, 함수는 멈춘 후 비교의 결과를 반환합니다.
- 대응하는 원소가 같은 경우, 비교 함수는 서로 다른 값을 찾거나 입력 클러스터의 끝에 도달할 때까지 다음 비교 값을 처리합니다.

숫자값 비교하기

대부분의 비교 함수는 한 개의 입력을 테스트하거나 두 개 입력을 비교하여 불리언 값을 반환합니다. 함수는 숫자를 비교하기 전에 먼저 같은 형으로 변환합니다. 함수는 각 입력을 정확한 비교를 위해 필요한 가장 큰 형으로 변환합니다. 숫자가 아닌 값 즉, NaN 값을 가지는 하나 또는 두 입력을 비교하면 비교가 같지 않음을 나타내는 결과가 반환됩니다.

문자열 비교하기

비교 함수는 ASCII 문자 코드의 값에 따라 **문자열을 비교**합니다. 비교 함수는 문자열을 비교하는데 그 문자열이 같지 않은 것을 만날 때나 하나의 문자열이 끝날 때까지 0번째 원소로 시작해서 한 번에 하나의 원소를 비교합니다. 이 함수는 긴 문자열을 짧은 문자열보다 큰 것으로 취급합니다.

예를 들어, a (십진수 97)는 A (65)보다 크며, 이것은 숫자 0 (48)보다 크고, 이는 스페이스 문자(32)보다 큽니다. LabVIEW는 문자열의 문자를 처음부터 하나씩 다른 값이 나올 때까지 비교하며, 다른 값이 나오면 비교를 끝냅니다. 예를 들어, LabVIEW는 문자열 abcd와 abef를 e의 값보다 작은 c를 찾을 때까지 비교합니다. 문자가 있는 것은 문자가 없는 것보다 큽니다. 따라서, 문자열 abcd는 abc보다 큽니다. 첫번째 문자열이 더 길기 때문입니다.

[10진수?] 및 [인쇄가능?] 함수와 같이 문자열 문자의 범주를 확인하는 함수는 문자열의 첫번째 문자만을 평가합니다.

범위 값과 데이터 강제 변환 결정하기

다음 단계를 따라 [범위내 확인과 강제 변환] 함수의 상위 리미트와 하위 리미트 입력으로 지정된 범위 내에 값이 있는지를 확인하고, 추가적으로 입력값을 범위 내의 값으로 강제 변환할 수 있습니다.



노트

다음 단계에서는 예로 상수값을 사용합니다. 문자열 또는 불리언 객체의 **값이 범위 내에 있는지를 결정**할 수 있으며 옵션으로 값을 범위 내의 값으로 강제로 변환할 수 있습니다.

1. 블록다이어그램에 [범위내 확인과 강제 변환] 함수를 추가합니다.
2. 함수의 x 입력에 숫자값을 연결합니다.
3. 함수의 상위 리미트 입력에 범위의 최대값을 연결합니다. x 가 상위 리미트 값보다 크면, 함수는 상위 리미트의 숫자로 강제 변환합니다. 기본값은 양의 무한대입니다.
4. 함수의 하위 리미트 입력에 범위의 최소값을 연결합니다. x 가 하위 리미트 값보다 작다면, 함수는 하위 리미트의 숫자로 강제 변환합니다. 기본값은 음의 무한대입니다.
5. (옵션) 함수의 강제 변환(x) 출력을 숫자를 받는 SubVI, 함수, 또는 인디케이터에 연결합니다. x 의 숫자가 상위 리미트와 하위 리미트 입력에 의해 설정된 범위 내에 있다면, x 의 값은 변경되지 않습니다. x 의 숫자가 범위 내에 있지 않으면, 함수는 상위 리미트 또는 하위 리미트의 숫자로 강제 변환합니다.
6. [범위내 확인과 강제 변환] 함수의 범위내? 출력을 불리언 인디케이터에 연결하여 x 의 숫자가 상위 리미트와 하위 리미트 입력에 의해 설정된 범위 내에 있는지 확인하거나, 입력으로 불리언 값을 받는 SubVI 또는 함수에 범위내? 출력을 연결합니다. x 의 숫자가 범위 안에 있으면, 범위내? 출력은 참 값을 반환합니다. 숫자가 범위 안에 있지 않으면, 범위내? 출력은 거짓 값을 반환합니다.
7. VI를 실행합니다.

배열에서 범위 값과 데이터 강제 변환 결정하기

다음 단계를 따라 [범위내 확인과 강제 변환] 함수의 상위 리미트와 하위 리미트 입력으로 지정된 범위 내에 배열의 값이 오게 하고 옵션으로 범위 안의 값으로 강제 변환합니다.



노트

다음 단계에서는 예로 상수값을 사용합니다. 문자열 배열 또는 불리언 배열의 **값이 범위 안에 있는지를 결정**할 수 있으며 옵션으로 값을 범위 안의 값으로 강제 변환할 수 있습니다.

1. 블록다이어그램에 [범위내 확인과 강제 변환] 함수를 추가합니다.
2. 함수의 x 입력에 배열을 연결합니다.
3. 함수의 상위 리미트 입력에 범위의 최대값을 연결합니다. x 의 값이 상위 리미트 값보다 크면, 함수는 상위 리미트의 숫자로 강제 변환합니다. 기본값은 양의 무한대입니다.
4. 함수의 하위 리미트 입력에 범위의 최소값을 연결합니다. x 의 값이 하위 리미트 값보다 작다면, 함수는 하위 리미트의 숫자로 강제 변환합니다. 기본값은 음의 무한대입니다.



노트 스칼라 값 또는 숫자의 배열을 상위 리미트 또는 하위 리미트에 연결할 수 있습니다. 상위 리미트 또는 하위 리미트에 스칼라 값을 연결하면, LabVIEW는 스칼라 값 리미트에 대해 x 에 연결한 배열에 있는 각 원소를 확인합니다. 상위 리미트 또는 하위 리미트에 배열을 연결하면, LabVIEW는 리미트로 연결한 배열의 대응하는 원소에 대해 x 에 연결한 배열의 각 원소를 확인합니다.

1. 함수의 강제 변환(x) 출력을 숫자의 배열을 받을 수 있는 SubVI, 함수, 또는 인디케이터에 연결합니다. x에 연결한 배열의 각 원소가 상위 리미트와 하위 리미트 입력에 의해 설정된 범위 안이면, x의 값이 변경되지 않습니다. x에 연결된 배열의 원소가 범위 내에 있지 않으면, 함수는 상위 리미트 또는 하위 리미트의 숫자로 그 원소를 강제 변환합니다.
2. 불리언 값의 배열에 [범위내 확인과 강제 변환] 함수의 범위내? 출력을 연결합니다. 배열의 각 불리언 값은 x에 연결한 배열의 원소에 대응하며 지정한 범위 내에 원소가 있는지 가리킵니다.
3. VI를 실행합니다.

원소나 집합을 비교하기 위한 비교 함수 설정하기

일부 비교 함수에는 데이터 배열 또는 클러스터를 **비교하는 데 사용할 수 있는 모드가 두 가지** 있습니다. 집합 비교 모드에서, 함수는 단일 불리언 값을 반환합니다. 원소 비교 모드에서, 함수는 개별 원소를 비교한 후 불리언 값의 배열이나 클러스터를 반환합니다.



노트

[최대 & 최소] 함수는 다른 비교 함수와는 다르게 동작합니다. [최대 & 최소] 함수는 사용자가 지정하는 모드와 출력에 따라 배열의 최대 원소나 배열의 최소 원소를 반환합니다.

입력이 두 개 이상인 비교 함수가 배열 및 클러스터에서 원소 또는 집합을 비교하도록 설정하려면, 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 비교 모드 >> 원소 비교 또는 비교 모드 >> 집합 비교를 선택합니다.

라벨과 디지털 디스플레이 보이기와 숨기기

대부분의 객체를 삭제할 수 있지만, 고유의 라벨과 디지털 디스플레이는 삭제할 수 없습니다.

다음 단계를 따라 컨트롤이나 인디케이터, 와이어 라벨과 디지털 디스플레이를 보이거나 숨깁니다.

1. 객체에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 보이는 항목 >> 라벨이나 보이는 항목 >> 디지털 디스플레이를 선택합니다.

구획에서 배경 이미지 반입하기

다음 단계를 따라 구획에 배경 이미지를 반입할 수 있습니다:

1. 구획의 스크롤 막대에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 구획 프로퍼티 대화 상자를 디스플레이합니다.

2. 배경 탭으로 가서 배경 리스트에서 이미지를 선택합니다. 이미지 미리보기는 미리보기 이미지를 디스플레이하고, 이미지는 구획에 나타납니다. 또한 다음 단계를 따라 배경 리스트에서 나타나지 않는 이미지를 선택할 수 있습니다.
 1. 탐색 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.
 2. 이미지를 탐색하고 확인 버튼을 클릭합니다. 배경 리스트에서 이미지가 나타납니다. LabVIEW는 배경 이미지로 BMP, JPEG, PNG 그래픽 포맷을 지원합니다. 지원되지 않는 그래픽 포맷의 이미지를 로드하려고 하면 LabVIEW는 에러를 반환합니다.



노트

배경 리스트에 나타나지 않는 이미지를 선택하면, LabVIEW는 해당 이미지를 배경 리스트에 영구적으로 추가하지 않습니다. 이미지를 리스트에 영구적으로 추가하려면 `labview\resource\backgrounds` 디렉토리에 이미지를 저장해야 합니다.

3. 위치 풀다운 메뉴를 사용해서 이미지 위치를 펼치기, 가운데, 또는 타일로 설정합니다.
4. 확인 버튼을 클릭하여 구획 프로퍼티 대화 상자를 닫고 배경을 적용합니다.

배경 이미지가 있는 구획이 포함된 VI를 저장하면 LabVIEW는 구획 배경 이미지를 VI와 함께 저장합니다.

또한 배경 이미지 프로퍼티를 사용하여 프로그램적으로 구획 배경을 설정할 수 있습니다. 배경 모드 프로퍼티를 사용하여 프로그램적으로 구획 배경 이미지 위치를 설정할 수 있습니다.

Linux에서 그래픽 반입하기

다음 단계를 따라 LabVIEW에 이미지(JPG, PNG, X Window System Dump XWD)를 반입합니다. `xwd` 명령을 사용하여 XWD 파일을 생성할 수 있습니다.

1. 편집>>클립보드에 그림 반입을 선택합니다.
2. 나타나는 대화 상자를 사용하여 반입하려는 이미지 파일을 탐색합니다. 파일을 선택하고 확인 버튼을 클릭합니다.
3. (옵션) 이미지를 붙여넣으려는 곳에 팬이나 블록다이어그램을 클릭합니다. 이미지를 붙여넣을 곳을 클릭하지 않으면, LabVIEW는 그것을 프런트패널이나 블록다이어그램의 가운데에 붙여넣습니다.
4. LabVIEW에서 편집>>붙여넣기를 선택합니다.

다음 단계를 따라 LabVIEW 클립보드에 그림 반입 기능을 사용하여 이미지 파일(GIF, JPG, XWD, BMP, PNG 또는 애니메이션 MNG)을 반입합니다.

1. 편집>>클립보드에 그림 반입을 선택합니다.
2. 나타나는 대화 상자를 사용하여 반입하려는 이미지 파일을 탐색합니다. 파일을 선택하고 확인 버튼을 클릭합니다.

3. (옵션) 이미지를 붙여넣으려는 곳에 팬이나 블록다이어그램을 클릭합니다. 이미지를 붙여넣을 곳을 클릭하지 않으면, LabVIEW는 그것을 프런트패널이나 블록다이어그램의 가운데에 붙여넣습니다.
4. LabVIEW에서 편집>>붙여넣기를 선택합니다.

macOS에서 그래픽 반입하기

이미지를 그래픽 어플리케이션으로부터 반입할 수 있고, LabVIEW에 이미지 파일을 끌어서 놓을 수 있고, 편집>>클립보드에 그림 반입 메뉴 항목을 사용할 수 있습니다.

다음 단계를 따라 LabVIEW에 이미지를 반입할 수 있습니다.

1. 편집용 그래픽 어플리케이션이나 웹 브라우저에서, 이미지를 클립보드에 복사합니다.
2. LabVIEW로 스위치합니다. 이미지는 LabVIEW 클립보드에서 자동적으로 이용 가능합니다.
3. (옵션) 이미지를 붙여넣으려는 곳에 팬이나 블록다이어그램을 클릭합니다. 이미지를 붙여넣을 곳을 클릭하지 않으면, LabVIEW는 그것을 프런트패널이나 블록다이어그램의 가운데에 붙여넣습니다.
4. 편집>>붙여넣기를 선택하여 이미지를 LabVIEW에 놓습니다.



노트

만약 이미지를 복사하여 붙여넣으면, 그 이미지는 투명성을 잃게 됩니다.

다음 단계를 따라 폴더에서 닫힌 그래픽 파일을 끌고 그것을 LabVIEW에 놓습니다.

1. 폴더에서, 반입하려는 그래픽 파일을 선택합니다.
2. LabVIEW 윈도우로 파일을 끌어와서 그것을 붙여넣으려는 곳에 커서를 놓습니다.
3. 마우스 버튼을 놓습니다. 파일로부터 이미지가 LabVIEW에 나타납니다.

다음 단계를 따라 LabVIEW 클립보드에 그림 반입 기능을 사용하여 이미지 파일(GIF, JPG, PNG 또는 애니메이션 MNG)을 반입합니다.

1. 편집>>클립보드에 그림 반입을 선택합니다.
2. 나타나는 대화 상자를 사용하여 반입하려는 이미지 파일을 탐색합니다. 파일을 선택하고 확인 버튼을 클릭합니다.
3. (옵션) 이미지를 붙여넣으려는 곳에 팬이나 블록다이어그램을 클릭합니다. 이미지를 붙여넣을 곳을 클릭하지 않으면, LabVIEW는 그것을 프런트패널이나 블록다이어그램의 가운데에 붙여넣습니다.
4. LabVIEW에서 편집>>붙여넣기를 선택합니다.

Windows에서 그래픽 반입하기

이미지를 그래픽 어플리케이션으로부터 반입할 수 있고, LabVIEW에 이미지 파일을 끌어서 놓을 수 있고, 편집>>클립보드에 그림 반입 메뉴 아이템을 사용할 수 있습니다.

다음 단계를 따라 그래픽 어플리케이션으로부터 애니메이션 이미지를 제외한 모든 이미지 포맷을 반입합니다.

1. 편집용 그래픽 어플리케이션이나 웹 브라우저에서, 이미지를 클립보드에 복사합니다.
2. LabVIEW로 스위치합니다. 이미지는 LabVIEW 클립보드에서 자동적으로 이용 가능합니다.
3. (옵션) 이미지를 붙여넣으려는 곳에 팬이나 블록다이어그램을 클릭합니다. 이미지를 붙여넣을 곳을 클릭하지 않으면, LabVIEW는 그것을 프런트패널이나 블록다이어그램의 가운데에 붙여넣습니다.
4. 편집>>붙여넣기를 선택하여 이미지를 LabVIEW에 놓습니다.



노트

만약 이미지를 복사하여 붙여넣으면, 그 이미지는 투명성을 잃게 됩니다.

다음 단계를 따라 윈도우 탐색기로부터 그래픽 파일(GIF, JPG, CLP, EMF, WMF, BMP, PNG 또는 애니메이션 MNG)을 끌어와 LabVIEW에 놓습니다.

1. Windows 탐색기에서, 반입하려는 그래픽 파일을 선택합니다.
2. LabVIEW 윈도우로 파일을 끌어와서 그것을 붙여넣으려는 곳에 커서를 놓습니다.
3. 마우스 버튼을 놓습니다. 파일로부터 이미지가 LabVIEW에 나타납니다.

다음 단계를 따라 LabVIEW 클립보드에 그림 반입 기능을 사용하여 이미지 파일(GIF, JPG, CLP, EMF, WMF, BMP, PNG 또는 애니메이션 MNG)을 반입합니다.

1. 편집>>클립보드에 그림 반입을 선택합니다.
2. 나타나는 대화 상자를 사용하여 반입하려는 이미지 파일을 탐색합니다. 파일을 선택하고 확인 버튼을 클릭합니다.
3. (옵션) 이미지를 붙여넣으려는 곳에 팬이나 블록다이어그램을 클릭합니다. 이미지를 붙여넣을 곳을 클릭하지 않으면, LabVIEW는 그것을 프런트패널이나 블록다이어그램의 가운데에 붙여넣습니다.
4. LabVIEW에서 편집>>붙여넣기를 선택합니다.

버튼의 라벨 편집하기

볼리언 컨트롤과 인디케이터는 **고유 라벨**뿐만 아니라 볼리언 텍스트 라벨도 가지고 있습니다. 볼리언 텍스트 라벨은 컨트롤이나 인디케이터의 값에 따라 변경됩니다. 초기에,

대부분의 불리언 컨트롤과 인디케이터는 참 상태에 ON을, 거짓 상태에 OFF를 디스플레이합니다. 수행 도구를 사용하여 버튼을 클릭하면, 컨트롤은 반대 상태로 토글됩니다.



팁

불리언 객체의 라벨이 컨트롤이나 인디케이터의 상태와 부합하도록 라벨을 붙입니다. 예를 들어 참에는 ON, 거짓에는 OFF를 사용합니다. 또한 불리언의 기본 상태를 괄호 안에 표시하십시오.

기본적으로, 라벨은 버튼 가운데 위치합니다. 다음 단계를 따라 라벨을 조작하는 연습을 합니다.

1. 누름 버튼을 프런트패널에 추가합니다.
2. 버튼에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목>> 불리언 텍스트를 선택합니다. Off 단어가 불리언 버튼에 나타납니다.
3. 위치 도구를 사용하여 텍스트를 선택하고 그 텍스트를 원하는 위치로 이동시킵니다. 또한, 바로 가기 메뉴에서 텍스트 가운데 고정을 선택하여 기본 위치로 텍스트를 되돌릴 수 있습니다. 텍스트를 잠금해제하려면 불리언 버튼에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 텍스트 고정 풀기를 선택합니다.
4. 라벨링 도구를 사용하여 텍스트의 상태를 변경합니다. 예를 들어, ON을 예로 변경합니다.
5. 불리언 텍스트를 선택하고 도구 모음의 텍스트 셋팅 폴다운 메뉴를 사용하여 텍스트의 폰트, 크기, 색을 변경합니다.
6. 버튼에서 마우스 오른쪽 버튼을 선택한 후 바로 가기 메뉴에서 보이는 항목>> 불리언 텍스트를 선택하여 두 상태 모두의 불리언 텍스트를 숨깁니다.

기본적으로, 컨트롤 팔레트에 위치한 라벨된 버튼의 각 상태에 대한 라벨을 변경할 수 있습니다. 확인 버튼 및 취소 버튼과 같이 라벨이 붙은 버튼의 라벨을 변경하려면, 각 상태에 따라 다른 텍스트 스트링을 사용할 수 있도록 버튼을 편집해야 합니다.

다음 단계를 따라 여러 텍스트 문자열을 허용하기 위해서 라벨된 버튼을 편집합니다.

1. 불리언 텍스트가 보이는지 점검합니다. 불리언 텍스트를 디스플레이하려면, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목>> 불리언 텍스트를 선택합니다.
2. 버튼에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>> 사용자 정의를 선택하여 **컨트롤 편집기** 윈도우를 디스플레이합니다.
3. 컨트롤 편집기 윈도우의 버튼에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 여러 텍스트 문자열을 선택합니다.
4. 파일>> 변경된 사항 적용을 선택합니다.
5. 버튼을 다른 VI에서도 사용하려는 경우, 버튼을 사용자 컨트롤로 저장합니다.

또한, 불리언 프로퍼티를 사용하여 불리언 라벨을 프로그램적으로 조작할 수 있습니다.

그래프와 차트

그래프와 차트 시작하기

그래프와 차트는 데이터를 디스플레이하고 업데이트하는 방법에서 차이가 납니다. 데이터의 전체 집합을 디스플레이하고 그래프를 반출하려면 그래프를 사용합니다. 연속적으로 데이터를 수집하는 빠른 프로세스의 경우 그래프를 사용하는 것이 가장 좋습니다.

데이터를 수집하는 것과 동시에 디스플레이하려면 차트를 사용합니다. 초당 몇 개의 데이터 포인트만을 플롯하는 느린 프로세스의 경우 차트를 사용하는 것이 가장 좋습니다. 그래프와 차트의 차이에 대한 더 자세한 정보는 ni.com에서 다음 관련 문서를 참조하십시오.

다음 토픽을 사용하여 그래프와 차트를 시작합니다.

그래프 또는 차트 선택하기 및 생성하기	그래프나 차트의 플롯 범례 변경하기
- 올바른 그래프 또는 차트 선택하기 - 그래프 생성하기 - 차트 생성하기	- 플롯 범례에 플롯 추가하기 - 플롯 범례에서 플롯 개수 설정하기
그래프 또는 차트 사용자 정의하기	그래프 또는 차트 탐색하기
- 그래프 또는 차트 설정하기 - 그래프와 차트 사용자 정의 - 그래프와 차트 축의 포맷과 정밀도 변경하기 - 그래프 또는 차트에서 X축과 Y축 설정하기 - 그래프 또는 차트의 X 스케일과 Y 스케일에 마커 추가하기 - 그래프와 차트에 여러 X 스케일과 Y 스케일 추가하기	- 그래프나 차트에서 스크롤하기 - 그래프 또는 차트의 보기를 줌 또는 이동하기
그래프 또는 차트 데이터 반출하기 및 지우기	

- 데이터와 이미지 반출하기
- 그래프와 차트의 데이터 지우기



관련 정보

[그래프 서브팔레트](#) (78 페이지 참조)

[그래프 생성하기](#) (1154 페이지 참조)

[차트 생성하기](#) (1172 페이지 참조)

[플롯 범례에 플롯 추가하기](#) (1143 페이지 참조)

[플롯 범례에서 플롯 개수 설정하기](#) (1149 페이지 참조)

[그래프 또는 차트 설정하기](#) (1146 페이지 참조)

[그래프와 차트 사용자 정의](#) (449 페이지 참조)

[그래프와 차트 축의 포맷과 정밀도 변경하기](#) (1145 페이지 참조)

[그래프 또는 차트에서 x축과 y축 설정하기](#) (1146 페이지 참조)

[그래프 또는 차트의 x 스케일과 y 스케일에 마커 추가하기](#) (1144 페이지 참조)

[그래프와 차트에 여러 x 스케일과 y 스케일 추가하기](#) (1144 페이지 참조)

[그래프나 차트에서 스크롤하기](#) (1148 페이지 참조)

[그래프 또는 차트의 보기를 줌 또는 이동하기](#) (1149 페이지 참조)

[데이터와 이미지 반출하기](#) (1132 페이지 참조)

[그래프와 차트의 데이터 지우기](#) (1145 페이지 참조)

플롯 범례에 플롯 추가하기

LabVIEW가 그래프 또는 차트에 그리는 데이터에 대한 플롯 범례에 플롯을 배치하거나 제외시킬 뿐만 아니라, **배치된 플롯을 사용자 정의**할 수 있습니다.

그래프 또는 차트의 플롯 범례에 플롯을 추가하려면, 위치 도구를 플롯 범례 경계 위로 옮깁니다. 크기 조정 핸들이 각 경계의 중간과 범례의 모서리에 나타나면, 핸들을 위, 아래, 왼쪽, 또는 오른쪽으로 끌어서 범례가 정확한 플롯 개수를 포함할 때 마우스 버튼을 놓습니다.

플롯 범례 스크롤 막대에서 증가 화살표를 클릭하여 빈 플롯을 보이게 합니다. LabVIEW는 플롯 범례에 있는 빈 플롯 범례를 비활성화합니다. 빈 플롯에 있는 빈 문양을 클릭하거나 플롯 이름을 입력하여 새 플롯을 생성합니다.

플롯 범례에 스크롤 막대 추가하기

스크롤 막대를 사용하여 한 번에 모든 플롯을 표시하지 않고도 플롯 범례의 플롯을 보십시오.

스크롤 막대를 플롯 범례에 추가하려면, 플롯 범례에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 보이는 아이템»수평 스크롤 막대 또는 보이는 아이템»수직 스크롤 막대를 선택합니다.

그래프 또는 차트의 X 스케일과 Y 스케일에 마커 추가하기

다음 단계를 따라 **그래프 또는 차트**의 x 스케일과 y 스케일에 마커를 추가합니다.

1. x 또는 y 스케일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 마커 간격»임의를 선택하여, 임의의 마커 간격을 사용하기 위한 스케일을 설정합니다. LabVIEW는 두 개의 끝 마커를 제외한 스케일 상의 모든 마커를 제거합니다.
2. 새 마커를 추가하려는 스케일 상의 위치에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 x 스케일»마커 추가 또는 y 스케일»마커 추가를 선택합니다. LabVIEW는 해당 위치에 새 마커를 추가합니다. 또한 <Ctrl> 키를 누른 채로 기존의 마커를 마커를 추가하려는 위치로 끌거나 또는 끝 마커 중의 하나를 마커를 추가하려는 위치로 끌 수 있습니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.
3. (옵션) 수행 도구 또는 라벨링 도구를 사용하여 새 마커의 값을 변경합니다.

마커를 삭제하려면, 마커에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 마커 삭제를 선택합니다.

그래프와 차트에 여러 X 스케일과 Y 스케일 추가하기

웨이브폼, XY, 강도, 디지털 웨이브폼 그래프는 여러 x와 y 스케일을 지원합니다. 혼합된 신호 그래프와 모든 차트는 여러 y 스케일만을 지원합니다. 그래프나 차트의 여러 스케일을 사용하여 공통적인 x 또는 y 스케일을 공유하지 않는 여러 개의 플롯을 디스플레이합니다.

다음 단계를 따라 **웨이브폼 그래프 또는 차트**에 스케일을 추가합니다.

1. 웨이브폼 그래프 또는 차트의 스케일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 스케일 복제를 선택합니다.
2. 새로운 스케일을 디스플레이의 반대편으로 옮기려면, 새로운 스케일에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 슬라이드 맞바꾸기를 선택합니다.
3. 플롯 범례의 플롯에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 x 스케일 또는 y 스케일을 선택하여 플롯을 새로운 스케일에 연결합니다. 플롯 범례가 보이지 않는 경우, 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템»플롯 범례를 선택합니다.



노트

프로퍼티 대화 상자를 사용하여 플롯을 스케일에 연결할 수도 있습니다. 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택합니다. 플롯 페이지에서, y 스케일 및 x 스케일 폴다운 메뉴를 사용하여 플롯을 올바른 스케일에 연결합니다.

1. 새로운 스케일의 범위를 변경하려면, 수행 도구 또는 라벨링 도구를 사용하여 스케일의 끝 값을 하이라이트하고 새 값을 입력합니다. 스케일을 수동으로 설정할 경우, 반드시 **오토스케일을 비활성화**해야만 합니다.

그래프와 차트 축의 포맷과 정밀도 변경하기

축의 포맷을 변경하여 그래프 또는 차트의 축에 다른 숫자형이나 시간 포맷을 디스플레이합니다. 정밀도를 변경하여 축이 나타내는 자릿수를 지정합니다.

다음 단계를 따라 **그래프 또는 차트** 축의 포맷과 정밀도를 변경합니다.

1. 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티, 3D 플롯 프로퍼티 또는 3D 그래프 프로퍼티를 선택합니다.
2. 디스플레이 포맷 또는 포맷 탭을 선택합니다.
3. 설정하려는 축을 선택합니다.
4. 타입에서 포맷을 선택합니다. 스케일 숫자의 **형**이 부동소수인 경우, 숫자를 축에서 16진수, 8진수 또는 2진수로 디스플레이할 수 없습니다.
5. 자릿수와 정밀도 타입 옵션을 사용하여 축에서 숫자의 정밀도를 변경합니다.



노트

고급 편집 모드를 사용하여 **포맷 문자열을 입력**하고 축에서 숫자의 정밀도를 사용자 정의합니다.

1. 확인을 클릭합니다.

그래프와 차트의 데이터 지우기

다음 단계를 따라 **그래프 또는 차트**의 데이터를 직접 지웁니다.

1. 데이터를 생성하고 그 데이터를 그래프 또는 차트에 디스플레이하는 VI를 실행합니다.
2. 실행되는 VI의 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 각각 그래프 지우기 또는 차트 지우기를 선택합니다. LabVIEW는 그래프 또는 차트의 데이터를 지웁니다. 실행이 끝난 VI의 바로 가기 메뉴에서 각각 데이터 처리>>그래프 지우기 또는 데이터 처리>>차트 지우기를 선택하십시오.

또한, VI를 실행하거나 호출할 때마다 **그래프와 차트를 자동으로 지울** 수 있습니다.

또한, 히스토리 데이터 프로퍼티를 사용하여 프로그램적으로 차트를 지울 수 있습니다.

프로그램적으로 그래프와 차트를 지우는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts 디렉토리에 있는 Clearing Waveform Charts VI를 참조하십시오.

 예제 열기  관련 예제 찾기

그래프 또는 차트 설정하기

그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 다음 바로 가기 메뉴 옵션 중 하나를 선택하여 **그래프 또는 차트를 설정**합니다.

- 다층 플롯 — 여러 개의 수직 스케일을 사용하여 차트에서 여러 개의 플롯을 디스플레이합니다. 이 바로 가기 메뉴 옵션에서 확인 표시를 제거하여 플롯을 겹쳐 놓고 하나의 수직 스케일을 사용하여 차트에서 여러 개의 플롯을 디스플레이합니다. 이 옵션은 차트에서만 사용 가능합니다.
- 차트 히스토리 길이 — 차트 히스토리 길이 대화 상자를 실행합니다. 이 옵션은 차트에서만 사용 가능합니다.
- 배열 전치 — 플롯하기 전에 x와 y 데이터를 바꿉니다. 또한, 배열 전치 프로퍼티를 사용하여 프로그램적으로 플롯하기 전에 x와 y 데이터를 바꿀 수 있습니다.
- 플롯 범례 자동 크기 조정 — 플롯 범례를 범례에 보이는 가장 긴 플롯 이름의 폭에 맞게 자동으로 크기 조정을 합니다.



노트

디지털 웨이브폼 그래프에서, 플롯 범례 자동 크기 조정은 플롯 범례가 **표준 보기**인 경우에만 보입니다.

또한, 그래프 또는 차트의 **x축과 y축을 설정**할 수 있습니다.

그래프 또는 차트에서 X축과 Y축 설정하기

그래프 또는 차트의 x 또는 y 스케일에서 마우스 오른쪽 버튼을 클릭한 후 다음의 바로 가기 메뉴 옵션 중 하나를 선택하여 x축 또는 y축을 설정합니다. 또한, 그래프 또는 차트에서 마우스 오른쪽 버튼을 클릭한 후 x 스케일 또는 y 스케일 바로 가기 메뉴에서 이 옵션에 접근할 수 있습니다. 또한, x 스케일과 y 스케일 프로퍼티를 사용하여 프로그램적으로 스케일을 선택할 수 있습니다.



노트

다음 바로 가기 메뉴 옵션은 3D 그래프에는 적용되지 않습니다.

- 마커 간격 — 균일(기본) 또는 임의의 마커 간격을 사용합니다. 기본적으로, 스케일 마커는 균등하거나, 또는 균등한 간격입니다. 임의의 마커 간격을 선택하여, 스케일에 특정 내부 마커를 나타낼 수 있습니다. **임의의 마커 간격**은 포인트 또는 임계점을 설정하는 것과 같이 슬라이드 또는 회전식 컨트롤에 일부 특정 포인트를 표기하는데 유용합니다.
- 마커 추가 — 스케일에 **마커를 추가**합니다. 마커 간격이 균일인 경우 이 옵션은 사용 가능하지 않습니다.
- 마커 삭제 — 스케일에서 **마커를 삭제**합니다. 마커 간격이 균일하고 스케일에 오직 두 개의 끝 마커만 포함되는 있는 경우, 이 옵션은 이용할 수 없습니다.

- 포맷—x 또는 y 축을 **포맷**하는데 사용할 수 있는 대화 상자를 디스플레이합니다.
- 스타일—스케일 상의 마커의 스타일을 설정합니다.
- 맵핑—선형 (기본) 또는 로그를 선택합니다.
- X 오토스케일 또는 Y 오토스케일—연결한 데이터에 기반하여 x 또는 y 스케일을 **오토스케일**합니다.
- 스케일에 맞춤—이 옵션 옆에 확인 표시를 하여 스케일에 사용되는 증가의 배수로 끝 마커를 반올림합니다.
- 스케일 라벨 보이기—스케일 라벨을 디스플레이합니다.
- 디지털 버스 확장—디지털 웨이브폼 데이터를 각 데이터 라인으로 디스플레이합니다. 이 옵션은 오직 디지털 웨이브폼 그래프에서만 사용할 수 있습니다. 또한, 디지털 버스 확장 프로퍼티를 사용하여 프로그램적으로 데이터가 어떻게 디스플레이되는지를 설정할 수 있습니다.
- 사이드 맞바꾸기—스케일을 디스플레이의 반대쪽으로 이동합니다. 이 옵션은 그래프 또는 차트에서 두 스케일을 사용할 경우에 유용합니다.



노트 그래프 또는 차트에 x 또는 y 스케일을 설정한 후, 바로 가기 메뉴에서 고급 >> 스케일 레이아웃 리셋을 선택하여 x 또는 y 스케일의 위치를 리셋할 수 있습니다.

- 스케일 복제—스케일을 복사하여 그래프 또는 차트에 **새 스케일로 추가**합니다.
- 스케일 삭제—스케일을 삭제합니다. 이 옵션은 그래프 또는 차트가 여러 x축과 y축을 가지는 경우에 사용 가능합니다.

또한, 그래프차트, 그래프스케일, 스케일 프로퍼티를 사용하여 프로그램적으로 그래프 또는 차트의 x 또는 y 축을 설정할 수 있습니다.

플롯 범례를 사용하여 아날로그 플롯 사용자 정의하기

플롯 범례를 사용하여 각 플롯이 그래프 또는 차트의 플롯 영역에서 어떻게 나타나는지 사용자 정의합니다. 플롯 범례 안에 있는 문양을 클릭한 후 바로 가기 메뉴에서 다음 옵션 중 하나를 선택하여, 그래프 또는 차트의 플롯 영역에 어떻게 플롯이 나타나는지 사용자 정의합니다.



노트

디지털 웨이브폼 플롯 설정을 통해 디지털 웨이브폼 그래프에 숫자와 플롯 라인이 어떻게 나타나는지 지정할 수도 있습니다.

- 플롯 보이기—그래프나 차트에 플롯을 디스플레이할지 설정합니다.



노트 그래프 또는 차트에 여러 개의 플롯이 포함된 경우, 플롯 범례에서 마우스 오른쪽 버튼을 클릭한 후 보이는 아이템 >> 플롯 보이기 확인란을 선택하면 각 플롯 옆에 확인란을 디스플레이할 수 있습니다. 디스플레이하지 않으려는 플롯의 확인란에서 확인 표시를 제거합니다.

- 일반 플롯—플롯 타입에 대한 옵션을 제공합니다. 라인, 산포, 라인과 기호, 채우기, 빗, 막대 플롯 타입을 선택할 수 있습니다.
- 색—플롯의 색을 선택할 수 있도록 색 선택기를 디스플레이합니다.
- 라인 스타일—기본 라인과 점선 라인 스타일을 제공합니다.
- 라인 폭—라인 폭을 제공합니다.
- 엘리어스 제거—라인 플롯을 자연스럽게 나타내기 위해서 이 아이템의 옆에 확인 표시를 합니다. 엘리어스 제거 라인 그리기는 많은 계산이 필요할 수 있습니다. 엘리어스 제거 라인 플롯을 사용하면 성능이 느려질 수 있습니다.



노트 엘리어스 제거 라인 그리기는 디지털 웨이브폼 그래프에서는 사용할 수 없습니다.

- 막대 플롯—막대 플롯 옵션을 제공합니다.
- 베이스 라인 채우기—플롯과 같은 색으로 플롯 아래 공간을 채우는 옵션을 제공합니다. 그래프 또는 차트가 여러 개의 플롯을 가지는 경우, 플롯의 아래 공간을 다른 플롯의 윗부분에서 채우기 위해서 바로 가기 메뉴의 아래에서 다른 플롯을 선택할 수 있습니다.
- 보간—데이터 포인트만을 플롯하는 것을 포함하여 보간 옵션을 제공합니다.
- 포인트 스타일—데이터 포인트의 포인트 스타일을 제공합니다.



노트 데이터 포인트의 크기를 변경하려면 라인 폭 옵션을 사용하여 해당 플롯의 라인 폭을 변경해야 합니다.

- X 스케일—X 축과 연계할 변수를 설정합니다.
- Y 스케일—Y 축과 연계할 변수를 설정합니다.

또한 플롯 프로퍼티를 사용하여 프로그램적으로 플롯이 플롯 영역에 어떻게 나타날지를 사용자 정의할 수 있습니다.

그래프나 차트에서 스크롤하기

그래프나 차트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 아이템 >> X 스크롤 막대를 선택하여 그래프나 차트 아래에 스크롤 막대를 디스플레이합니다. 수행 도구를 사용하여 스크롤 막대를 클릭하고 x축을 따라서 스크롤합니다.

또한, X 스크롤 막대 보이기 프로퍼티를 사용하여 프로그램적으로 x축 스크롤 막대를 디스플레이할 수 있습니다.



관련 정보

[그래프와 차트의 종류](#) (433 페이지 참조)

플롯 범례에서 플롯 개수 설정하기

런타임에서 플롯 범례 안에 보이는 플롯의 개수를 지정할 수 있습니다. 또한 그래프 또는 차트의 플롯 영역 안에 있는 플롯의 개수에 관계없이 플롯 범례 안에 보여줄 플롯의 개수를 지정할 수 있습니다.

그래프 또는 차트의 플롯 범례 안에서 보여줄 플롯의 개수를 지정하려면, 그래프 또는 차트의 플롯 영역에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티를 선택합니다. 모양 페이지에서, 플롯 보이기 필드에 있는 숫자를 변경하여 플롯 범례 안에서 보여줄 플롯의 최소 개수를 증가 또는 감소시킵니다. 플롯 보이기 필드에서 수동으로 직접 플롯 개수를 입력하거나, 증가 및 감소 버튼을 클릭하여 보여줄 플롯의 개수를 증가 또는 감소시킵니다.

또한 범례:행의 개수 프로퍼티를 사용하여 LabVIEW가 플롯 범례에 나타내는 플롯의 개수를 프로그램적으로 설정할 수 있습니다. 범례:최소 플롯을 사용하여 그래프 또는 차트의 플롯 영역 안에 있는 플롯의 개수에 관계없이 플롯 범례 안에 보여줄 플롯 개수의 최소치를 설정합니다.

그래프 또는 차트의 보기를 줌 또는 이동하기

그래프 또는 차트에서 보이지 않는 영역으로 줌 인, 줌 아웃 또는 이동하려면, **그래프 또는 차트**에서 마우스 오른쪽 버튼을 클릭하고 보이는 아이템 >> 그래프 팔레트를 선택하여 다음 그림과 같이 그래프 팔레트를 디스플레이합니다.



노트

또한  을 클릭하여 그래프에서 **커서**를 옮길 수도 있습니다.

줌 인 또는 줌 아웃

 을 클릭하고 다음 그림 중에서 줌 인 또는 줌 아웃을 선택합니다.

	그래프 또는 차트의 영역을 줌 인합니다.
	그래프 또는 차트의 x 축 영역을 줌 인합니다.
	그래프 또는 차트의 y 축 영역을 줌 인합니다.
	줌 인 또는 줌 아웃하여 그래프 또는 차트를 오토스케일 합니다.
	줌 인합니다. 줌 인하는 도중 줌 아웃하려면 <Shift> 키를 누르고 다시 줌 인하려면 <Shift> 키를 놓습니다.



줌 아웃합니다. 줌 아웃하는 도중 줌 인하려면 <Shift> 키를 누르고, <Shift> 키를 놓아 다시 줌 아웃합니다.

보이지 않는 영역으로 보기 이동하기

다음 단계를 완료하여 그래프 또는 차트의 보이지 않는 영역으로 보기를 이동합니다.

1. 를 클릭합니다.
2. 마우스를 놓지 않은 상태에서 그래프 또는 차트를 클릭합니다.
3. 마우스를 축에서 원하는 쪽으로 이동합니다.

그래프에 주석 추가하기

그래프에서 주석을 사용하여 **플롯 영역의 데이터 포인트를 하이라이트**합니다. 다음 단계를 따라 그래프에 주석을 추가합니다.

1. 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리>> 주석 생성을 선택하여 주석 생성 대화 상자를 디스플레이합니다. 주석 생성 대화 상자를 사용하여 주석 이름을 지정하고 어떻게 주석이 플롯 영역의 플롯에 고정 되는지 지정합니다.
2. 확인 버튼을 클릭하여 주석 생성 대화 상자를 닫습니다. 새로운 주석이 그래프에 나타납니다.
3. 주석에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 옵션을 사용하여 주석을 사용자 정의합니다:
 - 고정 스타일—주석이 플롯에 고정되는 방법을 설정하며 플롯 영역으로 이동될 수 있습니다. 고정 스타일에는 다음 옵션이 있습니다:
 - 모든 플롯에 고정—플롯 영역에서 주석을 가장 가까운 데이터 포인트를 가진 플롯으로 이동할 수 있도록 합니다.
 - 한 플롯에 고정—지정된 플롯을 따라서만 주석을 이동할 수 있습니다.
 - 독립—주석을 플롯 영역의 어디로든 이동시킬 수 있습니다. LabVIEW는 주석을 플롯 영역의 어느 플롯에도 고정시키지 않습니다.
 - 속성—각 주석의 모양과 동작을 사용자 정의합니다. 속성은 다음 옵션을 포함합니다:
 - 색—포인트, 화살표, 이름을 포함한 주석의 색을 선택할 수 있도록 **색 선택기**를 디스플레이합니다.
 - 주석 스타일—주석의 다양한 스타일을 제공합니다. LabVIEW는 선택된 스타일을 포인트 스타일 옵션과 함께 주석의 중심 포인트에 그리습니다.
 - 포인트 스타일—주석의 다양한 포인트 스타일을 제공합니다. LabVIEW는 선택된 스타일을 커서 스타일 옵션과 함께 주석의 중심 포인트에 그리습니다.

- 라인 스타일—주석의 다양한 기본 라인과 점선 라인 스타일을 제공합니다.
- 라인 폭—주석의 다양한 라인 폭을 제공합니다.
- 이름 보이기—플롯 영역에 주석 이름을 디스플레이합니다.
- 끌기 허용—주석을 끌 수 있습니다.
- 이름 고정—주석 이름을 절대적으로 위치시켜 주석을 이동하거나 플롯 영역을 스크롤해도 주석 이름은 플롯 영역에서 움직이지 않도록 합니다.
- 화살표 보이기—주석 이름에서 주석된 데이터 포인트를 가리키는 화살표를 디스플레이합니다.
- x 스케일—픽셀 위치를 주석의 데이터 값으로 맵핑하는 x 스케일을 선택할 수 있습니다. 이 옵션은 독립 주석에만 적용됩니다.
- y 스케일—픽셀 위치를 주석의 데이터 값으로 맵핑하는 y 스케일을 선택할 수 있습니다. 이 옵션은 독립 주석에만 적용됩니다.
- 주석 삭제—그래프에서 주석을 삭제합니다.



노트

그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 처리 >> 모든 주석 삭제를 선택하여 플롯 영역의 모든 주석을 삭제합니다.

1. 주석의 크로스헤어를 클릭하고 끌어 주석을 플롯 영역의 다른 데이터 포인트로 이동합니다. 주석 이름을 클릭하고 끌어 플롯 영역의 다른 위치로 이동합니다.

주석 리스트 프로퍼티를 사용하여 프로그램적으로 플롯 영역의 모든 주석에 대한 정보의 배열을 얻습니다.

그래프에 커서 추가

다음 단계에 따라 **그래프**에 **커서**를 추가합니다.

1. 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 커서 범례를 선택합니다.
2. 커서 범례에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 커서 생성을 선택한 후 커서 모드를 선택합니다. 커서 위치는 커서 모드에서 정의합니다. 커서에는 다음과 같은 모드가 있습니다.
 - 독립 — 플롯 위치에 구애받지 않고 플롯 영역에서 자유롭게 커서를 움직입니다.
 - 단일 플롯 — 커서와 연관되어 있는 플롯에만 커서를 위치시킵니다. 커서를 대응되는 플롯에 따라 움직일 수 있습니다. 커서 범례에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고정을 선택하여 하나 또는 모든 플롯을 커서와 연관시킵니다.
 - 멀티플롯 — 플롯 영역의 특정 데이터 포인트에서만 커서가 위치합니다. 멀티플롯 커서는 커서가 연관된 모든 플롯의 지정된 x값에서 값을 리포트합니다. 플롯 영역의 어느 플롯에나 커서를 놓을 수 있습니다. 커서 범례에서

마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고정을 선택하여 하나 또는 모든 플롯을 커서와 연관시킵니다. 이 모드는 복합 신호 그래프에서만 사용 가능합니다.



노트

커서의 모드를 생성한 후 변경할 수 없습니다. 반드시 **커서를 삭제**하고 다른 커서를 생성해야 합니다.

1. 스케일 범례 행에서 아무 곳이나 클릭하여 커서를 선택합니다. 수행 도구 또는 라벨링 도구를 사용하여 커서 범례에 이름과 좌표를 입력합니다.
2. 커서 범례에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 옵션을 선택하여 커서를 사용자 정의합니다:

- 보기—커서 모드가 멀티플롯일 때, 하나 또는 그 이상의 지정된 플롯이나 모든 플롯을 커서와 연관시킵니다. 멀티플롯 커서는 커서가 연관된 모든 플롯의 지정된 x값에서 값을 리포트합니다.

처음에 보려는 개별 플롯을 선택한 후 보기»모든 플롯을 대신 선택한 경우, 커서는 모든 플롯의 값을 리포트하기 시작합니다. 그 후 모든 플롯을 선택 해제하면, 커서는 처음에 선택한 플롯의 값을 리포트합니다.

- 고정—플롯에 커서를 고정시킵니다. 커서 모드가 단일 플롯일 때, 커서를 하나 또는 모든 플롯에 고정할 수 있습니다. 커서 모드가 멀티플롯일 때 고정은 커서가 x축을 따라 움직이는 방법에만 영향을 미칩니다.

노트 플롯 영역에 존재하는 데이터가 없는 경우, 커서는 모든 플롯 또는 플롯 범례에 열거된 첫 번째 플롯에만 고정됩니다. 모든 플롯 또는 플롯 범례에 열거된 첫 번째 플롯이 아닌 다른 플롯에 커서를 고정시키려면, 1개 이상의 플롯에 대한 데이터가 있어야 합니다. 예를 들어, 3번째 플롯에 커서를 고정시키려면 우선 플롯 영역에 3 개의 데이터 세트가 있어야 합니다.

- x 스케일—커서의 x 스케일을 설정합니다. 이 옵션은 커서 모드가 독립일 경우에만 유효합니다.
- y 스케일—커서의 y 스케일을 설정합니다. 이 옵션은 커서 모드가 독립일 경우에만 유효합니다.
- 속성—각 커서의 모양을 사용자 정의합니다. 속성은 다음 옵션을 포함합니다:
 - 색—커서의 색을 선택할 수 있도록 **색 선택기**를 디스플레이합니다.
 - 커서 스타일—다양한 커서 스타일을 제공합니다.
 - 포인트 스타일—커서의 교차점에 대한 다양한 포인트 스타일을 제공합니다.
 - 라인 스타일—다양한 기본 라인과 점선 라인 스타일을 제공합니다.
 - 라인 폭—다양한 라인 폭을 제공합니다.
 - 이름 보이기—그래프에 커서의 이름을 디스플레이합니다. 위치 도구를 사용하여 커서에 관계된 이름을 이동합니다.
 - 끌기 허용—마우스가 커서를 끌 수 있도록 합니다.
- 중심에 정렬—x 스케일과 y 스케일 변경없이 커서를 중앙으로 이동합니다. 커서 모드가 단일 플롯 또는 멀티플롯일 때, 이 옵션은 커서를 현재 위치한 플롯의 중앙에 놓고 커서 범례에서 커서 좌표를 업데이트합니다. 커서 모

드가 독립일 때, 이 옵션은 커서를 플롯 영역의 중앙에 놓고 커서 범례에서 커서 좌표를 업데이트합니다.

- 커서로 가기—커서를 그래프의 중앙에 보여주기 위해서 x 스케일과 y 스케일을 변경합니다.

또한 커서 프로퍼티를 사용하여 프로그램에서 커서를 사용자 정의할 수 있습니다.

플롯 영역을 혼합된 신호 그래프에 추가하기

혼합된 신호 그래프는 여러 개의 플롯 영역을 가질 수 있습니다. 주어진 플롯 영역은 디지털이나 아날로그 플롯 중 하나만을 디스플레이할 수 있습니다. 혼합된 신호 그래프는 아날로그와 디지털 데이터에 맞춰서 필요하면 자동적으로 플롯 영역을 생성합니다. 플롯 영역을 혼합된 신호 그래프에 추가하려면 혼합된 신호 그래프에서 마우스 오른쪽 버튼을 클릭하여 바로 가기 메뉴에서 플롯 영역에 추가를 선택합니다.

그래프에서 플롯 영역을 제거하려면, 플롯 영역에서 데이터를 모두 제거하고, 제거하려는 플롯 영역에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 플롯 영역 제거를 선택합니다.



노트

여러 개의 플롯 영역을 볼 수 있도록 그래프를 확대한 후 이 중 한 개의 플롯 영역을 제거하면 LabVIEW는 남아있는 플롯 영역의 크기를 조정하여 그래프 프레임을 채웁니다. 그래프의 크기를 줄여 플롯 영역의 크기를 줄일 수 있습니다.

디지털 웨이브폼 그래프에서 플롯 범례 보기 변경하기

디지털 웨이브폼 그래프는 두 가지의 플롯 범례 보기를 제공합니다. 플롯 범례 보기를 **표준 보기** 또는 **트리 보기** 중 하나로 변경하여 디지털 라인과 버스 정보를 디스플레이하십시오. 트리 보기에서는 디지털 버스를 그래프의 플롯 범례와 플롯 영역 모두에서 확장하고 수축할 수 있지만, 표준 보기에서는 디지털 라인을 볼 수만 있습니다.

플롯 범례를 트리 보기에서 표준 보기로 변경하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 플롯 범례를 표준 보기로 변경을 선택합니다. 트리 보기로 다시 되돌아가려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 플롯 범례를 트리 보기로 변경을 선택합니다.



노트

플롯 범례의 표준 보기에서 트리 보기로 변경하면 LabVIEW는 사용자 정의된 **사용자 라인 셋팅**을 포함하여 모든 **활당된 라인**을 제거합니다. 또한, 트리 보기가 활성화된 경우, 라인을 플롯 범례에 할당할 수 없습니다.

선택한 보기에 상관없이 플롯 범례를 비활성화하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급 >> 플롯 범례 비활성화를 선택하여 메뉴 아이템 옆에 확인표시를 합니다. 또는 범례:비활성화 프로퍼티를 사용하여 프로그램적으로 플롯 범례를 비활성화할 수 있습니다.

또한 디지털 트리 보기 프로퍼티를 사용하여 디지털 웨이브폼 그래프의 플롯 범례의 보기를 프로그램적으로 변경할 수 있습니다.

그래프 생성하기

그래프가 포함된 VI는 일반적으로 배열로 데이터를 수집한 후, 이 데이터를 그래프로 플롯합니다. 이 과정은 데이터를 먼저 저장한 후 데이터의 플롯을 생성하는 스프레드시트와 비슷합니다.

다음 단계를 따라 그래프를 생성합니다.

1. 다음 그래프 중에서 하나를 프런트패널 윈도우에 추가합니다:
 - 웨이브폼 그래프
 - XY 그래프
 - 디지털 웨이브폼 그래프
 - 혼합된 신호 그래프
 - 강도 그래프
2. 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴 항목을 사용하여 그래프를 사용자 정의합니다. x와 y 스케일을 설정하고, x와 y 스케일을 포맷하고, 플롯을 추가하고, 확대 또는 축소하고, 커서를 추가할 수 있습니다.

- 블록다이어그램을 만들어 데이터를 웨이브폼으로 디스플레이하거나, XY, 혼합된 신호 또는 디지털 웨이브폼 그래프에 디스플레이합니다.



팁

그래프에서 Microsoft Excel과 같은 위치로 데이터와 이미지를 반출합니다. 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반출을 선택하여 사용할 가능한 반출 옵션을 봅니다.



관련 정보

[그래프와 차트](#) (432 페이지 참조)

[웨이브폼 데이터를 그래프로 나타내기](#) (1164 페이지 참조)

[XY 데이터를 그래프로 나타내기](#) (1165 페이지 참조)

[디지털 데이터를 그래프로 나타내기](#) (1163 페이지 참조)

[그래프와 차트의 종류](#) (433 페이지 참조)

[그래프와 차트의 종류](#) (433 페이지 참조)

[그래프 또는 차트에서 X축과 Y축 설정하기](#) (1146 페이지 참조)

[그래프와 차트 축의 포맷과 정밀도 변경하기](#) (1145 페이지 참조)

[플롯 범례에 플롯 추가하기](#) (1143 페이지 참조)

[그래프 또는 차트의 보기를 줌 또는 이동하기](#) (1149 페이지 참조)

[그래프에 커서 추가](#) (1151 페이지 참조)

[웨이브폼 데이터를 그래프로 나타내기](#) (1164 페이지 참조)

[XY 데이터를 그래프로 나타내기](#) (1165 페이지 참조)

[여러 데이터 타입을 혼합된 신호 그래프에서 그래프로 나타내기](#) (1163 페이지 참조)

[디지털 데이터를 그래프로 나타내기](#) (1163 페이지 참조)

[데이터와 이미지 반출하기](#) (1132 페이지 참조)

플롯 범례를 사용하여 디지털 웨이브폼 그래프에서 플롯 사용자 정의하기

각 플롯을 설정해서 디지털 웨이브폼 그래프에 숫자와 플롯 라인이 어떻게 나타나는지를 지정할 수 있습니다. 디지털 웨이브폼 플롯을 설정하려면, 플롯 범례의 문양을 클릭한 후, 바로 가기 메뉴에서 다음 옵션 중 하나를 선택합니다:



노트

플롯 설정하기를 통해 각 플롯이 그래프 또는 차트의 플롯 영역에 어떻게 나타나는지 사용자 정의할 수도 있습니다.

- 플롯 보이기—그래프나 차트에 플롯을 디스플레이할지 설정합니다.

- 색—플롯의 색을 선택할 수 있도록 색 선택기를 디스플레이합니다.
- 라벨 포맷—버스에서 숫자의 포맷을 설정합니다. 16진수, 10진수, 8진수, 2진수 또는 고정 소수점 포맷으로 디스플레이하거나, 없음 포맷을 선택하여 라인 또는 버스에서 라벨을 제거할 수 있습니다. 또한 고정 소수점 설정을 선택하여 고정 소수점 설정 대화 상자를 디스플레이하고, 고정 소수점 라벨 포맷 옵션을 설정할 수 있습니다.
- 변이 타입—LabVIEW가 플롯에서 다른 값을 구별하는 방법을 설정합니다. 이 셋팅은 단일 비트 이상의 플롯에만 영향을 줍니다. 사각 에지 또는 기울어진 에지를 선택할 수 있습니다. 단순한 상태 변경을 보여주려면 사각 에지를 사용합니다. 기울어진 에지를 사용하여 상태 간에 지터 또는 정정 시간을 강조합니다.
- 변이 위치—높은값에서 낮은값 변이를 이전 포인트, 포인트 사이, 혹은 x축의 새 포인트 중 어디에 디스플레이할지 설정합니다. 기본적으로 x축의 새 포인트에 높은값에서 낮은값 변이를 디스플레이합니다.
- 라인 스타일—LabVIEW가 높은 또는 낮은 값을 구별하거나 특정한 플롯 라인을 오프셋하기 위해 플롯에서 굵은 라인을 사용할지 얇은 라인을 사용할지를 설정합니다. 기본 라인을 두껍게 하려면 가장 왼쪽 옵션을 선택합니다.
- X 스케일—X 축과 연계할 변수를 설정합니다.
- Y 스케일—Y 축과 연계할 변수를 설정합니다.

또한 플롯 프로퍼티를 사용하여 프로그램적으로 플롯이 플롯 영역에 어떻게 나타날지를 사용자 정의할 수 있습니다.

플롯 범례를 사용하여 혼합된 신호 그래프에서 플롯 사용자 정의하기

없음

디지털 데이터 플롯 사용자 정의하기

각 라인 또는 버스를 사용자 정의하여 숫자와 디지털 라인이 혼합된 신호 그래프에서 보여지는 방식을 지정할 수 있습니다. 그래프에서 디지털 라인 또는 버스를 사용자 정의하려면, 플롯 범례의 라인 또는 버스 문양에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 옵션을 선택합니다:

- 플롯 보이기—그래프나 차트에 플롯을 디스플레이할지 설정합니다.
- 색—플롯의 색을 선택할 수 있도록 색 선택기를 디스플레이합니다.
- 라벨 포맷—버스에서 숫자의 포맷을 설정합니다. 16진수, 10진수, 8진수, 2진수 또는 고정 소수점 포맷으로 디스플레이하거나, 없음 포맷을 선택하여 라인 또는 버스에서 라벨을 제거할 수 있습니다. 또한 고정 소수점 설정을 선택하여 고정 소수점 설정 대화 상자를 디스플레이하고, 고정 소수점 라벨 포맷 옵션을 설정할 수 있습니다.



노트

LabVIEW는 디지털 그래프 플롯에 충분한 공간이 있을 때에만 플롯 라벨을 디스플레이합니다.

- 변이 타입—LabVIEW가 플롯에서 다른 값을 구별하는 방법을 설정합니다. 이 셋팅은 단일 비트 이상의 플롯에만 영향을 줍니다. 사각 에지 또는 기울어진 에지를 선택할 수 있습니다. 단순한 상태 변경을 보여주려면 사각 에지를 사용합니다. 기울어진 에지를 사용하여 상태 간에 지터 또는 정정 시간을 강조합니다.
- 변이 위치—높은값에서 낮은값 변이를 이전 포인트, 포인트 사이, 혹은 x축의 새 포인트 중 어디에 디스플레이할지 설정합니다. 기본적으로 x축의 새 포인트에 높은값에서 낮은값 변이를 디스플레이합니다.
- 라인 스타일—LabVIEW가 높은 또는 낮은 값을 구별하거나 특정한 플롯 라인을 오프셋하기 위해 플롯에서 굵은 라인을 사용할지 얇은 라인을 사용할지를 설정합니다. 기본 라인을 두껍게 하려면 가장 왼쪽 옵션을 선택합니다
- X 스케일—X 축과 연계할 변수를 설정합니다.
- Y 스케일—Y 축과 연계할 변수를 설정합니다.

또한 플롯 프로퍼티를 사용하여 프로그램적으로 디지털 데이터 플롯이 플롯 영역에 어떻게 나타날지를 사용자 정의할 수 있습니다.

관련 정보 플롯 프로퍼티 고정 소수점 설정 대화 상자 **플롯 영역을 혼합된 신호 그래프에 추가하기** 여러 데이터 타입을 혼합된 신호 그래프에서 그래프로 나타내기 플롯 범례를 사용하여 디지털 웨이브폼 플롯 사용자 정의하기

아날로그 데이터 플롯 사용자 정의하기

각 플롯을 사용자 정의하여 숫자와 아날로그 플롯이 혼합된 신호 그래프에서 보여지는 방식을 지정할 수 있습니다. 혼합된 신호 그래프에서 아날로그 플롯을 사용자 정의하려면, 플롯 범례의 문양에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 중 하나의 옵션을 선택합니다:

- 플롯 보이기—그래프나 차트에 플롯을 디스플레이할지 설정합니다.
- 일반 플롯—플롯 타입을 설정합니다. 라인, 산포, 라인과 기호, 채우기, 빗, 막대 플롯 타입을 선택할 수 있습니다.
- 색—플롯의 색을 선택할 수 있도록 색 선택기를 디스플레이합니다.
- 라인 스타일—선택한 플롯 라인의 라인 스타일을 설정합니다. 기본 라인과 점선 라인 스타일에서 선택할 수 있습니다.
- 라인 폭—선택한 플롯 라인의 두께를 설정합니다.
- 엘리어스 제거—엘리어스 제거를 사용할지 여부를 설정합니다. 이 아이টে를 선택하여 라인 플롯을 자연스럽게 나타낼 수 있습니다. 엘리어스 제거는 복잡한 계산이 필요하므로 성능을 저하시킬 수 있습니다.
- 막대 플롯—LabVIEW가 막대 플롯을 디스플레이할지 여부 및 디스플레이 방식을 설정합니다.
- 베이스 라인 채우기—플롯과 같은 색으로 플롯 위 또는 아래 공간을 채우는 옵션을 설정합니다. 그래프 또는 차트가 여러 개의 플롯을 갖는 경우, 바로 가기 메뉴에서 가장 아래쪽에 있는 두 번째 플롯을 선택하여 해당 플롯에서 첫번째 플롯까지의 공간을 채울 수 있습니다.
- 보간—데이터 포인트만을 플롯하는 것을 포함하여 보간 옵션을 설정합니다.
- 포인트 스타일—선택한 플롯 라인의 데이터 포인트에 대한 포인트 스타일을 설정합니다.



노트 데이터 포인트의 크기를 변경하려면 라인 폭 바로 가기 옵션을 사용하여 해당 플롯의 라인 폭을 변경해야 합니다.

- X 스케일—X 축과 연계할 변수를 설정합니다.
- Y 스케일—Y 축과 연계할 변수를 설정합니다.

또한 플롯 프로퍼티를 사용하여 프로그램적으로 아날로그 데이터 플롯이 플롯 영역에 어떻게 나타날지를 사용자 정의할 수 있습니다.

관련 정보 플롯 프로퍼티 [플롯 영역을 혼합된 신호 그래프에 추가하기](#) [여러 데이터 타입을 혼합된 신호 그래프에서 그래프로 나타내기](#) [플롯 범례를 사용하여 아날로그 플롯 사용자 정의하기](#)

그래프에서 커서 삭제하기

다음 단계를 따라 [그래프](#)에서 [커서](#)를 삭제합니다.

1. 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 커서 범례를 선택합니다.
2. 나타난 커서 범례에서, 삭제하려는 커서의 옆에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 커서 삭제를 선택합니다.

디지털 웨이브폼 그래프에 디지털 라인과 버스 디스플레이하기

디지털 웨이브폼 그래프를 사용자 정의하여 플롯 영역에 보려는 데이터만을 디스플레이합니다. 다음 섹션의 다음 단계를 따라 디지털 웨이브폼 그래프를 사용자 정의하여 디지털 라인과 버스, 디지털 라인 또는 디지털 버스를 디스플레이합니다.

디지털 라인과 버스 디스플레이하기

[디지털 신호 그룹화] 익스프레스 VI를 사용하여, 디지털 라인 및 버스를 디지털 배열로 구성하고 라벨을 붙입니다. 이 익스프레스 VI는 설정 대화 상자에서 사용자가 지정한 순서에 따라 라인과 버스를 디지털 배열에 반환합니다.

기본으로, 디지털 웨이브폼 그래프는 축소된 디지털 라인과 버스를 디스플레이합니다. 이 기본 동작이 활성화되어있는지 점검하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭하여 고급 >> 라인 있는 버스 보이기 메뉴 아이템 옆에 확인 표시가 있는지 보십시오.

디지털 라인만 디스플레이하기

기본으로, LabVIEW는 고급>>라인 있는 버스 보이기를 활성화합니다. 디지털 라인만을 디스플레이하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급>>라인 있는 버스 보이기를 선택하여 확인 표시를 제거합니다.

디지털 버스만 디스플레이하기

기본적으로, LabVIEW는 고급>>라인 있는 버스 보이기 및 Y 스케일>>디지털 버스 확장을 활성화합니다. 디지털 버스만을 디스플레이하려면, 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 고급>>라인 있는 버스 보이기를 선택하고 확인 표시를 제거하여 고급>>라인 있는 버스 보이기를 비활성화합니다. 디지털 웨이브폼 그래프에서 마우스 오른쪽 버튼을 클릭하고 Y 스케일>>디지털 버스 확장을 선택하여 메뉴 아이템 옆의 확인 표시를 제거합니다.



노트

Y 스케일>>디지털 버스 확장은 고급>>라인 있는 버스 보이기가 비활성화된 상태이고 플롯 범례가 **표준 보기**인 경우에만 사용 가능합니다.

또는 라인 있는 버스 보이기 프로퍼티를 사용하여 디지털 웨이브폼 그래프의 플롯 영역에 나타나는 디지털 라인과 버스를 프로그램적으로 사용자 정의할 수 있습니다.

XY 그래프에 특별한 평면 디스플레이하기

XY 그래프에 **나이퀴스트 평면**, **니콜스 평면**, **s 평면**, **z 평면**을 디스플레이할 수 있습니다.



노트

평면 옵션 서브메뉴에서 평면을 선택해도 그 평면을 디스플레이하기 위해 그래프를 자동적으로 스케일하지는 않습니다. 특정 값은 이러한 평면의 가시 범위 밖에 위치합니다. 다음 단계는 평면에 보여주기 위해 사용할 수 있는 값을 포함합니다.

나이퀴스트 평면 디스플레이하기

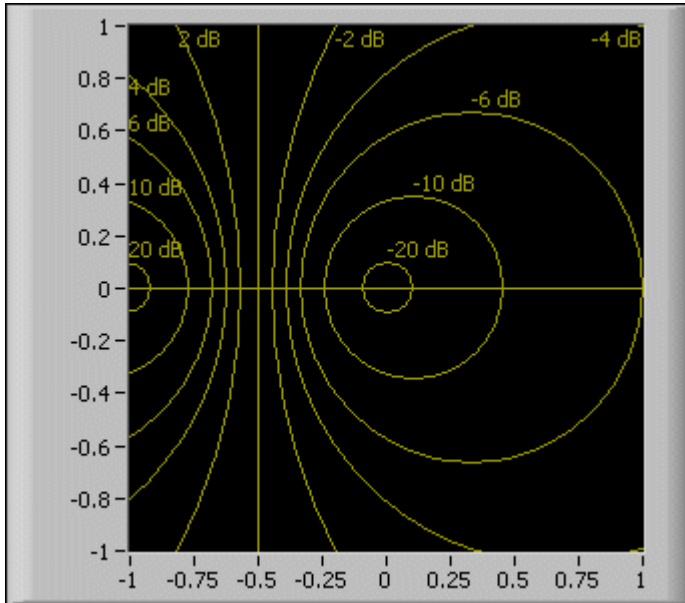
다음 단계를 따라 나이퀴스트 평면을 디스플레이합니다.

1. 프런트패널에 [XY 그래프]를 추가합니다.
2. x와 y 스케일의 최소값과 최대값을 더블 클릭한 후 최소값을 -1로 최대값을 1로 변경합니다.
3. XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션>>나이퀴스트를 선택합니다.
4. (옵션) 나이퀴스트 평면을 라벨없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션>>나이퀴스트 라벨 보이기 바로 가기 메뉴 아이템에서 확인 표시를 제거합니다.

5. (옵션) 나이퀴스트 평면을 직각좌표 라인없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션» 직각좌표 라인 보이기 바로 가기 메뉴 아이টে에서 확인 표시를 제거합니다.

나이퀴스트 평면을 제거하려면, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션» 없음을 선택합니다.

다음 프런트패널은 직각좌표 라인이 없는 나이퀴스트 평면을 보여줍니다.



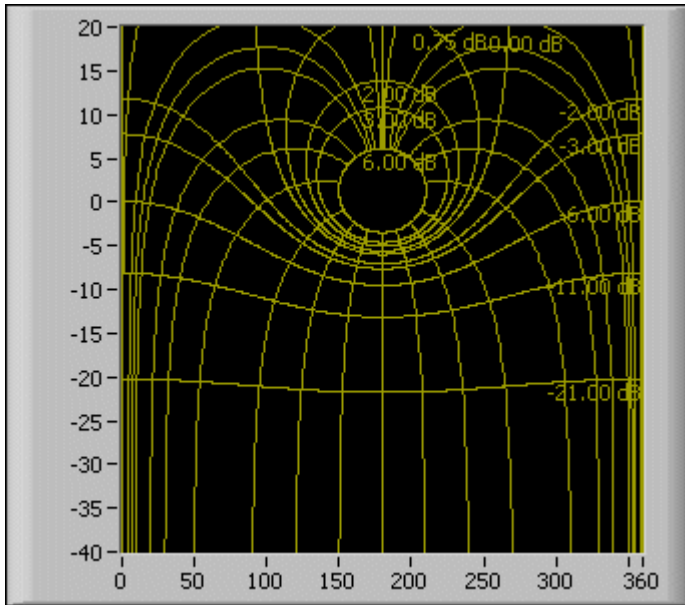
니콜스 평면 디스플레이하기

다음 단계를 따라 니콜스 평면을 디스플레이합니다.

1. 프런트패널에 [XY 그래프]를 추가합니다.
2. x 스케일의 최소값과 최대값을 더블 클릭한 후 최소값을 0으로, 최대값을 360으로 변경합니다.
3. y 스케일의 최소값과 최대값을 더블 클릭한 후 최소값을 -40으로, 최대값을 20으로 변경합니다.
4. XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션» 니콜스를 선택합니다.
5. (옵션) 니콜스 평면을 라벨없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션» 니콜스 라벨 보이기 바로 가기 메뉴 아이টে에서 확인 표시를 제거합니다.
6. (옵션) 니콜스 평면을 직각좌표 라인없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션» 직각좌표 라인 보이기 바로 가기 메뉴 아이টে에서 확인 표시를 제거합니다.

니콜스 평면을 제거하려면, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션» 없음을 선택합니다.

다음 프런트패널은 직각좌표 라인이 없는 니콜스 평면을 보여줍니다.



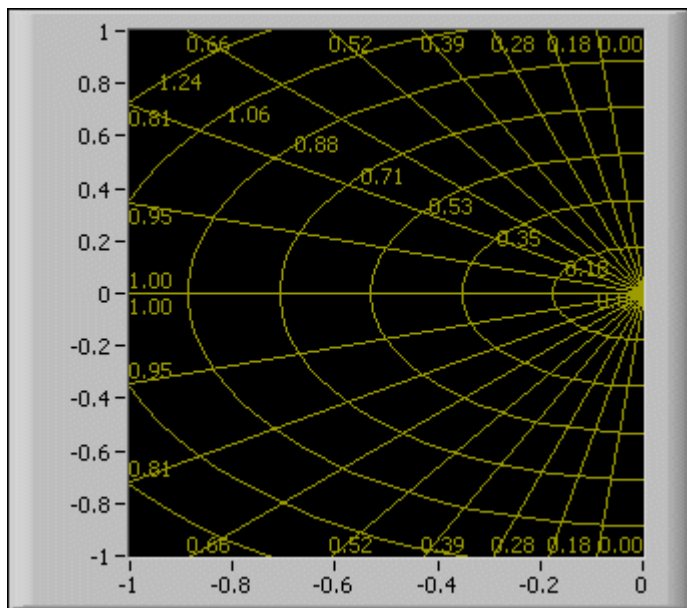
S 평면 디스플레이하기

다음 단계를 따라 S 평면을 디스플레이합니다.

1. 프런트패널에 [XY 그래프]를 추가합니다.
2. x 스케일의 최소값과 최대값을 더블 클릭한 후 최소값을 -1로 최대값을 0으로 변경합니다.
3. y 스케일의 최소값과 최대값을 더블 클릭한 후 최소값을 -1으로, 최대값을 1으로 변경합니다.
4. XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션» S 평면을 선택합니다.
5. (옵션) S 평면을 라벨없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션» S 평면 라벨 보이기 바로 가기 메뉴 아이템에서 확인 표시를 제거합니다.
6. (옵션) S 평면을 직각좌표 라인없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션» 직각좌표 라인 보이기 바로 가기 메뉴 아이템에서 확인 표시를 제거합니다.

S 평면을 제거하려면, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션» 없음을 선택합니다.

다음 프런트패널은 직각좌표 라인이 없는 S 평면을 보여줍니다.



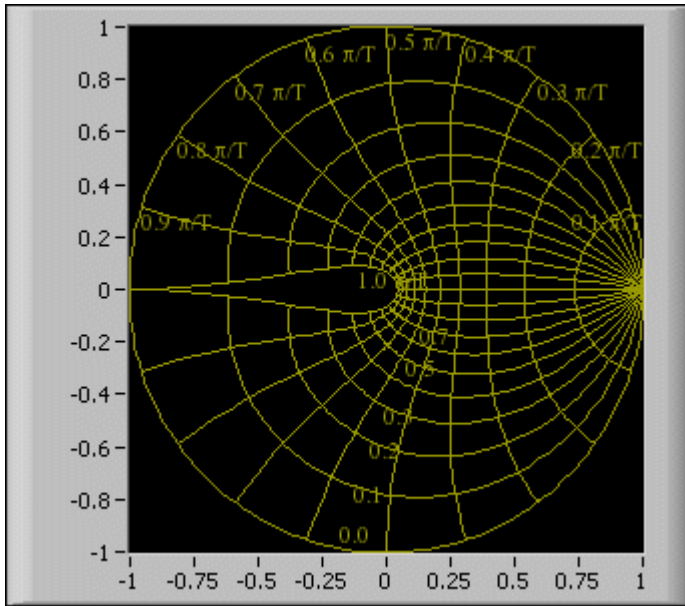
Z 평면 디스플레이하기

다음 단계를 따라 Z 평면을 디스플레이합니다.

1. 프런트패널에 [XY 그래프]를 추가합니다.
2. x와 y 스케일의 최소값과 최대값을 더블 클릭한 후 최소값을 -1로 최대값을 1로 변경합니다.
3. XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션»Z 평면을 선택합니다.
4. (옵션) Z 평면을 라벨없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션»Z 평면 라벨 보이기 바로 가기 메뉴 아이템에서 확인 표시를 제거합니다.
5. (옵션) Z 평면을 직각좌표 라인없이 보려는 경우, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 평면 옵션»직각좌표 라인 보이기 바로 가기 메뉴 아이템에서 확인 표시를 제거합니다.

Z 평면을 제거하려면, XY 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 평면 옵션»없음을 선택합니다.

다음 프런트패널은 직각좌표 라인이 없는 Z 평면을 보여줍니다.



또한, 평면 옵션:평면 타입 프로퍼티를 사용하여 XY 그래프에서 이 평면을 프로그램적으로 디스플레이하고 숨길 수 있습니다. 평면 옵션:라인 보이기, 평면 옵션:라벨 보이기, 직각좌표 라인:보이기 프로퍼티를 사용하여 평면 라인, 평면 라벨, 직각좌표 라인을 XY 그래프에 프로그램적으로 디스플레이하고 숨깁니다.

디지털 데이터를 그래프로 나타내기

다음 단계를 따라 **디지털 웨이브폼 그래프**에서 디지털 신호를 디스플레이합니다.

1. 디지털 신호를 얻거나 디지털 신호로 데이터를 변경합니다.
2. 프런트패널에 **디지털 웨이브폼 그래프**를 추가합니다.
3. 블록 다이어그램에 디지털 데이터 출력이나 디지털 웨이브폼 출력 결과값을 디지털 웨이브폼 그래프 터미널에 연결합니다.
4. VI를 실행합니다.

디지털 데이터를 그래프로 나타내는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Digital Waveform Graph 디렉토리에 있는 Create Digital Waveform VI를 참조하십시오.

 예제 열기  관련 예제 찾기

여러 데이터 타입을 혼합된 신호 그래프에서 그래프로 나타내기

여러 데이터를 혼합된 신호 그래프에서 그래프로 나타내는 것은 웨이브폼 데이터, XY 데이터 및 디지털 데이터를 다른 그래프에서 그래프로 나타내는 것과 비슷합니다. 웨이브

폼, XY, 또는 디지털 그래프가 받는 모든 데이터 타입은 혼합된 신호 그래프에 연결할 수 있습니다.

다음 단계를 완료하여 여러 데이터 타입을 혼합된 신호 그래프에 그리십시오.

1. 데이터를 직접 그래프에 연결하기 전에 [묶기] 함수를 사용하여 데이터 타입을 혼합된 신호 그래프로 병합할 수 있습니다.
2. 데이터를 [묶기] 함수의 원소에 연결합니다.
3. [묶기] 함수의 출력 클러스터를 혼합된 신호 그래프에 연결합니다. LabVIEW는 자동적으로 아날로그 데이터와 디지털 데이터의 조합을 수용할 플롯 영역을 생성합니다. 기본적으로, 혼합된 신호 그래프는 디지털 데이터를 플롯 영역에 디지털 라인 과 버스로 디스플레이합니다.

여러 데이터 타입을 혼합된 신호 그래프에 그래프로 나타내는 예제는 labview \examples\Controls and Indicators\Graphs and Charts\Mixed Signal Graph 디렉토리의 Mixed Signal Graph VI를 참조하십시오.

관련 정보 [웨이브폼 데이터를 그래프로 나타내기](#) [XY 데이터를 그래프로 나타내기](#) [디지털 데이터를 그래프로 나타내기](#) [그래프와 차트의 종류](#) [묶기] 함수

 예제 열기  관련 예제 찾기

웨이브폼 데이터를 그래프로 나타내기

다음 단계를 따라 웨이브폼 데이터를 단일 플롯 [웨이브폼 그래프](#)에 디스플레이합니다.

1. 프런트패널에 [웨이브폼 그래프](#)를 추가합니다.
2. 다음 객체를 가지고 블록다이어그램을 만듭니다:
 - [나누기] 함수
 - [사인] 함수
 - 웨이브폼 그래프 터미널을 제외한 블록다이어그램의 모든 객체를 둘러싸는 For 루프
3. [나누기] 함수의 x/y 출력을 [사인] 함수의 x 입력에 연결합니다.
4. [사인] 함수의 $\sin(x)$ 출력을 웨이브폼 그래프 터미널에 연결합니다. [오토인덱싱](#) 터널이 와이어가 루프 경계를 가로지르는 곳에 나타납니다.
5. For 루프의 반복 터미널을 [나누기] 함수의 x 입력에 연결합니다.
6. [나누기] 함수의 y 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 [생성 >> 상수](#)를 선택합니다. 숫자형 상수가 나타납니다.
7. 상수에 10을 입력한 후 <Enter> 키를 누릅니다.
8. For 루프의 카운트 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 [상수 생성](#)을 선택합니다. 숫자형 상수가 나타납니다.
9. 상수에 100을 입력한 후 <Enter> 키를 누릅니다.
10. VI를 실행합니다.

웨이브폼 데이터를 그래프로 나타내는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts 디렉토리에 있는 Waveform Graph Data Types VI를 참조하십시오.

 예제 열기  관련 예제 찾기

XY 데이터를 그래프로 나타내기

다음 단계를 따라 포인트 배열을 단일 플롯 **XY 그래프**에 디스플레이합니다.

1. 프런트패널에 **XY 그래프**를 추가합니다.
2. 다음 객체를 가지고 블록다이어그램을 만듭니다:
 - [나누기] 함수
 - [곱하기] 함수
 - [사인] 함수
 - [묶기] 함수
 - XY 그래프 터미널을 제외한 블록다이어그램의 모든 객체를 둘러싸는 For 루프
3. [나누기] 함수의 x/y 출력을 [사인] 함수의 x 입력에 연결합니다.
4. [사인] 함수의 $\sin(x)$ 출력을 [묶기] 함수의 첫번째 입력 터미널에 연결합니다.
5. [곱하기] 함수의 $x*y$ 출력을 [묶기] 함수의 두번째 입력 터미널에 연결합니다.
6. For 루프의 반복 터미널을 [나누기] 함수의 x 입력에 그리고 [곱하기] 함수의 x 입력에 연결합니다.
7. [나누기] 함수의 y 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **생성» 상수**를 선택합니다. 숫자형 상수가 나타납니다.
8. 상수에 10을 입력한 후 <Enter> 키를 누릅니다.
9. 상수를 [곱하기] 함수의 y 입력 터미널에 연결합니다.
10. [묶기] 함수의 출력 터미널을 XY 그래프 터미널에 연결합니다. **오토인덱싱** 터널이 와이어가 루프 경계를 가로지르는 곳에 나타납니다.
11. For 루프의 카운트 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **상수 생성**을 선택합니다. 숫자형 상수가 나타납니다.
12. 상수에 100을 입력한 후 <Enter> 키를 누릅니다.
13. VI를 실행합니다.

XY 데이터를 그래프로 나타내는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\XY Graph 디렉토리에 있는 XY Graph Data Types VI를 참조하십시오.

 예제 열기  관련 예제 찾기

그래프에서 타임스탬프와 플롯 라벨을 무시하기

다이나믹 데이터 또는 웨이브폼 데이터를 그래프에 연결하고 VI를 실행할 때, LabVIEW는 다이나믹 또는 웨이브폼 데이터의 설정에 따라 자동으로 그래프의 플롯 범례와 x 스케일 타임스탬프를 포맷합니다.

다이나믹 또는 웨이브폼 데이터가 포함하는 플롯 범례를 무시하려면, 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 속성 무시를 선택합니다. 속성 무시를 선택하면, 플롯 범례 라벨을 프로그램적으로 변경할 수 있습니다.

다이나믹 또는 웨이브폼 데이터가 포함하는 타임스탬프 설정을 무시하려면, 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타임스탬프 무시를 선택합니다.



노트

그래프에 디스플레이한 데이터가 절대 시간에 대해 포맷한 타임스탬프를 포함하는 경우, 타임스탬프 무시를 선택하여 타임스탬프를 무시합니다.

그래프 좌표를 플롯 이미지 좌표에 맵핑하기

플롯 이미지 프로퍼티를 사용하는 경우, LabVIEW는 그래프 플롯 영역의 원점에서 이미지를 그리기 시작합니다. 만약 플롯 이미지 프로퍼티를 사용하여 원점 (0,0)이 아닌 그래프 데이터 포인트를 기준으로 이미지를 그리고 싶으면, XY를 좌표에 맵핑 메소드를 사용하여 프런트패널의 좌표에 데이터 포인트의 그래프 좌표를 맵핑할 수 있습니다. XY를 좌표에 맵핑 메소드는 프런트패널 원점에 대해 상대적인 좌표를 반환합니다. 하지만, 정확하게 이미지를 그리기 위해서는 그래프 플롯 영역의 맨 왼쪽 위에 위치한 플롯 이미지 영역의 원점에 대해 상대적인 좌표를 만들어야 합니다. 이 차이를 확인하려면, XY를 좌표에 맵핑 메소드가 반환한 좌표에서 프런트패널의 원점으로부터 그래프 플롯 영역 오프셋을 빼야 합니다.

다음 단계를 따라 포함하는 구획의 좌표에 그래프 좌표를 프로그램적으로 맵핑합니다.



노트

분리자 막대를 사용하면 여러 구획이 생깁니다. 이후 포함하는 구획의 원점에서의 그래프 플롯 영역 오프셋을 기준으로 합니다.

1. 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인보크 노드>>XY를 좌표에 맵핑을 선택하여 XY를 좌표에 맵핑 인보크 노드를 생성합니다.
2. XY를 좌표에 맵핑 메소드의 XY 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 데이터 포인트의 그래프 좌표를 XY를 좌표에 맵핑 메소드의 XY입력에 연결합니다.
3. 그래프에서 마우스 오른쪽 버튼을 클릭하고 생성>>프로퍼티 노드>>플롯 영역 경계를 선택하여 플롯 영역 경계 프로퍼티 노드를 생성합니다.
4. 블록 다이어그램에 [이름으로 풀기] 함수를 추가합니다.
5. 플롯 영역 경계 프로퍼티 노드의 플롯 경계 출력을 [이름으로 풀기] 함수의 입력 클러스터 입력에 연결합니다.

6. [이름으로 풀기] 함수의 크기를 조정하여 왼쪽 및 위쪽 원소가 모두 나타나도록 합니다.
7. [묶기] 함수를 블록다이어그램에 추가합니다.
8. [이름으로 풀기] 함수의 왼쪽 및 위쪽 원소를 [묶기] 함수의 원소 입력에 연결합니다.
9. [빼기] 함수를 블록다이어그램에 추가합니다.
10. XY를 좌표에 맵핑 인보크 노드의 좌표 출력을 [빼기] 함수의 x 입력에 연결합니다.
11. [묶기] 함수의 출력 클러스터 출력을 [빼기] 함수의 y 입력에 연결합니다.
12. [빼기] 함수의 x-y 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다.

플롯 및 버스를 혼합된 신호 그래프의 다른 플롯 영역에 이동하기

혼합된 신호 그래프에서 플롯과 버스를 한 플롯 영역에서 다른 영역으로 이동시킬 수 있습니다. 예를 들어, 데이터를 보다 자세히 비교하고 싶은 경우 플롯이나 버스를 이동할 수 있습니다. 다음 단계를 따라 이 동작을 수행하십시오.



노트

디지털 플롯 라인이 버스 안에 있는 경우, 버스를 이동하여 라인을 이동해야 합니다.

1. 트리 컨트롤에서 옮기려는 플롯 또는 버스 이름 옆의 아이콘을 선택합니다.
2. 아이콘을 플롯 범례의 다른 플롯 영역으로 끕니다.

또한 웨이브폼 그래프, XY 그래프 및 차트 플롯 범례에서 플롯 또는 버스를 이동하는 것과 비슷하게 바로 가기 메뉴를 사용하여 혼합된 신호 그래프에서 플롯을 이동할 수 있습니다. 다음 단계를 통해 바로 가기 메뉴를 사용하여 플롯 또는 버스를 이동시키십시오.

1. 이동하고자 하는 플롯 범례의 플롯 또는 버스에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 Y 스케일과 플롯 또는 버스를 이동하려는 플롯 영역을 선택합니다.



노트 데이터 타입이 일치하지 않는 경우, LabVIEW는 바로 가기 메뉴의 플롯 영역을 비활성화합니다.

관련 정보 플롯 영역을 혼합된 신호 그래프에 추가하기

2D 그래프에 데이터 플롯하기

2D 그래프를 사용하여 2D에서 데이터를 플롯합니다. 2D 그래프를 프런트패널에 추가하면, 그래프가 다음의 보조 VI 중 한 개와 함께 블록다이어그램에 나타납니다.

- 2D 콤팩스 플롯 보조 VI
- 2D 에러 막대 플롯 보조 VI
- 2D 화살표 플롯 보조 VI
- XY 플롯 행렬 보조 VI

다음 단계에 따라 2D 그래프에 데이터를 플롯합니다.

1. 2D 그래프를 프런트패널에 추가합니다.
2. 블록다이어그램의 보조 VI에서 다형성 인스턴스를 선택합니다.
3. 2D 그래프 프로퍼티를 사용하여 그래프의 모양을 설정합니다. 프로퍼티 노드를 사용하여 블록다이어그램에서 모양을 설정합니다. 바로 가기 메뉴에서 프로퍼티를 선택하여 프로퍼티 대화 상자에서 모양을 설정합니다.
4. 각 입력에 데이터를 와이어로 연결하여 그래프를 플롯합니다.
5. VI를 실행합니다.



관련 정보

[그래프와 차트의 종류](#) (433 페이지 참조)

[어플리케이션과 VI 셋팅 변경하기](#) (186 페이지 참조)

3D 그래프에 데이터 플롯하기

3D 그래프를 사용하여 3D에서 데이터를 플롯합니다. 3D 그래프를 프런트패널에 추가하면, 그래프가 다음의 보조 VI 중 한 개와 함께 블록다이어그램에 나타납니다.

- 3D 산포 플롯 보조 VI
- 3D 줄기형 플롯 보조 VI
- 3D 헤성형 플롯 보조 VI
- 3D 표면 플롯 보조 VI
- 3D 컨투어 플롯 보조 VI
- 3D 메시 플롯 보조 VI
- 3D 워터폴 플롯 보조 VI
- 3D 화살형 플롯 보조 VI
- 3D 리본 플롯 보조 VI
- 3D 막대 플롯 보조 VI
- 3D 파이 플롯 보조 VI

다음 단계에 따라 3D 그래프에 데이터를 플롯합니다.



노트

다음 단계는 3D 표면 그래프, 3D 파라미터형 그래프, 3D 라인 그래프에는 적용되지 않습니다.

1. 3D 그래프를 프런트패널에 추가합니다.



노트 3D 그래프는 LabVIEW Full 및 Professional Development System에서만 제공됩니다.

1. 블록 다이어그램의 보조 VI에서 다형성 인스턴스를 선택합니다.
2. 3D 그래프 프로퍼티를 사용하여 그래프의 모양을 설정합니다. 프로퍼티 노드를 사용하여 블록 다이어그램에서 모양을 설정합니다. 바로 가기 메뉴에서 3D 플롯 프로퍼티를 선택하여, 3D 플롯 프로퍼티 대화 상자에서 모양을 설정합니다.
3. 각 입력에 데이터를 와이어로 연결하여 그래프를 플롯합니다.
4. VI를 실행합니다.



관련 정보

[그래프와 차트의 종류](#) (433 페이지 참조)

[3D 표면 그래프에 데이터 플롯하기](#) (1169 페이지 참조)

[어플리케이션과 VI 셋팅 변경하기](#) (186 페이지 참조)

3D 표면 그래프에 데이터 플롯하기

다음 단계를 완료하여 [3D 표면 그래프](#)에 데이터를 플롯합니다.

1. 프런트패널 윈도우에 [3D 표면 그래프를 추가](#)합니다.



노트

3D 그래프 컨트롤은 Windows의 LabVIEW Full 및 Professional Development System에서만 제공됩니다.

1. 블록 다이어그램에 있는 플롯 표면 설정 VI의 x 벡터 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 1D 배열 컨트롤을 생성합니다.
2. 플롯 표면 설정 VI의 y 벡터 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 1D 배열 컨트롤을 생성합니다.
3. 플롯 표면 설정 VI의 z 행렬 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>컨트롤을 선택하여 2D 배열 컨트롤을 생성합니다.
4. 프런트패널에서 각각의 배열 컨트롤에 배리언트 데이터를 입력합니다. [배열 차원](#)이 일치하는지 점검합니다. 2D 배열의 차원 크기는 1D 배열의 곱입니다.

5. 3D 그래프 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 3D 그래프 프로퍼티를 선택하여 3D 그래프 프로퍼티 대화 상자를 디스플레이합니다. 3D 그래프 프로퍼티 대화 상자의 옵션을 사용하여 3D 그래프의 모양을 사용자 정의합니다.
6. VI를 실행합니다. LabVIEW는 사용자가 입력한 데이터를 3D 그래프 표면에 디스플레이합니다.

3D 그래프에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 렌더링 윈도우를 선택하여, 독립적인 윈도우에서 3D 그래프를 볼 수 있습니다.

ActiveX 3D 표면 그래프에 X, Y, Z 데이터 플롯하기

다음 단계를 따라 **ActiveX 3D 표면 그래프**에 X, Y, Z 데이터를 플롯합니다.

1. 프런트패널에 **ActiveX 3D 표면 그래프**를 추가합니다.



노트

3D 그래프 컨트롤은 Windows의 LabVIEW Full 및 Professional Development System에서만 제공됩니다.

2. 블록다이어그램에서 [3D 표면] VI와 그 VI에 연결된 와이어를 삭제합니다. 이것은 3D 그래프를 위해 ActiveX 컨테이너 터미널을 남깁니다.
3. 블록다이어그램에 인보크 노드를 추가하고, 3D 그래프 컨테이너 터미널을 인보크 노드의 참조 입력에 연결합니다.
4. 인보크 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 메소드 선택 >> Plot3DMesh를 선택합니다. Plot3DMesh 메소드는 x, y, z 데이터 배열을 위해 **배리언트 데이터**를 받고 삼각형 형태의 표면을 플롯합니다.
5. 배리언트 데이터를 Plot3DMesh 메소드의 xVector, yVector, zVector 입력에 연결합니다. xVector 및 yVector의 값은 zVector 배열의 대응하는 지점을 지정합니다. xVector와 yVector의 기본값은 0, 1, 2, 3 등입니다. zVector 배열(인덱스 0,0)의 첫 번째 포인트는 (x 벡터[0], y 벡터[0]) 또는 (0,0)에 위치합니다. xVector를 -1, 1, 2, 3 등으로 변경할 경우 zVector의 첫 번째 포인트는 (-1,0)으로 이동합니다.
6. 크기 데이터를 Plot3DMesh 메소드의 wVector 입력에 연결하여 크기를 보여줍니다.
7. 그래프를 **스케일**하여 **배열 차원**이 일치하도록 합니다. 2D 배열의 차원 크기는 1D 배열의 곱입니다. 예를 들어, xVector 및 yVector는 zVector 2D 배열의 두 개 인덱스에 대응합니다. xVector의 원소 개수가 zVector 배열의 1차원 폭과 일치하지 않고, yVector의 원소 개수가 zVector 배열의 2차원 폭과 일치하지 않는 경우, 이 메소드는 xVector 및 yVector 모두의 값을 무시합니다.
8. VI를 실행합니다.

다음 단계를 따라 표면 대신에 각각의 포인트를 보여줍니다.

1. 프런트패널 그래프에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 CWGraph3D >> 속성을 선택합니다.
2. Plots 탭을 클릭하고 Plot style을 Point로 변경합니다.

3. 각 포인트에 대해 여러 색을 얻는 것을 피하려면, Color map style 풀다운 메뉴에서 None을 선택합니다.

노트 그래프에서 데이터가 보이도록 하려면 Point 페이지에서 기본 포인트 색과 크기를 변경해야 합니다.

4. 확인 버튼을 클릭합니다.
5. VI를 실행합니다.

또한 3D 그래프 X컨트롤에 **x, y, z 데이터를 플롯**할 수 있습니다.

3D 혜성형 그래프에서 애니메이션 다시 실행하기

다음 단계에 따라, VI를 실행한 후에 **3D 혜성형 그래프**의 애니메이션을 다시 실행합니다.

1. 혜성형 그래프에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 애니메이션 재생을 선택합니다.



노트

3D 그래프는 LabVIEW Full 및 Professional Development System에서만 제공됩니다.

혜성형 그래프를 사용하는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Math Plots - 3D 디렉토리에 있는 3D Comet Graph VI를 참조하십시오. 예제 열기 관련 예제 찾기

웨이브폼 데이터 차트에 그리기

다음 단계를 따라 웨이브폼 데이터를 단일 플롯, 단일 포인트 웨이브폼 차트에 그리십시오.

1. **웨이브폼 차트**를 프런트패널에 추가합니다.
2. 다음 객체를 가지고 블록다이어그램을 만듭니다:
 - [나누기] 함수
 - [사인] 함수
 - While 루프—블록다이어그램에 있는 객체를 모두 둘러싸도록 루프를 끌어서 놓음
3. [나누기] 함수의 x/y 출력을 [사인] 함수의 x 입력에 연결하고, [사인] 함수의 sin(x) 출력을 웨이브폼 차트 터미널에 연결합니다.
4. While 루프의 반복 터미널을 [나누기] 함수의 x 입력에 연결합니다.
5. [나누기] 함수의 y 입력 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택합니다. 숫자형 상수가 나타납니다.
6. 상수에 10을 입력한 후 <Enter> 키를 누릅니다.

7. 불리언 값을 While 루프의 조건 터미널에 연결합니다. 또한, 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 상수 생성 또는 컨트롤 생성을 선택하여 불리언 객체를 생성할 수 있습니다.
8. VI를 실행합니다.

웨이브폼 데이터를 차트에 그리는 예제는 labview\examples\Controls and Indicators\Graphs and Charts\Waveform Graphs and Charts 디렉토리에 있는 Waveform Chart Data Types and Update Modes VI를 참조하십시오.



예제 열기



관련 예제 찾기

차트 생성하기

차트는 새 데이터 포인트를 이미 그려진 데이터 포인트 뒤에 덧붙여서 디스플레이하고 히스토리를 생성합니다. 차트에서는 현재 읽은 값 또는 측정된 값을 이전에 수집된 데이터와 함께 볼 수 있습니다.

다음 단계를 따라 차트를 생성합니다.

1. 다음 차트 중에서 하나를 프런트패널 윈도우에 추가합니다:
 - 웨이브폼 차트
 - 강도 차트
2. 차트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴 항목을 사용하여 사용자 정의합니다. x와 y 스케일을 설정하고, x와 y 스케일을 포맷하고, 플롯을 추가하고, 확대 또는 축소하고, 차트를 통해 스크롤할 수 있습니다.

- 웨이브폼 차트에 데이터를 디스플레이하기 하기 위해서 블록다이어그램을 만듭니다.



팁

차트에서 Microsoft Excel과 같은 위치로 데이터와 이미지를 반출합니다. 차트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반출을 선택하여 사용 가능한 반출 옵션을 봅니다.



관련 정보

[그래프와 차트](#) (432 페이지 참조)

[그래프와 차트의 종류](#) (433 페이지 참조)

[그래프와 차트의 종류](#) (433 페이지 참조)

[그래프 또는 차트에서 X축과 Y축 설정하기](#) (1146 페이지 참조)

[그래프와 차트 축의 포맷과 정밀도 변경하기](#) (1145 페이지 참조)

[플롯 범례에 플롯 추가하기](#) (1143 페이지 참조)

[그래프 또는 차트의 보기를 줌 또는 이동하기](#) (1149 페이지 참조)

[그래프나 차트에서 스크롤하기](#) (1148 페이지 참조)

[웨이브폼 데이터 차트에 그리기](#) (1171 페이지 참조)

[데이터와 이미지 반출하기](#) (1132 페이지 참조)

웨이브폼의 숫자형 변경하기

웨이브폼 컨트롤의 웨이브폼 데이터 Y는 고정 소수점 숫자형 데이터 타입을 제외하면 모든 숫자형 데이터 타입의 배열이 될 수 있습니다. 웨이브폼 터미널의 내부는 데이터가 부동소수 또는 복소수인 경우 주황색으로, 정수인 경우 파란색으로 나타납니다.

다음 단계를 따라 웨이브폼의 Y 배열의 형을 변경합니다.

- 프론트패널이나 블록다이어그램에서, 웨이브폼 객체에서 마우스 오른쪽 버튼을 클릭합니다.
- 바로 가기 메뉴에서 Y 배열 형을 선택합니다.
- 데이터를 가장 잘 표현하는 [데이터 타입](#)을 선택합니다.

신호 수집 또는 시뮬레이션

[DAQ 어시스턴트] 익스프레스 VI를 사용하여 데이터 수집 디바이스를 통해 신호를 수집합니다. [신호 시뮬레이션] 익스프레스 VI를 사용하여 신호를 시뮬레이션합니다. 신호를

수집하거나 시뮬레이션한 후, 신호를 **그래프**로 나타내거나 **분석**하거나 또는 **텍스트 기반 측정 파일에 쓰기**를 수행할 수 있습니다.



노트

[DAQ 어시스턴트] 익스프레스 VI는 Windows에서만 사용 가능합니다.

서로 다른 DAQ 디바이스는 서로 다른 종류의 신호를 수집합니다. 일반적으로, 한 DAQ 디바이스로 두 종류의 신호(온도 읽기와 같은 단일 포인트와 웨이브폼과 같은 버퍼된 신호)를 수집할 수 있습니다. 또한 한 채널 또는 여러 채널로부터 데이터를 읽을 수 있습니다.

신호 분석하기

신호 분석 익스프레스 VI를 사용하여 수집하거나 시뮬레이션한 신호를 분석하고 측정합니다. 예를 들어, [스펙트럼 측정] 익스프레스 VI를 사용하여 신호의 FFT 측정 또는 파워 스펙트럼 측정을 수행할 수 있습니다.

연산 & 비교 익스프레스 VI와 숫자형, 불리언, 비교 함수를 사용하여 신호에 대해 수학적 인 분석을 수행합니다. 예를 들어, [수식] 익스프레스 VI를 사용하여 입력한 수식을 기반으로 신호를 분석할 수 있습니다.

각 익스프레스 VI가 신호를 분석하는 방법 또는 신호 분석에 알맞은 익스프레스 VI를 함수 팔레트에서 찾는 방법에 대한 정보는 기본 도움말 윈도우를 참조하십시오.



관련 정보

[신호 수집 또는 시뮬레이션](#) (1173 페이지 참조)

[팔레트에서 객체 찾기](#) (862 페이지 참조)

신호 디스플레이하기

신호를 **수집 또는 시뮬레이션**한 후, 그래프, 차트 또는 숫자형 인디케이터에 디스플레이할 수 있습니다. [신호 시뮬레이션], DAQ 어시스턴트, [신호 분석] 익스프레스 VI와 같은 많은 익스프레스 VI는 그래프, 차트, 온도계 또는 디지털 디스플레이와 같은 숫자형 인디케이터와 연결할 수 있도록 신호 출력을 포함합니다.

인디케이터는 신호의 타입에 의해 선택됩니다. 예를 들면, 일단 온도를 얻으면 온도계 또는 디지털 디스플레이에서 온도를 디스플레이합니다. 시간 주기를 넘어 여러 온도를 얻으면, 차트에 그 온도들을 디스플레이합니다. 웨이브폼 신호를 얻는다면, 사인파, 톱니파와 같은 웨이브폼이 시간 정보와 그에 연관된 값을 포함하므로 그래프에 신호를 디스플레이합니다.

플롯 범례에 신호 타입과 같은 이름으로 자동적으로 플롯의 이름을 가지도록 그래프를 설정할 수 있습니다. 예를 들면, 사인파를 얻거나 시뮬레이션하고 그래프에 그 신호 출력을 연결하면 신호의 플롯 이름이 플롯 범례에서 사인으로 나타납니다.

신호 필터링하기

필터 익스프레스 VI를 사용하여 신호를 필터링합니다. 그래프, 차트, 숫자형 인디케이터에서 신호 디스플레이를 할 수 있으며 하드웨어 디바이스 또는 파일에 신호를 쓸 수 있습니다.



관련 정보

[신호 디스플레이하기](#) (1174 페이지 참조)

[파일 I/O](#) (477 페이지 참조)

신호 병합하기

[신호 병합] 함수를 사용하여, 수집 또는 시뮬레이션을 통해 얻은 2개 이상의 다른 신호에서 오는 데이터를 하나의 와이어로 결합합니다. 결합된 신호를 여러 개의 플롯이 있는 그래프에 연결하거나 파일에 저장할 수 있습니다.

다음 단계를 따라 신호를 다른 VI, 함수, 그래프, 차트의 입력에 연결하기 전에 신호를 병합합니다.

1. 블록 다이어그램에 [신호 병합] 함수를 추가합니다.
2. 신호 병합 함수의 크기를 조정하여 연결하려는 신호에 대한 충분한 입력을 확보합니다.
3. 각 신호를 입력 신호 입력에 연결합니다.
4. 다른 VI, 함수, 그래프, 차트의 신호 입력에 결합된 신호 출력을 연결합니다.



노트

결합된 모든 신호를 개별적인 신호로 분리할 수 있습니다.



관련 정보

[노드 크기 조정하기](#) (849 페이지 참조)

[신호 분리하기](#) (1175 페이지 참조)

신호 분리하기

병합된 신호로부터 단일 신호를 추출하려면 [신호 분리] 함수를 사용합니다.

다음 단계를 따라 병합된 신호를 분리합니다.

1. 블록 다이어그램에 [신호 분리] 함수를 추가합니다.
2. 하나 이상의 신호에 접근하기 위해, [신호 분리] 함수의 크기를 재조정합니다. 예를 들면, 병합된 신호를 두 개의 신호로 분리하기 위해, 함수의 크기를 재조정하여 두 개의 출력으로 나타냅니다. 병합된 신호에 포함되어 있는 것보다 더 많은 출력

을 포함하는 함수의 크기를 재조정하면, 여분의 출력은 데이터를 가지고 있지 않습니다.

- 병합된 신호를 [신호 분리] 함수의 합쳐진 신호 입력에 연결합니다.



관련 정보

[신호 병합하기](#) (1175 페이지 참조)

[노드 크기 조정하기](#) (849 페이지 참조)

인스트루먼트 드라이버 변경 설명

새 인스트루먼트 드라이버 프로젝트 생성 마법사를 사용하여 인스트루먼트 드라이버를 생성한 후 이 설명을 따릅니다. LabVIEW 플러그 앤 플레이 인스트루먼트 드라이버 개발에 대한 더 자세한 정보는 National Instruments 웹 사이트를 참조하십시오.

변경 설명

- 지원되지 않는 VI 제거
- 일반적인 드라이버 VI 변경
- 인스트루먼트 특정 VI 변경
- 새 인스트루먼트 특정 VI 생성
- 예제 VI 생성
- VI 트리 VI 업데이트
- 팔레트 파일 업데이트
- Readme 파일 업데이트 또는 생성
- ni.com/idnet의 인스트루먼트 드라이버 네트워크에 **완성된 드라이버를 제출** 하십시오.

지원되지 않는 VI 제거하기

- 셀프테스트 또는 리셋과 같은 일반적인 VI를 포함하여 인스트루먼트가 지원하지 않는 모든 인스트루먼트 드라이버 VI를 프로젝트에서 제거합니다. 사용자 인스트루먼트의 개별 능력을 알고 있어야, 지원할 VI를 결정할 수 있습니다. 예를 들어 디지털 멀티미터 템플릿은 여러 포인트 측정을 위한 VI를 포함합니다. 여러 포인트 측정을 지원하지 않는 디지털 멀티미터를 위해 드라이버를 작성하는 경우, 이 VI를 드라이버 프로젝트와 드라이버 디렉토리 모두에서 제거합니다.
- 제거한 각 VI에 대해 프로젝트의 다른 인스트루먼트 드라이버 VI로부터의 해당 VI의 subVI 호출도 제거합니다.
- 파일을 제거한 후 프로젝트(.lvproj)와 프로젝트 라이브러리(.lvlib)를 저장합니다.

일반적인 드라이버 VI 변경하기

1. 템플릿에서 드라이버를 생성한 경우, 일반적인 드라이버 VI에는 프런트패널 및/또는 블록다이어그램의 노란색 텍스트 박스 안에 변경 설명이 있습니다. 일반적인 드라이버 VI는 다음과 같습니다: 초기화, 닫기, 셀프테스트, 리셋, 에러 쿼리, 개정 쿼리, 기본 인스트루먼트 설정. VI 문서를 읽고, 각 VI의 변경 설명을 따른 후, 설명을 삭제합니다.
2. 다음 추가적인 설명을 따르십시오:
 - Initialize VI—프런트패널로 전환하고 VISA 리소스 이름 컨트롤에 기본 주소 정보를 입력합니다. 디바이스를 위한 구문에 대해서는 VISA 리소스 이름 컨트롤 기본 도움말을 참조하십시오. 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리>현재값을 기본값으로 선택합니다. 이 컨트롤의 기본 도움말을 업데이트하여 인스트루먼트에서 지원되는 버스에 대한 정보만 표시하십시오. 시리얼 설정 클러스터의 시리얼 컨트롤을 수정하거나 제거하여 인스트루먼트의 기능과 일치하도록 합니다. 예를 들어, 인스트루먼트가 7 데이터 비트와 1 정지 비트만을 가지고 있는 경우, 데이터 비트와 정지 비트 컨트롤을 제거하고 블록다이어그램에서 이러한 파라미터를 상수로 설정합니다. 인스트루먼트가 2 보 전송속도만 지원하는 경우, 보 전송속도 컨트롤에서 지원되지 않는 속도를 제거합니다. 시리얼 설정 클러스터의 컨트롤을 인스트루먼트 기본으로 설정하고, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 수행>현재값을 기본값으로 선택합니다. 또한, 시리얼 설정 클러스터를 TCP/IP와 같은 시리얼이 아닌 버스 등의 다른 설정 셋팅에도 사용할 수 있습니다. 필요한 경우, 클러스터 이름과 기본 도움말을 업데이트하여 클러스터 내의 컨트롤을 반영합니다.
 - Close VI—인스트루먼트를 닫기 전에 특정한 상태로 설정해야 하는 경우, 블록다이어그램의 [VISA 닫기] 함수 앞에 필요한 코드를 추가합니다.
 - Self-Test VI—블록다이어그램에서 타임아웃 상수를 인스트루먼트가 셀프테스트를 끝낼 수 있을만큼 큰 값으로 설정합니다. 기본은 20,000 밀리초, 또는 20 초입니다.
 - Error Query VI—여러 SCPI 에러를 분석하도록 디자인되었습니다. 인스트루먼트가 에러 확인을 다른 방법으로 수행하는 경우, 이 기능을 수행하는 다른 방법에 대해 Fluke 45 Multimeter와 같은 다른 인스트루먼트 드라이버를 참조하십시오. Error Query VI는 하위 레벨 API VI를 제외한 모든 드라이버 VI에 SubVI로 포함되어 있습니다. SCPI 호환 인스트루먼트의 경우, 에러 쿼리 명령에 :SYST:ERR?를 사용합니다. 비 SCPI 호환 인스트루먼트의 경우, 에러 쿼리 응답을 분석하여 에러가 발생했는지 알아냅니다. 이 응답에 기초하여, 에러 클러스터의 불리언 상태 원소를 적당한 상태로 설정합니다. 인스트루먼트 에러에 리포트할 에러가 없는 경우 불리언은 거짓, 그 외의 경우에는 참이 됩니다.
 - Revision Query VI—[문자열로부터 스캔] 함수를 수정하여 인스트루먼트의 펌웨어 쿼리를 분석합니다.
3. 파일을 수정한 후 프로젝트(.lvproj)와 프로젝트 라이브러리(.lvlib)를 저장합니다.

인스트루먼트 특정 VI 변경하기

다음 단계를 따라 드라이버의 각 남은 VI를 검색합니다.

1. VI를 열고 VI로 정의된 작업을 수행하는데 필요한 컨트롤과 인디케이터를 추가, 수정, 제거하여 프런트패널을 편집합니다.
2. 인스트루먼트의 기본값에 맞도록 컨트롤의 기본값을 업데이트합니다. 각 컨트롤에 값을 설정한 후 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 > 현재값을 기본값으로 선택합니다.
3. 컨트롤과 인디케이터 라벨 및 설명을 업데이트하여 모든 변경 사항을 반영합니다.
4. 필요한 경우 VI 아이콘을 편집합니다.
5. 컨트롤 및 인디케이터를 제거하거나 추가한 경우, 커넥터 팬을 편집하여 맞춥니다.
6. 명령 문자열과 분석된 응답을 업데이트하여 블록 다이어그램을 편집하고 인스트루먼트 기능에 맞춥니다.
7. 필요한 경우 VI 문서를 편집합니다. Initialize VI에 대한 문서의 예는 다음과 같습니다: 이 인스트루먼트 드라이버는 시리얼 통신을 지원합니다. 시리얼 통신을 사용할 때 문제가 발생하는 경우, 19,200 이하의 보 전송속도와 하드웨어 핸드셰이킹 사용을 시도해 보십시오.
8. 템플릿에서 드라이버를 생성한 경우, 각 VI에는 프런트패널 및/또는 블록 다이어그램의 노란색 텍스트 박스 안에 변경 설명이 있습니다. VI 문서를 읽고, VI의 변경 설명을 따른 후, 설명을 삭제합니다.
9. VI를 저장하고 테스트합니다.
10. 파일을 수정한 후 프로젝트(.lvproj)와 프로젝트 라이브러리(.lvlib)를 저장합니다.

새 인스트루먼트 특정 VI 생성하기

1. 인스트루먼트 드라이버 VI 마법사를 사용하여 인스트루먼트가 지원하는 추가적인 VI를 생성합니다. 디지털 멀티미터 템플릿과 같은 클래스 템플릿에서 드라이버를 생성하는 경우, 인스트루먼트는 클래스 템플릿에서 다루지 않는 추가적인 기능을 지원할 수도 있습니다. 기존 VI와 구조를 참조하여 각 지원되는 기능에 대한 추가적인 VI를 생성해야 합니다. 예를 들어, 오실로스코프는 일반적으로 많은 타입의 트리거를 가지고 있으나 오실로스코프 템플릿은 에지 트리거링에 대한 지원만 포함하고 있습니다. 인스트루먼트가 추가적인 타입의 트리거를 지원하는 경우, 추가적인 인스트루먼트 드라이버 VI로 이 기능을 추가하십시오.
2. 다형성 VI를 사용하여 연관된 인스트루먼트 드라이버 VI를 그룹화합니다. 예를 들어, 여러 트리거 설정 VI는 하나의 다형성 VI로 제시될 수 있습니다.
3. 파일을 수정한 후 프로젝트(.lvproj)와 프로젝트 라이브러리(.lvlib)를 저장합니다.

다음은 클래스 템플릿에서 생성한 드라이버에 특정한 변경 사항입니다.

- 디지털 멀티미터—설정 및 데이터 팔레트의 하위 레벨 VI는 DMM의 더 많은 컨트롤을 허용합니다. 이 VI는 읽기(단일 포인트)와 읽기(여러 포인트) 템플릿 VI와 함께 사용되도록 디자인되지 않았습니다. 이런 VI는 이미 이 기능의 일부를 가지고 있기 때문입니다. 이 같은 하위 레벨 VI 사용에 대한 예제는 Read Triggered Multiple VI를 참조하십시오.

- 오실로스코프—프라이빗 Default Instrument Setup.vi는 오실로스코프를 단일 수집 모드로 설정해야 합니다. 이 명령은 :ACQ:STOPA SEQ;과 비슷할 수도 있습니다.
- 함수 생성기—인스트루먼트가 임의의 웨이브폼을 지원하는 경우, 별도의 VI를 생성하여 임의의 웨이브폼 지원을 구현해야 합니다. 임의의 웨이브폼 지원을 Configure Standard Waveform.vi에 추가하지 마십시오.

인스트루먼트가 자동으로 하나의 모드를 비활성화하고 다른 모드를 활성화하는 경우, 내부적으로 변조 상태와 스위치 VI를 활성화할 수 있습니다. 이렇게 하려면, 활성화 <x> 모드 컨트롤을 제거하고 블록 다이어그램에서 거짓 케이스를 삭제합니다.

예제 VI 생성하기

1. 인스트루먼트 드라이버 사용법을 설명하는 둘 또는 그 이상의 예제 VI를 생성합니다. 예제는 사용자가 인스트루먼트 작업을 수행하기 위해 필요한 권장 호출 시퀀스를 이해하는데 도움이 되어야 합니다. 인스트루먼트의 각 일반적 용도에 맞는 예제를 포함시켜 다양한 사용자들에게 도움을 줍니다.
2. 프로젝트 라이브러리가 아닌 인스트루먼트 드라이버 프로젝트에 예제 VI를 포함합니다.
3. 도구 >> NI 예제 탐색기를 위해 예제 VI 준비를 선택하여 예제 데이터 파일(.bin3)을 인스트루먼트 드라이버 프로젝트의 예제 디렉토리에 추가합니다.



노트

LabVIEW는 다음 접두어를 예약하므로 .bin3 파일의 이름에 다음 접두어를 사용할 수 없습니다: ms, test, lv, daq, cvi, tree, fr, de, ja, vi, im, dsc, rt, _, ko, zh, it, es.

VI 트리 VI 업데이트하기

1. VI 트리 VI를 열고 블록 다이어그램을 업데이트하여 인스트루먼트 드라이버의 변경 사항을 반영합니다. VI 트리는 어플리케이션의 subVI가 아닙니다. 이는 드라이버 VI의 계층구조의 간략한 개요를 볼 수 있도록 만들어졌습니다.
2. 셀프테스트 또는 리셋과 같은 일반적인 VI를 포함하여 인스트루먼트가 지원하지 않는 모든 VI를 제거합니다.
3. 드라이버에 추가한 모든 퍼블릭 VI를 추가합니다.
4. VI를 저장하고 닫습니다.

팔레트 파일 업데이트하기

1. 모든 VI가 완성된 후 함수 팔레트 파일(.mnu)을 업데이트하여 변경 사항을 인스트루먼트 드라이버에 반영합니다. 모든 퍼블릭 인스트루먼트 드라이버 VI는 팔레트 파일에서 접근할 수 있어야 합니다. 팔레트 파일에 예제 VI를 포함시키지 마십시오.
2. 드라이버에서 제거한 모든 VI의 링크를 제거합니다.

3. 필요한 경우 서브팔레트에 대한 팔레트 파일을 추가하여 새 VI에 대한 팔레트의 유용성을 높입니다. 팔레트 파일은 VI 트리 및 프로젝트 폴더 계층구조의 레이아웃과 일치해야 합니다.

Readme 파일 업데이트 또는 생성하기

1. Readme.html 파일을 html 편집기에서 엽니다. Readme 파일은 드라이버 개요, 사용을 위한 필요사항, 지원되는 기능, 개정 히스토리, 드라이버를 사용하기 위한 기타 정보를 포함하는 html 파일입니다. 템플릿에서 드라이버를 생성한 경우, 드라이버에는 수정해야 할 필드가 빨간 텍스트로 되어있는 Readme.html 파일이 포함됩니다. 드라이버를 직접 생성한 경우, Readme.html도 직접 만들어야 합니다.
2. Readme 파일의 알맞은 필드에 인스트루먼트 및 인스트루먼트 드라이버에 특정한 정보를 업데이트합니다. 템플릿에서 드라이버를 생성한 경우, 파일의 빨간 텍스트의 모든 인스턴스를 업데이트 및 대체합니다.
3. Readme 파일을 저장합니다.
4. 드라이버를 수정할 때마다 Readme 파일에 개정 히스토리를 업데이트합니다.

인스트루먼트 드라이버 네트워크에 인스트루먼트 드라이버 제출하기

자사의 인스트루먼트 드라이버에 보다 쉽게 접근할 수 있도록 하려는 인스트루먼트 제조업체나 드라이버 개발 노력을 다른 사용자와 공유하려는 개인 사용자의 경우, 드라이버를 인스트루먼트 드라이버 네트워크 ni.com/idnet에 제출하는 것을 고려하십시오.

인스트루먼트 드라이버 프로젝트 생성하기

인스트루먼트 드라이버를 만들기 전에, 현재 사용하고 있는 인스트루먼트에 드라이버가 없는지 확인합니다. NI 인스트루먼트 드라이버 검색기를 사용하여 인스트루먼트의 LabVIEW 플러그 앤 플레이 드라이버를 찾고, 다운로드하고, 설치합니다. 또한 드라이버에 대한 정보는 제조업체의 웹 사이트, National Instruments 웹 사이트의 인스트루먼트 드라이버 네트워크 및 NI 드라이버 및 업데이트에서 확인할 수 있습니다. 기존 LabVIEW 인스트루먼트 드라이버를 **검색하고 설치**할 수 있습니다.

새 인스트루먼트 드라이버 프로젝트 생성 마법사를 사용하여 인스트루먼트 드라이버 가이드라인을 따르는 LabVIEW 플러그 앤 플레이 인스트루먼트 드라이버 프로젝트를 생성합니다. 생성된 인스트루먼트 드라이버 프로젝트는 [인스트루먼트 드라이버] VI와 함수 팔레트 파일 .mnu를 포함합니다.

다음 단계를 따라 새 인스트루먼트 드라이버 프로젝트 생성 마법사를 사용하여 인스트루먼트 드라이버 프로젝트를 생성합니다.

1. 도구 >> 인스트루먼트이션 >> 인스트루먼트 드라이버 프로젝트 생성을 선택하여 인스트루먼트 드라이버 프로젝트 마법사를 디스플레이합니다.

2. 템플릿 드라이버 혹은 기존의 프로젝트 스타일 드라이버로부터 인스트루먼트 드라이버 프로젝트를 생성하느냐에 따라 프로젝트 타입 풀다운 메뉴에서 템플릿으로부터 새 드라이버 만들기 또는 새 드라이버, 기존 드라이버 복사를 선택합니다.



노트

만약 새로운 인스트루먼트 드라이버 프로젝트를 기존의 인스트루먼트 드라이버로부터 생성하고자 한다면, 새 인스트루먼트 드라이버 프로젝트 생성 마법사를 실행하기 전에 기존의 드라이버를 `labview\instr.lib` 디렉토리에 설치해야 합니다. NI 인스트루먼트 드라이버 탐색기를 사용하여 기존의 인스트루먼트 드라이버 프로젝트를 찾고 설치할 수 있습니다.

1. 프로젝트 타입 메뉴에서 새 드라이버, 기존 드라이버 복사를 선택한 경우, 소스 드라이버 풀다운 메뉴에서 사용 가능한 인스트루먼트 드라이버를 선택하고 다음 버튼을 클릭합니다. 프로젝트 타입 메뉴에서 템플릿으로부터 새 드라이버 만들기를 선택한 경우, 소스 드라이버 풀다운 메뉴에서 템플릿 타입을 선택하고 다음 버튼을 클릭합니다. 다음 테이블은 인스트루먼트 드라이버 프로젝트 마법사가 사용하는 템플릿 리스트입니다. 사용하는 인스트루먼트의 기능에 가장 적합한 템플릿을 선택하십시오. 일반 템플릿을 제외한 모든 템플릿은 특정 타입의 인스트루먼트나 디바이스를 위한 코어 드라이버 아키텍처와 함께 VI 및 팔레트 파일 .mnu를 생성합니다. 일반 템플릿은 인스트루먼트 타입에 관계없이 대부분의 인스트루먼트 드라이버 사이에서 가장 일반적인 VI와 팔레트 파일만을 포함하는 기본 드라이버를 생성합니다.

템플릿

설명

DC 파워 출력과 과전압 및 과전류 설정과 같은 기본 작업을 통제합니다. 이 템플릿은 출력 전압 및 전류 모니터링과 같은 기능도 포함합니다.

측정 함수, 범위, 분해능 설정과 같은 기본 작업을 통제합니다. 이 템플릿은 트리거 카운트 및 샘플 카운트 설정과 멀티포인트 측정 같은 고급 기능도 포함합니다.

표준 웨이브폼 출력과 같은 기본 작업을 통제합니다. 이 템플릿은 주파수와 진폭 변조 설정 기능도 포함합니다.

인스트루먼트로부터 웨이브폼 데이터 에지 트리거링하기 및 이전하기를 통한 웨이브폼 수집과 같은 기본 작업을 통제합니다.

클래스에 특정한 템플릿이 없는 메시지 기반 인스트루먼트를 위해 사용합니다. 일반적인 메시지 기반 인스트루먼트는 다음을 포함합니다: GPIB, USBTMC, VXI-11, 시리얼.

클래스에 특정한 템플릿이 없는 레지스터 기반 인스트루먼트를 위해 사용합니다. 일반적인 레지스터 기반 인스트루먼트는 다음을 포함합니다: VXI 및 PXI.

주파수 범위, 스위프 프로퍼티 설정과 같은 기본 작업을 통제합니다. 이 템플릿은 마커 설정 및 쿼리하기와 같은 고급 기능도 포함합니다.

측정 타입과 트리거 레벨과 같은 기본 작업을 통제합니다. 이 템플릿에는 시작과 정지 활성화 조건을 설정하는 기능도 포함되어 있습니다.

측정 함수, 범위, 오프셋 설정과 같은 기본 작업을 통제합니다. 이 템플릿에는 평균화와 보정 주파수를 설정하는 기능도 포함되어 있습니다.

2. 드라이버 식별자 텍스트 박스에 Agilent 34401, Tektronix 2000 시리즈와 같은 새 인스트루먼트 드라이버 프로젝트의 드라이버 식별자를 입력합니다. 마법사는 사용자가 입력한 드라이버 식별자를 인스트루먼트 드라이버 프로젝트 .lvproj 및 프로젝트 라이브러리 .lvlib의 기본 이름과 경로로 사용합니다. 파일을 디스크나 LabVIEW 환경 내에서 볼 때 인스트루먼트 제조업체 이름과 모델을 이름에 포함하여 사용자가 신속하게 드라이버를 식별할 수 있도록 도와줍니다. 각 인스트루먼트 드라이버 VI는 프로젝트 라이브러리 이름을 VI 이름의 일부로 포함하고 있습니다. 이름을 사용해서 드라이버의 폴더 구조를 생성하고 드라이버를 다양한 플랫폼에서 사용할 수 있기 때문에, 드라이버 이름에는 #, \$, &와 같은 특수 문자를 사용할 수 없습니다. 이름은 다른 설치된 드라이버 이름과 충돌해서는 안 되며 새 인스트루먼트 드라이버 프로젝트 생성 마법사가 이름 충돌을 확인합니다.
3. 드라이버 설명 텍스트 박스에 인스트루먼트 드라이버 프로젝트에 대한 설명을 입력한 후, 다음 버튼을 클릭합니다. 이 설명은 커서를 프로젝트 라이브러리 위로 이동할 때 기본 도움말 윈도우에 나타납니다.
4. 위에 있는 편집 버튼을 클릭하고 VI 아이콘 배너를 생성한 후, 아이콘 편집기를 열고 VI 아이콘 픽셀의 맨 위쪽의 8개 행을 편집합니다. 아이콘 편집기를 종료하면, LabVIEW는 맨 위쪽 8개 행 밑에서 이루어진 모든 편집 내용을 지웁니다. 마법사는 이 배너를 인스트루먼트 드라이버 프로젝트의 각 드라이버 VI에 대한 아이콘 배너로 사용합니다. 드라이버의 축약된 이름을 배너에 포함시켜 사용자가 쉽게 드라이버에 속한 VI를 식별할 수 있도록 돕습니다.
5. 하단의 편집 버튼을 클릭하고 VI 아이콘을 편집하여 최상위 팔레트 아이콘을 생성한 후, 다음 버튼을 클릭합니다. 마법사는 최상위 팔레트 아이콘을 인스트루먼트 드라이버 팔레트에서 보이는 아이콘으로 사용합니다. 드라이버와 인스트루먼트 타입을 잘 나타내도록 아이콘을 디자인하여 사용자가 팔레트에서 쉽게 드라이버를 식별할 수 있도록 합니다.
6. 새 프로젝트 위치 상자에서 설치된 드라이버의 경로를 확인한 후 마침 버튼을 클릭하여, 인스트루먼트 드라이버를 생성하고 새로 생성된 드라이버와 함께 새 **프로젝트 탐색기** 윈도우를 엽니다. 마법사는 인스트루먼트 드라이버 프로젝트가 필요로 하는 모든 파일을 생성하고, 에러나 경고 사항이 있으면 경고 리포트를 생성하며, **인스트루먼트 드라이버 변경 설명서**를 엽니다. 기존 드라이버와 다르게 새 드라이버를 변경하면 경고가 나타날 수 있습니다. 한 예로, 기존의 드라이버가 드라이버의 디렉토리 내에 저장하지 않은 VI를 포함한 경우, 마법사는 VI 복사본을 생성하지만 이 VI는 새 드라이버의 계층 구조에 위치하게 됩니다.
7. 인스트루먼트 드라이버 변경 설명서를 따라서 드라이버를 완성하십시오. 인스트루먼트 드라이버 VI 마법사를 사용하여 프로젝트를 위한 **인스트루먼트 드라이버 VI를 생성**할 수 있습니다.

인스트루먼트 드라이버 프로젝트를 완성한 후, 다른 사용자들이 이용할 수 있도록 ni.com/idnet에 있는 인스트루먼트 드라이버 네트워크에 드라이버를 올리는 것도 고려하십시오.

인스트루먼트 드라이버 VI 생성하기

인스트루먼트 드라이버 VI 마법사를 사용하여 인스트루먼트 드라이버 프로젝트를 위한 [인스트루먼트 드라이버] VI를 생성할 수 있습니다.

다음 단계를 따라 인스트루먼트 드라이버 VI 마법사를 사용하여 인스트루먼트 드라이버 VI를 생성합니다.

1. 인스트루먼트 드라이버 프로젝트를 생성하거나 엽니다. 새 인스트루먼트 드라이버 프로젝트 생성 마법사를 사용하여 **인스트루먼트 드라이버 프로젝트를 생성**할 수 있습니다. NI 인스트루먼트 드라이버 탐색기를 사용하여 기존의 인스트루먼트 드라이버 프로젝트를 찾고 설치할 수 있습니다.
2. **프로젝트 탐색기** 윈도우의 LabVIEW 인스트루먼트 드라이버 프로젝트 라이브러리 .lvlib에서 마우스 오른쪽 버튼을 클릭한 후, **로 가기 메뉴에서 새로 만들기 >> 인스트루먼트 드라이버 VI**를 선택하여 인스트루먼트 드라이버 VI 마법사를 디스플레이합니다.
3. 인스트루먼트 드라이버 VI 마법사의 시작 페이지에서 다음 옵션 중 하나를 선택하고 다음 버튼을 클릭합니다.
 - 인스트루먼트 드라이버 템플릿 VI로부터 생성—템플릿 VI로부터 인스트루먼트 드라이버 VI를 생성합니다.
 - 기존 VI로부터 복사-기존의 인스트루먼트 VI와 subVI를 복사하고 설정합니다.
 - 사용자 VI 설정—인스트루먼트 명령을 설정하고, 응답을 분석하는 방법을 지정하며, 연결된 인스트루먼트와 상호작용하고, 대응하는 인스트루먼트 I/O와 문자열 포맷/분석 코드를 새 인스트루먼트 드라이버 VI에 생성합니다.
 - 새 VI 추가—VISA 리소스 이름 컨트롤 및 인디케이터와 에러 클러스터 컨트롤 및 인디케이터만을 포함한 인스트루먼트 드라이버 VI를 생성합니다.
4. 인스트루먼트 드라이버 VI 마법사의 두번째 페이지에서:
 - 시작 페이지에서 인스트루먼트 드라이버 템플릿 VI로부터 생성을 선택한 경우, 템플릿으로부터 복제 페이지에 있는 사용 가능한 인스트루먼트 드라이버 VI 템플릿에서 VI를 선택한 후 다음 버튼을 클릭합니다.
 - 시작 페이지에서 기존 VI로부터 복사를 선택한 경우, VI 탐색 페이지의 탐색 버튼을 클릭하여 프로젝트에 추가하고자 하는 VI를 검색하고 선택한 후 다음 버튼을 클릭합니다.
 - 시작 페이지에서 사용자 VI 설정을 선택한 경우, VI 타입 선택 페이지에서 설정 VI 또는 측정 VI를 선택한 후 다음 버튼을 클릭합니다. 명령을 인스트루먼트에 보내는 VI를 생성하려면 설정 VI를 선택합니다. 명령을 인스트루먼트에 보낸 후 인스트루먼트의 응답을 읽는 VI를 생성하려면 측정 VI를 선택합니다.
 - 시작 페이지에서 새 VI 추가를 선택한 경우, 6 단계로 진행합니다.
5. 시작 페이지에서 사용자 VI 설정을 선택한 경우, 컨트롤 설정 페이지를 설정한 후 다음 버튼을 클릭합니다. 그렇지 않다면, 다음 단계로 계속 진행됩니다.
 1. 퍼센트 내에 파라미터로 사용할 텍스트 텍스트 박스에 인스트루먼트에 보낼 명령 문자열을 입력합니다. 파라미터 이름 주변에 % 문자를 삽입함으로써 명령 문자열 내에서 파라미터를 지정합니다. LabVIEW는 파라미터 설정 리스트에 생성한 파라미터 이름을 디스플레이합니다. 사용자가 생성한 각각의 파라미터에 대해서, LabVIEW는 이에 대응하는 컨트롤을 완성된 VI의 프런트패널과 커넥터 팬에 추가합니다.
 2. 데이터 타입 풀다운 메뉴에서 각각의 파라미터에 적합한 데이터 타입을 선택하십시오.
 3. 데이터 타입 풀다운 메뉴 옆에 있는 숫자형 상자에 각각의 파라미터의 기본 값을 설정합니다.
 4. 명령 미리보기 텍스트 박스에 설정된 명령 문자열을 확인합니다.

5. VI 타입 선택 페이지에서 설정 VI를 선택한 경우, 명령 테스트 섹션의 풀다운 메뉴에서 인스트루먼트를 선택합니다. VI 타입 선택 페이지에서, 측정 VI를 선택한 경우, 다음 버튼을 클릭한 후 명령 테스트 섹션의 풀다운 메뉴에서 인스트루먼트를 선택합니다.
6. 테스트 실행 버튼을 클릭하여 설정한 명령을 테스트합니다. LabVIEW는 명령 미리보기 텍스트 박스의 명령을 인스트루먼트에 보내고, 사용된 바이트 박스에 인스트루먼트에 쓰여진 바이트의 개수를 나타냅니다. 명령이 성공하면, 밝은 녹색 불이 들어옵니다. VISA가 에러를 반환하면, 붉은색 불이 들어옵니다. 에러 설명 버튼을 클릭하여 에러를 디스플레이합니다. VI 타입 선택 페이지에서 측정 VI를 선택한 경우, LabVIEW는 인스트루먼트의 응답을 읽고 이를 응답 텍스트 박스에 디스플레이합니다. 응답 데이터 타입 설정 섹션에서 응답 데이터 타입을 선택할 수 있습니다.
6. VI 프로퍼티 페이지에서 VI 이름, 경로, 아이콘, 설명을 편집한 후 다음 버튼을 클릭합니다.
7. 요약 페이지에서, 새로 만든 VI 정보를 확인한 후 마침 버튼을 클릭하여, 인스트루먼트 드라이버 프로젝트에 인스트루먼트 드라이버 VI와 모든 SubVI를 추가하고 VI를 엽니다. 템플릿 VI 또는 기존 VI에서 VI를 생성한 경우, LabVIEW는 프로젝트 라이브러리에 있는 기존 VI와 동일한 이름을 갖고 있으나 다른 커넥터 팬을 지닌 subVI의 이름을 변경합니다. LabVIEW 프로젝트 라이브러리에 있는 기존 VI와 동일한 이름 및 동일한 커넥터 팬을 갖는 subVI를 복사하지 않습니다. 새 VI는 LabVIEW가 추가하지 않은 모든 subVI가 있을 위치에 프로젝트 라이브러리에 있는 기존 VI를 사용합니다.

2D 그림 컨트롤에서 보이지 않는 픽셀 디스플레이하기

2D 그림 컨트롤로 나타내기에 그림이 너무 큰 경우, 그림의 일부가 잘리게 되어 사용자는 컨트롤의 디스플레이 영역에 포함된 픽셀만 볼 수 있습니다. 다음 방법을 따라 디스플레이 범위 안에 맞지 않는 픽셀을 디스플레이합니다:

- 위치 도구를 사용하여 컨트롤의 크기를 조절하고 전체 그림을 볼 수 있도록 VI를 다시 실행합니다. 또한, 프로그램적으로 그리기 영역 크기 프로퍼티를 사용하여 컨트롤의 크기를 조절할 수 있습니다.
- 컨트롤에서 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 스크롤 막대를 선택하여 컨트롤 안에 스크롤 막대를 나타냅니다. 또한, 스크롤 막대 보이기 프로퍼티를 사용하여 프로그램적으로 컨트롤에 스크롤 막대를 디스플레이할 수 있습니다.
- 원점 프로퍼티를 사용하여 컨트롤에 그림의 다른 부분이 보이도록 컨트롤 원점의 수직과 수평 좌표를 변경합니다.
- 줌 전환율 프로퍼티를 사용하여 컨트롤의 이미지를 크기 조절합니다.

2D 그림 컨트롤 리셋하기

2D 그림 컨트롤은 기본적으로 이미지를 그리기 전에 기존 디스플레이를 삭제합니다. 2D 그림 컨트롤이 이전 이미지 위에 새 이미지를 그리도록 하려면, 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 먼저 삭제를 선택하여 이 메뉴 아이템에서 확인 표시를 제거합니다.

또한, 먼저 삭제 프로퍼티를 사용하여 2D 그림 컨트롤이 프로그램적으로 이전 이미지를 삭제할지 여부를 지정할 수 있습니다.

사운드 입력과 사운드 출력 디바이스 사용하기

컴퓨터의 사운드 파일에 접근하는 대신 사운드 입력과 사운드 출력 디바이스를 사용하는 경우 다음 단계를 따릅니다.

1. [사운드 입력 설정] VI와 [사운드 출력 설정] VI를 사용하여 입력 또는 출력 작업을 위한 사운드 디바이스를 설정합니다. 사운드 입력 또는 사운드 출력을 위해 사운드 디바이스를 설정하는 경우, LabVIEW는 태스크 ID를 생성합니다. 태스크 ID는 지정된 디바이스상의 설정과 연계된 식별 번호를 반환합니다.
2. 디바이스에서 나오고 들어오는 사운드를 조작하려면, 태스크 ID를 다양한 사운드 VI에 통과시킵니다 사운드 입력 VI 또는 사운드 출력 VI를 사용하여 디바이스로부터 받거나 디바이스로 보내는 데이터를 조작합니다.
3. [사운드 입력 지우기] VI 또는 [사운드 출력 지우기] VI를 사용하여 디바이스를 닫습니다.

3D 그림 컨트롤로 3D 장면 생성하기

다음 단계에 따라 3D 그림 컨트롤 VI와 장면 객체 메소드를 사용하여 3차원 장면을 생성합니다.

1. 3D 그림 컨트롤을 프런트패널 윈도우에 추가합니다.
2. [객체 생성] VI를 블록다이어그램에서 3D 그림 컨트롤의 왼쪽에 추가합니다.
3. [객체 생성] VI의 장면:새 객체 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>장면 객체 클래스의 메소드>>그리기 가능 설정을 선택하여 그리기 가능 설정 메소드를 생성합니다.
4. 그리기 가능 설정 메소드를 블록다이어그램에서 [객체 생성] VI의 오른쪽에 추가합니다.
5. [객체 생성] VI의 장면:새 객체 출력을 그리기 가능 설정 메소드의 참조 입력에 연결합니다.
6. 기하 팔레트에서 VI를 선택하고 블록다이어그램에서 그리기 가능 설정 메소드의 왼쪽에 놓습니다.

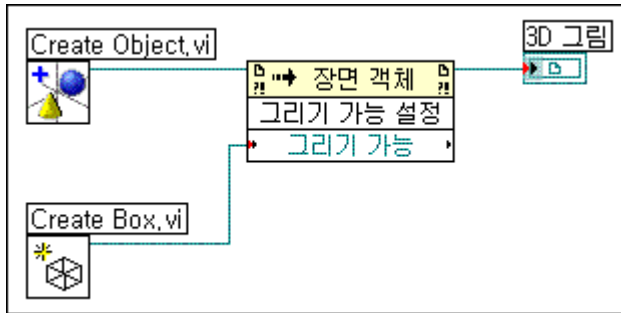


노트

[메시 생성] VI를 사용하는 경우, 메시 파라미터 설정 메소드를 사용하여 메시지를 설정해야 합니다.

1. 단계 5에서 추가한 VI의 참조 출력을 그리기 가능 설정 메소드의 그리기 가능 입력에 연결합니다.

2. 그리기 가능 설정 메소드의 참조 출력을 3D 그림 컨트롤 터미널에 연결합니다. 다음의 블록 다이어그램은 간단한 3D 장면을 보여줍니다.



3. 블록 다이어그램의 에러 입력과 에러 출력 터미널을 연결합니다.
4. 이 VI를 실행하여 3D 장면을 생성합니다.
5. (옵션) 카메라를 설정하여 다른 각도에서 3D를 보십시오.

기본 3D 장면을 생성한 후에는 다른 3D 그림 컨트롤 VI를 사용하여 3D 장면을 강화할 수 있습니다. 3D 장면에서 다른 3D 객체를 추가하고 객체 관계를 생성하고, 텍스처를 3D 객체에 적용하고, 빛 소스를 설정할 수 있습니다.

또한 3D 장면을 적용하고 디스플레이하도록 장면 윈도우를 설정할 수 있습니다.

3D 장면의 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

 예제 열기  관련 예제 찾기



관련 정보

[메시를 사용하여 3D 객체 그리기](#) (471 페이지 참조)

[VI에서 에러 I/O 설정하기](#) (1388 페이지 참조)

[3D 장면에서 카메라 설정하기](#) (1187 페이지 참조)

[3D 그림 컨트롤 VI](#) (164 페이지 참조)

[3D 객체간 관계 생성하기](#) (1192 페이지 참조)

[3D 객체의 표면 모양 바꾸기](#) (1189 페이지 참조)

[3D 장면에서 빛 소스 설정하기](#) (1188 페이지 참조)

[3D 장면 윈도우 설정하기](#) (1195 페이지 참조)

3D 장면에서 카메라 설정하기

3D 장면을 생성하면, 사용자 컨트롤 카메라가 3D 장면과 상호작용하는 방식을 설정할 수 있습니다.



노트

다음 과정은 3D 그림 컨트롤에서 제공된 3D 장면에서 카메라를 설정하는 방법을 설명합니다. 3D 장면을 볼 수 있도록 **장면 윈도우를 설정**하는 경우, 장면 윈도우 프로퍼티를 사용하여 장면 윈도우에서 카메라를 프로그램적으로 설정해야 합니다.

카메라와 3D 장면 윈도우를 설정하는 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.



예제 열기  관련 예제 찾기

3D 그림 컨트롤에서 제공된 3D 장면에서 카메라를 설정하려면 다음을 수행하십시오.

1. **3D 장면을 생성**하거나 3D 장면을 포함하는 VI를 엽니다.
2. 프런트패널에서, 3D 그림 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴의 카메라 컨트롤러 위로 커서를 움직여 사용 가능한 옵션을 디스플레이합니다.
3. 다음의 카메라 컨트롤러 옵션에서 선택하십시오:
 - 없음—3D 장면에서 카메라를 비활성화합니다.
 - 구형좌표—카메라가 고정된 포인트 주위를 회전하도록 설정합니다. 고정된 포인트는 3D 그림 컨트롤의 중앙에 있는 3D 장면에서의 포인트입니다.
 - 플라잉—카메라가 3D 장면에서 자유롭게 움직이도록 설정합니다. 이 옵션을 선택한 경우, 3D 객체에서 카메라를 너무 멀리 움직여 객체를 볼 수 없게 되지 않도록 주의하십시오.
 - 지향—카메라가 z 축에 상대적인 고정된 포인트 주위를 회전합니다. 이 옵션을 선택하는 경우, LabVIEW는 카메라가 왼쪽이나 오른쪽으로 기울어지는 것을 방지합니다.

3D 장면에서 카메라를 움직이면, LabVIEW가 장면을 다시 그려 3D 그림 컨트롤에서 보이는 바를 업데이트합니다.

4. (옵션) 자동으로 다시 그리기를 비활성화하려면, 3D 그림 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴의 카메라 컨트롤러»자동으로 다시 그리기를 선택하여 옵션 옆의 확인 표시를 제거합니다. 3D 장면에서 여러 개를 변경하고 변경을 모두 마칠 때까지 변경된 3D 장면을 보지 않으려면, 자동으로 다시 그리기 비활성화가 유용합니다.



노트 자동으로 다시 그리기를 비활성화하면, For 루프나 While 루프 등의 루프를 사용해서 3D 장면을 업데이트할 수 있습니다. 3D 장면을 구성하는 VI, 프로퍼티와 메소드를 루프에 넣습니다. LabVIEW는 루프가 반복할 때마다 3D 장면을 다시 그려, 현재의 카메라 각도가 반영되도록 보기를 업데이트합니다.

1. 이 VI를 실행하여 3D 장면을 생성합니다.
2. 수행 도구를 사용하여 3D 그림 컨트롤에서 클릭하고 끌며 3D 장면 주위에서 카메라를 움직일 수 있습니다. 수행 도구를 사용해서 클릭하고 끄는 동안 <Shift> 키를 눌러 3D 장면을 줌 인하고 줌 아웃할 수 있습니다. 수행 도구를 사용해서 클릭하고 끄는 동안 <Ctrl> 키를 눌러 화면에 있는 평면에서의 카메라 위치를 변경할 수 있습니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

또한 카메라 컨트롤러:타입과 카메라 컨트롤러:자동으로 다시 그리기 프로퍼티를 사용하여 3D 그림 컨트롤에서 제공된 3D 장면에서 카메라 컨트롤러 옵션을 프로그램식으로 설정할 수 있습니다.

3D 장면에서 빛 소스 설정하기

빛 소스를 설정하여 보이는 3D 장면에 햇빛이나 스포트라이트와 같은 시각 효과를 얻을 수 있습니다. 3D 장면에서 세 가지 타입의 빛을 생성할 수 있습니다: 방향 빛, 위치 빛, 스포트라이트. 다음의 과정에서는 3D 장면에서 방향 빛을 설정하는 방법을 설명하지만, 비슷한 단계를 수행하여 위치 빛과 스포트라이트를 설정할 수 있습니다.

3D 방향에서 방향 빛을 설정하려면 다음을 수행하십시오.

1. **3D 장면을 생성**하거나 3D 장면을 포함하는 VI를 엽니다.
2. 3D 객체를 3D 그림 컨트롤 터미널에 연결하는 와이어를 제거합니다.
3. 그리기 가능 설정 메소드의 참조 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>장면 객체 클래스의 메소드>>빛>>빛 추가를 선택하여 빛: 빛 추가 메소드를 생성합니다.
4. 빛:빛 추가 메소드를 그리기 가능 설정 메소드의 오른쪽에 추가하고, 그리기 가능 설정 메소드의 참조 출력을 빛:빛 추가 메소드의 참조 입력에 연결합니다.
5. [새 빛] VI를 블록다이어그램에 추가합니다.
6. [새 빛] VI의 빛 번호 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택하고 0에서 7사이의 값을 입력합니다. 이는 빛을 식별하는 정수를 인덱스하거나 설정합니다. 8개 이상의 빛을 인덱스할 수 없습니다.



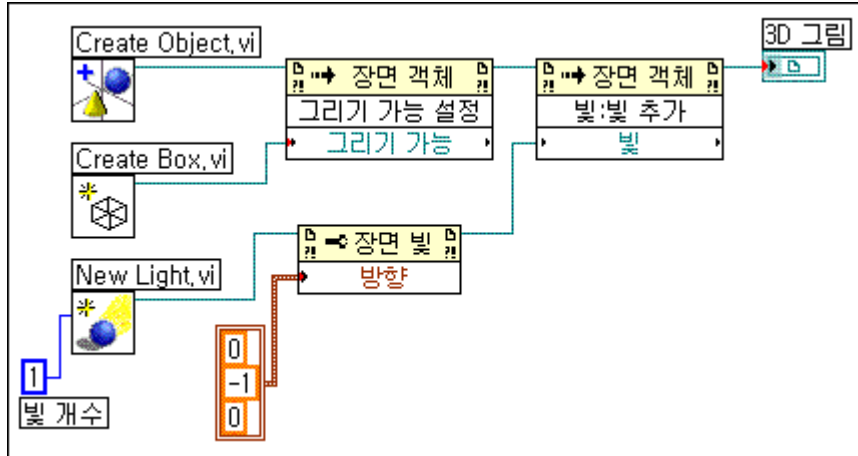
노트

LabVIEW는 인덱스가 정의되어 있지 않으면 빛 소스를 생성할 수 없습니다. 기본적으로, LabVIEW는 인덱스 0의 빛을 방향 빛으로 정의합니다. 빛을 인덱스 0으로 설정하면, 설정한 빛은 기본 빛을 덮어씁니다.

1. [새 빛] VI의 새 빛 참조 번호 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>장면 빛 클래스의 프로퍼티>>방향을 선택하여 방향 프로퍼티를 생성합니다.
2. 방향 프로퍼티를 [새 빛] VI의 오른쪽에 추가하고, 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 쓰기로 변경을 선택합니다.
3. [새 빛] VI의 새 빛 참조 번호를 방향 프로퍼티의 참조 입력에 연결합니다.
4. 방향 프로퍼티의 방향 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택하고 상수 값을 빛의 방향을 정의하는데 사용할 x, y, z 좌표로 변경합니다.

5. 방향 프로퍼티의 참조 출력을 빛:빛 추가 메소드의 빛 입력에 연결합니다.
6. 빛:빛 추가 메소드의 참조 출력을 3D 그림 컨트롤 터미널에 연결합니다.

다음의 블록다이어그램은 설정한 방향 빛이 있는 3D 장면을 보여줍니다. 이 3D 장면의 방향 좌표가 3D 객체 아래에 빛 소스를 추가합니다.



1. 블록다이어그램의 **에러 입력과 에러 출력 터미널을 연결**합니다.
2. 이 VI를 실행하여 3D 장면을 생성합니다.

3D 장면의 빛 소스 설정에 대한 예제를 보려면 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 Using Meshes VI를 참조하십시오.

 예제 열기  관련 예제 찾기

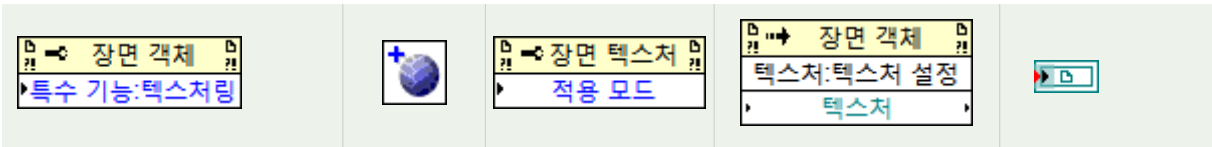
3D 객체의 표면 모양 바꾸기

새 3D 객체는 아무 무늬가 없는 단색의 표면을 가집니다. 모양을 바꾸려면 3D 객체에 **텍스처**를 적용해야 합니다. 텍스처는 3D 객체 주변을 감싸는 2D 이미지입니다.

사용 객체

다음 블록다이어그램을 사용하여 3D 객체의 표면 모양을 바꿉니다:

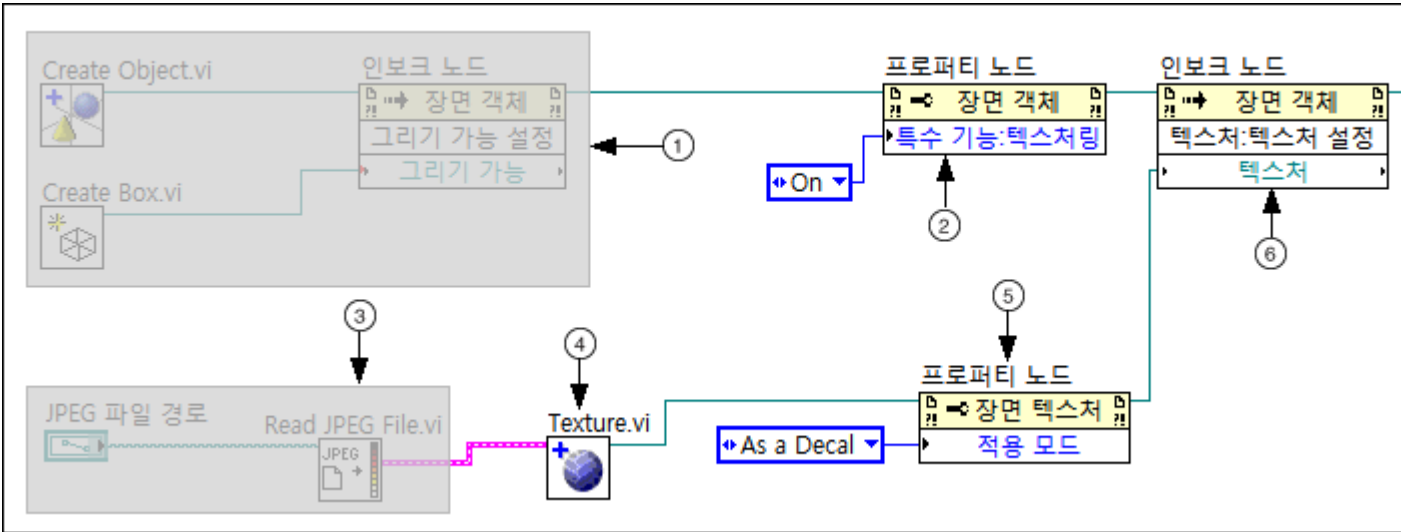
3D 객체에 대한 참조	특수 기능: 텍 스처링 프로퍼 티	[텍스처] VI	적용 모드 프로퍼티	텍스처: 텍스처 설정 메소드	3D 그림 컨트 롤



3D 객체 한 개
생성하기

실행 단계

다음 블록다이어그램을 사용하여 3D 객체의 표면 모양을 프로그램적으로 바꿉니다. 회색 섹션을 고유한 프로그래밍 목적에 맞도록 사용자 정의합니다.



다음은 위의 다이어그램에서 중요한 사항을 상세히 설명한 것입니다.

①	모양을 변경하려는 3D 객체에 대한 참조를 얻습니다. 프로그래밍 목적에 따라 이 참조를 얻는 방법이 달라집니다. 3D 객체 참조를 얻는 가장 일반적인 방법은 새 3D 객체를 생성하는 것입니다. 이 예제에서는 새 3D 상자를 생성하여 3D 객체 참조를 얻습니다.
②	특수 기능:텍스처링 프로퍼티를 사용하여 3D 객체에 텍스처를 추가할 수 있습니다. 특수 기능:텍스처링 프로퍼티를 On으로 설정하지 않으면, 텍스처를 3D 객체에 적용할 수 없습니다. 텍스처를 적용하려는 객체가 다른 객체의 하위 객체인 경우, 특수 기능:텍스처링 프로퍼티를 Inherit Value로 설정하여 직상위 객체의 텍스처 셋팅을 상속할 수 있습니다.

	원하는 텍스처를 정의하는 LabVIEW 이미지 데이터를 얻습니다. 다음 방법 중 하나를 사용하여 LabVIEW 이미지 데이터를 얻습니다: • 그래픽 포맷 팔레트에서 적절한 읽기 VI를 사용하여 JPG와 같은 표준 이미지 포맷을 LabVIEW 이미지 데이터로 변환합니다. • 기존의 3D 객체의 일부인 이미지에 접근하려면, 3D 그림 컨트롤 프로퍼티 또는 메소드를 사용합니다. • 사용자 이미지를 정의하려면 이미지 클러스터를 생성합니다. 이 예제는 [JPEG 파일 읽기] VI를 사용하여 JPEG 파일을 LabVIEW 이미지 데이터로 변환합니다.
④	[텍스처] VI는 LabVIEW 이미지 데이터를 장면 텍스처 참조로 변환합니다. 이미지 데이터를 장면 텍스처 참조로 변환해야 장면 텍스처 프로퍼티 및 메소드와 상호작용하여 3D 객체에서 텍스처가 나타나는 방식을 변경할 수 있습니다. 적용 모드 프로퍼티는 3D 객체에서 텍스처가 나타나는 방식을 지정합니다. 텍스처에 대해 적용 모드를 설정하지 않은 경우, LabVIEW가 텍스처 쉼기를 적용하여, 이미지 데이터가 객체의 배경색과 섞입니다. 텍스처가 원래 2D 이미지의 색조를 보존하게 하려면 적용 모드를 As a Decal 또는 As a replacement로 설정합니다. 이 예제는 텍스처의 적용 모드 프로퍼티를 사용하여 텍스처 이미지가 원래 색 그대로 그림으로 나타나도록 합니다.
⑥	텍스처:텍스처 설정 메소드는 참조된 3D 객체에 텍스처를 지정합니다.
⑦	장면 객체 참조를 사용하여 새 3D 객체를 디스플레이하거나 조작합니다. 이 참조를 사용하여 다음과 같은 작업을 수행할 수 있습니다: • 3D 객체 디스플레이—프론트패널에서 3D 그림 컨트롤을 생성하고, 참조를 연관된 블록다이어그램 터미널에 연결. • 3D 객체의 크기와 위치 조절—참조를 변환 VI 중 하나에 연결. • 여러 객체가 있는 3D 장면에서 객체 추가—참조를 사용하여 여러 3D 객체를 가진 3D 장면 생성. • 3D 객체의 다른 속성과 상호작용—참조를 장면 객체 프로퍼티 또는 메소드에 연결. 이러한 프로퍼티와 메소드는 색, 빛 소스, 텍스처와 같은 3D 객체의 속성 및 동작을 제어합니다. 이 예제는 3D 그림 컨트롤을 사용하여 텍스처가 있는 3D 객체를 디스플레이합니다.

팁 및 문제 해결 정보

- 더 복잡한 텍스처를 만들 수 있습니다. 장면 텍스처 참조의 장면 텍스처 프로퍼티를 사용하여 필터 모드, 수평 또는 수직 줄 바꿈 모드 및 변환과 같은 기타 텍스처 셋팅을 설정합니다.
- 장면의 여러 3D 객체에 원하지 않는 텍스처가 나타납니까? **3D 객체 계층구조**의 상위 객체에 텍스처를 적용했을 가능성이 있습니다. 상위 3D 객체를 변경하면 이 객체의 모든 하위 객체에 변경 사항이 적용됩니다. 이 문제를 해결하려면 텍스처를 상위 객체 참조 대신 하위 객체 참조에 적용합니다.
- 텍스처가 전혀 나타나지 않습니까? 특수 기능:텍스처링 프로퍼티는 사용자가 프로퍼티를 직접 On으로 설정하거나 상위 객체로부터 On 값을 상속받아, On 값으

로 설정되어야 합니다. 특수 기능:텍스처링 프로퍼티에 올바른 값을 설정했는지 확인하십시오.

- 3D 그림 컨트롤이 프런트패널에서 참조 번호로 나타납니까? 3D 그림 컨트롤은 블록 다이어그램의 터미널이 아니라, 프런트패널에서 생성해야 합니다. 프런트패널에서 3D 그림 컨트롤을 생성한 후, 연관된 블록 다이어그램 터미널을 장면 객체 참조에 연결하여 참조된 객체를 디스플레이합니다.
- 프런트패널의 3D 그림 컨트롤과 상호작용할 수 없습니까? 다른 각도에서 3D 장면을 볼 수 있도록 **카메라를 설정**해야 합니다.
- 항상 에러 입력과 에러 출력 터미널을 연결해야 합니다. 이 토픽의 예제는 프로그래밍 로직을 강조하기 위해 에러 핸들링을 생략했지만, 에러 핸들링을 사용하면 디버깅 기능을 활성화하고 데이터 흐름을 조절할 수 있습니다.

예제

텍스처가 있는 객체를 가진 3D 장면의 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

 예제 열기  관련 예제 찾기

관련 토픽

- [하나의 객체를 가진 3D 장면 생성](#)
- [여러 객체를 가진 3D 장면 생성](#)
- [객체를 볼 수 있도록 카메라 설정](#)
- [빛 소스 설정](#)
- [텍스트 추가](#)
- [별도의 장면 윈도우 생성](#)

3D 객체간 관계 생성하기

하나 이상의 3D 객체를 가진 **3D 장면을 생성**하려면, 3D 객체 관계의 계층구조를 설정해야 합니다. 모든 객체에 대한 참조를 3D 그림 컨트롤 터미널 또는 (**장면 윈도우를 설정**하는 경우) 장면 프로퍼티에 연결하는데 사용하는 3D 객체는 3D 장면에서 다른 모든 객체의 상위에 해당합니다. 객체의 참조를 계층구조에서 더 높은 객체에 연결하여 장면에 추가하는 3D 객체는 직하위 3D 객체입니다.

3D 객체가 다른 3D 객체의 직상위인 경우, 이 객체에 적용하는 모든 변환은 해당 3D 객체의 직하위에 영향을 미치게 됩니다. 직하위 3D 객체는 또한 관계 계층구조에서 그 객체 아래에 있는 하위 객체에 대한 직상위 객체입니다. 큰 3D 장면과 객체의 움직임이 서로 관련된 3D 장면에서 객체 관계 기능을 사용하여 보다 효율적으로 프로그램할 수 있습니다.

다음 단계를 따라 여러 객체가 있는 3D 장면을 생성합니다.

1. **3D 장면을 생성**하거나 3D 장면을 포함하는 VI를 엽니다.

2. 3D 객체를 3D 그림 컨트롤 터미널에 연결하는 와이어를 제거합니다.
3. 그리기 가능 설정 메소드의 참조 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>장면 객체 클래스의 메소드>>객체>>객체 추가를 선택하여 객체:객체 추가 메소드를 생성합니다.
4. 객체:객체 추가 메소드를 그리기 가능 설정 메소드의 오른쪽에 추가하고, 그리기 가능 설정 메소드의 참조 출력을 객체:객체 추가 메소드의 참조 입력에 연결합니다.
5. 객체:객체 추가 메소드의 참조 출력을 3D 그림 컨트롤 터미널에 연결합니다. 3D 객체가 직상위 3D 객체가 됩니다.
6. 다른 [객체 생성] VI를 블록다이어그램에서 객체:객체 추가 메소드의 왼쪽에 추가합니다.
7. [객체 생성] VI의 장면:새 객체 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>장면 객체 클래스의 메소드>>그리기 가능 설정을 선택하여 다른 그리기 가능 설정 메소드를 생성합니다.
8. 새 그리기 가능 설정 메소드를 [객체 생성] VI와 객체:객체 추가 메소드 사이에 추가하고, [객체 생성] VI의 장면:새 객체 출력을 그리기 가능 설정 메소드의 참조 입력에 연결합니다.
9. 기하 팔레트에서 VI를 선택하고 블록다이어그램에서 그리기 가능 설정 메소드의 왼쪽에 놓습니다.
10. 선택한 기하 VI의 새 xxx 참조 번호 출력을 그리기 가능 설정 메소드의 그리기 가능 입력에 연결합니다.
11. (옵션) 새 3D 객체를 3D 장면의 다른 위치로 이동합니다. [객체 이동] VI를 블록다이어그램에 추가합니다.
12. 새로운 그리기 가능 설정 메소드의 참조 출력을 [객체 이동] VI의 장면:객체 입력 입력에 연결합니다.
13. [객체 이동] VI의 이동 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>상수를 선택하고, 상수 값을 하위 객체를 이동하는데 사용할 x, y, z 좌표로 변경합니다.
14. [객체 이동] VI의 장면:객체 출력 출력을 객체:객체 추가 메소드의 객체 입력에 연결합니다.

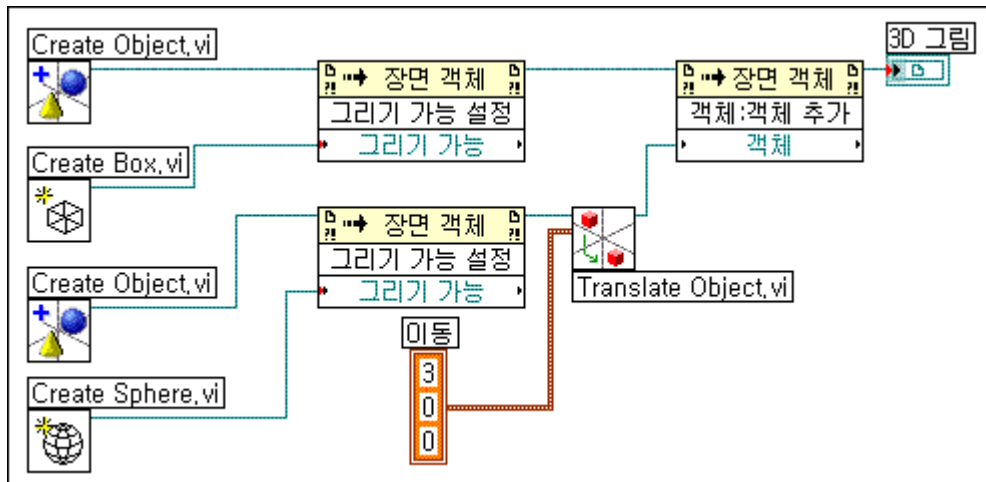


노트

3D 장면에 추가한 다른 객체를 이동하지 않는 경우, 이동하지 않은 3D 객체가 장면에서 서로의 윗부분에 나타나지만, 잘 안 보일 수도 있습니다.

1. 단계 11을 완료하지 않은 경우, 새로운 그리기 가능 설정 메소드의 참조 출력을 객체:객체 추가 메소드의 객체 입력에 연결합니다.

다음의 블록다이어그램은 두 개의 3D 객체가 있는 3D 장면을 보여줍니다.



1. 블록다이어그램의 **에러 입력과 에러 출력 터미널을 연결**합니다.
2. 이 VI를 실행하여 3D 장면을 생성합니다.

여러 개의 객체 관계를 가진 3D 장면의 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

예제 열기 관련 예제 찾기

3D 장면에 텍스트 추가하기

텍스트 객체를 생성하여 장면을 생성할 때 나타나는 **3D 장면에** 텍스트를 추가할 수 있습니다. 다음 단계를 따라 3D 장면에 텍스트를 추가합니다.

1. 3D 그림 컨트롤을 프런트패널에 추가합니다.
2. 객체 생성 VI를 블록다이어그램에서 3D 그림 컨트롤의 왼쪽에 추가합니다.
3. [객체 생성] VI의 장면:새 객체 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>장면 객체 클래스의 메소드>>그리기 가능 설정을 선택하여 그리기 가능 설정 메소드를 블록다이어그램에 추가합니다.

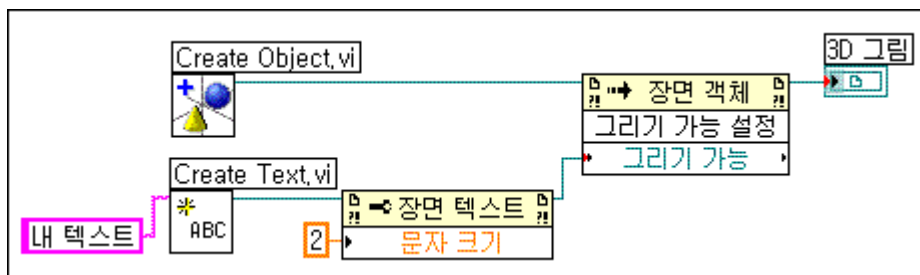


노트

또한 **클래스 탐색기 윈도우를 사용**하여 **프로퍼티와 메소드**를 검색하고 생성할 수 있습니다.

1. [객체 생성] VI의 장면:새 객체 출력을 그리기 가능 설정 메소드의 참조 입력에 연결합니다.
2. [텍스트 생성] VI를 블록다이어그램에 추가하여 새 텍스트 객체를 생성합니다.
3. [텍스트 생성] VI의 텍스트 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택하여 새 상수를 생성합니다.
4. 이전 단계에서 생성한 상수에 3D 장면에서 나타내려는 텍스트를 입력합니다.

5. [텍스트 생성] VI의 새 텍스트 참조 번호 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>장면 텍스트 클래스의 프로퍼티>>문자 크기를 선택하여 문자 크기 프로퍼티를 블록다이어그램에 추가합니다.
6. [텍스트 생성] VI의 새 텍스트 참조 번호 출력을 문자 크기 프로퍼티의 참조 입력에 연결합니다.
7. 문자 크기 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 쓰기로 변경을 선택합니다.
8. 문자 크기 프로퍼티의 문자 크기 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>상수를 선택하여 새 상수를 생성합니다.
9. 이전 단계에서 생성한 상수에 문자 크기의 값을 입력합니다.
10. (옵션) **문자 크기 프로퍼티 노드를 확장**하여 추가적인 장면 텍스트 프로퍼티를 설정하십시오.
11. 문자 크기 프로퍼티의 참조 출력을 그리기 가능 설정 메소드의 그리기 가능 입력에 연결합니다.
12. 그리기 가능 설정 메소드의 참조 출력을 3D 그림 컨트롤 터미널에 연결합니다. 다음 블록다이어그램은 My Text라는 텍스트를 포함하는 3D 장면을 보여줍니다. 이 텍스트는 문자 크기 2로 설정되어 있습니다.



1. (옵션) 다른 설정 옵션에 대해서는 그리기 가능 설정 메소드를 3D 그림 컨트롤 터미널에 연결하는 와이어를 제거한 후, 그리기 가능 설정 메소드의 참조 출력을 장면 객체 프로퍼티 또는 메소드에 연결합니다.



노트 텍스트 객체에는 **텍스처**를 적용할 수 없습니다.

1. 블록다이어그램의 **에러 입력과 에러 출력 터미널을 연결**합니다.
2. 이 VI를 실행하여 3D 장면을 생성합니다.

3D 장면 윈도우 설정하기

3D 장면을 생성한 후, 3D 장면을 제공할 다른 장면 윈도우를 설정할 수 있습니다.

다음 단계를 따라 3D 장면 윈도우를 설정합니다.

1. **3D 장면을 생성**하거나 3D 장면을 포함하는 VI를 엽니다.

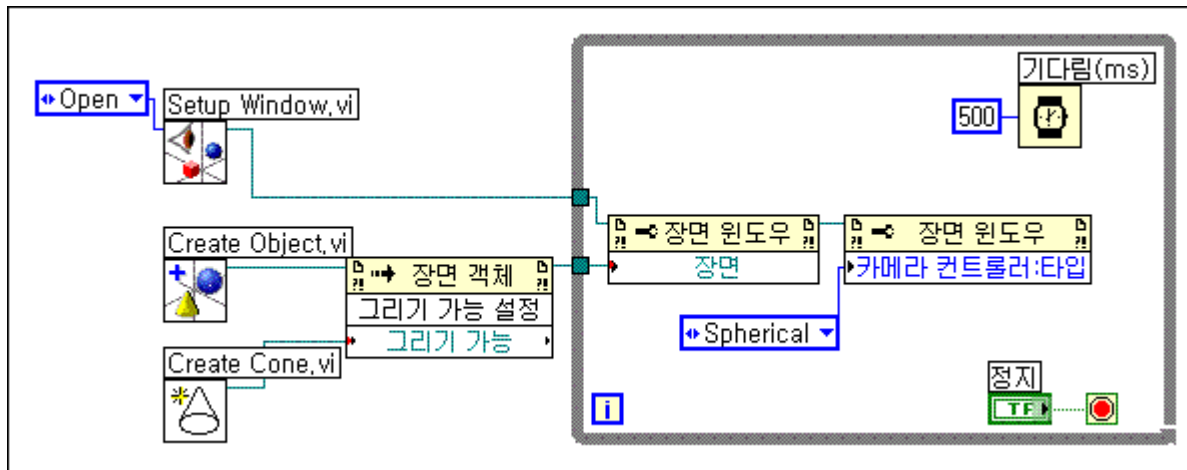
2. 3D 그림 컨트롤 터미널을 삭제하고 깨진 와이어를 제거하십시오.
3. [윈도우 설정] VI를 블록다이어그램에 추가합니다.
4. [윈도우 설정] VI의 윈도우 상태 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 상수 값을 Open, Maximized 또는 Minimized로 설정합니다.



노트

윈도우 상태를 Hidden으로 설정하면 VI를 실행할 때 해당 장면 윈도우를 볼 수 없습니다.

1. While 루프를 블록다이어그램에서 기존 블록다이어그램 객체의 오른쪽에 추가합니다.
2. While 루프의 **조건 터미널**에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 생성을 선택하여 불리언 컨트롤을 생성합니다. 이 컨트롤은 VI의 정지 버튼으로 기능합니다.
3. [윈도우 설정] VI의 새 윈도우 참조 번호 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>장면 윈도우 클래스의 프로퍼티>>장면을 선택하여 장면 프로퍼티를 생성합니다.
4. 장면 프로퍼티를 While 루프 안에 추가하고, 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 쓰기로 변경을 선택합니다.
5. While 루프 밖에서 그리기 가능 설정 메소드의 참조 출력을 장면 프로퍼티의 장면 입력에 연결합니다.
6. While 루프 바깥의 [윈도우 설정] VI 새 윈도우 참조 번호 출력에서 장면 프로퍼티의 참조 입력에 연결합니다.
7. (옵션) 장면에서 카메라를 설정합니다.
 1. 장면 프로퍼티의 참조 출력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>장면 윈도우 클래스의 프로퍼티>>카메라 컨트롤러>>타입을 선택하여 카메라 컨트롤러:타입 프로퍼티를 생성합니다.
 2. While 루프 안에서 카메라 컨트롤러:타입 프로퍼티를 장면 프로퍼티의 오른쪽에 추가하고, 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 쓰기로 변경을 선택합니다.
 3. 장면 프로퍼티의 참조 출력을 카메라 컨트롤러:타입 프로퍼티의 참조 입력에 연결합니다.
 4. 카메라 컨트롤러:타입 프로퍼티의 카메라 컨트롤러:타입 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>상수를 선택하고 상수 값을 구형 또는 플라잉으로 설정합니다.
8. (옵션) [기다림(ms)] 함수를 While 루프에 추가하여 장면 윈도우에서 카메라가 보다 자연스럽게 이동하도록 합니다. While 루프에 기다림 함수를 추가하지 않으면, 카메라를 움직일 때 3D 장면이 매끄럽지 않게 움직일 수 있습니다.
 1. [기다림(ms)] 함수의 대기 시간(ms) 입력에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>>상수를 선택하고, 상수 값을 1 또는 1보다 큰 정수로 설정합니다. 다음의 블록다이어그램은 회전 카메라 컨트롤러를 사용하고, While 루프 내에 [기다림 (ms)] 함수가 있도록 설정된 장면 윈도우를 보여줍니다.



1. 블록 다이어그램의 **에러 입력과 에러 출력 터미널을 연결**합니다.
2. 이 VI를 실행하여 장면 윈도우에서 3D 장면을 생성합니다.

이 과정은 장면 윈도우를 설정하는 방법에 대한 개요를 보여줍니다. 장면 윈도우 프로퍼티와 메소드를 사용하여 윈도우 경계값, 자동 투영 모드 또는 투영 행렬과 같은 장면 윈도우에 대한 셋팅을 추가적으로 설정합니다.

장면 윈도우에서 제공된 3D 장면의 예제는 labview\examples\Graphics and Sound\3D Picture Control 디렉토리에 있는 3D Model of Solar System VI를 참조하십시오.

예제 열기 관련 예제 찾기

볼리언 객체의 기계적 동작 변경하기

볼리언 컨트롤은 물리적 인스트루먼트의 동작과 유사한 프런트패널 윈도우를 생성하도록 볼리언 객체를 사용자 정의할 수 있는 6가지 종류의 기계적 동작을 갖습니다.



노트

바로 가기 키를 볼리언 컨트롤의 기계적 동작에 할당하면, 키보드가 그 값을 토글합니다. 예를 들어, 볼리언 컨트롤에 대해 바로 가기 키를 할당하고 해당 바로 가기 키를 래치 기계적 동작 중에 하나로 설정하면, 읽을 때 버튼이 언래치됩니다.

다음 단계를 통해 볼리언 컨트롤의 기계적 동작을 변경합니다.

1. 누름 버튼과 같은 볼리언 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 볼리언 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 볼리언 프로퍼티 대화 상자를 디스플레이합니다.
3. 볼리언 프로퍼티 대화 상자의 동작 탭을 클릭합니다.
4. 버튼 동작 리스트에서 어플리케이션에 적절한 **버튼 동작**을 선택합니다. 동작의 설명이 동작 설명 섹션에 나타나고 선택된 동작 미리보기 섹션에서 선택된 동작을 테스트할 수 있습니다. 동작 설명 섹션에 나타나는 아이콘에서, M은 컨트롤을

작동할 때 마우스 버튼의 이동을 나타내며, V는 컨트롤의 출력값을 나타내며, RD는 VI가 컨트롤을 읽는 시간의 포인트를 나타냅니다.



노트 로컬 변수가 있는 객체에서는 래치 기능을 사용할 수 없습니다. 불리언 값을 래치 기계적 동작으로 설정하는 경우, 값 프로퍼티와 값(신호) 프로퍼티는 VI가 실행 중이거나 실행되도록 예약되어 있을 때와 같이 유휴 상태에 있지 않은 경우에 에러를 반환합니다. 래치 기계적 동작으로 설정된 불리언 값이 있을 때 경합 조건이 발생할 수 있기 때문에 래치 기계적 동작으로 설정된 불리언 값을 프로그램적으로 읽을 수 없습니다.

불리언 컨트롤 기계적 동작에 대한 예제는 labview\examples\Controls and Indicators\Boolean 디렉토리에 있는 Mechanical Action VI를 참조하십시오.

예제 열기 관련 예제 찾기

혼합 확인란 설정하기

혼합 확인란을 사용하여 참, 거짓, 혼합 값을 디스플레이합니다. 다음 단계를 따라 혼합 확인란을 생성 및 설정합니다.

1. 혼합 확인란을 프런트패널 윈도우에 추가합니다.
2. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 혼합 확인란 프로퍼티 대화 상자를 디스플레이합니다. 모양 페이지에서 활성 상태 및 컨트롤의 값이 On, Off 또는 혼합일 때 LabVIEW가 디스플레이할 문자열을 설정할 수 있습니다.
3. 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 혼합 허용을 선택하여 혼합 확인란의 혼합 값을 대화식으로 설정할 수 있습니다. 바로 가기 메뉴의 혼합 허용 아이템에서 확인 표시를 제거하는 경우, LabVIEW는 혼합 확인란의 값을 혼합으로 지정할 수 있지만, 사용자가 이 값을 클릭하여 혼합으로 작동할 수 없게 됩니다.

또는 혼합 허용 프로퍼티를 사용하여 혼합 확인란의 혼합 값을 프로그램적으로 비활성화할 수 있습니다.

혼합 확인란의 혼합 값을 설명하는 예제는 labview\examples\Controls and Indicators\Boolean 디렉토리에 있는 System Mixed Checkbox VI를 참조하십시오.

예제 열기 관련 예제 찾기

라디오 버튼 컨트롤 생성하기

다음 단계를 따라 라디오 버튼 컨트롤을 생성합니다.

1. 라디오 버튼 컨트롤을 프런트패널 윈도우에 추가합니다.

- (옵션) 추가적인 불리언 컨트롤을 라디오 버튼 컨트롤에 놓습니다. 또한 라디오 버튼 컨트롤에서 기존 불리언 컨트롤을 **복제**할 수 있습니다.
- (옵션) 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음 항목을 선택하여 라디오 버튼 컨트롤을 설정합니다.
 - 선택하지 않기 허용 – 사용자에게 라디오 버튼 컨트롤에서 불리언 컨트롤을 선택하지 않거나 하나를 선택할 수 있는 옵션을 제공합니다.
 - 기계적 동작 – 라디오 버튼의 **기계적 동작**을 누를 때 스위치 또는 놓을 때 스위치로 설정합니다. 라디오 버튼 컨트롤의 개별 불리언 컨트롤에 대한 기계적 동작은 설정할 수 없습니다.
- 라벨링 도구를 사용하여 라디오 버튼 컨트롤의 각 불리언 컨트롤의 **라벨을 편집**합니다. 각 불리언 컨트롤은 반드시 고유 라벨을 가져야만 합니다. 선택하지 않기 허용 옵션을 선택하고 라디오 버튼 컨트롤의 불리언 컨트롤에 선택 없음의 라벨을 생성한 경우, V는 깨진 실행 버튼을 반환합니다.
- 케이스 구조를 생성**하여 라디오 버튼 컨트롤에서 사용자가 선택한 항목을 처리합니다. 라디오 버튼 컨트롤은 블록 다이어그램에 열거형 타입으로 나타납니다. 선택된 항목이 없는 경우, 라디오 버튼 컨트롤의 숫자값은 0이며 문자열 값은 선택 없음입니다.

또한, 선택하지 않기 허용 프로퍼티를 사용하여 프로그램적으로 사용자에게 라디오 버튼 컨트롤의 불리언 컨트롤을 선택하지 않을 수 있는 옵션을 제공할 수 있습니다.

라디오 버튼 컨트롤 사용의 예제는 다음 VI를 참조하십시오:

- 라디오 버튼 VI: labview\examples\Controls and Indicators\Boolean  예제 열기  관련 예제 찾기

불리언 컨트롤과 인디케이터 생성하기와 작동하기

다음 단계를 따라 프런트패널에 불리언 컨트롤 또는 인디케이터를 추가합니다.

- 누름 버튼과 같은 불리언 컨트롤이나 인디케이터를 프런트패널에 추가합니다.
- (옵션) 항목 이름을 입력합니다. 프런트패널 윈도우의 어느곳에서든 클릭하거나 <Ctrl-Enter> 키를 누르면 편집 세션이 종료됩니다. (macOS) <Option-Enter> 키를 누릅니다. (Linux) <Alt-Enter> 키를 누릅니다.
- 불리언 컨트롤을 참과 거짓 상태 사이를 토글하기 위해 수행 도구로 클릭합니다. 편집 모드에서, 불리언 컨트롤과 불리언 인디케이터를 작동시킬 수 있습니다. **자동 도구 선택 활성화** 상태이면, 불리언 컨트롤 또는 인디케이터의 위로 커서를 움직여야 할 필요가 있으며 임시로 수행도구로 바꾸기 위해 <Ctrl> 키를 누릅니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

불리언 객체 사용자 정의하기

컨트롤 편집기 윈도우 사용을 통해 고유의 불리언 스타일을 디자인할 수 있습니다. 컨트롤 편집기 윈도우에서 참, 거짓, 그리고 임의의 불리언 컨트롤 또는 인디케이터의 상태 변화를 위해 **그림 반입**을 이용할 수 있습니다.

디지털 테이블 만들기

디지털 테이블은 디지털 웨이브폼 그래프에서 플롯할 수 있는 데이터를 포함합니다. 직접 디지털 테이블을 만들거나, 아날로그 데이터를 디지털 데이터로 변환하거나, 또는 디지털 신호를 수집하여 디지털 테이블을 생성할 수 있습니다. 다음 단계를 따라 디지털 데이터의 테이블을 만들거나 수정합니다.

1. [디지털 데이터 만들기] 함수를 블록 다이어그램에 추가합니다.
2. (옵션) 기존의 디지털 데이터를 수정하려면, 디지털 데이터 입력을 연결합니다.
3. 데이터 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>> 상수 또는 생성>> 컨트롤을 선택합니다.
4. 2D 데이터 배열에 값을 입력합니다. 2D 배열의 각 행은 첫 번째 열의 최상위 비트 데이터와 마지막 열의 최하위 비트 데이터가 있는 데이터의 하나의 샘플에 대한 2진수 값을 나타냅니다. 각 값은 특정 디지털 데이터 상태를 나타냅니다. 유효한 값은 다음의 테이블에서와 같이 0에서 7을 포함합니다.

디지털 데이터 상태	설명
0 (로우로 유도)	강제적 로직 로우. 낮은 전압 레벨(V_{OL})로 유도.
1 (하이로 유도)	강제적 로직 하이. 높은 전압 레벨(V_{OH})로 유도.
Z (강제로 끄기)	강제적 로직 하이 임피던스. 드라이버를 끄.
L (로우를 비교)	로직 로우(에지) 비교. 낮은 전압 임계점보다 낮은 전압 레벨을 비교.
H (하이로 비교)	로직 하이(에지) 비교. 높은 전압 임계점보다 높은 전압 레벨을 비교.
X (알 수 없음 비교)	알 수 없는 로직 비교. 비교하지 않음.
T (비교 끄기)	로직 하이 임피던스(에지) 비교. 낮은 전압 임계점(V_{OL})과 높은 전압 임계점(V_{OH}) 사이의 전압 레벨을 비교.
V (유효함 비교)	로직 유효 레벨(에지) 비교. 낮은 전압 임계점(V_{OL})보다 낮거나 높은 전압 임계점(V_{OH})보다 높은 전압 레벨을 비교.



노트

배열 상수를 데이터 입력에 연결한 경우, 디지털 링 상수를 사용하여 간단하게 배열에 디지털 데이터를 입력할 수 있습니다. 디지털 링 상수를 사용하면 8개 디지털 데이터 상태에서 선택할 수 있습니다. 디지털 링 상수를 사용하려면 배열의 값에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 대체를 선택하여 현재 함수 팔레트 버전을 디스플레이합니다. 디지털 웨이브폼 팔레트를 탐색하여 디지털 링 상수를 선택합니다. LabVIEW는 해당 디지털 데이터 상태에 이전에 입력했던 모든 상수값을 변환합니다.

1. (옵션) 디지털 데이터를 압축하려면, 변이 입력을 연결하여 증가하지 않는(non-incremental) 변이를 정의합니다. 변이 배열은 데이터 배열과 동일한 행의 수를 포함해야 합니다. 노트를 확장하여 연결되지 않은 입력을 디스플레이합니다. 입력

을 클릭하여 변이 옆의 확인란에 확인 표시를 하여 변이 입력을 디스플레이합니다.

2. 디지털 데이터 출력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택하여 빈 디지털 테이블을 생성합니다.
3. 이 VI를 실행하여 디지털 테이블의 데이터를 디스플레이합니다.

관련 정보

디지털 테이블

디지털 데이터를 그래프로 나타내기

아날로그를 디지털로 VI

일반적인 DAQ 어플리케이션 생성하기

디지털 웨이브폼 그래프

리스트박스, 트리 컨트롤, 테이블

리스트박스, 트리 컨트롤, 테이블

리스트, 테이블 & 트리와 클래식 리스트, 테이블 & 트리 팔레트에 위치한 리스트박스, 트리 컨트롤, 테이블 컨트롤, 인디케이터를 사용하여 사용자에게 선택할 수 있는 아이템의 리스트를 제공합니다.



관련 정보

[리스트박스 생성하기](#) (1202 페이지 참조)

[트리 컨트롤 생성하기](#) (1201 페이지 참조)

[테이블 사용하기](#) (1213 페이지 참조)

트리 컨트롤 생성하기

트리 컨트롤을 사용하여 사용자가 선택할 수 있는 아이템의 계층구조 리스트를 제공합니다. 트리 컨트롤에 입력한 아이템을 아이템이나 노드의 그룹으로 묶습니다.

다음 단계를 따라 트리 컨트롤을 생성합니다.

1. 트리 컨트롤을 프런트패널에 추가합니다.
2. 라벨링 도구를 이용하여 트리 컨트롤의 아이템 이름을 입력하고 <Enter> 키를 누릅니다. LabVIEW는 아이템을 위해 유일한 태그를 생성합니다. 기본적인 태그는 왼쪽에서 시작하는, 텍스트를 가진 첫번째 셀의 텍스트입니다. 기존의 태그가 그

문자열을 이미 사용하고 있을 경우, LabVIEW는 숫자를 문자열에 덧붙여서 새 아이템에 유일한 태그를 생성합니다. 이 태그를 이용하여 프로그램적으로 아이템을 수정하거나 아이템 정보를 수집할 수 있습니다.

3. 입력하려는 각 아이템에 대해 2번째 단계를 반복합니다. LabVIEW는 그 위의 아이템과 같은 계층구조 레벨에서 각각의 새로운 아이템을 생성합니다. 새 아이템 위에 아이템이 없으면, LabVIEW는 최상위의 계층구조 레벨에서 아이템을 생성합니다.
4. 아이템을 한 단계 아래로 만들어 트리 컨트롤에서 아이템의 계층구조를 설정합니다.
5. (옵션) 트리 컨트롤을 설정합니다. 트리 컨트롤에서 기호 추가와 아이템의 편집, 비활성화, 삭제를 할 수 있습니다. 또한, 트리 컨트롤에서 확장/단축 기호 타입, 대소문자 구분, 아이템 끝기를 할 수 있습니다.
6. (옵션) 트리 컨트롤에서 사용자가 선택할 수 있는 아이템의 개수를 설정합니다.
7. 선택된 아이템을 다루기 위해 케이스 구조를 생성합니다. 0 또는 1 아이템 또는 1 아이템으로 설정된 선택 모드의 트리 컨트롤은 문자열로 블록다이어그램에 나타납니다. 프런트패널의 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 선택 모드 >> 데이터 타입을 선택하여 데이터 타입을 변경할 수 있습니다. 단일 선택 트리 컨트롤에서 데이터 값은 현재 선택한 옵션의 유일한 태그입니다. 어떤 아이템도 선택하지 않았다면, 값은 빈 문자열입니다. 0 또는 그 이상의 아이템 또는 1 또는 그 이상의 아이템으로 설정된 선택 모드의 트리 컨트롤은 문자열 배열로 블록다이어그램에 나타납니다. 이때 배열의 값은 현재 선택된 아이템을 나타냅니다. 어떤 아이템도 선택하지 않았다면, 값은 빈 배열입니다.

또한, 트리 컨트롤 메소드를 사용하여 트리 컨트롤에 아이템을 프로그램적으로 추가할 수 있습니다.



관련 정보

[트리 컨트롤 아이템을 한 단계 아래로 보내기](#) (1207 페이지 참조)

[리스트박스와 트리 컨트롤에 기호 추가하기](#) (1214 페이지 참조)

[리스트박스와 트리 컨트롤 아이템 활성화 또는 비활성화하기](#) (1204 페이지 참조)

[리스트박스와 트리 컨트롤에서 행 삭제하기](#) (1204 페이지 참조)

[트리 컨트롤에서 확장/축소 기호 설정하기](#) (1212 페이지 참조)

[리스트박스와 트리 컨트롤에서 대소문자 구분 설정하기](#) (1211 페이지 참조)

[텍스트 기반의 컨트롤에서 실행 중 끊어서 놓기 동작 설정하기](#) (1037 페이지 참조)

[런타임에서 리스트박스와 트리 컨트롤 안에서 선택할 수 있는 아이템의 개수 설정하기](#) (1212 페이지 참조)

[조건 코드 실행을 위한 케이스 구조 생성하기](#) (1244 페이지 참조)

리스트박스 생성하기

다음 단계를 따라 리스트박스를 생성합니다.

1. 리스트박스 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 라벨링 도구를 사용하여 리스트박스에 아이템 이름을 입력하고 <Enter> 키를 누릅니다.
3. 리스트박스에 입력하려는 각 아이템에 대해 2번째 단계를 반복합니다.
4. (옵션) 리스트박스에서 **사용자가 선택할 수 있는 아이템의 개수를 설정**합니다.
5. 선택된 아이템을 다루기 위해 **케이스 구조를 생성**합니다. 선택 모드가 0 또는 1 아이템이나 1 아이템으로 설정된 리스트박스는 32비트 부호있는 정수로 블록다이어그램에 나타납니다. 프런트패널의 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 선택 모드 >> 데이터 타입을 선택하여 데이터 타입을 변경할 수 있습니다. 단일 선택 리스트박스에 대한 데이터 값은 현재 선택된 옵션을 나타내는 번호입니다. 리스트의 첫번째 아이템은 제로 값을 가집니다. 아이템이 선택되지 않은 경우, 값은 -1입니다. 선택 모드가 0 또는 그 이상의 아이템 혹은 1 또는 그 이상의 아이템으로 설정된 리스트박스는 32비트 부호있는 정수의 배열로 블록다이어그램에 나타나며, 이때 배열의 값은 현재 선택된 아이템을 나타냅니다. 어떤 아이템도 선택하지 않았다면, 값은 빈 배열입니다.

또한, 리스트박스 프로퍼티를 사용하여 프로그램적으로 아이템을 리스트박스에 추가하고 리스트박스를 설정할 수 있습니다.

여러 열 리스트박스 생성하기

다음 단계를 따라 여러 열 리스트박스를 생성합니다.

1. 여러 열 리스트박스 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 옵션 아이템을 추가하기 위해서, 라벨링 도구를 사용하여 가장 왼쪽 열의 셀에 입력을 합니다.
3. 같은 행의 셀에 입력하여 각 옵션 아이템에 대한 정보를 추가합니다.
4. (옵션) 여러 열 리스트박스의 **선택 모드를 설정**합니다.
5. (옵션) **기호의 색을 변경**하고 **헤더와 셀의 색**을 변경합니다.
6. 선택된 아이템을 다루기 위해 **케이스 구조를 생성**합니다. 선택 모드가 0 또는 1 아이템이나 1 아이템으로 설정된 리스트박스는 32비트 부호있는 정수로 블록다이어그램에 나타납니다. 프런트패널의 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 선택 모드 >> 데이터 타입을 선택하여 데이터 타입을 변경할 수 있습니다. 단일 선택 리스트박스에 대한 데이터 값은 현재 선택된 옵션을 나타내는 번호입니다. 리스트의 첫번째 아이템은 제로 값을 가집니다. 아이템이 선택되지 않은 경우, 값은 -1입니다. 선택 모드가 0 또는 그 이상의 아이템 혹은 1 또는 그 이상의 아이템으로 설정된 리스트박스는 32비트 부호있는 정수의 배열로 블록다이어그램에 나타나며, 이때 배열의 값은 현재 선택된 아이템을 나타냅니다. 어떤 아이템도 선택하지 않았다면, 값은 빈 배열입니다.

또한 여러 열 리스트박스에 **헤더 텍스트를 추가**하고, **여러 열 리스트박스 헤더를 보이거나 숨기고**, 여러 열 리스트박스에 **행과 열을 추가**할 수 있습니다.

리스트박스과 트리 컨트롤에서 행 삭제하기

리스트박스에서, 삭제하려는 행에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 행 삭제를 선택하여 전체 행을 삭제합니다. 트리 컨트롤에서, 삭제하려는 행에서 마우스 오른쪽 버튼을 선택한 후 바로 가기 메뉴에서 아이템 삭제를 선택하여 행을 삭제합니다. 리스트박스 또는 트리 컨트롤에서 다음 행은 위로 이동합니다.



노트

트리 컨트롤에서 아이템을 삭제하면 삭제하는 아이тем 아래의 모든 아이тем도 삭제됩니다.

행을 삭제한 후에 트리 컨트롤의 값을 업데이트하지 않은 경우, 컨트롤이 오래된 데이터를 반환할 수도 있습니다. 선택된 아이тем을 삭제한다고 트리 컨트롤의 값이 변경되지는 않습니다. 아이тем을 삭제한 이후, 트리 값이 더 이상 존재하지 않는 아이тем에 대한 태그 일 수도 있습니다. 남아있는 트리 아이тем 중에 하나를 선택하여 트리 컨트롤의 값을 업데이트할 수 있습니다.



노트 행의 개별적인 셀은 삭제할 수 없습니다.

트리 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 아이тем 삭제를 선택하여 트리 컨트롤의 모든 행을 삭제합니다.

리스트박스과 트리 컨트롤 아이тем 활성화 또는 비활성화하기

리스트박스과 트리 컨트롤 아이тем에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이тем을 비활성화하기 위하여 아이тем 비활성화를 선택합니다. 리스트박스과 트리 컨트롤 아이тем에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 비활성화된 아이тем을 활성화하기 위하여 아이тем 활성화를 선택합니다.

또한, 프로그램적으로 리스트박스 아이тем을 비활성화하기 위하여 비활성 아이тем 프로퍼티를 사용할 수 있고 트리 컨트롤 아이тем을 비활성화하기 위하여 활성 아이тем:비활성화? 프로퍼티를 사용할 수 있습니다.

리스트박스, 테이블, 트리 컨트롤, 디지털 데이터 컨트롤에 구성요소 디스플레이하기

다음 단계를 따라 리스트박스, 테이블, 트리 컨트롤, 또는 디지털 데이터 컨트롤의 어떤 구성요소가 보일지를 선택합니다.



노트

보이는 아이тем 바로 가기 메뉴에서 사용할 수 있는 옵션은 컨트롤에 따라 다릅니다.

1. 리스트박스, 테이블, 트리 컨트롤, 또는 디지털 데이터 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다.
2. 보이는 아이템을 선택하고 바로 가기 메뉴에서 다음 아이템 중에 하나를 선택합니다:
 - 라벨—**라벨**을 디스플레이합니다.
 - 캡션—**캡션**을 디스플레이합니다.
 - 수직 스크롤 막대—수직 스크롤 막대를 디스플레이합니다.
 - 수평 스크롤 막대—수평 스크롤 막대를 디스플레이합니다.
 - 행 헤더—리스트박스, 테이블, 또는 트리 컨트롤의 행 헤더를 디스플레이합니다.
 - 변이—샘플 숫자를 디지털 데이터 컨트롤에 디스플레이합니다.
 - 열 헤더—단일 열 리스트박스의 열 헤더를 디스플레이합니다.
 - 열 헤더—여러 열 리스트박스, 테이블, 또는 트리 컨트롤의 열 헤더를 디스플레이합니다.
 - 샘플 숫자—샘플 숫자를 디지털 데이터 컨트롤에 디스플레이합니다.
 - 수직 라인—열을 분리하는 수직 라인을 디스플레이합니다.
 - 수평 라인—행을 분리하는 수평 라인을 디스플레이합니다.
 - 기호—리스트박스나 트리 컨트롤에서 아이템 옆에 있는 기호를 디스플레이할 수 있습니다. 리스트박스에서 이 아이템을 선택하면, 리스트박스는 기호를 위해 리스트의 왼쪽 편에 여분의 열을 추가합니다. 기본적으로, 아이템은 기호를 포함합니다. 바로 가기 메뉴나 **프로퍼티 노드**를 사용하여 **기호를 추가**할 수 있습니다.
 - 계층구조 라인 보이기—보이기 상태라면, 항상 아이템의 계층구조를 나타내는 라인을 트리 컨트롤의 아이템 왼쪽에 수직과 수평 라인으로 디스플레이합니다. 보이지 않기 상태라면, 계층구조 라인은 항상 보이지 않습니다. 기본값 상태라면, 운영 시스템의 트리가 계층구조 라인을 보이는 경우 LabVIEW가 계층구조 라인을 보여줍니다. 기본값은 보이기입니다.
 - 인덱스 디스플레이—테이블의 왼쪽 위에 보이는 셀에 행과 열의 인덱스를 디스플레이합니다.

또한, 테이블, 디지털 테이블, 리스트박스, 여러 열 리스트박스, 트리 컨트롤 프로퍼티를 사용하여 프로그램적으로 구성요소를 디스플레이할 수 있습니다.

테이블의 데이터 편집하기

다음 단계를 따라 테이블에 데이터를 잘라내기, 복사하기, 붙여넣기 합니다.

1. 복사 또는 잘라내기를 원하는 데이터가 있는 **셀, 행 또는 열을 선택**하거나 데이터를 붙여넣을 셀, 행, 또는 열을 선택합니다.
2. 선택한 셀에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 >> 데이터 잘라내기, 데이터 복사하기 또는 데이터 붙여넣기를 선택합니다.
 - 데이터 잘라내기—LabVIEW는 클립보드에 선택한 데이터만 복사하고 테이블에서 그 데이터를 삭제합니다. 행 또는 행의 데이터를 잘라낸다면, 그 아래의 모든 행은 위로 이동합니다. 데이터의 열을 잘라낸다면, 모든 열은 오

른쪽에서 왼쪽으로 이동합니다. 행 또는 열의 일부를 잘라낸다면, LabVIEW는 테이블에서 선택한 셀을 포함하는 전체의 행 또는 열을 삭제하지만 선택한 데이터를 클립보드에 복사해 놓습니다.

- 데이터 복사—LabVIEW는 클립보드에 선택한 데이터만 복사합니다.
- 데이터 붙여넣기—LabVIEW는 선택한 셀에 클립보드의 데이터를 붙여넣습니다. 타겟을 먼저 선택하지 않고 데이터를 붙여넣거나 클립보드의 데이터의 개수와 선택된 셀의 배치가 같지 않을 경우, LabVIEW는 데이터에 맞춰 필요한 만큼 테이블에 행과 열을 추가합니다.

실행 중에 리스트박스과 트리 컨트롤 셀 편집하기

다음 단계를 따라 실행 중에 리스트박스 또는 트리 컨트롤의 셀을 편집합니다.

1. 리스트박스 또는 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 편집 가능한 셀을 선택하면, 사용자가 실행 도중에 셀을 편집할 수 있습니다. 또한, 단일 열 리스트박스, 여러 열 리스트박스, 트리 컨트롤에서 셀 편집 허용 프로퍼티를 사용하여 사용자가 프로그램적으로 셀을 편집할 수 있습니다.
2. 실행 중에 편집하려는 셀 텍스트를 클릭한 후 잠시 후 다시 두번째 클릭을 합니다. 또한 편집 위치 프로퍼티를 사용하여 여러 열 리스트박스 또는 트리 컨트롤에서 편집할 셀을 프로그램적으로 선택할 수도 있습니다.
3. 원하는 텍스트를 입력합니다.

셀 편집 이벤트를 사용하여 단일 열 리스트박스, 여러 열 리스트박스 또는 트리 컨트롤 셀에서 텍스트가 바뀌는 이벤트를 처리할 수 있습니다.

테이블에 데이터 입력하기

다음 단계를 따라 테이블에 데이터를 빨리 입력하기 위해서 키보드를 사용합니다.

1. 수행 도구 또는 라벨링 도구를 사용하여 셀 내부를 클릭한 후 데이터를 입력합니다.
2. <Shift> 키와 함께 화살표 키를 눌러서 커서를 인접 셀로 이동합니다.
3. VI가 편집 모드인 경우, 키보드의 <Enter> 키를 눌러 커서를 아래 셀로 이동합니다. 실행 모드에서 **프런트패널 객체 사이에서 탭 이동**을 할 때에는 <Ctrl> 키와 아래 화살표 키를 눌러 테이블에서 탭 이동을 합니다. <Tab> 키를 눌러 커서를 인접 열로 이동하고, 키보드의 <Enter> 키를 눌러 인접 행으로 이동합니다. 탭 이동을 하여 테이블 밖으로 나오려면, <Ctrl> 키와 위 방향 화살표 키를 누릅니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.



노트

여러 라인 입력을 활성화하는 경우, 키보드의 <Enter> 키를 누르면 아래 셀로 커서를 이동하지 않고 같은 셀에 새로운 라인을 생성합니다.

1. 키보드의 <Ctrl-Enter> 키를 누르거나 숫자 키패드의 <Enter> 키를 눌러서 데이터 입력을 끝냅니다.

트리 컨트롤 아이템을 한 단계 아래로 보내기

다음 단계를 따라 아이템을 트리 컨트롤 계층구조에서 위쪽 또는 아래쪽으로 움직입니다.

1. 트리 컨트롤의 아이템에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴로부터 다음 옵션들 중의 하나를 선택합니다:
 - 아이템을 한 단계 아래로—아이템을 계층구조에서 한 레벨 낮은 하위 아이템으로 보냅니다. 그 아이템이 이미 하위 아이템 아래에 하위 아이템으로 나타나 있다면, 그 아이템을 하위 아이템으로 쓸 수 없습니다. 마침표(.) 키를 눌러서 수행 도구로 현재 아이템을 하위 아이템으로 보낼 수 있습니다.
 - 아이템을 한 단계 위로—아이템을 계층구조에서 한 레벨 상위 아이템으로 보냅니다. 아이템이 이미 계층구조에서 가장 상위에 나타나 있을 경우, 그 아이템은 상위로 이동시킬 수 없습니다. 쉼표(,) 키를 눌러서 수행 도구로 현재 아이템을 한 레벨 상위로 이동시킬 수 있습니다.

아이템을 설정해서 다른 아이템을 그 아이템의 하위 아이템으로 놓을 수 없다면, 그 아이템에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 하위 전용 아이템을 선택합니다. 하위 전용 아이템 아래로 또 다른 아이템을 끌어 오면, LabVIEW는 이 아이템을 하위 전용 아이템과 같은 계층구조 레벨에 놓습니다. 다른 아이템이 그 아래에 종속되는 아이템에서 마우스 오른쪽 버튼을 클릭하면 이 메뉴 아이템은 이용할 수 없습니다. 또한 활성 아이템:하위 아이템 전용? 프로퍼티를 사용하여 아이템을 설정할 수 있습니다. 따라서 그 아래의 다른 아이템을 프로그램적으로 하위 레벨로 놓을 수 없습니다.

또한, 활성 아이템:하위 레벨 프로퍼티를 사용하여 트리 컨트롤 아이템의 하위 레벨을 프로그램적으로 설정할 수 있습니다.

리스트박스과 트리 컨트롤에 아이템 삽입하기

다음 단계를 따라 리스트박스과 트리 컨트롤 끝에 아이템을 삽입합니다.

1. 컨트롤에서 마지막 아이템의 다음 셀을 클릭합니다.
2. 원하는 텍스트를 입력합니다.

다음 단계를 따라 리스트박스 또는 트리 컨트롤에 있는 기존 아이템 다음에 아이템을 삽입합니다.

1. 새 아이템을 삽입하려는 곳 다음에 있는 아이템을 클릭합니다.
2. <Shift-Enter> 키를 눌러서 현재 아이템 다음에 아이템을 삽입합니다. 트리 컨트롤에서 이 키 조합은 현재 아이템 계층구조에 동일한 레벨로 새로운 아이템을 놓습니다. 그리고 새로운 아이템 밑에 있는 모든 아이템이 한 줄 아래로 이동합니다.
3. 원하는 텍스트를 입력합니다.

리스트박스 또는 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 앞에 아이템 삽입을 선택하여 아이템을 삽입할 수 있습니다.

리스트박스와 여러 열 리스트박스에 행 또는 열을 삽입 및 삭제하기

다음 단계를 따라서 리스트박스 또는 여러 열 리스트박스에 행이나 열을 삽입합니다.

1. 새로운 행을 입력하려는 곳 아래의 행에서 마우스 오른쪽 버튼을 클릭하거나 열을 입력하려는 곳의 오른쪽에 있는 열에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 앞에 행 삽입 또는 앞에 열 삽입을 선택합니다.

행 또는 열을 삭제하려면 삭제하려는 행이나 열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 행 삭제 또는 열 삭제를 선택합니다.



노트

단일 열 리스트박스에서는 열을 삽입하거나 삭제할 수 없습니다.

테이블에서 데이터의 행이나 열을 삽입하거나 삭제하기

다음 단계를 따라 테이블에서 데이터의 행이나 열을 삽입합니다.

1. 새로운 행을 입력하려는 곳 아래의 행에서 마우스 오른쪽 버튼을 클릭하거나 열을 입력하려는 곳의 오른쪽에 있는 열에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 데이터 처리 >> 앞에 행 삽입이나 앞에 열 삽입을 선택합니다.

행이나 열을 삭제하려면, 삭제하려는 행이나 열에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 >> 행 삭제 또는 열 삭제를 선택합니다.

이 동작은 테이블의 데이터에만 영향을 주고 행이나 열의 모양에는 영향을 주지 않습니다. 예를 들면, 파란 행의 데이터 위에 붉은 행의 데이터를 가진 테이블이 있다고 가정합니다. 붉은 행에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리 >> 앞에 행 삽입을 선택하면, LabVIEW는 붉은 행에 빈 셀을 삽입하고 붉은 행의 데이터가 파란 행으로 이동합니다. LabVIEW는 기존의 붉은 행 위에 새로운 붉은 행을 삽입하지 않습니다.

트리 컨트롤 아이템 선택을 등위로 제한하기

기본적으로, LabVIEW는 트리 컨트롤에서 동일한 직상위 아이템을 공유하는 아이템의 경우만 여러 아이템을 대화식으로 선택할 수 있도록 제한합니다. 직상위가 서로 다른 아이템을 함께 선택할 수 있도록 허용하려면, 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴의 선택 모드 >> 등위 아이템 다중 선택 제한에서 확인 표시를 제거합니다. 또한, 등위 아이템 다중 선택 프로퍼티를 사용하여 프로그램적으로 여러 등위 아이템 선택을 제한하거나 다른 직상위를 갖는 아이템의 선택을 허용할 수 있습니다.

여러 아이템의 대화식 선택을 등위로 제한하는 경우, 로컬 변수를 사용하거나 트리 컨트롤의 값 프로퍼티를 설정하여 서로 다른 직상위를 가진 아이템을 프로그램적으로 선택할 수 있습니다. 하지만, 새로운 아이템을 선택에 대화식으로 추가할 수는 없습니다.

등위 아이템 다중 선택 제한 바로 가기 메뉴를 활성화했으나 여러 개의 비등위 아이템이 이미 선택된 경우, 이후의 선택은 정상적으로 작동하지 않을 수도 있습니다. 선택을 유효한 것으로 다시 설정하면 선택 작동은 정상으로 돌아옵니다.

트리 컨트롤 아이템 열기 및 닫기

다음 방법 중 하나를 사용하여 실행 중에 트리 컨트롤에서 개별 상위 아이템을 열거나 닫을 수 있습니다:

- 상위 아이템 왼쪽에 있는 확장/단축 기호를 클릭합니다.
- 상위 아이템 태그 또는 해당 **계층구조 라인**을 더블 클릭합니다.
- 숫자 키패드의 +와 - 키를 사용합니다.
- 왼쪽과 오른쪽 화살표 키를 사용합니다.

또한, 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 아이템 열기를 선택하여 트리 컨트롤에 있는 모든 아이템을 한 번에 열 수 있습니다. 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 아이템 닫기를 선택하여 트리 컨트롤의 모든 아이템을 한 번에 닫을 수 있습니다.

리스트박스, 테이블, 트리 컨트롤의 행과 열의 크기 조정하기

다음 단계를 따라 리스트박스, 테이블, 트리 컨트롤의 행과 열의 크기를 조정합니다.

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 행 높이 자동 크기 조정을 선택하여 이 메뉴 아이템에서 확인 표시를 제거합니다.
2. 위치 도구를 수평 라인 위로 이동하여 행의 크기를 조정하거나 수직 라인 위로 이동하여 열의 크기를 조정합니다. 도구가 알맞은 위치에 놓이면 다음 커서와 유사한 커서가 나타납니다: ➤
3. 수평 또는 수직 라인을 끌어 각각 행 또는 열의 크기를 조정합니다. 라인을 끄는 동안 <Shift> 키를 눌러 모든 행과 열을 같은 크기로 조정합니다. 테이블의 여러 행 또는 열을 같은 크기로 조정할 수 있습니다. 수행 도구를 사용하여 행 또는 열을 선택하고 <Shift> 키를 누르는 동안 위치 도구를 사용하여 선택된 행 또는 열을 크기 조정합니다.



노트

여러 라인 입력을 활성화할 경우, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 행 높이 자동 크기 조정을 선택하여 각 행을 내용에 맞게 크기 조정하고 각 셀에 모든 데이터를 디스플레이합니다. 이 옵션은 기본적으로 활성화 되어있습니다.

런타임으로 리스트박스과 트리 컨트롤 아이템 선택하기

다음 방법을 사용하여 VI가 실행 모드에 있는 동안 리스트박스나 트리 컨트롤에서 아이템을 선택합니다:

- 아이템을 클릭합니다. 여러 선택 리스트박스나 트리 컨트롤에서, <Ctrl> 키를 누르고 추가적인 아이템을 클릭하여 하나 이상의 아이템을 선택합니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다. <Shift> 키를 누르고 현재 아이템 위와 아래에 있는 아이템을 클릭해서 선택한 현재 아이템과 선택한 두 번째 결과 사이의 모든 아이템을 선택합니다.



노트

기본적으로, LabVIEW는 트리 컨트롤에서 동일한 직상위 아이템을 공유하는 아이템의 경우만 여러 아이템을 대화식으로 선택할 수 있도록 제한합니다. 직상위가 서로 다른 아이템을 함께 선택할 수 있도록 허용하려면, 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴의 선택 모드 >> 등위 아이템 다중 선택 제한에서 확인 표시를 제거합니다.

- 아이템을 클릭한 후, <Ctrl> 키를 누른 상태에서 위 아래 화살표 키를 사용하여 리스트박스나 트리 컨트롤 아이템을 탐색합니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다. 스페이스바를 눌러서 포커스를 가진 아이템을 선택된 아이템 리스트에 추가합니다. <Ctrl>를 계속 누른 상태에서 스페이스바를 다시 눌러 선택된 아이템 리스트에서 포커스를 가진 아이템을 제거합니다.
- 선택하려는 아이템 이름의 일부를 입력합니다. 왼쪽과 오른쪽 화살표 키를 사용하여 입력하는 문자와 일치하는 이전 아이템과 다음 아이템으로 이동합니다.

제로 아이템을 선택할 수 있는 선택 모드를 가진 리스트박스나 트리 컨트롤에서, <Ctrl> 키를 누르고 아이템을 클릭하여 그것을 선택해제합니다. (macOS) <Command> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.



노트 컨트롤의 프로퍼티 대화 상자의 키 조작 페이지에서 <PageUp>, <PageDown>, <Home> 또는 <End>를 바로 가기 키로 지정한 경우, 더 이상 이 키를 리스트박스에서 아이템을 선택하기 위해 사용할 수 없습니다.

트리 컨트롤에서 직상위 아이템 선택하기

다음 단계를 따라 사용자가 하위에 아이템을 가진 아이템과 하위 전용이 아닌 아이템을 선택하고 그 아이템 아래에 다른 아이템을 넣지 않도록 합니다.

- 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다.
- 바로 가기 메뉴에서 선택 모드 >> 상위 아이템의 선택 허용을 선택합니다. 이 아이템 옆의 확인 표시를 제거하면, 사용자는 종속 아이템으로 설정하는 유일한 아이템을 선택할 수 있습니다.

또한 직상위 아이템의 선택 허용 프로퍼티를 사용하여 직상위 아이템을 프로그램적으로 선택할 수 있습니다.

테이블의 셀, 행, 열 선택하기

수행 도구를 사용하여 셀을 클릭하고 커서를 끌어서 여러 셀을 선택합니다. 또한, <Shift> 키를 누르고 커서를 끌어와서 여러 셀을 선택할 수 있습니다. 테이블의 보이는 내용 밖에서 커서를 움직이면, 테이블은 스크롤하고 선택을 연장합니다. 경계선이 선택한 셀 주위로 나타납니다.



노트

커서를 이동할 때 선택한 부분이 테이블의 볼 수 있는 내용 밖으로 확장되는 것을 피하려면, 테이블에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 선택 스크롤링을 선택하여 확인 표시를 제거합니다. 경계선으로 선택을 아웃라인하지 않으려면, 테이블에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고급 >> 선택 보이기를 선택하여 확인 표시를 제거합니다.

다음 단계를 따라 데이터의 전체 행이나 열을 선택합니다.

1. 수행 도구를 사용하여 행의 왼쪽 경계와 열의 위쪽 경계를 클릭합니다. 도구가 적절히 위치를 정하면 화살표 커서가 나타납니다.
2. 행의 높이나 열의 넓이를 가로질러 커서를 끌어옵니다.

테이블에서 모든 데이터를 선택하기 위해, 수행 도구를 사용하여 테이블의 왼쪽 상단의 코너를 클릭합니다. 도구가 적절히 위치를 정하면 두배의 화살표 커서가 나타납니다.

리스트박스과 트리 컨트롤에서 대소문자 구분 설정하기

다음 단계를 따라 런타임으로 리스트박스나 트리 컨트롤에 문자를 입력할 때 LabVIEW가 대문자와 소문자를 다루는 방법을 설정합니다.

1. 리스트박스나 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 키보드 모드를 선택하고 다음 옵션 중에서 선택합니다:
 - 시스템 기본값(기본 셋팅)을 선택하면, 리스트박스나 트리 컨트롤은 주어진 플랫폼에서 다른 리스트박스와 트리 컨트롤에서와 같은 작동을 합니다. (Windows와 macOS) 리스트박스와 트리 컨트롤은 대소문자를 구분합니다. (Linux) 리스트박스와 트리 컨트롤은 대소문자를 구분합니다.
 - 대소문자 구분을 선택할 때, 사용자에게 의해 입력된 대문자와 소문자가 아 이템과 일치하는 아 이템만을 선택합니다.
 - 대소문자 구분 없음을 선택하면, 대소문자 구분이 무시됩니다.

또한, 키보드 모드 프로퍼티를 사용하여 리스트박스, 여러 열 리스트박스, 트리 컨트롤의 대소문자 구분을 프로그램적으로 설정할 수 있습니다.

런타임에 열 폭 설정하기

사용자가 런타임에 열 폭을 조절할 수 있도록 하려면 여러 열 리스트박스, 트리 컨트롤, 또는 테이블에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 이동가능한 열 분리를 선택합니다.

트리 컨트롤에서 확장/축소 기호 설정하기

다음 단계를 따라 기호의 타입을 설정하여 종속 아이টে를 가진 각 아이টে 옆에 디스플레이합니다. 확장/축소 기호를 사용하여 트리에서 **상위 아이টে를 열고 닫습니다**.

1. 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다.
2. 확장/축소 기호 타입을 선택하고 바로 가기 메뉴에서 다음 아이টে 중의 하나를 선택합니다.
 - 없음—기호를 디스플레이하지 않습니다.
 - LabVIEW—플랫폼에 독립적인 삼각형 기호를 디스플레이합니다.
 - Mac OS—Mac OS 스타일의 삼각형 기호를 디스플레이합니다.
 - Windows—Windows 스타일의 플러스와 마이너스 기호를 디스플레이합니다.
 - 기본—OS의 기본 기호를 디스플레이합니다.

또한, 확장/축소 기호:기호 타입 프로퍼티를 사용하여 각 상위 아이টে 옆에 디스플레이하는 기호의 타입을 프로그램적으로 설정할 수 있습니다.

트리 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 확장/축소 기호 타입 >> 하위 레벨 0에서 보이기를 선택하여 계층구조의 최상위 레벨 아이টে의 확장/축소 심볼을 보이거나 숨길 수 있습니다. 이 옵션 옆의 확인 표시를 제거한 경우에도, 아이টে를 더블 클릭하여 하위 레벨 0에서 아이টে를 확장하거나 축소할 수 있습니다. 더블 클릭 기능을 비활성화하려면, 참 상수를 아이টে 닫기? 이벤트의 버림? 파라미터에 연결합니다.

또한, 확장/축소 기호:루트에서 기호 보이기 프로퍼티를 사용하여 프로그램적으로 기호를 보이거나 숨길 수 있습니다.

런타임에서 리스트박스와 트리 컨트롤 안에서 선택할 수 있는 아이টে의 개수 설정하기

다음 단계에 따라 런타임으로 **리스트박сна 트리 컨트롤에서 선택**할 수 있는 아이টে이 얼마나 될지 지정합니다.

1. 프런트패널 윈도우의 리스트박сна 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 선택 모드를 선택하고 다음 옵션 중에서 선택합니다:

- 0 또는 1 아이템—작동이 1 아이템인 경우와 같습니다. 다만, 차이점은 선택된 아이템이 하나도 남지 않게, 현재 아이템을 선택 해제할 수 있다는 점입니다. 커서로 아이템을 끌어와서 리스트박스에서 아이템을 제거할 수 있습니다.
- 1 아이템—(기본) 리스트박스나 트리 컨트롤에 있는 하나의 아이템이 (하이라이트되어) 선택됩니다. 다른 아이템을 클릭할 때, 새 아이템이 (하이라이트되어) 선택되고 이전 아이템은 선택 해제됩니다.
- 0 또는 그 이상의 아이템—여러 아이템을 선택할 수 있습니다.
- 1 또는 그 이상의 아이템—여러 아이템을 선택할 수 있지만, 적어도 하나의 아이템은 선택되어야 합니다.

또한, 선택 모드 프로퍼티를 사용하여 리스트박스나 트리 컨트롤에서 선택할 수 있는 아이템의 개수를 프로그램적으로 설정할 수 있습니다.

또한, 트리 컨트롤에서 등위로 **여러 아이템의 선택을 제한**할 수 있습니다.

리스트박스 또는 트리 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 선택 모드 >> 전체 행 하이라이트를 선택하여 아이템을 선택했을 때 전체 행을 하이라이트합니다.

테이블 사용하기

테이블은 문자열의 2차원 배열입니다. 문자열 함수를 사용하여 테이블을 다룹니다.

테이블의 각 셀이 문자열이고 각 셀은 열과 행에 상주합니다. 프런트패널에 테이블을 추가할 때 헤더를 입력하고 이것을 라벨링 도구나 수행 도구를 사용하여 변경할 수 있습니다. 테이블의 행과 열 헤더는 테이블 데이터의 일부가 아닙니다. 테이블 헤더는 프로퍼티 노드를 사용하여 읽거나 설정할 수 있는 데이터의 분리된 조각입니다.

테이블의 왼쪽 상단의 코너에 디스플레이되어 있는 인덱스는 어느 셀을 볼 수 있는지 가리킵니다. 배열에서 수행하는 것처럼 이 인덱스를 다룰 수 있습니다.



팁

테이블에서 Microsoft Excel과 같은 위치로 데이터와 이미지를 반출합니다. 테이블에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반출을 선택하여 사용 가능한 반출 옵션을 봅니다.



관련 정보

[테이블에 배열 디스플레이하기](#) (1053 페이지 참조)

[데이터와 이미지 반출하기](#) (1132 페이지 참조)

리스트박스, 테이블, 트리 컨트롤에 헤더 텍스트 추가하기

다음 단계를 따라 리스트박스, 테이블, 또는 트리 컨트롤에 헤더 텍스트를 추가합니다.

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목 >> 행 헤더와 보이는 항목 >> 열 헤더(또는 열이 여러 개인 컨트롤의 경우에도 보이는 항목 >> 열 헤더)를 선택하여 행 헤더 및 열 헤더를 디스플레이합니다.



노트

단일 열 리스트박스는 행 헤더를 가지지 않습니다.

1. 라벨링 도구를 사용하여 텍스트를 추가하려는 헤더를 클릭합니다.
2. 원하는 텍스트를 입력합니다.



노트 왼쪽 위 헤더 셀에는 텍스트를 입력할 수 없습니다.

리스트박스과 트리 컨트롤에 기호 추가하기

다음 단계를 따라 리스트박스 또는 트리 컨트롤에 기호를 추가합니다.

1. 다음 컨트롤 중에서 하나를 프런트패널 윈도우에 추가합니다:
 - 리스트박스
 - 트리 컨트롤
2. 라벨링 도구를 사용하여 리스트박스와 트리 컨트롤에 텍스트를 입력합니다.
3. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목 >> 기호를 선택합니다.
4. 리스트박스 또는 트리 컨트롤의 항목에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 항목 기호를 선택하고 원하는 기호를 선택합니다.



노트

여러 열 리스트박스에서, LabVIEW는 첫번째 열에 기호를 추가합니다.

프로그램에서 리스트박스에 기호를 추가하기 위해서 항목 기호 프로퍼티를 사용할 수 있습니다.

또한 트리 컨트롤, 리스트박스, 여러 열 리스트박스 메소드를 사용하여 트리 컨트롤 또는 리스트박스의 항목 기호를 사용자 정의할 수 있습니다.

리스트박스, 여러 열 리스트박스, 트리 컨트롤, 테이블 색칠하기

다음 단계를 따라 리스트박스, 여러 열 리스트박스, 트리 컨트롤, 테이블의 헤더와 셀의 색을 변경합니다.

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 보이는 아이템>>행 헤더를 선택하여 행 헤더를 보여주거나, 보이는 아이템>>열 헤더(열이 여러 개인 컨트롤의 경우에도 보이는 아이템>>열 헤더)를 선택하여 열 헤더를 보여줍니다.
2. 색칠 도구를 사용하여 헤더가 아닌 셀에서 마우스 오른쪽 버튼을 클릭하고 색 선택기를 디스플레이하여 모든 셀에 대한 색을 선택합니다.
3. 색칠 도구를 사용하여 행 헤더에서 마우스 오른쪽 버튼을 클릭하고 색 선택기를 디스플레이하여 모든 행 헤더에 대한 색을 선택합니다.
4. 색칠 도구를 사용하여 열 헤더에서 마우스 오른쪽 버튼을 클릭하고 색 선택기를 디스플레이하여 모든 열 헤더에 대한 색을 선택합니다.
5. 색칠 도구를 사용하여 컨트롤의 왼쪽 위 헤더 셀에서 마우스 오른쪽 버튼을 클릭하고 색 선택기를 디스플레이하여 왼쪽 위 헤더 셀에 대한 색을 선택합니다.

또는 활성 셀 프로퍼티를 사용하여 활성 셀의 열과 행을 프로그램적으로 설정할 수 있습니다. 셀 배경색 프로퍼티를 사용하여 활성 셀의 배경색을 프로그램적으로 설정할 수 있습니다.

여러 라인 입력을 위해서 셀 설정하기

리스트박스, 테이블, 또는 트리 컨트롤에서 키보드의 <Enter> 키를 누를 때 셀에 새로운 라인을 생성하기 위해서 셀을 설정하려면, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 여러 라인 입력을 선택합니다.

여러 라인 입력을 활성화하는 경우, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 행 높이 자동 크기 조정을 선택하여 각 행을 내용에 맞게 크기 조정하고 각 셀에 모든 데이터를 디스플레이합니다. 이 옵션은 기본적으로 활성화 되어있습니다.

또한 여러 라인 입력 프로퍼티를 사용하여, <Enter> 키를 누를 때 새로운 라인이 셀에 프로그램적으로 생성되도록 리스트박스, 여러 열 리스트박스, 테이블, 또는 트리 컨트롤의 셀을 설정할 수 있습니다.

구조

블록다이어그램에 구조 추가하기

다음 단계를 따라 블록다이어그램에 구조를 추가합니다.

1. 구조 팔레트에서 구조를 선택합니다. 커서는 구조의 작은 아이콘이 됩니다.
2. 구조의 상단 코너를 블록다이어그램을 놓으려는 곳에서 클릭하여 커서를 놓고 오른쪽이나 왼쪽으로 움직입니다.
3. 구조가 원하는 크기가 되었을 때 다시 블록다이어그램을 클릭합니다.

구조를 추가한 후에, 구조의 **크기 조정**을 할 수 있습니다.

구조 크기 조정하기

다음 단계를 따라 블록다이어그램의 구조의 크기를 조정합니다.

1. 위치 도구를 구조 경계 위로 이동합니다. 크기 조정 핸들이 구조의 코너와 각 구조 경계의 중간에 나타납니다.
2. 커서를 크기 조정 핸들 위로 이동하여 커서를 크기 조정 커서로 변경합니다.
3. 크기 조정 커서를 사용하여 크기 조정 핸들을 끕니다. 크기 조정 핸들을 끌어당겨서 구조의 크기를 조정하는 경우, 구조는 실시간으로 커지거나 줄어듭니다.



노트

기본적으로, 구조는 내부의 객체보다 작게 크기 조정할 수 없습니다. 구조의 작동을 비활성화하려면, 구조 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 자동 크기 조정을 선택하여 확인 표시를 제거합니다. 객체보다 작은 구조 안으로 객체를 끌어오면, 사용자가 객체를 블록다이어그램에 놓은 후에 객체를 안에 포함할 수 있도록 구조의 크기가 재조정됩니다.

1. 마우스 버튼을 놓습니다.



노트 **정렬 눈금이 활성화된** 경우, 눈금의 크기 단위로만 구조의 크기를 조정할 수 있습니다.

두 프레임의 공유된 경계에서 크기 조정 핸들을 사용하여 플랫폼 시퀀스 구조에서 한 프레임의 크기를 조정할 때, 어떤 프레임이 크기 조정 핸들을 디스플레이하는지 확인하여 올바른 프레임을 크기 조정하도록 합니다. 프레임의 왼쪽 경계에서 크기 조정 핸들을 사용하여 프레임을 축소하려면, 핸들을 오른쪽으로 끕니다. 같은 크기 조정 핸들을 사용하여 프레임을 확대하려면, 핸들을 왼쪽으로 끕니다. 프레임을 축소하거나 확대할 때, 플랫폼 시퀀스 구조의 다른 프레임은 현재 크기로 유지되고 크기 조정된 프레임에 맞춰 이동합니다.

구조 안에 객체 추가하기

객체를 구조 안으로 끌어오거나 객체 주위에 구조를 만들어서 구조 안에 객체를 추가합니다. 구조를 객체 위로 움직여서 구조 안에 객체를 넣을 수 없습니다.

다음 단계에 따라 구조 안에 객체를 추가하는 연습을 합니다.

1. 블록다이어그램에 케이스 구조를 추가합니다.
2. 구조 안에 [틱 카운트 (ms)] 함수를 추가합니다.
3. [틱 카운트 (ms)] 함수를 구조의 경계에 가깝게 움직입니다. 구조의 경계 부근에서 구조 안에 객체를 추가하거나 움직이면, 블록다이어그램에 객체를 추가한 이후에 해당 객체가 필요로 하는 공간을 추가하기 위해 자동으로 구조의 크기가 재조정됩니다. 구조의 크기 자동 조정 기능을 비활성화하기 위해서는 구조의 경계에서 마우스 오른쪽 버튼을 클릭하고 확인 표시를 제거하기 위해 바로 가기 메뉴에서 자동 크기 조정을 선택합니다.

4. 구조 밖으로 [틱 카운트 (ms)] 함수를 움직입니다.
5. [틱 카운트 (ms)] 함수 주위에 또 다른 구조를 추가합니다.
6. 두번째 구조를 선택하고 그것을 삭제합니다. 구조 안의 함수도 삭제되었다는 것을 확인하십시오.



노트

구조에서 객체를 삭제하지 않고 **구조를 제거**할 수 있습니다.

1. 남아 있는 구조 밖에 또 다른 [틱 카운트 (ms)] 함수를 추가합니다.
2. 구조를 [틱 카운트 (ms)] 함수 위로 움직입니다. 구조의 그림자는 이 함수가 구조 위에 있고 그 안에 있지 않다는 것을 나타냅니다.
3. 케이스 구조를 선택하고 그것을 삭제합니다. 이 함수가 구조 안에 있지 않았기 때문에 그 함수를 삭제하지 않았다는 것을 확인하십시오.

구조를 다른 구조로 대체하기

다음 단계를 따라 구조 또는 루프에서 객체를 삭제하지 않고 구조 또는 루프를 다른 구조로 대체합니다.



노트

In Place 원소 구조나 이벤트 구조는 다른 구조로 대체할 수 없습니다.

1. 대체하려고 하는 구조 또는 루프에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 구조 또는 루프의 이름으로 대체를 선택합니다.

구조에서 객체를 삭제하지 않고 구조를 삭제하기

다음 단계를 따라 구조에서 객체를 삭제하지 않고 구조를 삭제합니다.

1. 삭제하려는 구조에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 구조 이름 제거를 선택합니다.

For 루프와 While 루프에서, LabVIEW는 루프의 내용을 블록다이어그램에 복사하고 터널을 통하여 연결된 원소를 자동으로 연결합니다.

여러 케이스를 가진 구조를 제거하는 경우, 보이는 케이스, 프레임 또는 서브다이어그램과 블록다이어그램 터미널만 보존됩니다. 모든 다른 케이스, 프레임이나 서브다이어그램은 삭제됩니다.

루프

루프에서 나가는 배열을 자동으로 연결하기

루프에서 나가는 배열이나 문자열을 자동으로 연결하려면, 배열이나 문자열을 루프 출력 터널에 직접 연결하고, 터널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 터널 모드 >> 연결하기를 선택합니다. 연결하기를 선택하면 모든 입력이 순서대로 추가되어, 연결된 배열 입력과 동일한 차원의 출력 배열이 생기거나, 한 개로 연결된 출력 문자열이 생깁니다. 이 외에 마지막 값과 인덱싱의 터널 모드가 있습니다. 마지막 값은 마지막 루프 반복의 마지막 값을 보여줍니다. 인덱싱은 더 높은 차원의 배열을 만듭니다. 조건에 따라 LabVIEW가 언제 루프 출력 터널에 값을 쓸지를 변경할 수 있습니다.

자동으로 배열을 연결하는 예제는 labview\examples\Structures\Shift Registers and Tunnels 디렉토리에 있는 Loop Tunnel Modes VI를 참조하십시오.

루프를 사용하여 배열이나 집합의 개별 원소 처리하기

배열 또는 집합의 개별 원소를 루프 안팎으로 전달하도록 루프를 설정할 수 있습니다.

배열 또는 집합의 개별 원소를 처리하도록 입력 터널 설정하기

배열 또는 집합을 For 루프나 While 루프에 연결하는 경우, 오토인덱싱을 활성화하면 배열 또는 집합의 모든 원소를 읽고 처리할 수 있습니다. 업스트림 노드의 배열 또는 집합을 루프 경계의 오토인덱싱 입력 터널로 연결하고 입력 터널의 오토인덱싱을 활성화하면, 해당 배열 또는 집합의 원소는 첫 번째 원소로 시작하여 한 번에 하나씩 루프에 들어갑니다.

다음 단계에 따라 루프를 통해 배열에 원소를 전달합니다:

1. 루프 경계에 있는 터널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 배열 원소를 처리하려는 방법에 해당하는 옵션을 선택합니다:
 - 바로 가기 메뉴에서 인덱싱 비활성화를 선택하여 오토인덱싱을 비활성화하면, 배열 또는 집합 전체가 루프로 전달되고 같은 개수의 원소로 구성된 배열이 반환됩니다.
 - 바로 가기 메뉴에서 인덱싱 활성화를 선택하여 오토인덱싱을 활성화하면, 배열 또는 집합의 개별 원소가 각각 루프로 전달되고, 단일 원소가 반환됩니다.



노트

루프가 실행되지 않을 때 기본 데이터 출력값을 얻지 않으려면, For 루프에 출력 터널이 생길 때마다 오토인덱싱을 활성화하십시오.

루프 출력 터널에 괄호 모양이 있으면 오토인덱싱이 활성화된 것입니다.

배열의 대체값을 반환하도록 출력 터널 설정하기

다음 단계에 따라 루프의 출력 터널을 설정하여, 배열의 인덱스 값, 배열의 마지막 값 또는 연결된 배열을 반환하도록 합니다.

1. 출력 터널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 각각 터널 모드 » 마지막 값, 인덱싱 또는 연결하기를 선택합니다.

터널이 반환할 수 있는 각 출력 옵션에 대한 더 자세한 정보는 다음 테이블을 참조하십시오.

출력 이름	설명
인덱스 하기	출력 배열은 루프가 반복될 때마다 새 원소를 수신합니다. 그러므로, 오토인덱싱된 출력 배열의 크기는 항상 반복 횟수와 같은 크기입니다. 예를 들어, 루프가 10회 실행되면 출력 배열은 10개의 원소를 가집니다.
마지막 값	출력 터널은 루프의 마지막 반복의 원소만을 반환합니다.
연결하기	LabVIEW는 모든 입력을 순서대로 추가하여, 연결된 배열과 같은 차원의 출력 배열을 생성합니다. 연결하기 터널 모드는 [배열 만들기] 함수와 같이 배열을 연결합니다.

출력 터널과 다음 노드 사이에서 와이어의 굵기는 루프가 값을 인덱싱하는지, 마지막 값을 반환하는지, 또는 값을 연결하는지 여부를 나타냅니다. 인덱싱 모드의 와이어는 같은 차원의 배열 대신 더 높은 차원의 배열을 포함하고 있기 때문에 연결하기 모드보다 더 굵습니다. 출력 터널이 마지막 값을 반환하는 경우, 와이어의 굵기는 인덱싱 또는 연결 시보다 더 가늘어집니다.

	노트 루프는 1차원 배열에서 스칼라를 인덱싱하고, 2차원 배열에서 1차원 배열을 인덱싱합니다. 출력 터널에 대해서는 반대로 적용됩니다. 스칼라 원소는 1차원 배열로 순차적으로 쌓이고, 1차원 배열은 2차원 배열로 쌓입니다.

오토인덱싱을 비활성화하여 배열을 만드는 예제는 labview\examples\Arrays\Arrays.lvproj를 참조하십시오.

 예제 열기  관련 예제 찾기

루프에 연결된 배열 또는 집합의 오토인덱싱 타입 선택하기

루프 타입	배열 또는 집합 설정	오토인덱싱 동작	예제
For 루프	입력 터널에 한 개의 배열 또는 집합을 연결하고 카운트 터미널은 연결하지 않음	For 루프의 입력 터널에 연결된 배열 또는 집합에서 오토인덱싱을 활성화하면, LabVIEW가 배열 또는 집합 크기를 실행 횟수로 사용하기 때문에 카운트 터미널을 연결할 필요가 없습니다. 기본적으로 LabVIEW는 For 루프에 연결되는 모든 배열 또는 집합에 대해 오토인덱싱을 활성화합니다.	—
	노트 배열 또는 집합을 한 번에 한 개의 원소씩 처리할 필요가 없다면 오토인덱싱을 비활성화해도 좋습니다.		
For 루프	연결된 카운트 터미널	카운트 터미널이 연결된 상태에서 오토인덱싱을 활성화하면, LabVIEW가 실행되는 실제 반복 횟수는 배열 또는 집합의 크기나 카운트 터미널의 값 중에서 더 작은 쪽이 됩니다.	예를 들어, For 루프에 10개의 원소가 있는 배열 또는 집합을 연결하고 카운트 터미널에 15의 값을 연결하면, 루프는 10번만 실행됩니다.

루프 타입	배열 또는 집합 설정	오토인덱싱 동작	예제
For 루프	여러 배열 또는 집합	두 개 이상의 터널에 대해 오토인덱싱을 활성화하면, LabVIEW가 실행되는 반복 횟수는 배열 또는 집합의 크기 중에서 더 작은 쪽이 됩니다.	예를 들어, 각각 10개와 20개의 원소를 가진 2개의 오토인덱싱된 배열을 루프에 입력하고, 카운트 터미널에 값 15를 연결하면, 루프는 10번만 실행되고, 첫 번째 배열의 모든 원소와 두 번째 배열의 처음 10개 원소까지만 인덱싱됩니다. 다른 예제에서, 2개의 소스에서 나온 데이터 중에서 처음 100개 원소를 그래프에 플롯하고자 한다면, 카운트 터미널에 100을 연결하십시오. 그러나 더 작은 데이터 소스에 50개의 원소 밖에 없다면 루프는 50회 실행되고 각 데이터 소스의 처음 50개 원소까지만 인덱싱합니다. [배열 크기] 함수를 사용하여 배열의 크기를 확인하십시오.

루프 타입	배열 또는 집합 설정	오토인덱싱 동작	예제
While 루프	연결된 한 개의 배열 또는 집합	While 루프로 들어가는 배열 또는 집합에 대해 오토인덱싱을 활성화하면, While 루프는 For 루프와 같은 방법으로 배열 또는 집합을 인덱싱합니다. 하지만 While 루프의 반복 횟수는 배열 또는 집합의 크기로 제한되지 않습니다. 특정 조건이 발생할 때까지 While 루프가 반복해서 실행되기 때문입니다. 루프의 반복 횟수가 배열 또는 집합의 크기보다 크면, While 루프는 배열 또는 집합의 원소를 모두 처리한 후, 다음 반복에서 배열 또는 집합 원소 타입의 기본값을 사용합니다. [배열 크기] 또는 [집합 크기] 함수를 사용하여 기본값이 While 루프에 전달되지 않도록 할 수 있습니다. 이 함수는 배열 또는 집합의 원소 개수를 나타냅니다. 반복 횟수가 배열 또는 집합의 크기와 같으면 While 루프의 실행을 멈추도록 설정합니다. While 루프에서 오토인덱싱은 기본적으로 비활성화되어 있습니다.	—
	팁 출력 배열 또는 집합의 크기를 미리 결정할 수 없기 때문에 For 루프 출력의 오토인덱싱을 활성화하는 것이 While 루프의 경우보다 더 효율적입니다. 반복을 너무 많이 하면 시스템 메모리가 부족할 수 있습니다.		

관련 정보

배열

집합을 이용한 데이터의 그룹화

For 루프: 지정된 횟수만큼 작업 반복하기

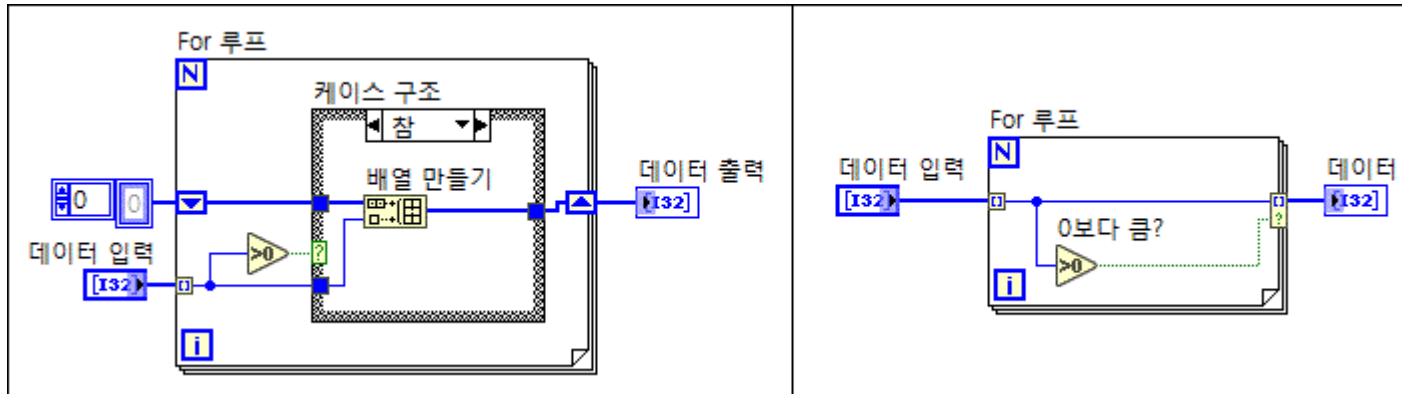
While 루프: 조건이 발생할 때까지 작업 반복하기

루프에서 나가는 배열을 자동으로 연결하기

루프 출력 터널에 조건적으로 값 연결하기

루프 출력 터널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 모드에서 터널 모드 >> 조건적을 선택하여, LabVIEW가 사용자가 지정한 조건에 근거하여 루프 출력 터널에 쓸 값을 결정하도록 설정할 수 있습니다.

조건적 메뉴 항목을 사용하여 블록 다이어그램을 단순화할 수 있습니다. 예를 들어, 다음 블록 다이어그램은 오른쪽 다이어그램의 조건적 출력 터널이 왼쪽 다이어그램의 케이스 구조와 동일한 목표를 달성하는 것을 보여줍니다.



모든 터널 모드에서 조건적 메뉴 항목을 선택할 수 있습니다. 다음 테이블은 각 터널 모드에서 조건적 메뉴 항목을 선택할 때 미치는 영향을 보여줍니다.

터널 모드	결과
마지막 값	마지막 반복에서의 마지막 값을 터널에 조건적으로 저장합니다. 마지막 값을 읽는 것은 피드백 노드 에 마지막으로 어떤 값이 쓰였는지 알고 싶을 때 유용합니다.
인덱스하기	배열을 조건적으로 배열로 인덱스합니다.
연결하기	모든 입력을 순서대로 추가하여 연결된 배열과 같은 차원의 출력 배열을 생성합니다.

조건에 따라 루프 출력 터널에 값 쓰기와 관련한 예제는 labview\examples\Structures\Shift Registers and Tunnels 디렉토리에 있는 Loop Tunnel Modes VI를 참조하십시오.

각 반복에서 불리언 값 확인하기

루프가 매 반복마다 터미널의 값을 검사하기를 원하면, 루프 안에 터미널을 놓습니다.

다음 단계에 따라 매 반복마다 루프가 불리언 터미널 값을 확인하도록 합니다.

1. 불리언 컨트롤을 프런트패널에 놓습니다.

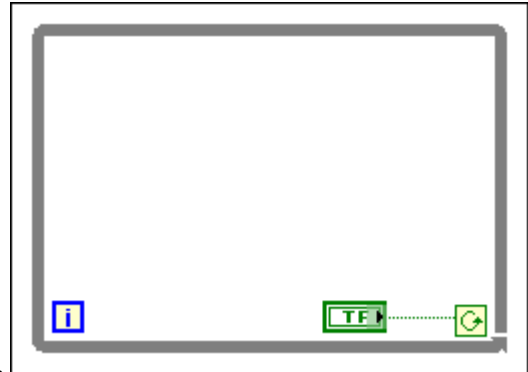
2. 블록다이어그램에서 불리언 터미널을 루프 안에 놓습니다.
3. 불리언 터미널을 루프의 조건 터미널에 연결합니다.



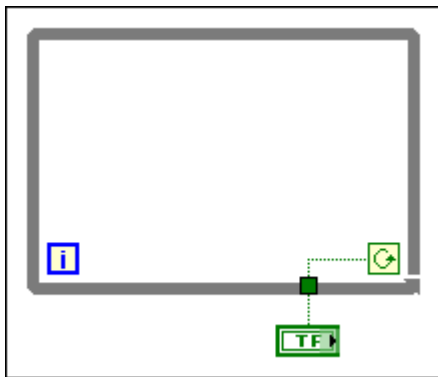
노트

다음 블록다이어그램에서처럼 프론트패널 컨트롤 값을 거짓으로 변경하

면 While 루프를 멈출 수 있습니다.



참인 경우 계속으로 설정된 While 루프의 조건 터미널을 참으로 설정되고 루프 밖에 위치한 불리언 터미널에 연결하면, 다음 블록다이어그램과 같은 무한 루프가 발생합니다.



또한 참인 경우 정지로 설정된 While 루프의 조건 터미널을 거짓으로 설정되고 루프 밖에 위치한 불리언 터미널에 연결해도 무한 루프가 발생할 수 있습니다.

무한 루프를 멈추기 위해, 도구 모음에서 강제 종료 버튼을 클릭하여 VI를 강제 종료해야 합니다. 컨트롤 값을 변경하면 무한 루프를 멈추게 되는데 왜냐하면 루프가 멈추고 VI를 다시 실행할 때까지 그 값이 전달되기 때문입니다.

루프의 연속 동작 변경하기

While 루프, For 루프 또는 (Real-Time, Windows) Timed 루프의 조건 터미널이 참인 경우 정지(●)이면 루프는 조건 터미널이 참 값을 받을 때까지 서브다이어그램을 실행합니다. 조건 터미널이 참인 경우 계속(●)이면, 루프는 조건 터미널이 거짓 값을 받을 때까지 서브다이어그램을 실행합니다.



노트

조건 터미널이 있는 For 루프는 조건 터미널이 정지값을 받거나 설정한 횟수만큼 반복될 때까지(두 경우 중 먼저 발생하는 것을 기준으로) 실행됩니다.

다음 단계를 따라 조건 터미널이 있는 루프의 연속 동작을 변경합니다.

1. 조건 터미널에서 마우스 오른쪽 버튼을 클릭합니다. 또한 While 루프나 Timed 루프의 루프 경계에서 마우스 오른쪽 버튼을 클릭할 수 있습니다.
2. 바로 가기 메뉴에서 참인 경우 정지 또는 참인 경우 계속을 선택합니다.

또한 수행 도구를 사용하여 조건 터미널을 클릭해서 조건을 변경할 수 있습니다.

불리언 값이나 에러 클러스터를 조건 터미널이 있는 While 루프나 For 루프의 조건 터미널에 연결해야 합니다.

루프의 타이밍 컨트롤하기

기본으로 LabVIEW는 루프와 블록다이어그램을 최대한 빨리 실행합니다. 그러나, 예를 들어 데이터 값을 차트에 플롯하는 속도와 같은, 프로세스 실행 속도를 컨트롤해야 할 경우도 있습니다.

다음 중 한 가지 방법으로 루프의 타이밍을 컨트롤할 수 있습니다:

- [기다림] 함수를 사용하여, 루프가 다시 실행되기 전에 기다릴 시간을 밀리초 단위로 지정할 수 있습니다.
- (Real-Time, Windows) **Timed 루프**를 사용하여 시간 제한과 지연이 있는 코드를 실행합니다.

루프의 조건 터미널로 에러 핸들링하기

에러가 조건 터미널로 전달되면 While 루프나 For 루프의 반복을 컨트롤할 수 있습니다. 다음 단계에 따라 조건 터미널을 사용하여 에러를 핸들링합니다.

1. 에러 클러스터 터미널을 선택합니다.
2. While 루프나 For 루프의 조건 터미널에 에러 클러스터를 와이어로 연결합니다.



노트

에러 클러스터를 조건 터미널에 연결하면 에러 클러스터 상태 파라미터의 참(TRUE) 또는 거짓(FALSE) 값만 터미널에 전달됩니다. 그리고 바로 가기 메뉴 아이콘도 참인 경우 정지, 참인 경우 계속에서 에러 발생 시 정지, 에러 발생 시 계속으로 변경됩니다.

For 루프의 작업 실행 횟수 지정하기

For 루프는 카운트 터미널에 연결한 횟수만큼 내부의 서브다이어그램을 반복적으로 실행합니다.

다음 단계를 따라 For 루프로 작업이 실행되는 횟수를 지정합니다.

1. 블록다이어그램에 For 루프를 추가합니다.
2. **For 루프 내부에 객체를 추가**하고, For 루프가 반복하는 서브다이어그램을 생성합니다.
3. 다음 중 한 가지 방법으로 For Loop의 반복 횟수를 결정합니다:
 - 숫자값을 카운트 터미널 에 연결합니다.
 - 배열을 **오토인덱싱**하여 루프 경계의 입력 터널에 배열을 연결합니다.



노트

카운트 터미널에 숫자값이 연결되지 않았거나 배열의 크기가 반복 카운트를 결정하는 오토인덱싱 입력 터널이 없는 vi는 실행되지 않습니다.

For 루프에 **조건 터미널을 추가**하여 불리언 조건이 발생하거나 에러가 발생할 때 루프가 정지하도록 설정할 수 있습니다. 조건 터미널이 있는 For 루프는 정의한 조건이 발생하거나 또는 반복이 모두 완료할 때까지(두 경우 중 먼저 발생하는 것을 기준으로) 실행됩니다.

조건이 충족될 때 For 루프 정지하기

조건 터미널을 추가하여 조건이 발생할 때 **For 루프**가 정지하도록 설정할 수 있습니다. 조건 터미널이 있는 For 루프는 조건이 발생하거나 또는 반복이 모두 완료할 때까지 중 먼저 발생하는 경우에 실행됩니다. 예를 들어 에러가 발생하지 않는 이상 일정한 횟수만큼 루프가 실행되도록 하려면 조건 터미널이 있는 For 루프를 사용한 후 에러 클러스터를 조건 터미널에 연결합니다.

다음 단계에 따라 조건이 발생하면 For 루프를 정지합니다.

1. 블록다이어그램에 For 루프를 추가합니다.
2. 루프 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 조건 터미널을 선택합니다. 조건 터미널 이 루프 안에 나타나며 카운트 터미널 모양이  기호에서  기호로 바뀝니다.
3. **For 루프 내부에 객체를 추가**하고, For 루프가 반복하는 서브다이어그램을 생성합니다.
4. 불리언 컨트롤이나 [비교] 함수에서의 출력과 같은 불리언 데이터 또는 에러 클러스터를 조건 터미널에 연결합니다. 또한 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 상수 생성 또는 컨트롤 생성을 선택하여 불리언 객체를 생성할 수 있습니다.
5. (옵션) 기본적으로, 조건 터미널은 참인 경우 정지로 설정되어 있습니다. 터미널에서 마우스 오른쪽 버튼을 클릭하고 참인 경우 계속을 선택하여 조건 터미널의 **동작을 변경**할 수 있습니다. 조건 터미널이 참인 경우 계속 일 때, 루프는 터미널

이 거짓값을 받을 때까지 실행됩니다. 또한 수행 도구를 사용하여 조건 터미널을 클릭해서 조건을 변경할 수 있습니다.



노트

For 루프 내부의 조건 터미널을 연결하는 경우, 루프 실행이 반복되는 횟수를 미리 지정할 수 없습니다. **출력 배열을 오토인덱싱**하고 루프가 실행된 후 원소 개수를 확인하거나, 인디케이터를 루프 반복 터미널에 연결하고 루프가 실행된 후 카운트를 확인하여, 루프가 실행 완료한 **반복 횟수를 확인**할 수 있습니다.

조건 터미널에 연결된 조건이 발생하지 않는 경우에만, 카운트 터미널에 연결된 숫자 또는 오토인덱싱된 입력 배열의 원소 개수가 최대 루프 반복 횟수로 쓰입니다. 최대 반복 횟수 없이 조건이 발생할 때까지 루프가 반복되도록 하려면 While 루프를 대신 사용하십시오.

병렬 반복: For 루프 실행 속도 향상

병렬 반복을 활성화하여 For 루프의 실행 속도를 향상할 수 있습니다. LabVIEW는 일반적으로 For 루프 반복을 여러 그룹으로 나눈 후, 순서대로 이러한 그룹을 실행합니다. For 루프에서 병렬 반복을 활성화하면, LabVIEW는 이러한 그룹을 동시에 실행하여 계산 집약적인 For 루프의 실행 속도가 향상됩니다.

다음 단계를 완료하여 For 루프의 병렬 반복을 실행합니다:

1. 병렬 For 루프 반복 활성화하기.
2. For 루프가 병렬 반복으로 실행될 수 있는지 확인하기.
3. 필요한 경우, 프로그램적으로 병렬 For 루프 반복 설정하기.

성능 고려사항을 검토하여 병렬 반복 실행이 For 루프에 이득이 될지 확인합니다.

예제

병렬 반복이 있는 For 루프를 사용하는 것에 대한 예제는 labview\examples\Structures\Parallel For Loop 디렉토리에서 Parallel For Loop Reduction VI를 참조하십시오.



예제 열기



관련 예제 찾기



관련 정보

[병렬 For 루프 반복 활성화하기](#) (1228 페이지 참조)

[For 루프가 병렬 반복으로 실행될 수 있는지 확인하기](#) (1228 페이지 참조)

[병렬 For 루프 반복을 설정하여 성능 변경하기](#) (1229 페이지 참조)

[병렬 For 루프 반복의 성능 고려사항](#) (1230 페이지 참조)

병렬 For 루프 반복 활성화하기

병렬 For 루프 반복을 활성화하여 For 루프 실행 속도를 향상할 수 있습니다. 다음 단계를 완료하여 병렬 반복을 활성화합니다:

1. For 루프에서 마우스 오른쪽 버튼을 클릭하고 반복 병렬 처리 설정을 선택합니다. LabVIEW에 For 루프 반복 병렬 처리 대화 상자가 나타납니다.
2. 루프 반복 병렬 처리 활성화 확인란에 확인 표시를 합니다.
3. 생성된 병렬 루프 인스턴스 개수를 이 VI를 실행하게 될 컴퓨터가 가질 수 있는 논리적 프로세서의 최대 개수와 같도록 설정합니다. 예를 들어, 최대 8 개의 프로세서가 있는 컴퓨터에서 사용할 어플리케이션을 개발하는 경우, 이 숫자를 8로 설정합니다.
4. 확인 버튼을 클릭합니다. 병렬 인스턴스 P 터미널은 설정된 For 루프의 카운트 N 터미널 아래에 나타납니다. 병렬 반복을 활성화하기 위해 병렬 인스턴스 터미널에 값을 연결할 필요가 없습니다. 그러나, 이 터미널을 사용하여 병렬 For 루프의 성능을 변경할 수 있습니다.

병렬 반복을 활성화한 후, For 루프가 병렬 반복으로 실행될 수 있는지 **확인**하십시오.

For 루프가 병렬 반복으로 실행될 수 있는지 확인하기

일부 For 루프는 병렬 반복 실행할 수 없습니다. For 루프에서 병렬 반복을 활성화한 후, 다음 단계를 완료하여 For 루프가 병렬 반복 실행될 수 있는지 확인합니다.

1. 환경 셋팅이 경고를 디스플레이하도록 설정되어 있는지 확인합니다.
2. 보기>>에러 리스트를 선택하여 에러 리스트 윈도우를 엽니다.
3. 에러와 경고 섹션을 검토합니다. 다음 에러 또는 경고가 나타나지 않는 경우, For 루프는 병렬 반복 실행될 수 있습니다.
 - 에러: 병렬 For 루프에는 루프 반복 사이의 배열 의존성이 포함될 수 없습니다
 - 에러: 병렬 For 루프에 루프 반복 사이에 의존성이 있을 수 없습니다
 - 에러: 병렬 For 루프에 다층 시프트 레지스터가 있을 수 없습니다
 - 에러: 병렬 For 루프에 초기화되지 않은 시프트 레지스터가 있을 수 없습니다
 - 에러: 병렬 For 루프에 조건 터미널이 있을 수 없습니다
 - 에러: 병렬 For 루프에 피드백 노드가 있을 수 없습니다
 - 에러: 병렬 For 루프에 래치 옵션을 가진 불리언 컨트롤이 있을 수 없습니다
 - 에러: 이 기능이 타겟에서 지원되지 않습니다
 - 경고: 부수적인 결과를 발생하는 노드
4. 병렬 반복을 활성화할 때 에러 또는 경고가 발생하는 경우, 병렬 반복을 비활성화하거나 코드를 수정합니다. 언제 병렬 루프 반복을 사용해야 하는지에 대한 더 자세한 정보는 ni.com의 다음 지원 문서를 참조하십시오.

경고를 발생하는 병렬 반복 For 루프의 예제는 labview\examples\Structures\Parallel For Loop 디렉토리에 있는 Parallel For Loop Iteration Order VI를 참조하십시오.

 예제 열기  관련 예제 찾기



노트

프로젝트 또는 SubVI를 포함하는 VI와 같이 여러 VI가 있는 경우, 병렬 가능한 루프 찾기 결과 윈도우를 사용하여 병렬 반복 실행할 수 있는 For 루프를 찾습니다.



관련 정보

[병렬 For 루프 반복 활성화하기](#) (1228 페이지 참조)

[경고 디스플레이하기](#) (908 페이지 참조)

병렬 For 루프 반복을 설정하여 성능 변경하기

병렬 For 루프 반복을 활성화하는 경우, 병렬 인스턴스 P와 그룹 크기 C 터미널을 사용하여 프로그램식으로 성능을 변경할 수 있습니다. 터미널의 기본 설정은 대부분의 경우 최적의 성능 향상 결과를 낳기 때문에, 프로그램적인 설정은 극히 일부의 경우에만 필요합니다. 그러나, 병렬 인스턴스와 그룹 크기 터미널을 사용하여 기본 설정과 다르게 설정을 사용자 정의할 수 있습니다.

병렬 인스턴스 터미널을 사용하여 성능 변경하기

병렬 For 루프 반복을 활성화하면, 병렬 인스턴스 터미널이 For 루프 안에 나타납니다. 숫자값을 병렬 인스턴스 터미널에 연결하여 병렬 인스턴스 실행 횟수를 프로그램적으로 설정합니다. 컴파일 할 때, LabVIEW는 For 루프 반복 병렬 처리 대화 상자의 생성된 병렬 루프 인스턴스 개수 필드에 연결한 최소값과 같은 병렬 인스턴스 개수를 생성합니다. 프로세서는 병렬 인스턴스를 동시에 실행하여 성능을 향상합니다.

병렬 인스턴스 터미널의 입력에 값을 연결하지 않으면, LabVIEW는 자동으로 컴퓨터의 논리적 프로세서의 수를 감지하여, 이 수를 병렬 인스턴스 터미널 기본값으로 사용합니다. 대부분의 경우 병렬 인스턴스 실행 횟수가 컴퓨터의 프로세서 수와 같을 때 성능이 최적화되기 때문에, 병렬 인스턴스 터미널의 입력을 연결하지 않지 말아야 합니다.

For 루프에 기다림 작업을 수행하는 코드가 있을 때, 루프 인스턴스 실행 횟수가 논리적 프로세서 수보다 큰 경우에 성능이 최적화됩니다. 예를 들어, 병렬 For 루프 인스턴스가 외부 하드웨어로부터 데이터를 수집하기 위해 기다릴 때, 초과 구독하여 프로세서가 첫 번째 병렬 인스턴스를 기다리는 동안 두 번째 병렬 인스턴스를 실행할 수 있습니다. 병렬 For 루프 인스턴스가 다른 계산 집약적인 코드와 동시에 실행되는 경우, 실행하는 병렬 인스턴스 개수가 논리적 프로세서의 개수보다 적은 경우, 성능이 최적화됩니다. 예를 들어 For 루프와 SubVI가 동시에 실행되는 경우, 병렬 인스턴스 개수가 논리적 프로세서 개수보다 적으면 For 루프에 할당되는 리소스가 제한되고, 다른 작업에 사용하기 위해 리소스가 예약됩니다.

그룹 크기 터미널을 사용하여 반복 스케줄 사용자 지정하기

LabVIEW는 루프 반복을 루프 반복으로 구성되는 그룹으로 나눕니다. 병렬 반복이 활성화되면, 프로세서는 이러한 그룹을 동시에 실행하여 실행 속도를 향상시킵니다. LabVIEW는 크기가 큰 그룹부터 작은 그룹의 순서로 실행을 스케줄하도록 기본 설정되어 있습니다. 큰 그룹을 먼저 실행하면 스케줄 오버헤드가 감소하고, 작은 그룹을 먼저 실행하면 프로세서 유휴 시간이 감소합니다. 작은 그룹을 큰 그룹보다 먼저 실행하도록 스케줄하는 것과 같이 기본 설정과 다른 반복 스케줄을 사용하는 것이 For 루프에 유용한 경우에만 프로그램적으로 그룹 크기를 설정해야 합니다.

병렬 For 루프 반복을 활성화한 후, 다음 단계를 따라 그룹 크기 터미널을 사용해서 프로그램적으로 반복 스케줄을 설정합니다:

1. For 루프에서 마우스 오른쪽 버튼을 클릭하고 반복 병렬 처리 설정을 선택합니다. LabVIEW에 For 루프 반복 병렬 처리 대화 상자가 나타납니다.
2. 반복 분할 스케줄 섹션에서 그룹 크기(C) 터미널로 분할 지정을 선택합니다. 그룹 크기 터미널은 병렬 인스턴스 터미널 아래에 나타납니다.
3. 숫자값 또는 숫자값 배열을 그룹 크기 터미널에 연결합니다.

숫자값을 그룹 크기 터미널에 연결하면, 이 값이 각 그룹에 포함할 반복의 횟수를 지정합니다. 숫자값 배열을 연결하면 그룹 크기를 보다 정밀하게 컨트롤할 수 있습니다. 배열의 각 값은 각 그룹에 포함할 반복 횟수를 지정하고, 이때 첫번째 그룹은 인덱스 0에서 시작합니다.



노트

너무 많은 그룹 크기를 가진 배열을 연결하는 경우, LabVIEW는 남은 값을 무시합니다. 너무 적은 그룹 크기를 연결하는 경우, 배열의 마지막 원소를 사용하여 남은 그룹의 크기를 결정합니다.

병렬 For 루프 반복의 성능 고려사항

일부 경우, For 루프에서 **병렬 반복을 활성화**해도 성능 향상에 큰 효과가 없을 수도 있습니다. 다음 가이드라인을 따라 병렬 반복 실행되는 For 루프의 성능 장점을 최적화합니다:

- 병렬 루프 반복을 활성화하면 실행 오버헤드가 생깁니다. 계산 집약적이지 않은 For 루프에서 병렬 루프 반복을 활성화하지 마십시오. 실행 시간이 줄어들 경우에만 오버헤드를 상쇄하여 성능을 높일 수 있습니다.
- 중첩된 For 루프를 사용하는 경우, 바깥 쪽 루프에서만 병렬 반복을 활성화합니다.
- [텍스트 파일에 쓰기] 함수 및 재호출이 아닌 SubVI 호출과 같이 **차단된 노드**로 인해 루프 반복 중 병렬 처리가 제한될 수 있습니다. 병렬 반복 실행되는 For 루프 안에서는 차단된 노드를 사용하지 않도록 합니다.

병렬 루프 반복으로 성능을 개선하는 것에 관한 자세한 정보는 ni.com의 기술 백서를 참조하십시오.

While 루프의 조건 지정하기

While 루프에는 조건 터미널이 특정 불리언 값을 받을 때까지 실행되는 서브다이어그램이 있습니다.

다음 단계를 따라 While 루프의 조건을 지정합니다.

1. 블록다이어그램에 While 루프를 추가합니다.
2. **While 루프 안에 객체를 추가**하여 While 루프가 반복할 서브다이어그램을 생성합니다.
3. 불리언 컨트롤과 같은 불리언 데이터나 [비교] 함수에서의 출력을 조건 터미널 에 연결합니다. 또한 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 상수 생성 또는 컨트롤 생성을 선택하여 불리언 객체를 생성할 수 있습니다.
4. (옵션) 기본적으로, 조건 터미널은 참인 경우 정지로 설정되어 있습니다. 터미널에서 마우스 오른쪽 버튼을 클릭하고 참인 경우 계속을 선택하여 조건 터미널의 **동작을 변경**할 수 있습니다.



노트

조건을 지정하지 않는 경우, While 루프는 무한 루프가 됩니다. 루프가 시작되기 전에 한 번만 값을 읽기 때문에 컨트롤의 값을 바꾸어도 무한 루프를 멈출 수 없습니다. 무한 루프를 멈추려면, 도구 모음의 실행 강제 종료 버튼을 클릭하여 VI를 강제 종료해야 합니다.

VI는 각 반복의 끝에 있는 조건 터미널을 확인하기 때문에, While 루프는 항상 최소한 한 번 실행합니다. 조건 터미널이 연결되지 않은 경우 VI를 실행할 수 없습니다.

첫 번째 반복에서 **코드 실행을 방지**하려면 루프 안에 케이스 구조를 놓습니다.

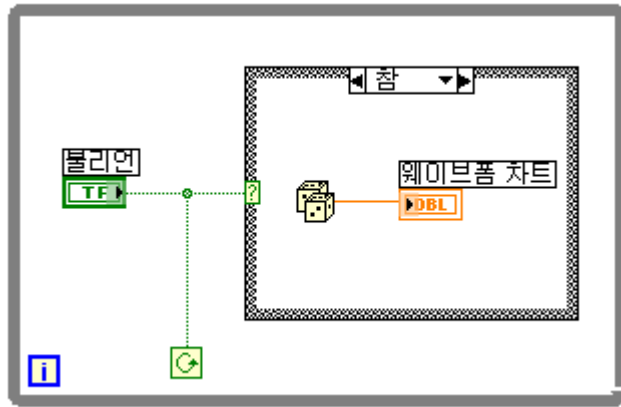
처음 While 루프 반복에서 코드 실행 막기

VI가 반복을 마칠 때마다 조건 터미널을 확인하기 때문에, **While 루프**는 항상 최소한 한 번은 실행됩니다.

조건을 테스트해서 조건이 처음 반복에서 거짓이면 코드를 실행하지 않는 While 루프를 다음 단계에 따라 생성합니다.

1. 블록다이어그램에 While 루프를 추가하고 사각형 영역을 끌어서 늘립니다.
2. While 루프 안에 케이스 구조를 추가하고, While 루프 안에서 사각형 영역을 끌어서 늘립니다.
3. 불리언 컨트롤을 생성하여 VI를 컨트롤합니다.
 1. 케이스 구조의 선택자 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 컨트롤 생성을 선택하여 불리언 컨트롤을 생성합니다.
 2. 불리언 컨트롤을 While 루프의 조건 터미널에 연결합니다. 와이어 연결을 보기 좋게 하기 위해, 불리언 컨트롤 아래의 조건 터미널을 움직이고 기존의 와이어 부분으로부터 와이어 결가지를 생성합니다.

4. 조건 터미널을 참이면 계속으로 **동작 변경**합니다.
5. 참 케이스 안에 [난수(0-1)] 함수를 추가합니다.
6. 웨이브폼 차트를 프런트패널에 추가합니다.
7. 블록 다이어그램에서 난수(0-1) 함수를 웨이브폼 차트에 연결합니다. 함수와 차트 터미널이 선택한 케이스 안에 있다는 것을 확인합니다.



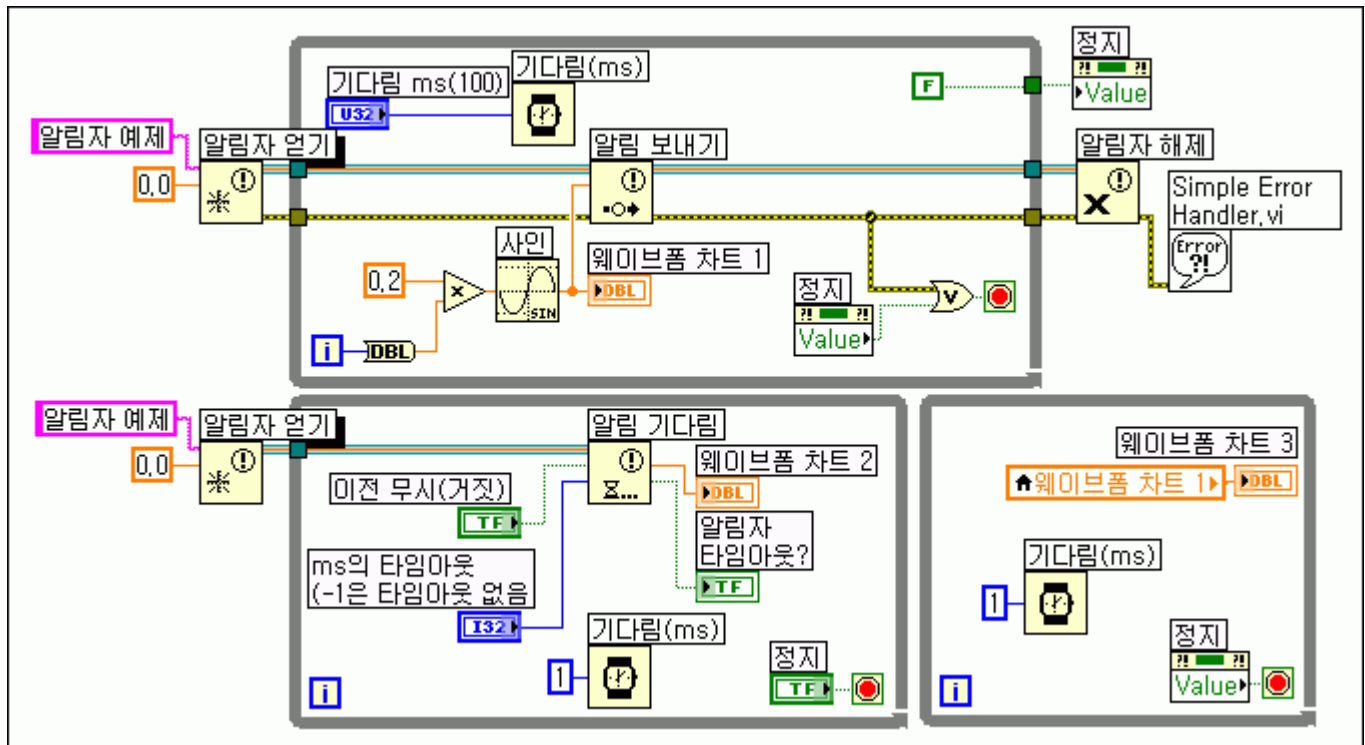
다음 단계를 따라 While 루프가 올바르게 작동하는지 확인하고 VI를 실행합니다.

1. 프론트패널을 살펴봅니다. 불리언 컨트롤의 기본값은 OFF입니다.
2. 실행 버튼을 여러번 클릭하여 v_i 를 실행합니다. 차트는 어떤 값도 디스플레이하지 않습니다. While 루프는 한번 실행하지만 그러나, 케이스 구조는 불리언 컨트롤이 OFF로 설정되어 있는 동안 실행하지 않습니다.
3. 수행 도구를 가지고, 불리언 버튼을 ON으로 클릭합니다.
4. 실행 버튼을 클릭합니다. While 루프가 계속 실행되는 동안, 난수가 생성되고 차트에 이것을 디스플레이합니다.
5. 불리언 버튼을 클릭하여 v_i 를 정지합니다.

예제: While 루프에서 알립 동작 함수 처리하기

다음 예제는 While 루프에서 알림 동작 함수와 정지 불리언 컨트롤을 사용할 때 많이 발생하는 실수를 보여줍니다.

두번째 루프에서 정지 불리언 컨트롤의 값이 참으로 변경되면, VI는 항상 정지하지 않고 사용자가 VI를 완전히 정지하기 위해 실행 강제 종료 버튼을 클릭해야 합니다. 이것은 루프의 각 함수가 그 마지막 실행을 끝낼 때까지 루프가 실행을 끝내지 못했고, 두번째 루프에서 [알림 기다림] 함수가 정지 조건에 연결되지 않았기 때문입니다.



다음 단계에 따라 이 문제를 해결하십시오.

1. [알림 기다림] 함수의 에러 출력 클러스터를 정지 버튼에 연결합니다.
2. [알림자 해제] 함수의 강제 삭제? 입력을 참으로 설정하여 루프 실행 완료 후 알림자를 삭제합니다.

다음 루프 반복으로 값 전달하기

For 루프, While 루프, 또는 (Real-Time, Windows) Timed 루프에서 시프트 레지스터를 사용하여 한 루프 반복에서 다음 루프 반복으로 값을 전달합니다.

다음 단계를 따라 시프트 레지스터를 생성하여 다음 루프 반복으로 값을 전달합니다.

1. 블록다이어그램에 구조를 추가합니다. 예를 들면, 블록다이어그램에 While 루프를 추가합니다.
2. 루프의 왼쪽 또는 오른쪽 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 시프트 레지스터 추가를 선택합니다.

여러 개의 시프트 레지스터를 사용하여 반복되는 값을 기억할 수 있으며 시프트 레지스터의 초기값 리셋을 사용하여 시프트 레지스터를 초기화할 수 있습니다.



노트
(FPGA Module) FPGA Module은 플립 플롭 그룹을 사용하여 시프트 레지스터를 구현합니다.



- 관련 정보**
- [시프트 레지스터: 루프 반복 사이에 값 전달하기](#) (370 페이지 참조)
 - [루프와 기타 구조](#) (364 페이지 참조)
 - [다음 루프 반복으로 여러 값 전달하기](#) (1234 페이지 참조)
 - [시프트 레지스터의 초기값 리셋하기](#) (1235 페이지 참조)

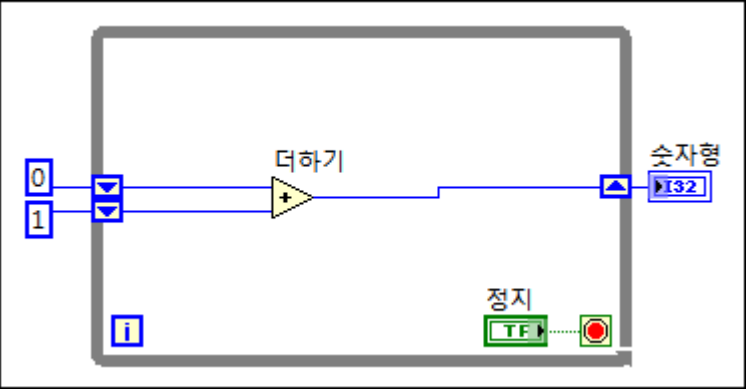
다음 루프 반복으로 여러 값 전달하기

루프의 왼쪽 편에 다중 시프트 레지스터 터미널을 생성하여 여러 개의 이전 반복을 저장하고 해당 값을 다음 반복으로 넘길 수 있습니다. 이 기법은 데이터 포인트 평균값 연산에 유용합니다.

다음 단계를 따라 **시프트 레지스터**를 설정하여 하나 이상의 이전값을 다음 반복으로 전달합니다.

1. **시프트 레지스터를 생성**합니다.
2. 왼쪽 또는 오른쪽 시프트 레지스터 터미널에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 원소 추가를 선택하여 루프의 왼쪽 편에 추가적인 터미널을 생성합니다.

다중 시프트 레지스터는 루프의 왼쪽에만 나타납니다. 왜냐하면 오른쪽 터미널은 다음 블록 다이어그램에서처럼 단지 현재 루프 반복에서 생성된 데이터를 다음 루프 반복에 전달하는 역할만 하기 때문입니다.



반복	마지막 값		
	왼쪽 위 시프트 레지스터	왼쪽 아래 시프트 레지스터	오른쪽 시프트 레지스터
0		0	1
1		1	0
2		1	1
3		2	1
4		3	2

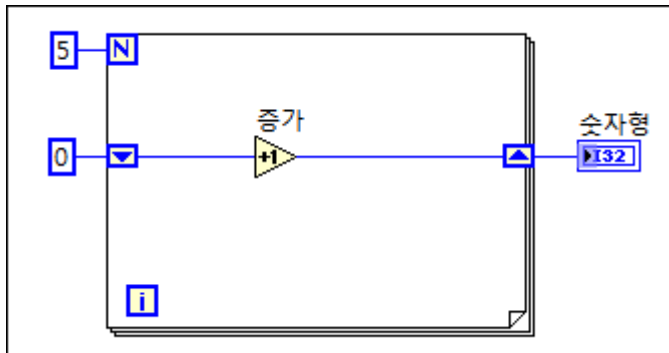
이전 블록 다이어그램의 왼쪽 터미널에 다른 원소를 하나 더 추가하면, 최근 두 번의 반복 값을 다음 반복으로 전달할 수 있으며, 가장 최근 반복 값은 가장 위쪽 시프트 레지스터에 저장됩니다. 아래쪽 터미널은 이전 반복에서 전달된 데이터를 저장합니다.

시프트 레지스터의 초기값 리셋하기

초기화된 시프트 레지스터를 사용하여 VI가 실행될 때마다 시프트 레지스터의 초기값을 리셋합니다.

다음 단계를 따라 **시프트 레지스터**의 초기값을 리셋합니다.

1. **시프트 레지스터를 생성**합니다.
2. 다음 블록 다이어그램에서처럼 루프 밖에서 왼쪽 터미널에 값을 연결합니다.



위의 블록 다이어그램에서, For 루프는 5번 실행되고 시프트 레지스터가 전달하는 값이 매번 하나씩 증가합니다. For 루프가 5번 반복한 후, 시프트 레지스터는 마지막 값인 5를 인디케이터에 전달하고 VI를 끝냅니다. 시프트 레지스터는 이 VI를 실행할 때마다 0의 값으로 시작하게 됩니다.

초기화하지 않은 시프트 레지스터가 있는 루프를 사용하여 VI를 실행할 때마다 시프트 레지스터의 초기 입력이 이전 실행의 마지막 값이 되도록 합니다. 초기화되지 않은 시프트

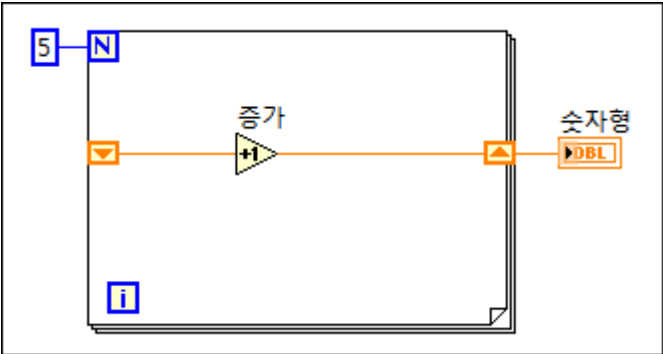
트 레지스터가 VI의 순차적인 실행 사이에서 상태를 보존하려고 연결되어 있지 않은 왼쪽 시프트 레지스터 터미널에 입력을 남겨둡니다.



노트

레지스터를 초기화하지 않으면, 루프가 마지막으로 실행되었을 때 레지스터에 기록된 값이나 루프가 실행된 적이 없으면 해당 데이터 타입의 **기본값**이 사용됩니다.

다음 블록다이어그램은 초기화되지 않은 시프트 레지스터를 보여줍니다.



이전 블록다이어그램에서, For 루프는 5번 실행되고 시프트 레지스터가 전달하는 값이 매번 하나씩 증가합니다. 처음 VI를 실행하면 시프트 레지스터는 32비트 정수의 기본값인 0부터 시작합니다.

For 루프가 5번 반복한 후, 시프트 레지스터는 마지막 값인 5를 인디케이터에 전달하고 VI를 끝냅니다. 다음에 VI를 실행하면, 시프트 레지스터는 이전 실행에서 마지막 값이었던 5를 초기값으로 시작합니다. For 루프가 5번 반복하면, 시프트 레지스터는 마지막 값인 10을 인디케이터에 전달합니다.

VI를 다시 실행하면 시프트 레지스터는 10의 값으로 시작합니다. 초기화되지 않은 시프트 레지스터는 VI를 닫을 때까지 이전 반복 값을 유지하거나 **LabVIEW가 블록다이어그램을 다시 컴파일합니다.**

시프트 레지스터를 터널로 대체하기

터널이란 구조 안팎으로 데이터를 주고받는 터미널입니다. 한 루프 반복에서 다음 루프 반복으로 값을 전달할 필요가 없을 때 시프트 레지스터를 터널로 대체합니다.

다음 단계를 따라 시프트 레지스터를 터널로 대체합니다:

1. 시프트 레지스터에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 터널로 대체를 선택합니다.

터널을 시프트 레지스터로 대체하기

한 루프 반복에서의 값을 다음 반복으로 전달해야 할 때 터널을 시프트 레지스터로 대체합니다.

다음 단계를 따라 터널을 시프트 레지스터로 대체합니다:

1. 터널에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 시프트 레지스터로 대체를 선택합니다.

마우스의 오른쪽 버튼을 클릭한 터널의 반대쪽 루프 경계에 대응되는 터널이 없을 경우, LabVIEW는 자동으로 대응되는 한 쌍의 시프트 레지스터 터미널을 생성합니다. 다음과 같이 마우스 오른쪽 버튼을 클릭한 터널의 루프 경계 반대쪽에 터널이 있는 경우, LabVIEW는 마우스 오른쪽 버튼을 클릭한 터널을 시프트 레지스터 터미널로 대체하고, 커서는 시프트 레지스터 아이콘이 됩니다.



커서가 시프트 레지스터 아이콘이 되면, 다음 중 하나를 수행합니다:

- 루프의 반대쪽 터널을 클릭하여 터널을 시프트 레지스터로 대체합니다.
- 블록 다이어그램을 클릭하여 루프 경계의 시프트 레지스터를 다른 시프트 레지스터 터미널의 정면 건너편에 놓습니다.



노트

시프트 레지스터 터미널이 터널 뒤에 나타나는 경우, 시프트 레지스터는 연결되지 않습니다.

While 루프에서 오토인덱싱 활성화된 터널을 시프트 레지스터로 변환하는 경우, 시프트 레지스터가 오토인덱싱할 수 없기 때문에 루프 밖의 노드에 연결한 모든 와이어는 깨집니다. 와이어가 깨지면 다음 단계를 따릅니다:

1. 깨진 와이어와 깨진 와이어에 연결된 출력을 삭제합니다.
2. 오른쪽 시프트 레지스터의 출력 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>인디케이터를 선택합니다.

관련 정보

시프트 레지스터: 루프 반복 사이에 값 전달하기

루프를 사용하여 배열이나 집합의 개별 원소 처리하기

While 루프

For 루프

최종 값을 이전 실행에서 피드백 노드로 전달하기

피드백 노드를 블록 다이어그램에 추가하면, 노드에 초기화 터미널이 부착되고, 이전 실행의 마지막 값을 전달해서 VI가 컴파일되거나 로드될 때 노드가 글로벌하게 초기화됩니다. 프로퍼티 대화 상자의 설정 페이지에서 피드백 노드를 설정할 수도 있습니다.

입력값이 있고 VI가 컴파일되거나 로드되면, 피드백 노드가 글로벌하게 초기화되도록, 피드백 노드를 다음 단계로 설정합니다.

1. 블록다이어그램에 피드백 노드를 추가합니다.
2. 입력 값을 초기화 터미널에 와이어로 연결합니다. 두 개 혹은 그 이상의 실행이나 반복에 지연을 설정할 때, 배열을 초기화 터미널에 연결하여 각 지연 원소를 임의의 값으로 초기화합니다.



노트

피드백 노드가 글로벌하게 초기화되고 사용자가 초기값을 설정하는 경우, 피드백 노드는 실행을 시작할 때마다 VI의 첫번째 호출에서 해당 값으로 초기화되도록 기본설정됩니다. 입력값을 초기화 터미널에 연결하지 않은 경우, 첫번째 실행시 피드백 노드의 초기 입력은 해당 데이터 타입의 기본값입니다.

3. 초기화 터미널에서 마우스 오른쪽 버튼을 클릭한 후 글로벌하게 초기화 >> 첫번째 컴파일 또는 로드시 초기화를 선택합니다.



관련 정보

[피드백 노드: 루프 반복 사이에 데이터 저장하기](#) (371 페이지 참조)

[블록다이어그램 객체](#) (52 페이지 참조)

루프 실행 시마다 피드백 노드의 입력값 리셋하기

피드백 노드는 글로벌하게 **VI 컴파일 또는 로드 시 초기화**되도록 기본설정되어 있습니다. 그러나 루프 실행에서 초기화되도록 노드를 설정할 수도 있습니다.

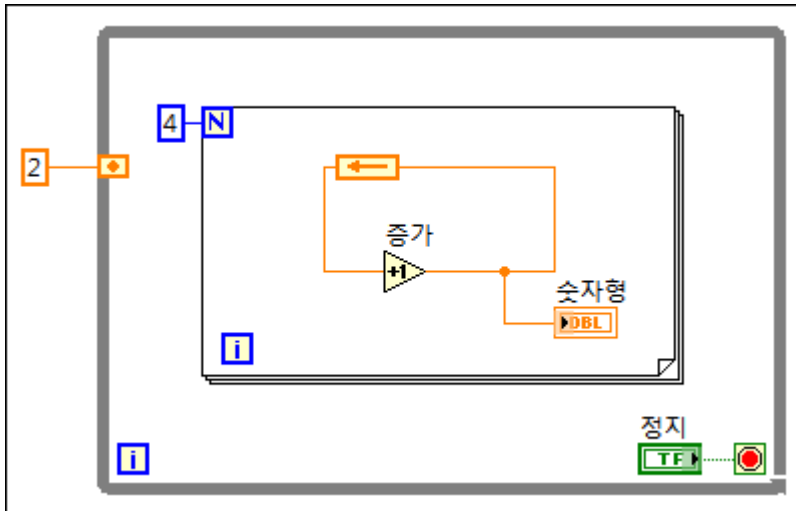
다음 단계에 따라 초기화 터미널을 루프의 왼쪽 경계로 이동하면, 루프가 실행될 때마다 입력값이 리셋됩니다.

1. 피드백 노드를 블록다이어그램에 있는 루프 또는 중첩된 하위 루프 및 구조 그룹에 추가합니다.
2. 초기화 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 한 루프 초기화 출력 이동을 선택하여 다음 루프 출력으로 초기화 터미널을 이동합니다.
3. 루프 밖에서 입력값을 초기화 터미널에 연결합니다.

초기화 터미널을 루프의 왼쪽 경계로 움직인 후, 초기화 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 글로벌하게 초기화 >> 컴파일 또는 로드시 초기화를 선택하여, 초기화 터미널을 루프 내 원래 위치로 다시 이동할 수 있습니다. 한 루프 초기화 입력 이동을 선택하여 초기화 터미널을 다음 루프 입력으로 이동합니다.

초기화 터미널을 가장 바깥쪽 루프의 경계로 이동하는 경우, 한 루프 초기화 입력 이동을 선택하여 초기화 터미널을 다음 루프 입력으로 이동합니다.

루프가 실행되는 동안 피드백 노드가 초기화하도록 설정하면, 피드백 노드는 매 루프 실행의 첫 번째 반복 전에 초기화됩니다. 초기화 터미널이  아이콘으로 변하고, 다음 그림의 루프에서 보이는 것처럼, 터미널에 연결한 데이터 타입에 따라 아이콘의 색이 바뀝니다.



위의 블록다이어그램에서, 피드백 노드는 For 루프 안에 있지만, 초기화 터미널은 가장 바깥쪽 While 루프의 경계에 있습니다. 그러므로, 피드백 노드는 While 루프가 실행될 때마다 입력값 2로 초기화되는 반면, [증가] 함수의 결과는 각 For 루프 실행마다 이전 값에 추가됩니다. 피드백 노드와 초기화를 분리하면 피드백 노드가 특정한 케이스에서만 실행되기를 원할 때 하위 케이스 구조에서 유용합니다.

시프트 레지스터를 피드백 노드로 대체하기

다음 단계를 따라 **시프트 레지스터**를 피드백 노드로 대체합니다.

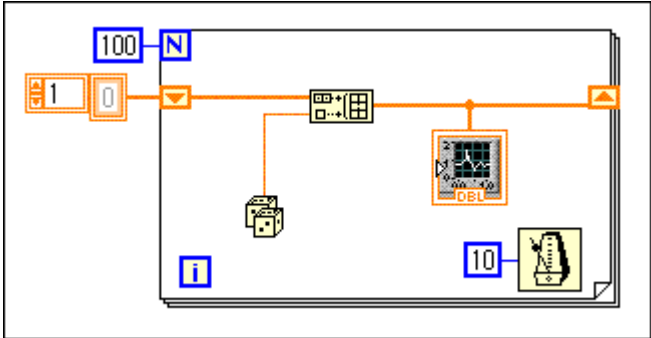
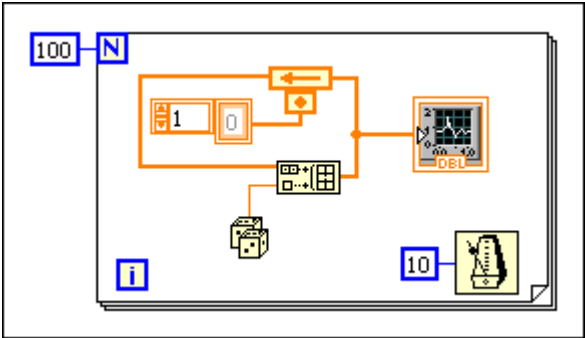
1. 시프트 레지스터에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 피드백 노드로 대체를 선택합니다.

피드백 노드를 시프트 레지스터로 대체하기

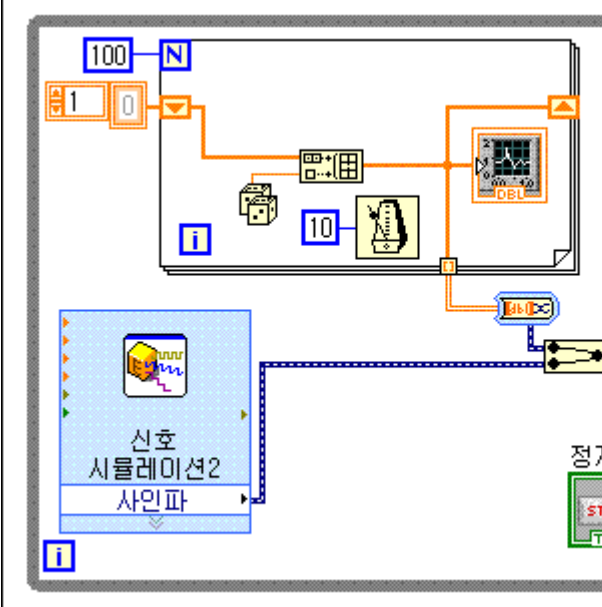
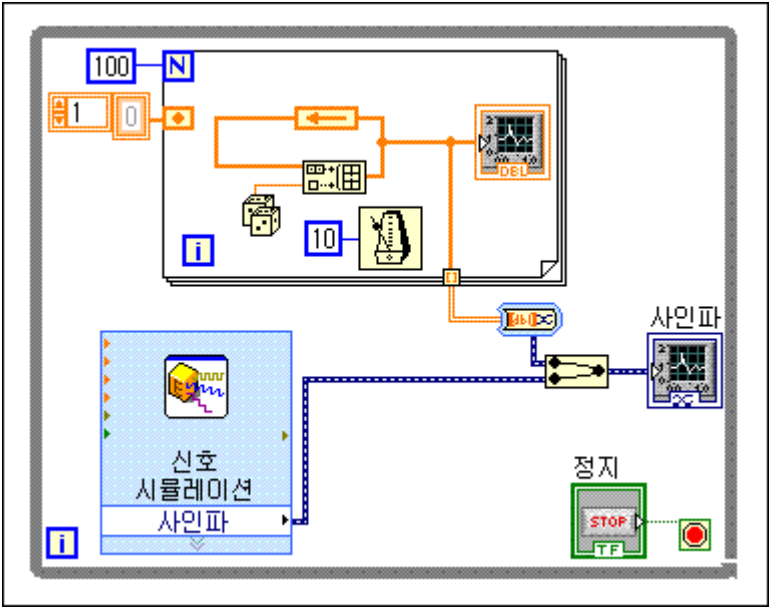
다음 단계를 따라 피드백 노드를 시프트 레지스터로 대체합니다.

1. 피드백 노드에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 시프트 레지스터로 대체를 선택합니다.

피드백 노드를 시프트 레지스터로 대체하기 전 **초기화 터미널**을 루프의 경계로 이동하지 않은 경우, 다음의 블록다이어그램에서 보이는 것처럼 LabVIEW는 시프트 레지스터를 이전 노드를 포함했던 루프의 경계에 놓습니다.



피드백 노드를 시프트 레지스터로 대체하기 전 루프의 경계로 초기화 터미널을 이동한 경우, 다음의 블록다이어그램에서 보이는 것처럼 LabVIEW는 시프트 레지스터를 루프의 경계에 놓습니다.



피드백 노드 화살표 방향 변경하기

와이어에서 데이터 흐름을 나타내는 피드백 노드의 화살표 방향을 변경할 수 있습니다. 화살표의 방향을 바꾸더라도 피드백 노드와 그에 연결된 와이어의 외형적 모양만 변경되며 데이터의 흐름 방향은 변하지 않습니다.

다음 단계를 따라 피드백 노드의 화살표 방향을 변경하십시오.

1. 블록다이어그램에 피드백 노드를 추가합니다.

2. 프로퍼티 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 방향 변경을 선택합니다.



노트

LabVIEW는 피드백 노드를 연결한 방식에 기반하여 자동으로 화살표 방향을 변경합니다. 예를 들어, 와이어에서 데이터 흐름의 방향을 변경하면, LabVIEW는 피드백 노드 화살표의 방향을 변경합니다.

서브다이어그램과 케이스 구조

서브다이어그램 추가 및 복제하기

구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이전 케이스 추가, 다음 케이스 추가(또는 프레임이나 서브다이어그램)를 선택하여 **서브다이어그램**을 추가합니다. 또한 라벨에서 <Shift-Enter> 키를 눌러 구조에 서브다이어그램을 추가할 수도 있습니다.

다음 단계를 따라 케이스, 다층 시퀀스, 조건적 비활성화 구조, 다이어그램 비활성화 구조 또는 타입 특수화 구조에 보이는 서브다이어그램의 복사본을 만들고 원래 서브다이어그램 뒤에 삽입합니다.

1. 복제하려는 **서브다이어그램으로 이동**합니다.
2. 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 케이스 복제(또는 프레임이나 서브다이어그램)를 선택합니다. 복제 서브다이어그램은 블록 다이어그램 터미널의 패널 객체를 제외한 원래 서브다이어그램의 모든 객체를 포함합니다.



노트

케이스 선택자 라벨에서 <Ctrl-Shift-Enter> 키를 눌러 케이스 구조의 보이는 서브다이어그램 부분을 복사할 수도 있습니다.

1. **조건적 비활성화 구조를 설정**하는 경우, 구조 경계에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 서브다이어그램 복제를 선택하여 조건 설정 대화 상자를 디스플레이합니다. 기호를 선택하고 조건 설정 대화 상자에서 값을 입력합니다.

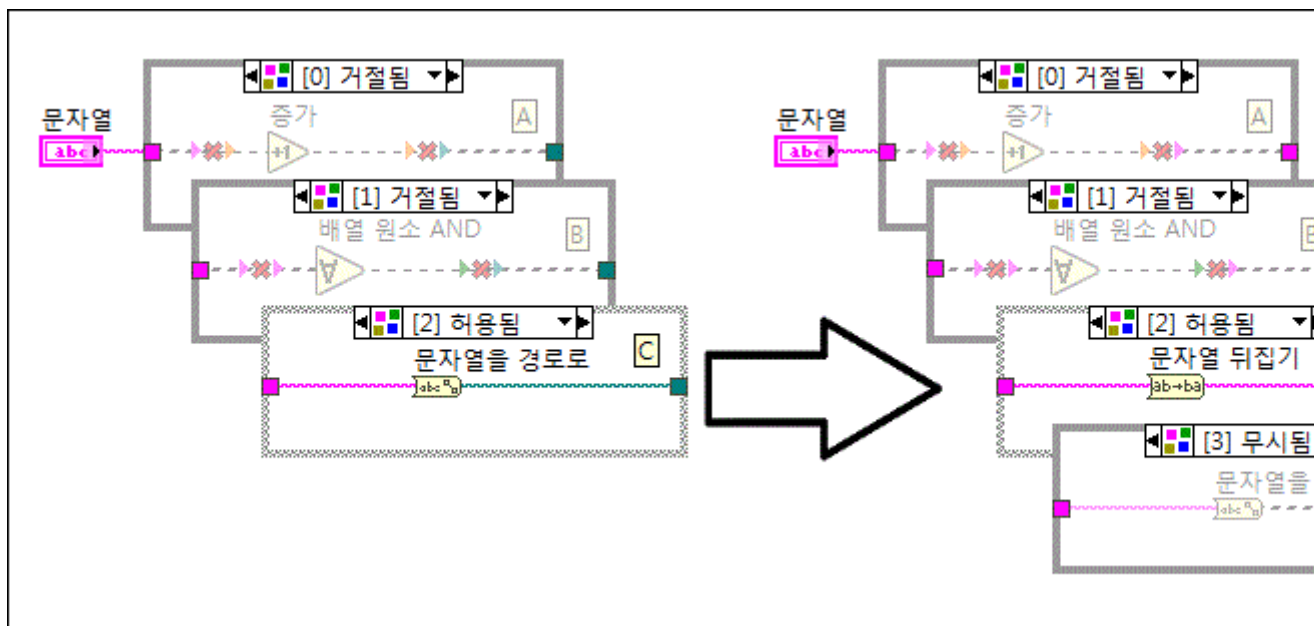


노트 LabVIEW 프로젝트에서 조건적 비활성화 구조를 사용하지 않는 경우, RUN_TIME_ENGINE, TARGET_BITNESS 및 TARGET_TYPE의 기호만 사용할 수 있습니다. 또한 LabVIEW 프로젝트에서 조건적 비활성화 구조를 사용하는 경우에는 조건적 비활성화 기호 페이지를 사용하여 기호를 생성하고 정의할 수도 있습니다.

서브다이어그램을 추가하거나 복제하는 경우 다음 유의사항을 고려합니다.

- 다층 시퀀스 구조에서 프레임을 추가하거나 **삭제**하거나 또는 **재배치**할 때, LabVIEW는 프레임 라벨의 숫자를 자동적으로 조정합니다.
- 플랫폼 시퀀스 구조에 프레임을 추가하거나 삭제할 때, 구조는 자동적으로 크기 조정됩니다.
- 조건적 비활성화 구조의 서브다이어그램을 재배치하면, 여러 서브다이어그램이 '참'일 수 있지만 LabVIEW는 '참'인 첫번째 서브다이어그램만을 컴파일하고 실행합니다.
- 타입 특수화 구조에서 서브다이어그램을 추가하거나 삭제하면 서브다이어그램 라벨과 vi 실행 방법이 영향을 받습니다. 이는 타입 특수화 구조가 각 서브다이어그램을 순서대로 컴파일하려고 시도하고 구문 에러가 있는 서브다이어그램은 거부하기 때문입니다. 구조는 구문 에러가 없는 첫번째 서브다이어그램을 받고 나머지 서브다이어그램을 무시합니다.

예를 들어, 다음 타입 특수화 구조는 서브다이어그램 A, B 및 C를 포함합니다. 구조는 처음 두 개의 서브다이어그램을 거부하고 마지막 하위 다이어그램 C(인덱스 2)를 받습니다. 서브다이어그램 C 앞에 서브다이어그램 D를 추가하면 서브다이어그램 C의 인덱스는 3이 됩니다. 구조는 서브다이어그램 C를 컴파일하려고 시도하기 전에 에러 없는 서브다이어그램 D를 컴파일하기 때문에 서브다이어그램 C를 무시하고 서브다이어그램 D를 받습니다.



- **이벤트 구조를 추가하거나 복제**할 수도 있습니다.

서브다이어그램 사이에서 움직이기

다음 중 한 가지 방법을 사용하여 사용 가능한 **서브다이어그램**을 스크롤합니다.

- 가능한 경우 구조 선택자 라벨에서 감소 및 증가 화살표를 클릭합니다.
- (Windows) 커서를 선택자 라벨 위로 이동하고 <Ctrl> 키를 누른 상태에서 마우스 휠을 움직입니다. (Linux) 커서를 선택자 라벨 위로 이동하고 <Meta> 키를 누릅니다.

다음 중 한 가지 방법을 사용하여 특정 서브다이어그램으로 이동합니다:

- 케이스 이름나 프레임 이름 옆의 아래 방향 화살표를 클릭한 후 바로 가기 메뉴에서 서브다이어그램을 선택합니다.
- 구조의 경계를 마우스 오른쪽 버튼으로 클릭하고 케이스 보이기 또는 프레임 보이기를 선택합니다. 바로 가기 메뉴에서 케이스 또는 프레임 라벨을 선택합니다.

서브다이어그램 삭제하기

다음 단계에 따라 **서브다이어그램**을 삭제합니다.

1. 삭제하려는 서브다이어그램으로 이동합니다.
2. 구조의 경계에서 마우스 오른쪽 버튼을 클릭한 후 다음 옵션 중 하나를 선택합니다:
 - 이 케이스 삭제
 - 이 프레임 삭제
 - 이 서브다이어그램 삭제
 - 이 이벤트 케이스 삭제

하나의 서브다이어그램만이 존재한다면, 보이는 서브다이어그램을 삭제할 수 없습니다.

서브다이어그램 재배치하기

다음 단계에 따라 구조에서 **서브다이어그램**을 재배치합니다.

1. 구조의 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 케이스 재배치 또는 서브다이어그램 재배치를 선택하여 케이스 재배치 대화 상자를 디스플레이합니다.
2. 케이스 리스트에서 케이스 또는 서브다이어그램을 선택하고 새 위치로 끕니다. 수평 라인이 나타나 리스트에서 케이스 또는 서브다이어그램을 삽입하는 위치를 나타냅니다. 또한 정렬 버튼을 클릭하여 케이스 또는 서브다이어그램을 알파벳 순으로 재배치할 수 있습니다.
3. 확인 버튼을 클릭합니다.

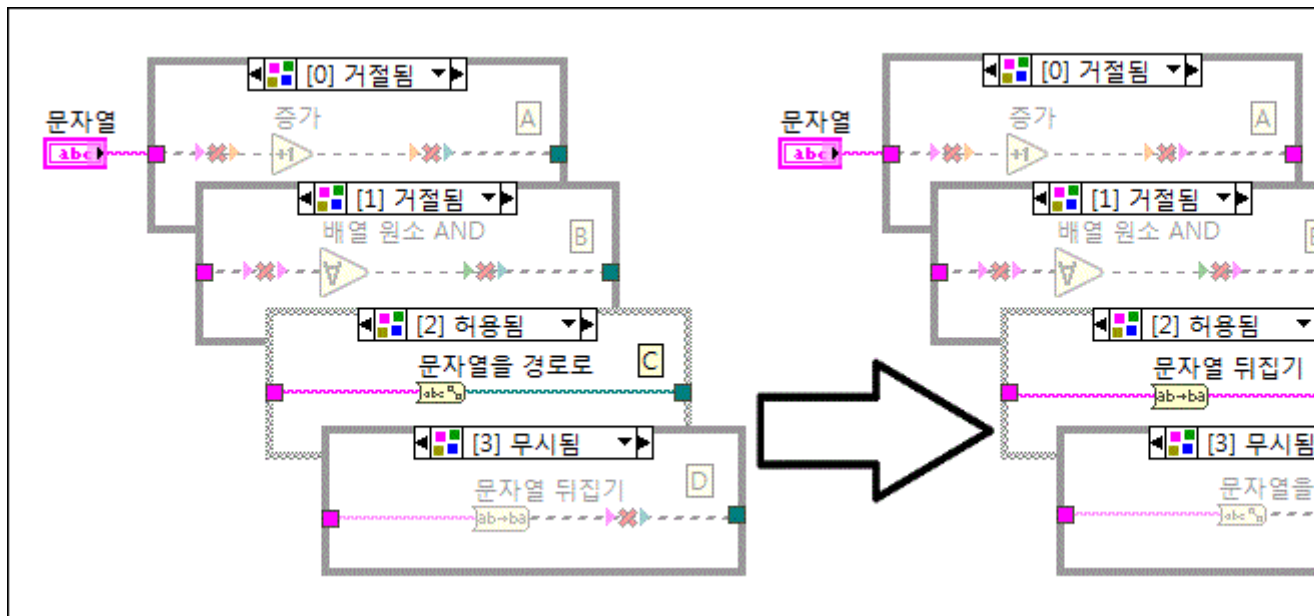
케이스 구조의 경우 케이스를 **서로 바꾸거나 이동**하여 다시 배치할 수 있습니다.

다층 시퀀스 구조에서 프레임을 재배치하려면 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 프레임을 바꾸기를 선택하고, 새 프레임 번호를 선택합니다.

구조의 서브다이어그램을 재배치할 때 다음과 같은 사항을 유의하십시오.

- 셋 또는 그 이상의 케이스가 있는 케이스 구조에서 케이스를 재배치할 수 있습니다. 둘 또는 그 이상의 케이스가 있는 이벤트 구조에서 케이스를 재배치할 수 있습니다. 둘 또는 그 이상의 서브다이아그램이 있는 **비활성화 구조**에서 서브다이아그램을 재배치할 수 있습니다.
- 조건적 비활성화 구조의 서브다이아그램을 재배치하면, 여러 서브다이아그램이 '참'일 수 있지만 LabVIEW는 '참'인 첫번째 서브다이아그램만을 컴파일하고 실행합니다.
- 타입 특수화 구조에서 서브다이아그램을 다시 배치하면 서브다이아그램 라벨과 VI 실행 방법에 영향을 줍니다. 이는 타입 특수화 구조가 각 서브다이아그램을 순서대로 컴파일하려고 시도하고 구문 에러가 있는 서브다이아그램은 거부하기 때문입니다. 구조는 구문 에러가 없는 첫번째 서브다이아그램을 받고 나머지 서브다이아그램을 무시합니다.

다음 예에서 타입 특수화 구조에는 서브다이아그램 A, B, C 및 D가 있습니다. 이 구조는 서브다이아그램 C를 받고 D를 무시합니다. 서브다이아그램 C와 D를 전환하면 서브다이아그램 C 및 D의 인덱스 역시 전환됩니다. 구조는 서브다이아그램 C를 컴파일하려고 시도하기 전에 에러 없는 서브다이아그램 D를 컴파일하기 때문에 서브다이아그램 C를 무시하고 서브다이아그램 D를 받습니다.



조건 코드 실행을 위한 케이스 구조 생성하기

케이스 구조를 사용하여, 구조로 전달되는 입력값에 따라 실행되는 두 개 이상의 케이스 서브다이아그램을 지정합니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.

케이스 구조를 생성할 때 다음 단계로 케이스 서브다이아그램의 값을 지정합니다.

1. 블록다이아그램에 케이스 구조를 추가합니다.
2. 입력값을 케이스 선택자 에 연결하여 어떤 서브다이아그램이 실행될지 결정합니다. 케이스 구조의 왼쪽 경계 아무 곳이나 케이스 선택자를 놓을 수 있습니다. 다

음 테이블에서 보이는 것처럼, 서로 다른 **데이터 타입**을 케이스 선택자에 연결하여 특정 타입의 서브다이어그램을 생성합니다.

데이터 타입

케이스 구조의 서브다이어그램

구조에 참 케이스와 거짓 케이스가 있습니다.

구조에 에러 케이스와 에러 없음 케이스가 있습니다. 또한, 에러 클러스터를 사용하여 **에러 처리하기**가 가능합니다.

개수의 제한 없이 구조에 케이스가 있습니다.



노트

기본 설정상, 케이스 선택자에 와이어로 연결하는 문자열 값은 대소문자를 구분합니다. 대소문자 구분없는 일치 허용하려면, 케이스 선택자에 문자열 값을 와이어로 연결한 다음에 케이스 구조의 경계선에서 마우스 오른쪽 버튼을 클릭하여, 바로 가기 메뉴에서 대소문자 구분없는 일치를 선택합니다.

팁 터널을 케이스 선택자로 변환하려면 터널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 케이스 선택자로 대체를 선택합니다. 이전의 케이스 선택자가 연결되어 있는 경우, LabVIEW는 이전에 연결된 케이스 선택자를 터널로 변환합니다. 또한 케이스 선택자로 대체 메소드를 사용하여 프로그램적으로 터널을 케이스 선택자로 변환할 수 있습니다.

3. **케이스 구조 안에 객체를 추가**하여, 케이스 구조를 실행할 수 있는 서브다이어그램을 생성합니다. 필요한 경우, **서브다이어그램을 추가하거나 복제**합니다.
4. 각 케이스에서, 라벨링 도구를 사용하여 케이스 구조 위에 있는 선택자 라벨에 **하나의 값 또는 값의 리스트와 범위를 입력**합니다. 각 선택자 라벨에 값을 입력하면 구조에 전달된 입력값에 따라 특정 서브다이어그램이 실행됩니다.

노트 문자열 데이터 타입이 있는 케이스 서브다이어그램의 경우, **백슬래시 코드**를 사용하여 백슬래시 및 디스플레이가 불가능한 문자를 나타냅니다. 예를 들어, \\를 사용하여 한 개의 백슬래시를 나타내고, \r로 캐리지 리턴을 나타냅니다.

각 케이스 구조 서브다이어그램의 값 입력하기

다음 테이블에서 보이는 것처럼 **케이스** 구조의 선택자 라벨에 단일 값 또는 값의 리스트와 범위를 입력할 수 있습니다.



노트

선택자 터미널이 에러 클러스터에 연결된 경우, LabVIEW는 숫자형 값 또는 범위에 에러를 추가하여 선택자 라벨의 숫자형 값을 에러 코드로 변환합니다.

케이스 구조의 케이스 선택자에 연결된 와이어의 데이터 타입을 바꾸면, 케이스 구조는 자동적으로 선택자 라벨 값을 새 데이터 타입으로 변경합니다. 예를 들어, 숫자값 19를

문자열로 바꾸면 문자열 값은 "19"가 됩니다. 문자열을 숫자값으로 변환하면, LabVIEW는 숫자를 나타내는 문자열 값만을 변환합니다. 다른 값은 문자열로 남아있습니다. 숫자를 불리언 값으로 변환하면, LabVIEW는 0을 거짓으로, 1을 참으로 변환하고 모든 다른 숫자값은 문자열이 됩니다.

케이스 선택자에 연결된 객체와 다른 타입의 선택자 라벨 값을 입력하는 경우 값은 빨간색으로 나타납니다. 이는 해당 값을 삭제하거나 편집할 때까지 VI가 실행되지 않음을 나타냅니다.



노트 부동소수는 반올림 문제가 발생할 수 있으므로 부동소수 값을 케이스 선택자 라벨 값으로 사용할 수 없습니다. 부동소수값을 케이스 선택자에 연결하면, LabVIEW는 가장 가까운 정수로 반올림합니다. 선택자 라벨에 부동소수값을 입력하면, 값이 빨간색으로 표시되며 이는 구조를 실행하기 전에 이 값을 삭제하거나 편집해야 한다는 것을 나타냅니다. 또한, 고정소수점 수를 케이스 라벨 값으로 사용할 수 없습니다. 고정소수점 수를 케이스 선택자에 연결하는 경우, VI 실행 버튼이 깨져서 나타납니다.

케이스 구조에서 연결되지 않은 출력 터널의 연결 피하기

케이스 구조에 여러 개의 입력 및 출력 터널을 만들 수 있습니다. 모든 케이스 서브다이어그램에서 입력이 사용 가능하지만, 모든 서브다이어그램이 모든 입력을 사용해야 하는 것은 아닙니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.

그러나 케이스 구조의 서브다이어그램 경계에서 출력 터널을 생성하면, 다른 모든 케이스 경계에서 동일한 위치에 터널이 보입니다. 하나의 출력 터널이라도 연결되지 않으면, 구조의 모든 출력 터널의 중심이 흰색 사각형으로 나타납니다.

다음 중 한 가지 방법으로 터널에 기본 데이터 타입 값을 사용하여 연결되지 않은 모든 출력 터널의 연결을 피합니다:

- 출력 터널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 연결되지 않으면 기본값 사용을 선택하여 연결되지 않은 모든 터널의 터널 데이터 타입에 기본값을 사용합니다.
- 구조의 터널을 설정하여 연결되지 않은 케이스의 **입력과 출력 터미널을 자동으로 연결**합니다(최상의 추천 방법).



노트

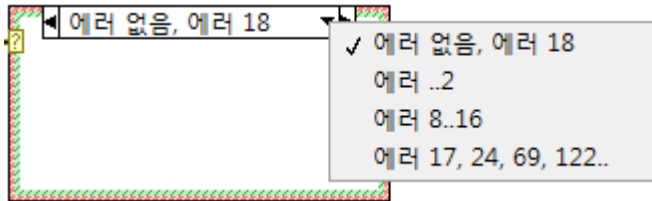
각 케이스에서 같은 출력 터널에 대해 다른 데이터 소스를 정의할 수 있지만, 데이터 타입은 각 케이스에 대해 호환 가능해야 합니다.

케이스 구조로 에러 핸들링하기

다음 단계를 따라 **케이스** 구조를 사용하여 에러를 핸들링합니다.

1. 에러 클러스터 터미널을 선택합니다.

2. 에러 클러스터를 케이스 구조의 선택자 터미널에 와이어로 연결합니다. 케이스 선택자 라벨은 두 케이스를 모두 디스플레이하고(에러 및 에러 없음), 케이스 구조의 경계는 케이스에 따라 색이 바뀝니다(에러의 경우 빨강, 에러 없음의 경우 녹색). 케이스 구조는 모든 에러의 에러 서브다이어그램을 실행합니다.
3. (옵션) 케이스 구조가 특정 에러 또는 에러 리스트에 대해 특정 서브다이어그램을 실행하도록 설정하려면 케이스를 더 추가하고 **선택자 라벨에 에러 값이나 범위를 입력**합니다. 에러 없음 케이스에도 에러 값을 추가할 수 있습니다. 선택자 라벨이 에러 코드를 포함하는 경우 에러 없음 케이스의 경계는 녹색과 빨간색입니다.



노트

에러 케이스의 선택자 라벨을 사용자 정의하지 않고 에러 케이스를 더 추가하면 에러 케이스가 기본으로 변경되고, 이는 기존 에러 케이스로 처리되지 않는 나머지 모든 에러를 처리합니다.

4. (옵션) 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 기수를 선택하여 선택자 라벨에서 에러 코드의 디스플레이 포맷을 설정합니다. 디스플레이 포맷이 10진수이면, 포맷을 나타내는 문양이 선택자 라벨의 왼쪽에 나타납니다.
5. 에러 케이스에 코드를 추가하여 여러 다른 케이스에 대해 동작을 정의합니다.

케이스 구조 서브다이어그램 사이에 케이스 추가하기

다음 단계를 따라 **케이스** 구조 안에 있는 기존의 케이스 서브다이어그램 사이에 케이스를 추가합니다:

1. 순서의 끝에 **새로운 케이스를 추가**합니다.
2. 새로운 케이스가 보이면, 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 케이스 재배치를 선택합니다.
3. 새 케이스의 위치를 선택합니다. 이후의 서브다이어그램은 순서를 따라 다음으로 이동합니다.
4. 확인 버튼을 클릭하여 변경을 저장합니다.



팁

선택자 라벨에서 <Shift-Enter> 키를 눌러 케이스 구조에 케이스를 추가할 수도 있습니다.

또한, **기존 케이스를 다음 순서로 이동**시킬 수 있습니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.

케이스 구조의 기본 서브다이어그램 지정하기

가능한 입력값을 모두 나열하지 않으려면, **케이스** 구조에서 기본 케이스 서브다이어그램을 지정하여 범위 외 값을 처리합니다. 예를 들어, 케이스 선택자가 정수이고 서브다이어그램을 1, 2, 3으로 지정하면, 입력값이 4이거나 다른 지정되지 않은 정수값일 경우, 실행할 기본 서브다이어그램을 지정해야 합니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.

다음 단계에 따라 기본 서브다이어그램을 지정합니다.

1. **기존 서브다이어그램으로 이동**합니다.
2. 선택자 라벨에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 기본 케이스로 만들기를 선택하여 기존 서브다이어그램이 기본 서브다이어그램으로 지정합니다.

또한 다음 단계에 따라 기본 서브다이어그램을 지정할 수 있습니다.

1. 기본으로 정하려는 **서브다이어그램으로 이동**하고 라벨링 도구로 선택자 라벨을 클릭합니다.
2. 선택자 라벨에서 기본을 입력합니다.



노트

기본 주위에 따옴표를 추가하지 마십시오. 그렇게 하면 값이 기본 서브다이어그램이 아닌 문자열로 인식됩니다.

1. <Enter> 키를 누릅니다.



노트 불리언 컨트롤을 케이스 선택자에 연결한 경우, 기본 서브다이어그램을 지정할 수 없습니다. 불리언 컨트롤을 참 또는 거짓으로 설정하여 어떤 서브다이어그램을 실행할지 결정합니다.

케이스 구조 서브다이어그램 교환하기

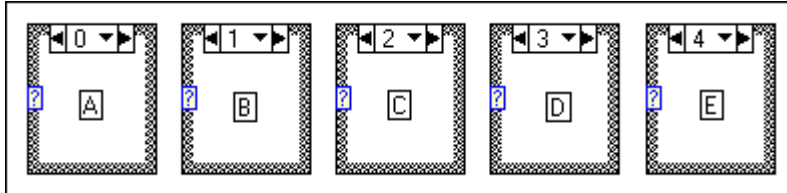
다른 서브다이어그램 또는 바로 가기 메뉴에 나타나는 케이스의 순서에 영향을 미치지 않고 **케이스** 구조의 보이는 케이스 서브다이어그램을 다른 케이스와 교환할 수 있습니다. 두 개 케이스의 서브다이어그램만을 변경합니다. 서브다이어그램이 3개 이상 있는 케이스 구조에서 케이스를 교환할 수 있습니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.

다음 단계에 따라 보이는 서브다이어그램을 다른 서브다이어그램과 교환합니다.

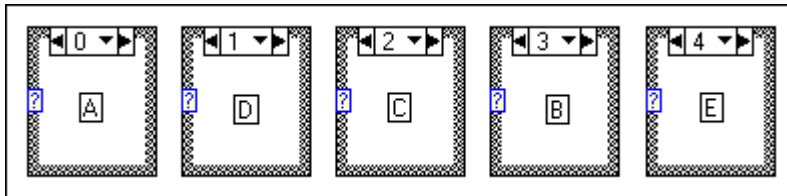
1. 다른 서브다이어그램과 교환하려는 서브다이어그램이 속한 **케이스로 이동**합니다.

2. 구조 경계에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 케이스와 함께 다이어그램 교환을 선택하고 보이는 서브다이어그램과 교환하려는 서브다이어그램의 케이스를 선택합니다.

다음 **케이스** 구조의 예제에서, 케이스 0은 서브다이어그램 A를 가지고 있고, 케이스 1은 서브다이어그램 B를 가지고 있고, 이렇게해서 서브다이어그램 E를 가지고 있는 케이스 4까지 보여줍니다.



서브다이어그램이 있는 케이스 1을 케이스 3과 교환할 수 있는데, 이렇게 하면 다음 예에서 보이는 것처럼 LabVIEW가 서브다이어그램을 교환합니다.



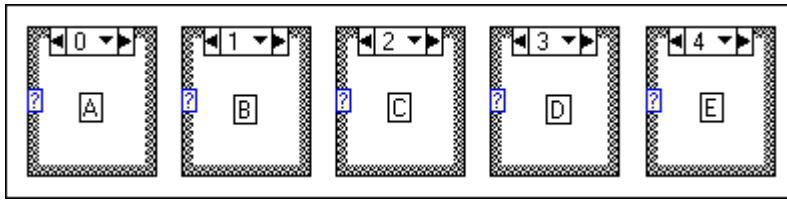
케이스 구조 서브다이어그램 이동하기

케이스 선택자에 연결된 데이터 타입이 숫자형이고 열거형이 아닌 경우, **케이스** 구조에서 보이는 케이스 서브다이어그램을 케이스 순서상 다른 위치로 이동할 수 있습니다. 서브다이어그램이 3개 이상 있는 케이스 구조에서 서브다이어그램을 이동할 수 있습니다. 케이스 구조를 사용하는 예제는 labview\examples\Structures 디렉토리에 있는 Case Structure - Selector Data Types VI를 참조하십시오.

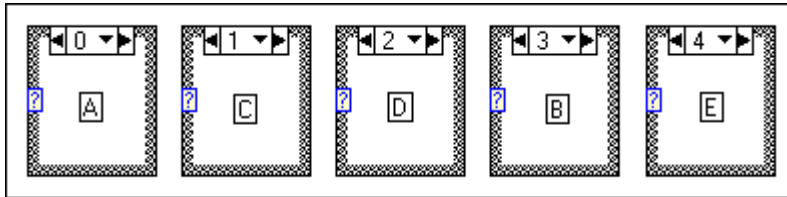
다음 단계에 따라 보이는 서브다이어그램을 다른 위치로 이동합니다.

1. 다른 위치로 이동하려는 서브다이어그램의 **케이스로 이동**합니다.
2. 구조 경계에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 다이어그램을 케이스로 이동을 선택하고, 보이는 서브다이어그램과 함께 이동시키려는 서브다이어그램의 케이스를 선택합니다.

다음 **케이스** 구조의 예제에서, 케이스 0은 서브다이어그램 A를 가지고 있고, 케이스 1은 서브다이어그램 B를 가지고 있고, 이렇게해서 서브다이어그램 E를 가지고 있는 케이스 4까지 보여줍니다.



케이스 1인 서브다이아그램을 케이스 3의 위치로 이동할 수 있으며, 이렇게 하면 다음 예에서 보이는 것처럼 케이스 2와 3에 해당하는 서브다이아그램이 차례로 위로 이동합니다.



플랫 시퀀스 구조의 프레임을 통해 데이터 전달하기

플랫 시퀀스 구조를 사용하는 경우, 시퀀스 구조 밖의 데이터는 각 프레임이 실행되면서 프레임에 들어옵니다. 데이터는 프레임이 실행된 후에 프레임을 나옵니다.

다음 단계를 따라 플랫 시퀀스 구조에서 프레임을 통해 연속적으로 데이터를 전달합니다.

1. 플랫 시퀀스 구조에 연결할 터미널을 선택합니다.
2. 한 프레임의 터미널을 구조 내의 아무 프레임에나 연결합니다. 프레임을 통과하여 와이어를 연결하거나 한 프레임에서 다른 프레임으로 와이어를 연결할 수 있습니다.

다층 시퀀스 구조의 프레임을 통해 데이터 전달하기

다층 시퀀스 구조에서 시퀀스 로컬 터미널을 사용하여 한 프레임에서 다음에 오는 임의의 프레임에 데이터를 전달합니다.



노트

다층 시퀀스 구조는 구조에 연결된 모든 데이터가 도달할 때까지 실행을 시작하지 않습니다. 각 프레임으로부터 연결된 데이터는 모든 프레임이 실행을 종료해야만 떠납니다.

다음 단계를 따라 시퀀스 로컬 터미널을 추가하여 다층 시퀀스 구조의 프레임을 통해 데이터를 전달합니다.

1. 다음 프레임으로 데이터를 전달하려는 **프레임으로 이동**합니다.
2. 오른쪽 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 시퀀스 로컬 추가를 선택합니다.

3. 전달하려는 데이터를 시퀀스 로컬 터미널에 연결합니다. 밖을 향하는 화살표 가 터미널에 나타납니다.
4. 다음 프레임에서, 이전 프레임에서 데이터를 전달하려는 터미널에 시퀀스 로컬 터미널을 연결합니다. 다음 프레임에 있는 터미널은 안으로 향하는 화살표를 포함하며, 이것은 터미널이 해당 프레임에서 데이터 소스라는 것을 나타냅니다.



노트 시퀀스 로컬 터미널을 연결했던 프레임 이전의 프레임에서는 시퀀스 로컬 터미널을 사용할 수 없습니다.

시퀀스 로컬 터미널을 제거하기 위해서, 시퀀스 로컬 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 제거를 선택합니다.

시퀀스 구조를 언제 사용할지 결정하기

시퀀스 구조는 실행의 순서를 강제로 지정하고 병렬 작업을 막습니다. 예를 들어, PXI, GPIB, 시리얼 포트 및 DAQ 디바이스와 같은 I/O 장치를 사용하는 비동기 작업은 시퀀스 구조가 금지하지 않는다면 다른 작업과 동시에 실행될 수 있습니다.

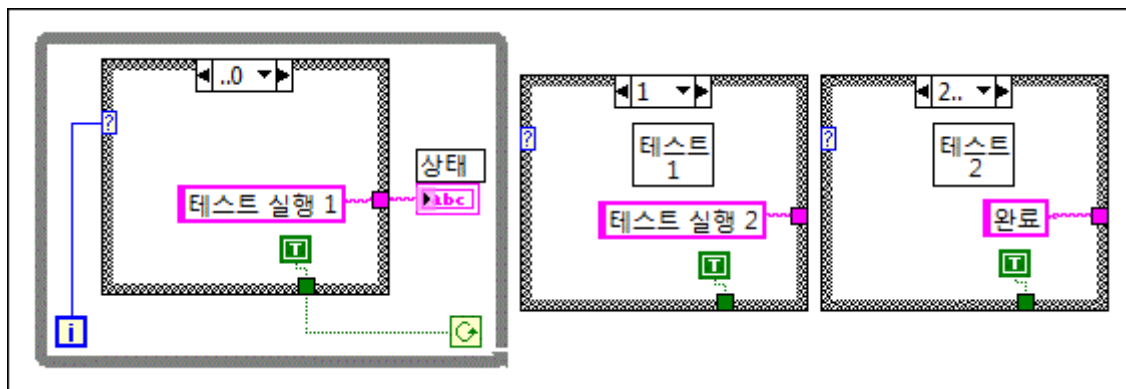
이러한 경우, LabVIEW의 고유한 병렬 처리 기능을 활용할 필요가 있습니다. 다음 중 한 가지 방법으로 시퀀스 구조의 지나친 사용을 피합니다:

- 실행 순서를 컨트롤 할 때는 노드 사이의 데이터 의존성을 이용하는 방법을 고려하십시오. 예를 들어, **예러 I/O**와 같은 **흐름 파라미터**를 사용하여 실행 순서를 컨트롤할 수 있습니다.
- 케이스 구조와 While 루프를 사용합니다. 다음 그림과 같이 시퀀스 구조의 여러 프레임에서 인디케이터를 업데이트할 수는 없습니다.



이전 블록 다이어그램

램에서 테스트 어플리케이션에 사용된 VI에는 진행 중인 현재 테스트의 이름을 표시하는 상태 인디케이터가 있습니다. 각 테스트가 다른 프레임에서 호출된 SubVI 이면, 다층 시퀀스 구조에서 보이는 깨진 와이어를 통해 알 수 있듯이 각 프레임으로부터 인디케이터를 업데이트할 수 없습니다. 다층 시퀀스 구조의 모든 프레임의 실행이 끝난 다음에만 내부 데이터 값이 구조 밖으로 빠져나올 수 있으므로, 오직 하나의 프레임에서만 상태 인디케이터에 값을 할당할 수 있습니다. 대신에, 다음 블록 다이어그램에서처럼 케이스 구조와 While 루프를 사용합니다.



케이스 구조의 각 케이스는 시퀀스 구조의 프레임에 대응됩니다. While 루프가 반복하면서 각 케이스를 순차적으로 실행합니다. 상태 인디케이터는 각 케이스에 대한 vi의 상태를 디스플레이합니다. 각 케이스의 실행이 끝나면 구조 밖으로 데이터가 전달되므로 대응하는 SubVI를 호출하기 전에 상태 인디케이터가 바로 업데이트됩니다. 시퀀스 구조와 달리, 케이스 구조는 모든 케이스가 While 루프를 종료할 수 있는 데이터를 전달할 수 있습니다. 예를 들어, 첫 테스트를 실행하는 중에 에러가 발생하면 케이스 구조가 거짓을 조건 터미널에 전달하여 루프를 끝냅니다. 그러나 시퀀스 구조는 에러가 발생하더라도 전체 프레임을 실행해야 합니다.

조건적 비활성화 구조의 조건 설정하기

조건적 비활성화 구조는 플랫폼과 사용자 정의된 다른 기호를 포함하는 조건의 설정에 따라 하나의 서브다이아그램을 실행합니다.

다음 단계에 따라 기호와 값을 선택하여 조건적 비활성화 구조의 조건을 설정합니다.

1. 블록다이어그램에 조건적 비활성화 구조를 추가합니다.
2. (옵션) LabVIEW 프로젝트에서 구조를 사용하는 경우, 사용자 정의 조건적 비활성화 기호를 생성할 수 있습니다.
3. 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 이 서브다이어그램의 조건 편집을 선택하여 조건 설정 대화 상자를 디스플레이합니다.
4. 기호를 선택하고 기호의 값을 입력합니다. 다음 테이블은 기본 기호와 각 기호에 대해 유효한 값을 나열합니다. 값은 대소문자를 구별하는 문자열이므로 다음 유효한 값 중 하나를 아래와 같이 정확히 입력해야 합니다.

기호

PowerPCx86nullARM

기호

FPGA_TARGETDEV_COMPUTER_SIM_IODEV_COMPUTER_REAL_IOTHIRD_PARTY_SIMULATION

VIRTEX2VIRTEX5VIRTEX6SPARTAN3SPARTAN6SPARTAN3SPARTAN6ZYNQKINTEX7

타겟이 정의한 값을 찾으려면 FPGA 타겟 프로퍼티 대화 상자의 조건적 비활성화 기호 페이지를 참조하십시오.

LinuxMacnullPharLapVxWorksWin

참거짓

3264

WindowsFPGAIPBuilderEmbeddedRTMacUnixPocketPCDSP

**노트**

LabVIEW 프로젝트에서 조건적 비활성화 구조를 사용하지 않는 경우, RUN_TIME_ENGINE, TARGET_BITNESS 및 TARGET_TYPE의 기호만 사용할 수 있습니다.

5. (옵션) 더하기 부호를 클릭하여 다른 조건을 추가합니다. 예를 들어, 서브다이어그램이 Windows 및 Mac과 같은 복수의 플랫폼에 적용되도록 하려면, TARGET_TYPE 기호의 값을 한 조건에서 Windows로 설정하고, 더하기 부호를 클릭한 후, OR 연산자를 선택하고, TARGET_TYPE 기호의 값을 두 번째 조건에서 Mac으로 설정하면 됩니다.
6. 확인 버튼을 클릭합니다.
7. 조건적 비활성화 구조 내에 객체를 추가하여, 조건적 비활성화 구조가 실행할 수 있는 서브다이어그램을 생성합니다. 필요한 경우, 서브다이어그램을 추가하거나 복제합니다.



노트 불리언 연산자를 통해 조건을 합쳐 LabVIEW가 여러 조건을 평가할 수 있도록 식을 생성할 수 있습니다. 하나 이상의 조건이 참인 경우, 참인 첫번째 조건이 해당 구조에 대한 활성화 서브다이어그램을 결정합니다. 서브다이어그램에 대한 불리언 식을 정의하려는 경우, AND 연산자로 결합하는 기호/값 비교가 OR 연산자로 결합하는 기호/값 비교에 앞서 평가됩니다.

실행 시 컴파일되지 않도록 블록다이어그램의 코드 섹션을 비활성화하려면, 다이어그램 비활성화 구조를 사용하십시오.

조건적 비활성화 구조 사용에 대한 예제는 labview\examples\Structures\Disable Structures 디렉토리에 있는 Conditional Disable Structure VI를 참조하십시오.



예제 열기  관련 예제 찾기

조건적 비활성화 구조의 조건 사용자 정의하기

조건적 비활성화 구조를 **LabVIEW 프로젝트**에서 사용하는 경우, LabVIEW에서 사용자 조건적 비활성화 기호를 정의할 수 있습니다. 조건적 비활성화 기호는 기호 이름과 기호값으로 구성됩니다. 기호 이름과 값을 비교하게 되면 조건이 참이어야 LabVIEW가 서브다이어그램을 컴파일하고 실행합니다.

다음 단계에 따라 조건적 비활성화 기호를 생성하여 조건을 사용자 정의합니다.

1. 프로젝트 » 프로퍼티를 선택하여 프로젝트 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 조건적 비활성화 기호를 선택하여 조건적 비활성화 기호 페이지를 디스플레이합니다.
3. 새 기호 텍스트 박스에서 생성하려는 기호의 이름을 입력합니다. 기호에 대한 이름을 선택하여 수행하는 함수를 나타낼 수 있습니다. 기호의 이름은 대소문자를 구분합니다.

4. 새 값 텍스트 박스에 나타내고자 하는 기호 값을 입력합니다.



노트

값은 모두 대소문자를 구별하는 문자열입니다. 따라서 LabVIEW는 입력한 숫자값을 숫자가 아닌 문자열을 처리합니다. 예를 들어, 사용자 기호의 값을 0.0으로 정의한 후에 조건 설정 대화 상자에서 조건을 0으로 설정하면, 그 조건은 거짓으로 판별됩니다.

5. 추가 버튼을 클릭하여 조건적 비활성화 기호 리스트에 새로운 기호를 추가합니다.
6. 확인 버튼을 클릭하여 조건적 비활성화 기호 페이지를 닫습니다. 이제, 조건적 비활성화 구조에서 방금 정의한 **기호를 사용**할 수 있습니다.

또한 적절한 타겟 프로퍼티 대화 상자의 타겟 레벨에서 조건적 비활성화 기호를 생성할 수 있습니다. 예를 들어, FPGA 타겟을 사용하는 경우, FPGA 타겟 프로퍼티 대화 상자에서 조건적 비활성화 기호를 생성할 수 있습니다.

한 프로젝트에서 같은 기호가 프로젝트 레벨과 타겟 레벨에 모두 정의된 경우, 조건적 비활성화 구조는 타겟에 대해 정의된 기호만을 인식합니다.

In Place 원소 구조로 데이터 복사본 개수 줄이기

In Place 원소 구조는 일반적인 데이터 다루기 작업을 수행하기 때문에 LabVIEW 컴파일러가 여러 데이터 값을 복사하거나 해당 값을 메모리에 보관할 필요가 없습니다.

다음 단계를 따라 In Place 원소 구조를 설정하여 데이터 다루기를 수행합니다.

1. In Place 원소 구조를 블록 다이어그램에 추가합니다.
2. In Place 원소 구조의 경계에서 마우스 오른쪽 버튼을 클릭하고 사용하려는 경계 노드를 선택합니다. 경계 노드의 각 세트는 구조의 왼쪽에 경계 노드를, 구조의 오른쪽에 경계 노드를 포함합니다.
3. **객체를 In Place 원소 구조에 추가**하여 배열, 클러스터, 웨이브폼 또는 배리언트 원소를 조작합니다.
4. 경계 노드에 있는 모든 필수 터미널을 연결합니다. 그렇지 않은 경우, VI는 깨지고 실행되지 않을 것입니다.

다양한 타입의 여러 경계 노드를 하나의 In Place 원소 구조에 추가할 수 있습니다.

In Place 원소 구조 사용의 유의점과 권장사항

다음 리스트는 **In Place 원소** 구조를 사용할 때 알아야 할 유의점과 권장사항을 설명합니다:

- 클러스터를 In Place 원소 구조의 풀기/묶기 원소 경계 노드로 푸는 경우, 한 번 이상 클러스터 원소를 풀 수 없습니다. 예를 들어 대상 클러스터에 다른 클러스터가 포함되어 있고 그 하위 클러스터의 원소를 푸는 경우, 이미 하위 클러스터의 원소 중 하나를 풀었기 때문에 전체 하위 클러스터는 풀 수 없습니다.

- 단일 배열, 클러스터, 배리언트, 또는 웨이브폼을 위해 하나 이상의 경계 노드를 사용하지 마십시오. 각 개별 아이템에 하나의 경계 노드를 작동합니다. 단일 배열, 클러스터, 배리언트, 또는 웨이브폼에 하나 이상의 경계 노드를 사용하는 경우, 전체 배열, 클러스터, 배리언트, 또는 웨이브폼의 복사본이 메모리에 저장되므로 In Place 원소 구조의 긍정적인 성능 효과가 무효가 되어버립니다.
- 배리언트에 여러 타입의 데이터가 포함되어 있을 수 있기 때문에 반드시 배리언트에서 대체하려는 데이터 타입을 배리언트를 원소로 / 원소에서 배리언트로 경계 노드의 데이터 타입 입력에 연결해야 합니다. 데이터 타입 입력에 연결된 타입의 데이터가 배리언트에 포함되지 않은 경우, VI는 에러를 반환합니다.
- In Place 원소 구조를 다른 In Place 원소 구조 안에 넣을 수 있습니다. 하위 In Place 원소 구조를 사용하여 여러 복잡한 데이터 타입의 원소에 접근할 수 있습니다. 예를 들어, 클러스터 내의 배열 원소에 접근하려는 경우, 바깥의 In Place 원소 구조를 사용하여 클러스터를 풀고 안쪽의 In Place 원소 구조를 사용하여 배열의 원소에 접근합니다.
- 사용자가 노드에 연결하는 데이터의 값을 LabVIEW가 변경할 것임을 표시하려면, In Place 원소 구조의 노드에 있는 변경자로 표시 바로 가기 메뉴 옵션을 사용하는 것이 좋습니다. 예를 들어, **다이나믹 디스패치 터미널**은 클래스 계층구조 안에서 다이나믹 디스패치 SubVI의 모든 구현을 실행할 수 있습니다. 다이나믹 디스패치 VI의 직상위 구현은 어떠한 작업도 없고 데이터를 변경하지 않을 수도 있습니다. 하지만, 직하위 구현은 데이터를 변경할 수도 있습니다. 직상위 구현을 노드에 연결하고 변경자로 표시 옵션을 선택하는 경우, LabVIEW가 어느 한 시점에서 클래스 계층 구조에 있는 다이나믹 디스패치 터미널에 와이어로 연결된 값을 변경할 것임을 나타낼 수 있습니다. 그러면 변경자로 표시를 사용함으로써 LabVIEW가 생성하는 데이터의 복사본 개수를 최소화하게 되어 성능이 최적화됩니다.

Timed 구조 프레임 추가하고 삭제하기 (Real-Time, Windows)

Timed 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 다음에 프레임 추가 또는 이전을 선택하여 프레임을 추가합니다. 또한, 두 프레임 사이의 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프레임 삽입을 선택하여 두 프레임 사이에 프레임을 삽입할 수 있습니다.

Timed 구조 프레임을 삭제하려면, 프레임의 아래 또는 위 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이 프레임 삭제를 선택합니다. 하나의 프레임만이 존재하는 경우에는 프레임을 삭제할 수 없습니다.



Timed 구조 홈

Timed 구조 프레임 병합하기 (Real-Time, Windows)

두 개의 Timed 구조 프레임을 병합하려면, 두 프레임 사이의 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프레임 병합을 선택합니다.

첫번째 프레임의 오른쪽 데이터 노드에 연결된 값 또는 두번째 프레임의 왼쪽 데이터 노드에 연결된 값을 가지고 있는 프레임을 병합하는 경우, LabVIEW는 값을 깨진 와이어와 함께 보여줍니다.

Timed 구조 홈

Timed 구조 노드의 터미널 디스플레이하고 숨기기 (Real-Time, Windows)

값을 Timed 구조의 입력과 오른쪽 데이터 노드에 연결하여 구조의 옵션을 설정할 수 있습니다. 또한, 구조는 왼쪽 데이터와 출력 노드의 다른 출력 값을 반환합니다.

기본으로, Timed 구조는 입력, 출력, 데이터 노드의 여러 입력 또는 출력만 디스플레이합니다. 입력, 출력, 또는 데이터 노드에서 마우스 오른쪽 버튼을 클릭하고 모든 입력 보이거나 모든 출력 보이기를 선택하여 노드의 모든 입력 또는 출력을 디스플레이합니다.

또한, 노드에 선택된 입력 또는 출력만을 디스플레이할 수 있습니다. 입력, 출력, 또는 데이터 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 입력 추가 또는 출력 추가를 선택하여 노드에 다음으로 사용 가능한 입력 또는 출력을 디스플레이합니다. 새 입력 또는 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 입력 선택 또는 출력 선택을 선택합니다. 바로 가기 메뉴는 현재 선택된 입력 또는 출력을 확인 표시와 함께 디스플레이합니다. 디스플레이하려는 입력을 선택합니다.

노드의 입력 또는 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 입력 제거 또는 출력 제거를 선택하여 노드의 입력 또는 출력을 숨깁니다.

Timed 구조의 노드 입력 및 출력에 대한 참조 정보는 Timed 루프와 Timed 시퀀스 토픽을 참조하십시오.

Timed 구조 홈

이벤트

이벤트 케이스 추가 및 복제하기

다음 단계를 따라서 이벤트 케이스를 추가하거나 복제합니다.

1. 이벤트 구조 경계에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 이벤트 케이스 추가 또는 이벤트 케이스 복제를 선택합니다. 이벤트 편집 대화 상자가 나타납니다.
3. **이벤트 케이스를 편집합니다.**

이벤트 케이스를 복제하면, LabVIEW는 케이스에 설정한 이벤트는 복제하지 않습니다. 이벤트를 처리하기 위해서 반드시 **새로운 케이스를 설정**해야만 합니다. 동일한 이벤트를 여러 케이스가 처리하도록 설정한 경우, VI는 깨집니다.

이벤트 구조에서 처리되는 이벤트 설정하기

이벤트 구조에서 처리할 이벤트를 설정하기 이전에, LabVIEW에서 **이벤트 사용에 대한 유의점과 권장사항**을 살펴 보십시오.

다음 단계를 따라 이벤트를 처리하는 이벤트 구조 케이스를 설정합니다.

1. (옵션) 사용자 이벤트, 라디오 버튼 컨트롤을 갖는 불리언 컨트롤, 또는 어플리케이션, VI, 컨트롤에 대한 참조를 기반으로 생성된 사용자 인터페이스 이벤트를 처리하기 위해서 이벤트 구조를 설정하려는 경우, 처음에 반드시 **이벤트를 다이내믹하게 등록**해야 합니다.
2. 이벤트 구조의 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이 케이스에 의해 핸들되는 이벤트 편집을 선택하여 현재 케이스를 편집할 수 있는 이벤트 편집 대화 상자를 엽니다. 또한 바로 가기 메뉴에서 이벤트 케이스 추가를 선택하여 **새 케이스를 생성**할 수 있습니다.
3. 이벤트 소스에 이벤트 소스를 지정합니다.
4. 키 다운, 타임아웃, 또는 값 변경과 같이 이벤트 소스에 설정하려는 이벤트를 이벤트 리스트에서 선택합니다. 이벤트 소스 리스트에서 다이내믹 이벤트 소스를 선택하면 이벤트 리스트는 선택한 이벤트를 보여줍니다. 이것은 **이벤트를 등록**했을 때 선택한 이벤트와 동일한 이벤트입니다. 이벤트를 다이내믹하게 등록하고 이벤트 등록 참조 번호 출력을 다이내믹 이벤트 터미널에 연결한 경우, 소스는 다이내믹 섹션에 나타납니다.
5. 현재 케이스에서 처리할 추가적인 이벤트를 추가하려는 경우, 파란색 더하기 버튼  을 클릭한 후 단계 3과 4를 반복하여 각 추가 이벤트를 지정합니다. 대화 상자 상단의 이벤트 지정자 섹션은 케이스에서 처리할 모든 이벤트를 나열합니다. 이 리스트에서 아이템을 클릭할 때 이벤트 소스 섹션은 선택한 이벤트 소스를 하이라이트하기 위해 업데이트합니다. 단계 3과 단계 4를 반복하여 각 이벤트를 다시 정의하거나 빨간색 X 버튼  을 클릭하여 선택된 이벤트를 제거할 수 있습니다.
6. 확인 버튼을 클릭하여 설정을 저장하고 대화 상자를 닫습니다. 설정한 이벤트 케이스는 이벤트 구조의 위에 있는 이벤트 선택자 라벨에서 선택 옵션으로 나타나며, 이벤트 데이터 노드는 이 케이스에서 처리된 모든 이벤트에 공통적인 데이터를 디스플레이합니다.
7. (옵션) 타임아웃 이벤트를 사용하여 이벤트 구조를 설정하여 이벤트가 발생하는 특정 시간을 기다릴 수 있습니다. 이벤트 구조의 왼쪽 위의 타임아웃 터미널에 값을 연결하여 타임아웃 이벤트를 생성하기 전에 이벤트 구조가 이벤트 발생을 기다려야 하는 밀리초 수를 지정합니다. 타임아웃 기본값은 -1이며, 이는 이벤트 발생을 무한히 기다리도록 지정합니다.
8. 설정하려는 각 이벤트 케이스에 대해 1 단계부터 6 단계까지 반복합니다.

이벤트 사용에 대한 예제는 다음 VI를 참조하십시오:

- labview\examples\Dialog and User Interface\Events\\Events.lvproj  예제 열기  관련 예제 찾기
- labview\examples\Structures\Event Structure\Event Structure.lvproj  예제 열기  관련 예제 찾기

사용자 이벤트 생성하기

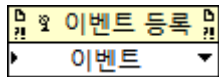
사용자가 직접 정의한 데이터를 사용하는 사용자 이벤트를 생성하고 이름을 지정할 수 있습니다. 사용자 이벤트를 사용하면 어플리케이션의 각각 다른 부분이 **비동기적 통신**을 할 수 있도록 합니다. 사용자가 직접 프런트패널 객체와 상호작용해야 하는 **사용자 인터페이스 이벤트**와는 달리, 사용자 이벤트를 사용하면 사용자는 객체의 프로그램적인 변경 사항에 응답하는 어플리케이션을 생성할 수 있습니다. 하나의 이벤트 구조에서 사용자 인터페이스 이벤트와 프로그램적으로 생성되는 사용자 이벤트를 모두 처리할 수 있습니다.

사용 객체

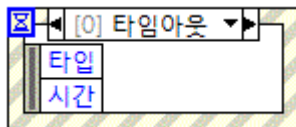
다음 블록다이어그램 객체를 사용하여 사용자 이벤트를 생성합니다.



—사용자 이벤트 생성



—이벤트 등록



—이벤트 구조



—사용자 이벤트 발생



—이벤트 등록 해제



—사용자 이벤트 삭제

실행 단계

다음 블록다이어그램을 생성하여 사용자가 정의하는 데이터를 프로그램적으로 전달하는 사용자 이벤트를 만듭니다.

While Loop

위의 블록다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.

1 —사용자 이벤트를 정의하려면, 블록다이어그램 객체를 [사용자 이벤트 생성] 함수에 연결합니다. [사용자 이벤트 생성] 함수의 입력은 개별 원소이거나 클러스터이며, 해당 입력의 이름과 데이터 타입이 사용자 이벤트의 이름과 데이터 타입을 정의합니다. 또한 LabVIEW는 **고유 라벨**을 사용하여 사용자 이벤트가 사용하는 데이터를 식별하므로, 입력 데이터 또는 입력 데이터의 각 원소에는 설명적인 고유 라벨을 붙입니다. 사용자 이벤트의 입력 데이터에는 라벨을 붙여야 이벤트 구조의 이벤트 데이터 노드에 라벨이 나타납니다. 위의 블록다이어그램에서 사용자 이벤트의 이름은 내 사용자 이벤트이고 이벤트 데이터 아이템의 이름은 문자열입니다.

2 —[사용자 이벤트 생성] 함수의 사용자 이벤트 출력은 그 사용자 이벤트의 이름과 데이터 타입을 사용하는 엄격한 타입의 참조 번호입니다. 이벤트를 등록하고 사용자 이벤트를 생성할 때에는, 이벤트 이름 및 데이터 타입에 대한 참조 번호를 사용하십시오.

3 —[이벤트 등록] 함수를 사용하여 **다이나믹하게 이벤트를 등록**합니다. 사용자 이벤트와 사용자 인터페이스 이벤트의 조합을 [이벤트 등록] 함수에 연결할 수 있습니다.

4 —[이벤트 등록] 함수의 이벤트 등록 참조 번호 출력을 이벤트 구조의 왼쪽에 있는 다이나믹 이벤트 터미널에 연결합니다. 다이나믹 이벤트 터미널을 디스플레이하려면, 이벤트 구조에서 마우스 오른쪽 버튼을 클릭한 후 다이나믹 이벤트 터미널 보이기를 선택합니다. 사용자 이벤트 데이터 아이템은 이벤트 구조의 왼쪽 경계에 있는 이벤트 데이터 노드에 나타납니다.

5 — 이벤트 데이터 노드에 더 많은 데이터 아이템을 추가하려면, 노드를 수직으로 크기 조정합니다. 이벤트 데이터 노드를 사용하여 특정 이벤트와 연관된 이벤트 데이터 원소에 접근할 수 있습니다.

6 — 이벤트 편집 대화 상자를 사용하여 이벤트를 처리할 이벤트 구조의 케이스를 설정합니다. 사용자 이벤트의 이름이 대화 상자의 이벤트 소스 섹션에서 다이내믹 부표제 아래에 나타납니다.

7 — [사용자 이벤트 발생] 함수를 사용하여 이벤트를 처리를 위해 설정된 이벤트 구조를 통해서 사용자 이벤트 및 연계된 데이터를 어플리케이션의 다른 부분으로 전달합니다. [사용자 이벤트 발생] 함수는 사용자 이벤트 참조 번호와 이벤트 데이터의 값을 받습니다. 데이터 값은 반드시 사용자 이벤트의 데이터 타입과 일치해야만 합니다.



노트

사용자 이벤트가 등록되지 않은 경우 [사용자 이벤트 발생] 함수는 작동하지 않습니다. 사용자 이벤트가 등록되었지만 대기 중인 이벤트 구조가 없을 경우, LabVIEW는 이벤트 구조가 이벤트를 처리할 때까지 사용자 이벤트와 데이터를 큐에 저장합니다.

8 — [이벤트 등록 해제] 함수 및 [사용자 이벤트 삭제] 함수를 사용하여 연계된 이벤트 및 사용자 이벤트 참조를 해제할 수 있습니다. LabVIEW는 최상위 VI가 실행을 끝내면 자동으로 모든 이벤트를 등록 해제하고 기존의 사용자 이벤트를 삭제합니다. 하지만 메모리 자원을 절약하기 위해, National Instruments는 특히 오랜 시간 동안 실행되는 어플리케이션의 사용자 이벤트를 명확하게 등록 해제하고 삭제할 것을 권장합니다.

유의점 및 권장사항

- 사용자 이벤트를 정적으로 등록할 수는 없습니다.
- 별개의 [이벤트 등록] 함수를 사용하여 하나의 사용자 이벤트를 여러 번 등록할 수 있습니다. 이런 경우, 이벤트 등록 참조 번호에 연계된 각 큐는 사용자 이벤트의 복사본과 [사용자 이벤트 발생] 함수가 매번 실행하는 연계된 이벤트 데이터를 받습니다.
- 사용자 이벤트는 **알림 이벤트**이며, 사용자 인터페이스 이벤트 또는 다른 사용자 이벤트로 이벤트 구조의 동일한 이벤트 케이스를 공유할 수 있습니다.
- 기존의 사용자 인터페이스 이벤트와 이름 및 데이터 타입이 같은 이벤트 데이터 아이템을 가진 사용자 이벤트를 만들어서 프런트패널과 사용자의 상호 작용을 시뮬레이션할 수 있습니다. 예를 들어, OldVal과 NewVal로 명명된 두 불리언 필드의 클러스터를 사용하여 MyValChg로 불리는 사용자 이벤트를 생성할 수 있습니다. 이때 두 불리언 필드는 값 변경 사용자 인터페이스 이벤트가 불리언 컨트롤에 연계시키는 것과 같은 이벤트 데이터 아이템입니다. 같은 이벤트 구조 케이스를 시뮬레이션된 MyValChg 사용자 이벤트와 실제 불리언 값 변경 이벤트에서 공유할 수 있습니다. 이벤트 구조는 [사용자 이벤트 생성] 함수가 사용자 이벤트를 발생시키거나 사용자가 컨트롤의 값을 변경하는 경우 이벤트 케이스를 실행합니다.

예제

다이내믹한 이벤트 등록에 관한 예제는 labview\examples\Dialog and User Interface\Events\Events.lvproj를 참조하십시오.

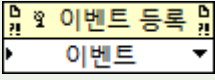
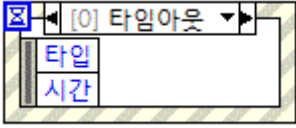
사용자 이벤트 사용에 대한 예제는 labview\examples\Dialog and User Interface\Events 디렉토리에 있는 User Event Generation VI를 참조하십시오.

다이나믹하게 이벤트 등록하기

다이나믹 이벤트 등록은 어플리케이션, VI 또는 컨트롤과 같은 VI 서버 참조를 사용하여 실행 시에 LabVIEW에서 이벤트가 발생할 객체를 지정할 수 있습니다.

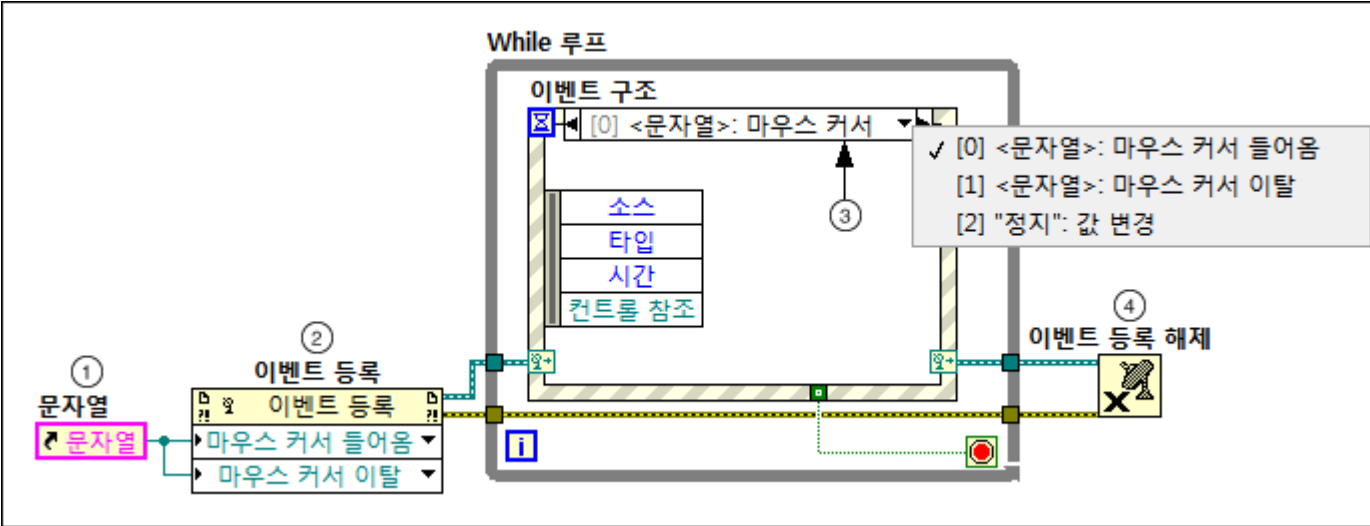
사용 객체

다음 블록다이어그램 객체를 사용하여 다이나믹하게 이벤트를 등록합니다.

이벤트 등록	While 루프	이벤트 구조	이벤트 등록 해제
			

다음 단계

다음 블록다이어그램을 생성하여 문자열 컨트롤에서 마우스 커서 들어옴 및 마우스 커서 이탈 이벤트를 다이나믹하게 등록합니다.



위의 블록다이어그램에 대한 중요한 상세 정보가 다음 리스트에 설명되어 있습니다.

- ①
- LabVIEW에서 이벤트를 발생시키려는 객체에 대한 **VI 서버** 참조를 얻습니다. 위의 그림에서의 문자열 컨트롤 상수와 같은 컨트롤 참조 상수를 얻으려면, 프로퍼티 노드를 사용하여 VI의 컨트롤을 쿼리하거나 해당 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>>참조를 선택합니다.

②

[이벤트 등록] 함수를 사용하여 이벤트를 다이나믹하게 등록합니다. 이 함수의 크기를 조정하고 다른 이벤트를 선택하여 하나의 참조에 여러 개의 이벤트를 등록시킬 수 있습니다. 이벤트 소스 입력의 아래쪽 화살표를 클릭하고, 위의 그림에서 보이는 마우스 커서 들어옴, 마우스 커서 이탈 이벤트와 같이 LabVIEW로 생성하려는 이벤트를 선택합니다. 등록 후 LabVIEW는 이벤트 구조가 이벤트를 처리할 때까지 이벤트를 큐에 저장합니다.



팁

커서를 이벤트 등록 참조 번호 위로 움직이면 기본 도움말 윈도우에서 등록된 이벤트를 볼 수 있습니다.

③

이벤트 선택자 라벨에서 마우스 오른쪽 버튼을 클릭하여 이벤트 편집 대화 상자를 디스플레이합니다. 대화 상자를 사용하여 다이나믹하게 등록된 이벤트를 처리할 수 있도록 **이벤트 구조를 설정**합니다. 대화 상자의 이벤트 소스 섹션은 각 다이나믹 등록 이벤트 소스를 나열하는 다이나믹 부표제를 포함합니다. 이벤트 구조로 처리할 이벤트를 선택한 후에는, 이벤트 데이터를 처리할 구조를 설계합니다.

④

[이벤트 등록 해제] 함수를 사용하여 이벤트 발생을 종료합니다. [이벤트 등록 해제] 함수가 실행되면, LabVIEW는 이벤트 등록 참조 번호가 지정하는 모든 이벤트를 등록 해제하고, 관련된 이벤트 큐를 없애고, 그리고 큐에 남아 있는 다른 이벤트들도 없앱니다.

다이나믹하게 등록된 이벤트의 유의점과 권장사항

- **기존 등록을 수정**하려는 때에만 [이벤트 등록] 함수의 이벤트 등록 참조 번호 입력을 연결합니다.
- 다이나믹 이벤트를 처리하는 각 이벤트 구조를 각각의 [이벤트 등록] 함수에 연결합니다. [이벤트 등록] 함수의 이벤트 등록 참조 번호 가지를 분기하면, 여러 개의 이벤트 구조가 하나의 큐에서 복수의 이벤트를 생성하도록 하여 예상치 못한 동작을 일으킬 수 있는 경합 조건을 발생시킬 수 있습니다.
- 이벤트 구조에서 처리할 이벤트를 설정하기 이전에, LabVIEW에서 **이벤트 사용에 대한 유의점과 권장사항**을 살펴 보십시오.

예제

다이나믹한 이벤트 등록에 관한 예제는 labview\examples\Dialog and User Interface\Events\Events.lvproj를 참조하십시오.



예제 열기  관련 예제 찾기

다이나믹하게 이벤트 등록 수정하기

다이나믹하게 이벤트를 등록하는 경우, LabVIEW가 이벤트 및 이벤트를 발생시키는 객체를 등록할 때 등록 정보가 변경되도록 수정할 수 있습니다. 예를 들어, 프런트패널 컨트롤

롤에서 마우스 이동 이벤트를ダイナミック하게 등록할 수 있고, 사용자가 어떤 버튼을 먼저 클릭했을 때에만 LabVIEW가 이벤트를 생성하도록 등록을 수정할 수 있습니다.



노트
정적으로 등록된 이벤트를 실행 중에 수정할 수 없습니다.

다음 단계에 따라 등록 정보를 수정합니다:

- 1. 특정 상황에서만 발생하기를 원하는 이벤트에 자리 표시자를 생성합니다.
- 2. LabVIEW가 언제 해당 이벤트를 등록하기를 원하는지 지정하는 이벤트 구조를 설계합니다.
- 3. 해당 이벤트의 등록을 먼저 해제한 후, 모든 이벤트의 등록을 해제합니다.

예제

다이내믹한 이벤트 등록에 관한 예제는 labview\examples\Dialog and User Interface\Events\Events.lvproj를 참조하십시오.



예제 열기



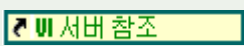
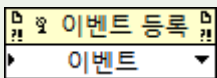
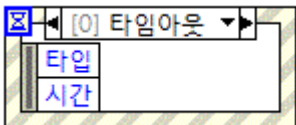
관련 예제 찾기

단계 1: 특정 상황에서만 발생하기를 원하는 이벤트에 자리 표시자를 생성합니다.

자리 표시자를 사용하여 어플리케이션에 특정 조건이 발생하였을 때만 모니터하려는 이벤트를 등록합니다. 자리 표시자는 특정 이벤트에 대해 [이벤트 등록] 함수의 엔트리를 생성하지만, LabVIEW가 자리표시자의 객체에 대해 해당 이벤트를 발생시키는 것은 허용하지 않습니다. [이벤트 등록] 함수의 참조 번호 출력을 다른 [이벤트 등록] 함수의 참조 번호 입력에 연결하면, 그 참조 번호를 받는 함수는 기존의 등록 정보를 그대로 갖게 됩니다. 그러므로 자리표시자 객체를 유효한 객체 참조로 대체하여 [이벤트 등록] 함수의 등록 정보를 수정해야 합니다.

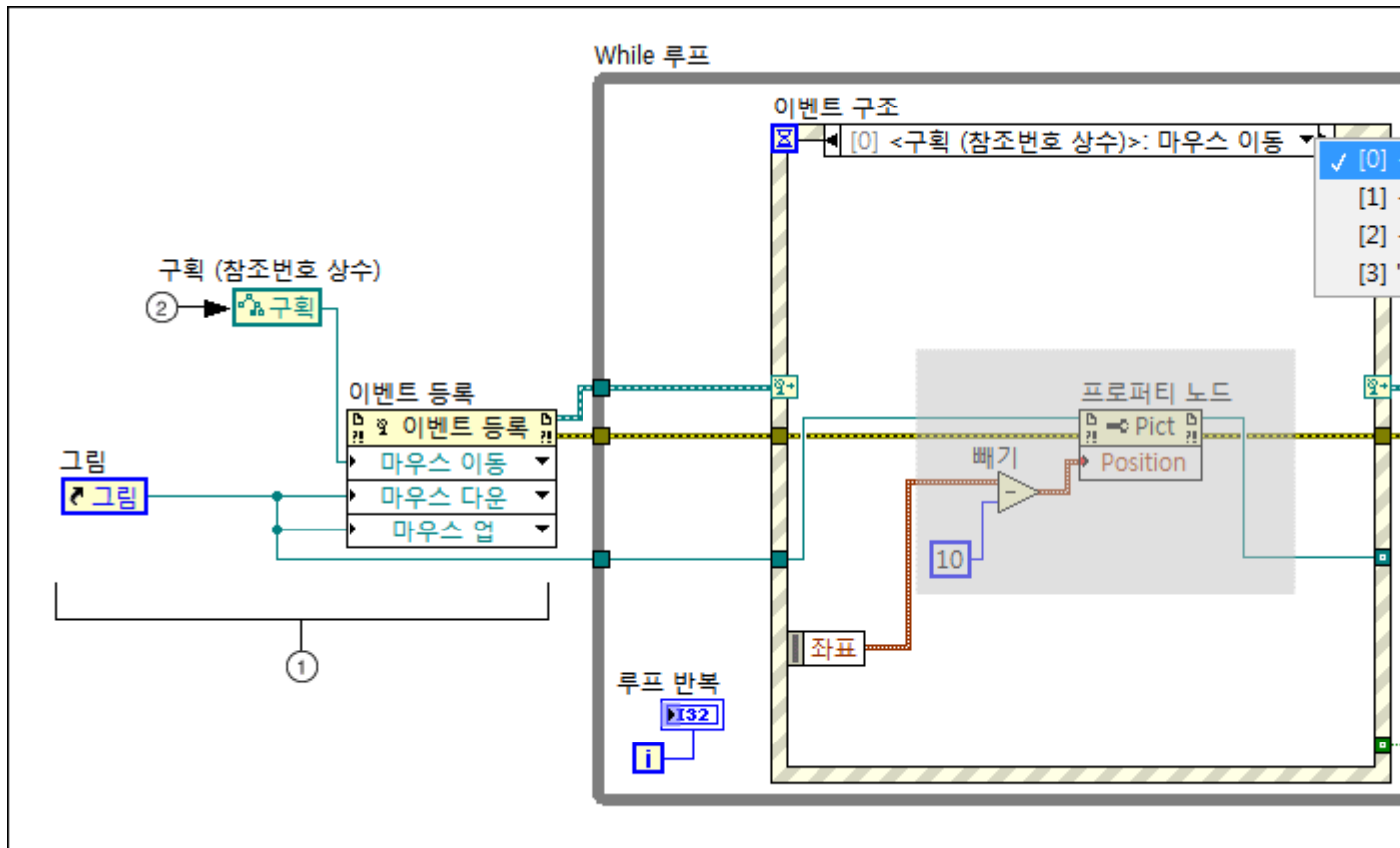
사용 객체

다음 블록 다이어그램 객체를 사용하여 사용자 이벤트 등록을 다이내믹하게 수정합니다.

VI 서버 참조	이벤트 등록	While 루프	이벤트 구조
			

실행 단계

다음 블록 다이어그램 객체를 사용하여 이벤트를 다이나믹하게 등록하고 어플리케이션의 다른 위치에 등록하려는 이벤트에 대한 자리 표시자를 생성합니다. 이 예는 등록을 다이나믹하게 수정하여, 사용자가 프런트패널의 그림 컨트롤을 클릭하였을 때만 VI 구획의 마우스 이동 이벤트가 트리거되도록 합니다.



다음 리스트는 이전 단계의 중요한 상세 정보에 대한 설명입니다.

1. 그림 컨트롤 참조에 대해 마우스 다운 및 마우스 업 이벤트를 **다이나믹하게 등록**합니다.
2. 이벤트에 대한 자리 표시자를 생성하려면, [이벤트 등록] 함수에 참조 번호 상수를 연결합니다. 참조 번호 상수의 값이 참조 번호 아님이기 때문에, LabVIEW는 마우스 이동 이벤트를 유효하지 않은 객체에 등록합니다. 그러므로 데이터 흐름에 의해 LabVIEW가 유효한 객체에 이벤트를 등록하게 될 때까지 마우스 이동 이벤트는 발생하지 않습니다. 이전 블록 다이어그램에서, 현재 vi의 구획에 대한 엄격한 타입의 참조 번호 상수는 마우스 이동 이벤트에 연결되어 있습니다. 이 참조 번호 상수를 생성하려면, vi 서버 참조를 클릭한 후 바로 가기 메뉴에서 구획을 선택합니다. 구획 참조에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 상수를 선택합니다.

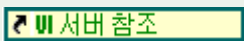
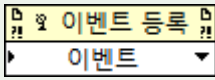
특정 상황에서만 발생하기를 원하는 이벤트에 대한 자리 표시자를 등록한 후, LabVIEW가 언제 이벤트를 등록하기를 원하는지 지정하는 이벤트 구조를 설계합니다.

단계 2: LabVIEW가 언제 해당 이벤트를 등록하기를 원하는지 지정하는 이벤트 구조를 설계합니다.

이 예제의 마우스 다운 이벤트 구조에는 마우스 이동 이벤트를 등록하는 [이벤트 등록] 함수가 들어 있습니다. 사용자가 프런트패널의 그림 컨트롤을 클릭하면, 이벤트 구조가 실행되며 마우스 다운 이벤트를 처리하고 마우스 다운 이벤트 케이스의 코드가 실행되며 마우스 이동 이벤트를 구획 참조에 등록합니다.

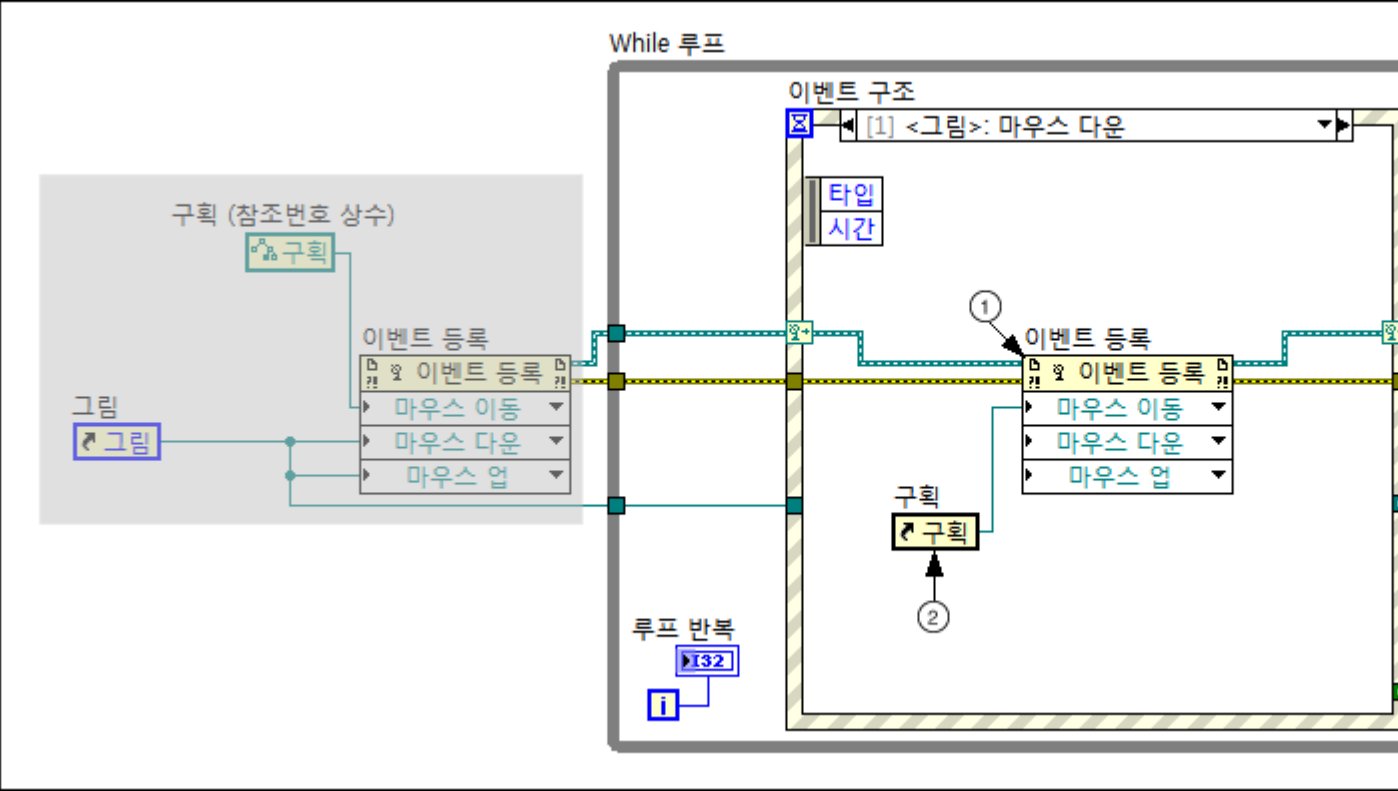
사용 객체

다음 블록 다이어그램 객체를 사용하여 사용자 이벤트 등록을ダイナミック하게 수정합니다.

VI 서버 참조	이벤트 등록
	

실행 단계

다음 블록 다이어그램을 생성하여 LabVIEW가 마우스 이동 이벤트를 실행 시 유효한 객체에 등록하도록 합니다.



다음 리스트는 이전 단계의 중요한 상세 정보에 대한 설명입니다.

①	기존의 등록을 수정하려면, [이벤트 등록] 함수의 이벤트 등록 참조 번호 출력을 다른 [이벤트 등록] 함수의 왼쪽 위 입력에 연결합니다. 이벤트 등록 참조 번호 입력을 연결하면, 함수는 자동으로 크기 조정되어 원래 이벤트 등록 참조 번호를 생성한 [이벤트 등록] 함수에 지정된 것과 동일한 참조 타입의 동일한 이벤트를 보여줍니다. 이벤트 등록 참조 번호 입력이 연결되어 있는 동안에는 수동으로 함수를 크기 조정하거나 다시 설정할 수 없습니다.
②	마우스 이동 이벤트에 구획 참조를 연결하여 등록된 자리 표시자를 대체합니다. 사용자가 그림 컨트롤을 클릭하면 이벤트 구조가 실행되며 마우스 다운 이벤트를 처리하고, LabVIEW는 마우스 이동 이벤트를 등록합니다.

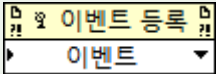
LabVIEW가 언제 이벤트를 등록할지 지정하도록 이벤트 구조를 설계하고, **해당 이벤트의 등록을 먼저 해제한 이후 모든 이벤트의 등록을 해제합니다.**

단계 3: 해당 이벤트의 등록을 먼저 해제한 후, 모든 이벤트의 등록을 해제합니다.

LabVIEW에서 특정 상황에서만 발생하기를 원하는 이벤트가 등록된 후에는, 사용자가 해당 이벤트의 등록 해제를 지정할 때까지 계속하여 이벤트가 생성됩니다. 이벤트 케이스를 설계하여, LabVIEW에서 이벤트가 등록 해제될 상황을 지정할 수 있습니다. 이 예에서는 사용자가 프런트패널의 그림 컨트롤을 클릭하면 LabVIEW에서 마우스 이동 이벤트가 등록되고, 사용자가 마우스 버튼을 누르면 마우스 이동 이벤트의 등록이 해제됩니다.

사용 객체

다음 블록 다이어그램 객체를 사용하여 사용자 이벤트 등록을ダイナミック하게 수정합니다.

참조 번호 아님 상수	이벤트 등록	이벤트 등록 해제
		

실행 단계

다음 블록 다이어그램을 생성하여 마우스 이동 이벤트의 등록을 해제하고, 마우스 다운 이벤트가 발생한 경우가 아니라면 LabVIEW에서 해당 이벤트가 생성되지 않도록 합니다.

프런트패널 잠그기

기본적으로, 이벤트가 큐를 입력할 때, LabVIEW는 이벤트를 생성시킨 객체가 있는 프런트패널을 잠급니다. LabVIEW는 모든 이벤트 구조가 이벤트를 다루기를 끝낼 때까지 프런트패널을 잠겨 있는 상태로 유지합니다. 프런트패널이 잠겨 있는 동안 LabVIEW는 프런트패널의 동작을 처리하지는 않지만, 이러한 상호작용을 버퍼에 저장한 후 프런트패널이 잠금해제되면 이를 처리합니다. 예를 들어 이벤트 구조에서 사용자가 텍스트를 입력해야 하는 어플리케이션을 시작하는 경우, 어플리케이션이 프런트패널에 나타나기 전에 타이핑을 시작할 수도 있습니다. 이 이벤트의 이벤트 케이스가 완료될 때까지 프런트패널 잠금 옵션을 활성화한 경우, 일단 어플리케이션이 시작되고 프런트패널에 나타나면 키 조작을 발생한 순서대로 처리합니다. 이 이벤트의 이벤트 케이스가 완료될 때까지 프런트패널 잠금 옵션이 비활성화된 경우, LabVIEW가 이벤트 케이스 완료에 따라 키 조작 실행을 큐에 저장하지 않기 때문에 키 조작이 프런트패널의 다른 곳에서 처리될 수도 있습니다.



노트

프런트패널을 잠근 이벤트 케이스를 실행하는데 많은 시간이 걸리는 경우, [사용 중 설정] 함수를 사용하거나 **대화 상자를 디스플레이**하여 프런트패널이 잠겼음을 알리는 것을 고려하십시오.

프런트패널 잠금은 윈도우를 움직이고, 스크롤 막대를 사용하고, 실행 강제 종료 버튼을 클릭하는 것과 같은 특정 동작에는 영향을 미치지 않습니다.



주의 이벤트 처리를 위해 신속하게 실행되는 이벤트 구조가 없고 프런트패널 잠금이 활성화된 경우, VI의 사용자 인터페이스가 반응하지 않을 수 있습니다. 이런 상황이 발생할 경우, 실행 강제 종료 버튼을 클릭하여 VI를 정지합니다. 이벤트 구조에서 마우스 오른쪽 버튼을 클릭한 후 이벤트 편집 대화 상자의 이 이벤트의 이벤트 케이스가 완료될 때까지 프런트패널 잠금 확인란에서 확인 표시를 제거하여 프런트 패널 잠금을 비활성화할 수 있습니다.

알림 이벤트의 프런트패널 잠금을 비활성화하여, 이벤트가 큐를 입력할 때 이벤트를 생성한 컨트롤을 포함하여 모든 컨트롤이 여전히 이벤트를 생성하도록 만들 수 있습니다. 여러 이벤트가 발생할 때, LabVIEW는 이를 처리하기 위해 프런트패널이 잠기기를 기다리지 않습니다. 다음 단계를 따라 프런트패널 잠금을 비활성화 합니다.

1. 이벤트 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이 케이스에 의해 핸들되는 이벤트 편집을 선택합니다.
2. 이 이벤트의 이벤트 케이스가 완료될 때까지 프런트패널 잠금 확인란에서 확인 표시를 제거합니다.



노트 LabVIEW는 해당 케이스가 완료될 때까지 필터 이벤트의 처리를 끝낼 수 없기 때문에 필터 이벤트에서는 이 옵션을 비활성화할 수 없습니다. LabVIEW가 필터 이벤트를 처리하는 동안 다른 이벤트를 처리하도록 허용하면 이벤트가 제대로 실행되지 않을 수 있습니다.

실행 중 큐에 추가된 이벤트 보기

다음 단계를 따라 이벤트 큐에서 대기 중인 이벤트를 봅니다:

1. VI를 실행합니다.
2. 이벤트 구조 경계에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 이벤트 검사 윈도우를 선택합니다. 이벤트 큐에서 처리되지 않은 이벤트 테이블은 큐에서 대기 중인 이벤트를 디스플레이합니다. 이벤트 구조가 이벤트를 처리하면, 이벤트는 이벤트 로그에 나타납니다.
4. (옵션) 이벤트 구조를 포함하는 활성 VI 풀다운 메뉴를 사용하여, 해당 이벤트에 대해 등록된 이벤트 구조 또는 사용자가 윈도우를 연 이후 이벤트를 처리한 이벤트 구조들을 포함하는 활성 VI의 이름을 볼 수 있습니다.
5. (옵션) 파일에 로그 저장을 클릭하여 이벤트 로그를 텍스트 파일에 저장합니다.
6. (옵션) 모두 지우기를 클릭하여 이벤트 구조를 포함하는 모든 실행 VI에 대한 이벤트 로그를 지우고, 실행되지 않는 이벤트 구조에 대한 이벤트 로그를 삭제합니다. 지우기를 클릭하여 현재 실행 중인 VI에 대한 이벤트 로그만 삭제합니다.

LabVIEW에서 이벤트를 사용할 때의 유의점과 권장사항

LabVIEW는 그래픽 프로그래밍 인터페이스이기 때문에 이벤트 처리가 다른 프로그래밍 언어에서의 이벤트 처리와 다릅니다. 다음 리스트는 LabVIEW 어플리케이션에 이벤트를 포함시킬 때의 몇몇 유의점과 권장사항을 설명합니다.

- 이벤트가 발생할 때마다 이벤트 구조가 이벤트를 처리할 수 있도록 하십시오.
- 래치되는 불리언 컨트롤인 경우 그 터미널을 값 변경 이벤트 케이스 내에서 읽으십시오.
- 래치된 불리언 컨트롤에 대한 취소 동작을 처리하기 위해서 케이스 구조를 사용합니다.
- 여러 알림 이벤트를 처리하기 위해서 하나의 케이스를 설정할 때 경고를 사용합니다.
- 다른 이벤트 데이터를 갖는 여러 필터 이벤트를 처리하기 위해서 하나의 케이스를 설정할 수 없습니다.
- 이벤트 구조를 포함하는 While 루프가 래치된 정지 불리언 컨트롤의 값에 의해 끝난 경우, 이벤트 구조의 래치된 정지 불리언 컨트롤을 처리하는 것을 기억합니다.
- 프로그램으로 특정 프런트패널 객체를 모니터할 필요가 없을 경우, [프런트패널 작업 기다림] 함수 사용을 고려합니다.
- 사용자 인터페이스 이벤트는 오직 직접적인 사용자 상호작용에만 적용됩니다.
- 마우스 다운? 필터 이벤트를 갖는 이벤트 케이스에서 대화 상자를 사용하는 것을 피합니다.
- 하나의 루프 안에 두 개의 이벤트 구조를 놓지 마십시오.
- 같은 VI에서 여러 개의 이벤트 구조를 사용하지 않도록 합니다.
- 다이내믹 등록을 사용할 때, 각 이벤트 구조에 대해 [이벤트 등록] 함수를 갖는지 확인합니다.

- 같은 이벤트를 동시에 VI와 컨트롤 클래스에 등록할 경우, LabVIEW는 VI 이벤트를 먼저 생성합니다.
- 서브패널 컨트롤을 사용할 때, 서브패널 컨트롤을 포함하는 최상위 VI가 이벤트를 처리합니다.
- 현재 이벤트를 처리하면서 다른 이벤트를 발생하거나 처리하려는 경우, [이벤트 콜백 등록] 함수 사용을 고려합니다.
- 알림이나 필터 이벤트사이에서 선택할 때에는 주의하십시오. 알림 이벤트를 처리하도록 설정된 이벤트 케이스는 LabVIEW가 사용자 상호작용을 처리할지 또는 어떻게 처리할지에 영향을 줄 수 없습니다. LabVIEW가 사용자 상호작용을 처리할지 또는 어떻게 처리할지를 수정하려면, 필터 이벤트를 사용합니다.
- 중요 종료 코드에 대해 패널 닫기 알림 이벤트를 사용하지 마십시오. 단, 패널을 닫을 때 VI가 강제 종료되지 않도록 조치한 경우에는 사용할 수 있습니다. 예를 들어, 사용자가 프론트패널을 닫기 전에 어플리케이션이 VI에 대한 참조를 열기하도록 합니다. 또는 패널 닫기? 필터 이벤트를 사용하여 프론트패널을 닫기 전에 이벤트가 발생하도록 할 수 있습니다.
- 이벤트 처리를 위해 신속하게 실행되는 이벤트 구조가 없고 프론트패널 잠금이 활성화된 경우, VI의 사용자 인터페이스가 반응하지 않을 수 있습니다. 이벤트 구조가 사용자 인터페이스 이벤트에 신속하게 반응하려면 이벤트 구조를 자주 실행되는 루프 안에 두어야 하고 프론트패널 잠금을 비활성화해야 합니다.



관련 정보

[이벤트가 발생할 때마다 이벤트 구조가 이벤트를 처리할 수 있도록 하기](#) (1272 페이지 참조)

[래치된 불리언 컨트롤을 이용한 이벤트 사용하기](#) (1277 페이지 참조)

[래치된 불리언 컨트롤을 이용한 이벤트 사용하기](#) (1277 페이지 참조)

[한 케이스가 여러 알림 이벤트 처리하기](#) (1276 페이지 참조)

[한 케이스가 여러 필터 이벤트 처리하기](#) (1275 페이지 참조)

[이벤트 구조에서 래치된 정지 불리언 컨트롤 핸들링](#) (1274 페이지 참조)

[이벤트 사용 또는 프론트패널 작업 기다림 함수 사용](#) (1277 페이지 참조)

[사용자가 직접 상호 작용할 때에만 이벤트 사용하기](#) (1276 페이지 참조)

[마우스 다운? 이벤트 케이스에서 대화 상자를 가지고 사용자에게 입력요청하기](#) (1276 페이지 참조)

[하나의 루프에 두 개의 이벤트 구조를 사용하는 것을 피하기](#) (1272 페이지 참조)

[가능하면 하나의 이벤트 구조만 사용하기](#) (1276 페이지 참조)

[다이나믹하게 이벤트 등록하기](#) (1262 페이지 참조)

[서브패널 컨트롤과 함께 이벤트 사용하기](#) (1277 페이지 참조)

[어떤 타입의 사용자 인터페이스 이벤트를 사용할지 결정하기](#) (407 페이지 참조)

[다이나믹하게 VI 로딩 및 호출하기](#) (170 페이지 참조)

[프론트패널 잠그기](#) (1269 페이지 참조)

[이벤트 타입](#) (405 페이지 참조)

하나의 루프에 두 개의 이벤트 구조를 사용하는 것을 피하기

National Instruments는 루프에 단지 하나의 이벤트 구조만을 사용하도록 권장합니다. 이벤트가 이 설정에서 발생할 때, 이벤트 구조는 이벤트를 다루고, 루프는 반복되고, 다음 이벤트가 일어날 때까지 이벤트 구조는 기다립니다. 단일 루프에 2개의 이벤트 구조를 놓으면 이 2개의 이벤트 구조가 하나의 이벤트를 다룰 때까지 루프는 반복할 수 없습니다. 이벤트 구조를 위해 고정되어 있는 프런트패널을 활성화하면, 사용자가 프런트패널과 어떻게 상호작용할 것인가에 의해 VI의 사용자 인터페이스는 응답하지 않을 수 있습니다.

예를 들면, 2개의 이벤트 구조를 단일 While 루프에 놓고 처음 이벤트 구조를 설정하여 마우스 다운 이벤트를 다루고 두 번째 이벤트 구조를 설정하여 키 다운 이벤트를 다루면, 처음 이벤트 구조는 사용자가 마우스 버튼을 클릭할 때 마우스 다운 이벤트를 받습니다. 처음 이벤트 구조는 올바른 이벤트 케이스를 실행하고 실행을 끝냅니다. 그동안, 두 번째 이벤트 구조는 키 눌림이 일어나기를 기다립니다. 사용자가 키를 누를 때, 두 번째 이벤트 구조는 키 다운 이벤트를 받습니다. 두 번째 이벤트 구조가 이벤트를 다룰 때, While 루프는 반복됩니다. 사용자가 마우스 다운 이벤트, 키 다운 이벤트, 마우스 다운 이벤트, 키 다운 이벤트, 등을 생성하는 상호 교대를 계속하면, 이벤트 구조가 발생함에 따라 그 이벤트를 다루고 While 루프가 계속 반복되기 때문에 VI가 자연스럽게 실행됩니다.

그러나, 2개의 마우스 다운 이벤트로 하여금 키 다운 이벤트 사이에 개입하지 않고 순차적으로 발생하도록 하여 사용자가 마우스 버튼을 2번 클릭하면, 사용자 인터페이스가 멈춥니다. 사용자가 처음으로 마우스 버튼을 클릭하면, 이벤트 구조는 먼저 마우스 다운 이벤트를 받고, 이벤트를 다루고 실행을 끝냅니다. 그러나, 두 번째 이벤트 구조는 키 다운 이벤트를 계속해서 기다리고 While 루프가 다시 반복하지 않도록 막습니다. 사용자가 마우스 버튼을 두 번 클릭하면, LabVIEW는 두 번째 마우스 다운 이벤트를 생성하고 처음 이벤트 구조가 이벤트를 다룰 때까지 프런트패널을 잠급니다. 이 포인트에서 VI는 교착 상태가 됩니다. 처음 이벤트 구조가 While 루프가 반복할 때까지 실행될 수 없고, 두 번째 이벤트 구조가 키 다운 이벤트를 받고 다룰 때까지 While 루프는 반복할 수 없습니다. 프런트패널이 잠겨 있기 때문에, 키 다운 이벤트는 발생할 수 없습니다. 사용자가 VI를 강제 종료할 때까지 프런트패널이 잠겨 있고 응답하지 않습니다.

프런트패널이 잠기고 사용자 인터페이스가 멈추는 것을 피하기 위해, 단일 이벤트 구조에서 VI가 다루려고 하는 **모든 이벤트를 설정**하거나 While 루프안에 단지 하나의 이벤트 구조가 있다는 것을 확인합니다. 덧붙여서, 이벤트가 일어날 때 이벤트를 다루도록 **이용할 수 있는 이벤트 구조**가 항상 있다는 것을 확인합니다.

이벤트가 발생할 때마다 이벤트 구조가 이벤트를 처리할 수 있도록 하기

VI가 실행하거나 실행하도록 예약되었을 때 LabVIEW는 이벤트를 큐하기 시작합니다. 이벤트 구조는 **데이터 흐름**에 따라 구조가 실행될 때 큐된 이벤트를 처리합니다. 그러므로, LabVIEW는 이벤트 구조가 이벤트를 처리하기 위해 기다리기 전에도 이벤트를 생성할 수 있습니다.

LabVIEW는 VI가 실행되는 시간 전체에 대해 이벤트를 큐하기 때문에, 데이터 흐름으로 인해 이벤트 구조가 실행을 종료한 후에도 이벤트를 큐할 수 있습니다. 예를 들어 이벤트 구조를 포함하는 루프는 VI가 실행을 지속하는 동안에도 종료될 수 있습니다. 큐된 이벤트가 모두 처리되도록 하려면, 이벤트 구조가 실행을 종료한 후 곧 **VI의 실행이 정지되도록 설계**합니다.

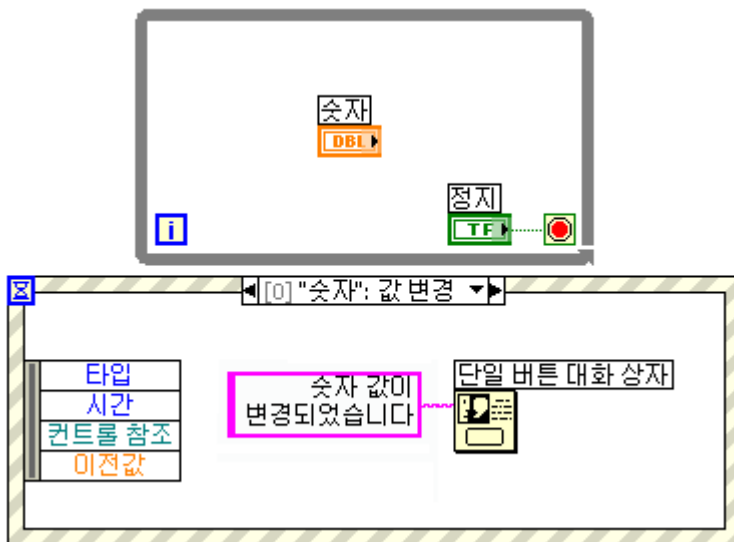


주의

이벤트 처리를 위해 신속하게 실행되는 이벤트 구조가 없고 **프론트패널 잠금**이 활성화된 경우, VI의 사용자 인터페이스가 반응하지 않을 수 있습니다. 이런 상황이 발생할 경우, 강제 종료 버튼을 클릭하여 VI를 정지합니다. 이벤트 구조에서 마우스 오른쪽 버튼을 클릭한 후 이벤트 편집 대화 상자의 이 이벤트의 이벤트 케이스가 완료될 때까지 프론트패널 잠금 확인란에서 확인 표시를 제거하여 프론트패널 잠금을 비활성화할 수 있습니다. 한편 필터 이벤트에 대한 프론트패널 잠금은 해제할 수 없습니다.

While 루프를 사용하여 이벤트 구조로 모든 이벤트를 처리하기

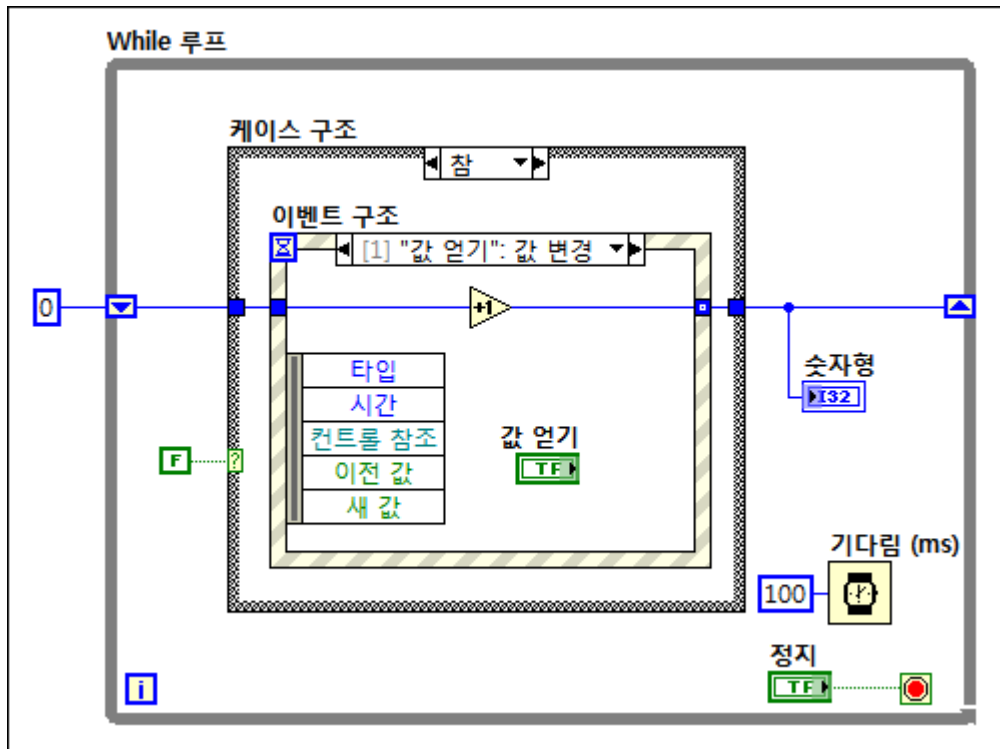
이벤트가 더 발생할 수 없게 될 때 종료하는 While 루프에 이벤트 구조를 놓습니다. 예를 들어, 다음 블록 다이어그램에서 이벤트 구조는 While 루프 밖에 있으며 숫자값 변경 케이스에 대해서 프론트패널 잠금이 활성화되어 있습니다.



숫자 컨트롤의 값을 변경할 경우, 이벤트가 발생합니다. 이벤트 구조는 한번 실행되고 값 변경 이벤트를 처리합니다. 숫자 컨트롤의 값을 다시 변경할 경우, 또 다른 이벤트가 발생하며 프론트패널 잠금이 활성화되어 있으므로 **사용자 인터페이스가 잠깁니다**. 이벤트 구조는 이미 한번 실행되었으며 While 루프내에 있지 않기 때문에 두번째 이벤트를 처리하기 위해서 다시 실행될 수 없습니다. 정지 불리언 컨트롤을 클릭하여 While 루프와 VI를 멈춘 경우, 블록 다이어그램에 아직 처리되지 않은 두번째 이벤트가 있기 때문에 VI는 멈추지 않으며 블록 다이어그램이 이벤트를 처리할 때까지 정지 불리언 컨트롤을 클릭할 때 LabVIEW는 이벤트를 처리하지 않습니다. 이러한 현상은 이벤트 구조를 While 루프 내에 놓아서 피할 수 있습니다.

케이스 구조의 사용을 피해 이벤트 구조로 이벤트를 처리할 시기를 결정하기

케이스 구조 내부에 이벤트 구조를 놓지 마십시오. 다음 블록 다이어그램의 예에서, 케이스 구조 내부에 이벤트 구조가 있고, 값 변경 케이스에 대해서 프론트패널 잠금이 활성화되어 있습니다.

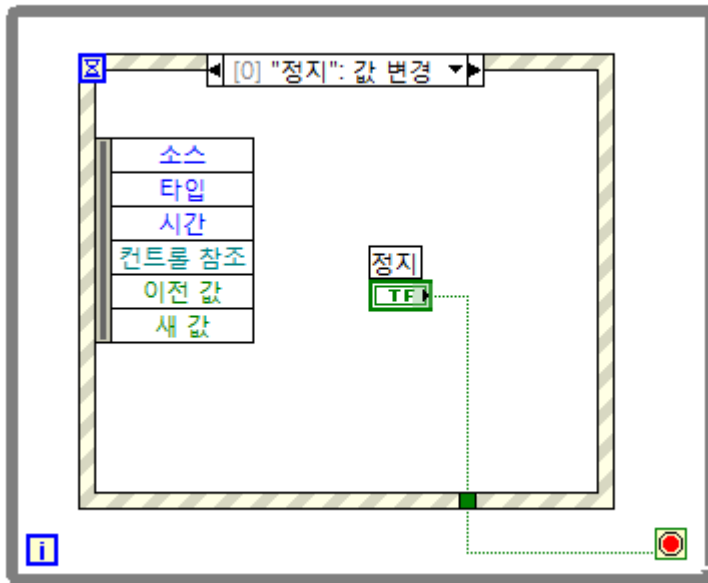


값 얻기 컨트롤을 클릭하면, 프런트패널은 무한히 잠금 상태가 됩니다. 케이스 구조에 거짓 불리언 상수를 연결하면, 이벤트 구조가 참 케이스 안에서 이벤트를 처리하는 것을 방지할 수 있습니다. 이벤트에 대해 패널 잠금이 활성화되어 있기 때문에 프런트패널은 이벤트 구조를 기다려야 하지만, 데이터 흐름으로 인해 이벤트 구조가 이벤트를 처리하도록 실행되지 못합니다. 이 어플리케이션에 대해 프런트패널 잠금을 비활성화하고 값 얻기 컨트롤을 클릭하면, 프런트패널은 응답 가능한 상태가 되고 정지 버튼을 누를 때 완료됩니다. 그러나 이벤트 구조가 실행되어 이벤트를 처리한 것이 아니므로 숫자형 값은 증가되지 않습니다.

이벤트 구조에서 래치된 정지 불리언 컨트롤 핸들링

래치된 정지 불리언 컨트롤을 처리하려면, 반드시 이벤트 구조 케이스를 설정해야 합니다. 래치된 정지 불리언 컨트롤이 있는 이벤트를 사용할 경우, 반드시 래치된 불리언 컨트롤의 기계적 동작에 대한 이벤트 구조 안에 컨트롤을 놓아서 올바르게 작동하도록 해야 합니다.

다음 예제는 이벤트 구조에서 래치된 정지 불리언 컨트롤을 다루는데 권장할 만한 방법을 보여줍니다. 이 예에서 사용자가 정지 버튼을 클릭하면, 어플리케이션은 실행을 중지하고 래치된 정지 불리언 버튼은 거짓으로 리셋됩니다.



이벤트 구조는 While 루프 안에 있고 값 변경 이벤트 구조가 래치된 정지 불리언 컨트롤에 대해 설정됩니다. 이 어플리케이션을 실행하면, 다음 순서로 실행이 완료됩니다:

1. 사용자가 정지 버튼을 클릭하면, 래치된 정지 불리언 컨트롤의 값이 참으로 변경되고, 이벤트 구조가 실행되어 "정지": 값 변경 이벤트 구조를 처리합니다.
2. "정지": 값 변경 이벤트 구조는 프런트패널 컨트롤을 읽고, 컨트롤 터미널에 참 값을 전달하고, 정지 버튼을 거짓으로 리셋합니다.
3. While 루프 조건 터미널이 거짓 값을 읽고 루프를 정지합니다.

관련 정보

래치된 불리언 컨트롤을 이용한 이벤트 사용하기

이벤트가 발생할 때마다 이벤트 구조가 이벤트를 처리할 수 있도록 하기

이벤트 구조에서 처리되는 이벤트 설정하기

불리언 객체의 기계적 동작 변경하기

한 케이스가 여러 필터 이벤트 처리하기

이벤트 구조의 한 케이스는 "키 다운?" 및 "키 반복?"과 같이 동일한 이벤트 데이터를 가진 이벤트가 아닌 이상 여러 필터 이벤트를 처리할 수 없습니다. 필터 이벤트에서, 프런트패널이 이벤트 데이터를 처리하기 전에 확인하거나 변경할 수 있고, 또는 이벤트 데이터 전체를 취소하여 VI에 영향을 주는 변경을 막을 수 있습니다. 데이터를 변경할 수 있기 때문에, 이벤트 구조는 알림 이벤트의 경우처럼 데이터를 병합할 수는 없습니다.

한 케이스가 여러 알림 이벤트를 처리하기

이벤트 구조의 한 케이스가 여러 알림 이벤트를 처리할 수 있습니다. 그러나, 이벤트가 같은 타입이 아닌 경우, 케이스가 처리하는 모든 이벤트에 공통된 이벤트 데이터 필드만 케이스의 왼쪽에 나타납니다. 같은 케이스에서 다른 데이터 타입을 가진 컨트롤의 여러 값 변경 이벤트를 처리하는 경우, 이전값과 새값 이벤트 데이터 필드는 배리언트 데이터 타입을 반환합니다.

하나의 케이스가 다른 VI 서버 클래스의 객체에 대한 여러 이벤트를 처리할 경우, 참조 타입은 모든 객체의 공통된 상위 클래스입니다. 예를 들어, 이벤트 구조에서 한 케이스를 숫자형 컨트롤과 색 램프 컨트롤의 이벤트를 처리하도록 설정할 경우, 이벤트 소스의 컨트롤 참조 타입은 숫자형입니다. 왜냐하면 숫자형 컨트롤과 색 램프 컨트롤은 숫자형 클래스에 속하기 때문입니다. 또한 단일 이벤트 케이스가 다이나믹하게 등록된 DAQmx 태스크의 이벤트를 처리하거나 LabVIEW 컨트롤과 같은 다른 이벤트 소스의 이벤트를 처리하도록 설정할 수 없습니다.

마우스 다운? 이벤트 케이스에서 대화 상자를 가지고 사용자에게 입력요청하기

사용자에게 입력을 요청하는 대화 상자를 띄우는 이벤트 케이스는 VI에서 예상하지 못한 동작을 발생시킬 수 있습니다. 예를 들면, 숫자 컨트롤과 함께 마우스 다운? 이벤트를 사용하고 이벤트 케이스를 대화 상자를 가지고 입력요청으로 설정하면, 이벤트 구조는 증가와 감소 화살표를 포함한 숫자 컨트롤을 클릭할 때 반응하고 대화 상자를 디스플레이합니다. 마우스 버튼이 계속 눌러진 동안 대화 상자가 나타나기 때문에, 대화 상자는 마우스 버튼을 놓을 때까지 포커스를 유지합니다. 그리고 그 대화 상자가 대신 알림 신호를 받기 때문에 숫자 컨트롤은 마우스 버튼의 상태 변화에 대해 전혀 알림 신호를 받지 못합니다. 대화 상자를 닫은 후에, 숫자 컨트롤은 이전 마우스 다운? 이벤트에 대해 계속 반응하고, 마우스 버튼을 더이상 누르고 있지 않아도 컨트롤을 무한히 계속 증가시키거나 감소시킬 수 있습니다.

이 문제를 피하기 위해, National Instruments는 마우스 다운? 이벤트를 다루는 이벤트 케이스 안에서 대화 상자를 사용하지 말 것을 권장합니다. 대신에, 마우스 업 이벤트 케이스 안이나 이벤트 구조 밖에서 대화 상자를 사용합니다.

사용자가 직접 상호 작용할 때에만 이벤트 사용하기

사용자 인터페이스 이벤트는 사용자가 직접 활성화된 프런트패널과 상호 작용할 때 발생합니다. 일반적으로, LabVIEW는 VI 서버, 공유 변수, 글로벌 변수, 로컬 변수, DataSocket 등을 사용하여 VI 또는 프런트패널 객체를 프로그램적으로 변경하는 경우에는 이벤트를 발생시키지 않습니다. 유일한 예외는 값(신호) 프로퍼티입니다.

가능하면 하나의 이벤트 구조만 사용하기

한 개의 VI 안에서 여러 이벤트 구조를 사용할 수 있으며 특히 이벤트를 다이나믹하게 등록한 경우에 유용합니다. 한편 정적으로 등록하는 이벤트는 여러 이벤트 구조를 필요로 하는 경우가 거의 없으므로, 가능하면 한 개의 이벤트 구조를 사용하여 이벤트를 처리하십시오.

한 개의 VI 안에서 여러 이벤트 구조를 사용할 때는 다음 가이드라인을 따르십시오:

- 이벤트 구조를 각 개별 While 루프 안에 놓습니다.
- 모든 이벤트 구조가 동시에 실행되도록 합니다.
- 모든 이벤트 구조가 동시에 이벤트 처리를 종료하도록 합니다.

이벤트 사용 또는 프런트패널 작업 기다림 함수 사용

사용자가 프런트패널과 상호작용할 때까지 VI를 기다리려는 경우 [프런트패널 작업 기다림] 함수를 사용합니다. 이 함수는 시스템 리소스를 절약하기 위해서 고안되었지만 어떤 이벤트가 발생했는지에 대해서는 어떠한 피드백도 제공하지 않습니다.

마우스 클릭, 키누름 등과 같이 프런트패널 상의 특정 사용자 동작에 대한 반응으로 다른 코드를 실행하려는 경우 이벤트 구조를 사용합니다.

래치된 불리언 컨트롤을 이용한 이벤트 사용하기

래치 기계적인 동작으로 설정된 불리언 컨트롤에 이벤트를 트리거하면, 불리언 컨트롤은 블록 다이어그램이 불리언 컨트롤의 터미널을 읽을 때까지는 기본적으로 리셋되지 않습니다. 정확하게 동작하기 위해서는 이벤트 케이스 내부에서 기계적 동작의 터미널을 읽어야만 합니다. 래치된 불리언 컨트롤의 값 변경 이벤트를 설정하면 이벤트 편집 대화 상자에 노트가 나타납니다.

래치 기계적인 동작으로 설정된 불리언 컨트롤에 취소 동작을 수행하면 이벤트 구조는 취소 동작을 값 변경 이벤트로 처리하며 이것은 예상치 못한 결과를 반환할 수 있습니다. 취소 동작을 정확히 처리하기 위해서는 반드시 케이스 구조를 이벤트 케이스 내에서 사용해야만 합니다. 케이스 구조의 참 케이스를 사용하여 불리언 컨트롤을 어떻게 처리할지를 지정하고 거짓 케이스를 사용하여 취소 동작을 어떻게 처리할지를 지정합니다.

래치된 정지 불리언 컨트롤을 어떻게 처리할지에 대한 정보는 **이벤트 구조에서 래치된 정지 불리언 컨트롤 처리하기** 유의점을 참조하십시오.

서브패널 컨트롤과 함께 이벤트 사용하기

이벤트를 처리하기 위해 VI를 설정하고 그 후 같은 이벤트를 처리하기 위해 **서브패널 컨트롤에 로드**한 VI를 설정하는 경우, 서브패널 컨트롤을 포함하는 최상위 VI가 이벤트를 처리합니다.

문자열

문자열

문자열은 ASCII 문자의 나열을 그룹화합니다. 문자열의 정보 및 데이터 포맷은 플랫폼에 의존적이지 않습니다. LabVIEW는 문자열을 특별한 포맷으로 저장하며, 이 포맷에서 문자의 배열의 첫번째 4 바이트는 문자열에서 몇 개의 문자가 나타날 지를 저장합니다. 그러므로, n 개의 문자가 있는 문자열을 메모리에 저장하려면 $n + 4$ 바이트가 필요합니다.



관련 정보

[숫자를 문자열로 변환하기](#) (1278 페이지 참조)

[문자열 포맷하기](#) (1279 페이지 참조)

[문자열에 디스플레이할 수 없는 문자를 삽입하기](#) (1279 페이지 참조)

[문자열에서 텍스트를 검색하고 대체하기](#) (1280 페이지 참조)

[패턴 일치를 위한 특수 문자](#) (1281 페이지 참조)

숫자를 문자열로 변환하기

다음 단계를 따라 [문자열로 포맷] 함수를 사용하여 숫자를 문자열로 변환합니다.

1. 숫자형 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 문자열 인디케이터를 프런트패널 윈도우에 추가합니다.
3. [문자열로 포맷] 함수를 블록다이어그램에 추가합니다.
4. [문자열로 포맷] 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 포맷 문자열 편집을 선택하여 포맷 문자열 편집 대화 상자에 포맷 문자열을 디스플레이합니다. 이 대화 상자를 사용하여 포맷, 소수점, 데이터 타입, 변환된 값의 폭을 지정하는 포맷 문자열을 생성합니다.
 1. 선택된 작업 풀다운 메뉴에서 소수점형 숫자로 포맷을 선택합니다.
 2. 먼저 자리맞춤 풀다운 메뉴에서 오른쪽 자리맞춤을 선택하고 채우기 풀다운 메뉴에서 제로로 채우기를 선택합니다.
 3. 옵션 섹션의 최소 필드 폭 사용 확인란에 확인 표시를 하고 8을 입력합니다. <Enter> 키를 누릅니다. 대응하는 포맷 문자열 필드는 이제 %08f를 포함해야 합니다. 8자리보다 적은 숫자를 숫자 컨트롤에 입력할 경우, 이 엔트리는 첫번째 숫자 이전의 남은 공간을 제로로 채웁니다.
 4. 옵션 섹션의 지정된 정밀도 사용 확인란에 확인 표시를 하고 4를 입력합니다. <Enter> 키를 누릅니다. 대응하는 포맷 문자열 필드는 이제 %08.4f를 포함해야 합니다. 이 엔트리는 숫자 컨트롤의 숫자를 소수점 이후 4자리를 갖는 문자열로 변환합니다.
 5. 확인 버튼을 클릭합니다. 함수는 문자열 상수를 생성하고 문자열 상수를 포맷 문자열 입력에 연결합니다.

5. 1 단계에서 생성한 숫자 컨트롤을 [문자열로 포맷] 함수의 입력 1 입력에 연결합니다.
6. 2 단계에서 생성한 문자열 인디케이터를 [문자열로 포맷] 함수의 결과 문자열 출력에 연결합니다.
7. 프런트패널로 돌아가서 VI를 실행합니다. 문자열 인디케이터는 4자리의 소수 자릿수를 가지는 숫자를 디스플레이합니다.

또한, [문자열/숫자 변환] 함수를 사용하여 숫자와 문자열 형을 변환할 수 있습니다.

문자열 포맷하기

데이터를 다른 VI, 함수, 또는 어플리케이션에서 사용하기 위해서 데이터를 문자열로 바꾸고 그 VI, 함수, 또는 어플리케이션에서 읽을 수 있는 포맷으로 바꾸어야 할 때가 있습니다. 예를 들어, Microsoft Excel은 문자열에 탭, 콤마, 또는 스페이스와 같은 구분 문자가 있어야 합니다. 엑셀은 이 경계 부호를 사용하여 숫자나 단어를 셀로 구분합니다.

대부분의 경우, 문자열의 포맷을 위하여 [문자열] 함수의 포맷 문자열 파라미터에 하나 또는 그 이상의 포맷 지정자를 입력해야 합니다. 포맷 지정자는 문자열을 포맷하기 위한 방법을 가리키는 코드입니다.

다음 함수는 문자열을 포맷합니다:

- 배열을 스프레드시트 문자열로
- 스프레드시트 문자열을 배열로
- 문자열로부터 스캔
- 문자열로 포맷
- 값을 문자로 포맷
- 값 스캔
- 파일로부터 스캔
- 파일로 포맷



관련 정보

[포맷 지정자 구문](#) (425 페이지 참조)

문자열에 디스플레이할 수 없는 문자를 삽입하기

특수 이스케이프 코드를 사용하여, 포맷 문자열 파라미터에 디스플레이할 수 없는 문자, 백래쉬, 퍼센트 문자를 삽입합니다. 플랫폼에 의존하는 라인 끝 문자에 대해 코드가 존재하지 않습니다. 라인 끝 상수를 사용하십시오.

[문자열로부터 스캔] 함수에서, 포맷 문자열 파라미터의 스페이스는 입력 문자열의 화이트 스페이스(스페이스, 탭, 폼 피드)의 크기와 일치합니다.



관련 정보

[백슬래시\('\'\) 코드 디스플레이](#) (1311 페이지 참조)

문자열의 지정된 포인트에서 텍스트 대체하기

다음 단계를 따라 문자열에서 텍스트를 대체합니다.

1. 블록다이어그램에 [부분 문자열 대체] 함수를 추가합니다.
2. 텍스트를 대체하고자 하는 문자열을 [부분 문자열 대체] 함수의 문자열 입력에 연결합니다.
3. 대체할 텍스트를 포함하는 문자열을 [부분 문자열 대체] 함수의 부분 문자열 입력에 연결합니다.
4. [부분 문자열 대체] 함수의 오프셋 입력에 텍스트 대체를 시작하려는 문자열까지의 문자 수를 입력합니다. 예를 들어, 문자열의 5번째 문자 다음부터 텍스트 대체를 시작하려면, 오프셋에 5를 입력합니다.
5. [부분 문자열 대체] 함수의 결과 문자열 출력을 문자열 인디케이터 또는 문자열을 입력으로 받는 VI 또는 함수에 연결합니다.
6. VI를 실행합니다. [부분 문자열 대체] 함수의 대체된 부분 문자열 출력은 대체한 텍스트를 반환합니다.

또한, [문자열의 텍스트를 찾고 대체](#)할 수 있습니다.

문자열에서 텍스트를 검색하고 대체하기

다음 단계를 따라 문자열에서 텍스트를 검색하고 대체합니다.

1. 블록다이어그램에 [문자열 검색과 대체] 함수를 추가합니다.
2. [문자열 검색과 대체] 함수의 입력 문자열 입력에 검색하려는 문자열을 연결합니다.
3. 검색 문자열 입력에 검색하려는 문자열을 연결합니다.
4. 대체 문자열 입력에 대체하려는 문자열을 연결합니다.
5. 텍스트의 모든 어커런스를 대체하기 위해, 문자열 검색과 대체 함수의 모두 대체? 입력을 참으로 설정합니다. 첫번째 어커런스를 대체하기 위해, 모두 대체?를 거짓으로 설정합니다.
6. [문자열 검색과 대체] 함수의 결과 문자열을 문자열을 입력으로 받는 문자열 인디케이터, VI, 함수에 연결합니다.
7. 블록다이어그램의 [모든 에러 입력과 에러 출력 터미널을 연결](#)합니다.
8. VI를 실행합니다. [문자열 검색과 대체] 함수의 대체 후의 오프셋 출력은 LabVIEW가 대체 문자열에서 처음 일치하는 텍스트를 발견한 문자열의 포인트를 나타냅니다.

또한, 문자열의 지정된 포인트에서 텍스트를 대체하고 VI 안에서 텍스트를 찾고 대체할 수 있습니다.

패턴 일치를 위한 특수 문자

[패턴 일치] 함수의 정규식 입력에는 다음과 같은 특수 문자를 사용합니다.



노트

[패턴 일치] 함수는 한정된 정규식 세트와 호환 가능하며, 문자 그룹화, 대체 패턴 일치, 역참조 또는 최소 일치 기호를 지원하지 않습니다. 보다 복잡한 패턴 일치 작업에는 [패턴 일치] 함수 대신 [정규식 일치] 함수를 사용하십시오.

특수 문자	설명	예
. (마침표)	새 라인 문자를 제외한 모든 단일 문자와 일치합니다. 대괄호 안에서 .는 리터럴로 사용됩니다.	문자열: Welcome to LabVIEW. 정규식: t.... 일치 부분 문자열: to La [z.]를 정규식으로 입력하면, 마침표는 리터럴이고 . 또는 z와 일치합니다. 이 예에서 [z.]는 .를 일치로 반환합니다.
*	이전에 오는 단일 문자나 문자 클래스를 0번 이상 나타낼 수 있는 것으로 표시합니다. 별표는 0번 나타나는 패턴과 일치할 수 있기 때문에, 별표가 포함된 정규식은 전체 패턴이 별표로 표시가 된 경우 빈 문자열을 반환할 수 있습니다. 이 양기호가 일치시키는 문자 수에는 제한이 없습니다.	문자열: Hello LabVIEW! 정규식: e1* 일치 부분 문자열: e11w* 또는 (welcome)*과 같은 식은 함수에서 다른 일치가 없는 경우 빈 문자열과 일치합니다.
+	이전에 오는 단일 문자나 문자 클래스를 0번 또는 그 이상 나타낼 수 있는 것으로 표시합니다. 이 양기호가 일치시키는 문자 수에는 제한이 없습니다.	문자열: Hello LabVIEW! 정규식: e1+ 일치 부분 문자열: e11
?	이전에 오는 단일 문자나 문자 클래스를 0번 또는 1번 나타낼 수 있는 것으로 표시합니다. 기본적으로 이 양기호가 일치시키는 문자 수에는 제한이 없습니다.	문자열: Hello LabVIEW! 정규식: e1? 일치 부분 문자열: e1

특수 문자	설명	예
[]	지정한 문자 집합의 모든 문자와 일치할 수 있는 문자 클래스를 생성합니다. 예를 들어, [abc]는 a나 b 또는 c와 일치합니다. [패턴 일치] 함수는 다음의 예외를 제외하고 대괄호 안에 있는 특수 문자를 리터럴로 해석합니다:	문자열: version=14.0.1 정규식: [0-9]+\.[0-9]+\.[0-9]+ 일치 부분 문자열: 14.0.1 [0-9] 식은 모든 숫자와 일치합니다. 더하기 부호는 이전 문자 클래스인 [0-9]와 1회 이상 가능한 최대 횟수로 일치합니다. \. 식은 리터럴 . 문자와 일치합니다. 더하기 부호는 이전 문자 클래스인 [0-9]와 1회 이상 가능한 최대 횟수로 일치합니다. 이 정규식은 3부분으로 구성된 버전 번호와 일치시키는데 사용할 수 있습니다.
-(대수)	자릿수, 소문자나 대문자 사이에 사용되어 범위를 나타냅니다; 예를 들어, [0-5], [a-g], [L-Q]와 같습니다.	
~	괄호 안에 들어있는 문자 또는 문자의 범위를 제외한 그 외의 모든 문자(디스플레이할 수 없는 문자 포함)와 일치합니다. 예를 들어, [~0-9]는 0에서 9까지를 제외한 모든 문자와 일치합니다.	
^	괄호 안에 들어있는 문자 또는 문자의 범위를 제외한 그 외의 모든 디스플레이할 수 있는 문자(스페이스 문자 포함)와 일치합니다. 예를 들어, [^0-9]는 0에서 9를 제외하고 스페이스 문자를 포함한 모든 디스플레이할 수 있는 문자와 일치합니다.	
^	^가 정규식에 첫 번째로 나오는 문자인 경우, 문자열의 오프셋과 일치하게 됩니다. 정규식이 오프셋의 문자로 시작하는 문자열의 일부와 일치하는 경우가 아니라면, 일치되지 않습니다. ^가 첫 번째 문자가 아닌 경우, 일반 문자와 같이 취급됩니다.	문자열: Hello LabVIEW! 정규식: ^[^]+ 일치 부분 문자열: Hello 이 정규식은 입력 문자열의 시작 부분에서부터 스페이스 문자를 제외하고 가능한 많은 개수의 문자와 일치합니다. 이 정규식을 사용하여 문자열의 첫 번째 워드, 숫자 또는 다른 문자의 조합을 분리할 수 있습니다.

특수 문자	설명	예
\$	\$가 정규식의 마지막 문자인 경우, 문자열의 마지막 원소와 일치하게 됩니다. 정규식이 문자열의 마지막 문자(마지막 문자 포함)까지 일치하는 경우가 아니라면, 일치하지 않습니다. \$는 마지막 문자가 아닌 경우, 일반 문자로 다루어집니다.	문자열: Hello LabVIEW! 정규식: [^]+\$ 일치 부분 문자열: LabVIEW! 이 정규식은 입력 문자열의 끝 부분에서부터 스페이스 문자를 제외하고 가능한 많은 개수의 문자와 일치합니다. 이 정규식을 사용하여 문자열의 마지막 워드, 숫자 또는 다른 문자의 조합을 분리할 수 있습니다.
\	이 리스트에 있는 모든 특수 문자의 해석을 취소합니다. 예를 들어, \?는 물음표와 일치하고, \.는 마침표와 일치하며, \\는 백슬래쉬와 일치합니다. 또한 공간 및 디스플레이할 수 없는 문자에 대해서는 다음과 같은 구성을 사용할 수 있습니다:	문자열: Welcome to the LabVIEW Help! 정규식: come\nto\tthe\sLabVIEW\sHelp\21 일치 부분 문자열: come to the LabVIEW Help! come\n 식은 새 라인 문자의 앞에 있는 리터럴 문자와 일치합니다. to\t 라는 식은 탭 앞에 있는 리터럴 문자 to와 일치합니다. 2개의 \s 식은 the와 LabVIEW, LabVIEW와 Help! 사이에 있는 스페이스와 일치합니다. \21 라는 식은 21이 느낌표에 대한 16 진수 코드이므로 느낌표와 일치합니다.
\b	백스페이스	
\f	폼 피드	
\n	새 라인	
\s	스페이스	
\r	캐리지 리턴	
\t	탭	
\xx	xx는 0에서 9의 숫자 및 A에서 F의 대문자를 사용한 16진 코드의 모든 문자	

정규식 일치 및 문자열 검색과 대체를 위한 특수 문자

정규식을 사용하여 패턴 일치를 구현하는데에는 문자의 특정한 조합이 필요합니다. 정규식을 사용하면 문자열의 일부 또는 서브셋을 일치 또는 검색할 수 있습니다. 예를 들어

정규식을 사용하여 코드 문자열의 변수를 일치시키거나 보고서의 특정 값을 일치시킬 수 있습니다.



노트
[정규식 일치] 함수와 [문자열 검색과 대체] 함수는 문자열의 널 문자를 지원하지 않습니다. 또한 정규식은 빈 문자열을 일치의 결과로 반환할 수 있습니다. 정규식이 일치 결과를 반환하지 않는 경우 일치 후의 오프셋에 -1을 반환합니다.

정규식 지원은 PCRE 라이브러리 패키지에 의해 제공됩니다. PCRE 라이브러리 패키지의 재배포 라이선스에 대한 더 자세한 정보는 <National Instruments>_Legal Information 디렉토리를 참조하십시오.

Perl Compatible Regular Expressions에 대한 더 자세한 정보는 www.pcre.org의 Perl 5 Regular Expression 문서를 참조하십시오.

정규식 및 검색 문자열 입력용 특수 문자

[정규식 일치] 함수의 정규식 입력 및 [문자열 검색과 대체] 함수의 검색 문자열 입력에는 다음 테이블의 특수 문자를 사용합니다. 해당 함수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 정규식을 선택하여 [문자열 검색과 대체] 함수에 다음 특수 문자를 사용할 수 있습니다.

노트 다음 특수 문자는 [정규식 일치] 함수와 [문자열 검색과 대체] 함수와 함께만 사용할 수 있습니다. [패턴 일치] 함수는 보다 한정된 정규식을 사용하지만 [정규식 일치] 함수보다 실행 속도가 빠릅니다. [패턴 일치] 함수의 정규식에 대한 더 자세한 정보는 패턴 일치를 위한 특수 문자 토픽을 참조하십시오.

특수 문자	설명	예
(마침표)	새 라인 문자를 제외한 모든 단일 문자와 일치합니다. 대괄호 안에서 .는 리터럴로 사용됩니다.	입력 문자열: Welcome to LabVIEW. 정규식: t....일치: to La 정규식으로 [z.]를 입력하면 마침표는 리터럴이므로 식은 .나 z와 일치합니다. 이 예에서 [z.]는 .를 일치로 반환합니다.

특수 문자	설명	예
*	앞에 오는 단일 문자나 문자 그룹 또는 문자 클래스를 0번 이상 나타낼 수 있는 것으로 표시합니다. 별표는 0번 나타나는 패턴과 일치할 수 있기 때문에 별표가 포함된 정규식은 전체 패턴이 별표로 표시가 된 경우 빈 문자열을 반환할 수 있습니다. 기본적으로 이 양기호가 일치시키는 문자 수에는 제한이 없습니다.	입력 문자열: Hello LabVIEW! 정규식: e1*일치: e11 w* 또는 (welcome)* 식은 함수가 다른 일치를 찾지 못하는 경우 빈 문자열을 반환합니다.

특수 문자	설명	예
+	앞에 오는 단일 문자나 문자 그룹 또는 문자 클래스를 1번 이상 나타낼 수 있는 것으로 표시합니다. 기본적으로 이 양기호가 일치시키는 문자 수에는 제한이 없습니다.	입력 문자열: Hello LabVIEW! 정규식: e1+ 일치: e11

특수 문자	설명	예
?	앞에 오는 단일 문자나 문자 그룹 또는 문자 클래스를 0번 또는 1번 나타낼 수 있는 것으로 표시합니다. 기본적으로 이 양기호가 일치시키는 문자 수에는 제한이 없습니다. 양기호 직후에 사용되는 경우, ?는 양기호가 최소의 횟수로 일치되도록 변경합니다. 변경이 가능한 양기호에는 *, + 및 {}가 있습니다.	입력 문자열: Hello LabVIEW! 정규식: e1?일치: e1 입력 문자열: <pre>HelloLabVIEW</pre> 정규식: .+?일치: Hello 두 번째 예에서 정규식에서 ?를 제거하면 HelloLabVIEW가 새롭게 일치하게 되는데, 이는 + 직후에 ?가 없는 경우 +는 최대 수의 문자와 일치하기 때문입니다. 이 정규식은 태그 사이의 문자열과 일치시키는데 사용할 수 있습니다.

특수 문자	설명	예
{n,N}	앞에 오는 단일 문자, 문자 그룹 또는 문자 클래스를 지정된 횟수만큼 나타낼 수 있는 것으로 표시합니다. 여기서 n은 최소 횟수이고, N은 최대 횟수입니다. 1개의 수를 지정할 수도 있습니다. 범위를 지정할 경우 이 양기호는 가능한 최대 횟수에 일치합니다.	입력 문자열: HelloLabVIEW! 정규식: (.+?){2} 일치: HelloLab 입력 문자열: HelloLabVIEW! 정규식: (.+?){1,3} 일치: HelloLabVIEW 두 번째 예에서 최소 일치 횟수는 1이고, 최대 일치 횟수는 3입니다. 정규식은 지정된 횟수 한도 내에서 최대 횟수와 일치하기 때문에 이 정규식은 3개의 일치를 반환합니다.

특수 문자	설명	예
[]	지정한 문자 집합의 모든 문자와 일치할 수 있는 문자 클래스를 생성합니다. 예를 들어, [abc]는 a나 b 또는 c와 일치합니다. -를 사용하여 문자의 범위를 지정할 수 있습니다. 예를 들어 [a-z]는 모든 소문자 단일 문자와 일치합니다. [정규식 일치] 함수와 [문자열 검색과 대체] 함수는 ^, -, \를 제외하고 대괄호 안에 있는 특수 문자를 리터럴로 해석합니다.	입력 문자열: version=14.0.1 정규식: [0-9]+(\.[0-9]+)* 일치: 14.0.1 [0-9]라는 식은 어떤 한 자릿수와 일치합니다. 더하기 부호는 이전 문자 클래스인 [0-9]와 1회 이상 가능한 최대 횟수로 일치합니다. 괄호는 문자의 그룹을 생성하며, 이 문자 그룹은 . 문자 및 연속적인 모든 자릿수와 부분 일치합니다. \. 식은 리터럴 . 문자와 일치합니다. 더하기 부호는 이전 문자 클래스인 [0-9]와 1회 이상 가능한 최대 횟수로 일치합니다. 별표 이전 문자 그룹인 (\.[0-9]+)와 0회 이상 일치하기 때문에, 이 정규식은 . 문자가 없는 정수와도 일치합니다. 이 정규식을 사용하여 임의의 정수, 소수, 버전 번호, IPv4 주소 또는 . 문자로 구분된 기타 수열을 일치시킬 수 있습니다.

특수 문자	설명	예
	지정한 문자 집합의 전체 문자와 일치할 수 있는 문자 그룹을 생성합니다. 문자 그룹 직후에 나오는 양기호는 그룹 전체에 해당됩니다. 또한 괄호는 각 개별 문자 그룹이 부분 일치를 반환하는 부분 일치를 생성합니다. 다른 문자 그룹 안에 문자 그룹을 넣으면, 정규식은 그 그룹보다 밖의 그룹의 부분 일치 구문을 먼저 생성합니다. [정규식 일치] 함수 노드를 확장하여 부분 일치 출력에 접근할 수 있습니다. 또한 역참조를 사용하는 식에서 이후 부분 일치를 다시 참조할 수 있습니다. 정규식에	입력 문자열: Hello LabVIEW! 정규식: (e1.)..(L..) 일치: e1lo Lab 부분 일치 1: e1l 부분 일치 2: Lab 입력 문자열: Hello LabVIEW! 정규식: (. (e1.) .) . (L..) 일치: Hello Lab 부분 일치 1: Hello 부분 일치 2: e1l 부분 일치 3: Lab

특수 문자	설명	예
	가능한 다른 일치를 분리합니다. 이 문자는 여러 문자 그룹 중 하나와 일치시킬 때 유용합니다. ()를 포함하는 정규식은 가능한 일치의 순서에 관계없이 입력 문자열에서 첫 번째로 일치한 항목을 반환합니다. 예를 들어, dog cat과 cat dog 정규식은 the dog chased the cat에서 모두 dog를 반환합니다.	입력 문자열: value=FALSE total=12.34 token=TRUE 정규식: (value token)=(TRUE FALSE) 일치: value=FALSE 부분 일치 1: value 부분 일치 2: FALSE 정규식은 입력 문자열에서 일치하는 첫 번째 항목을 반환합니다. 입력 문자열에서 token=TRUE가 value=FALSE의 앞에 있는 경우, 정규식은 위와 달리 token=TRUE를 일치한 항목으로 반환합니다.

특수 문자	설명	예
^	패턴의 첫 문자로 사용될 때 일치할 문자열의 처음으로 한정합니다. [정규식 함수 또는 [문자열 검색과 대체] 함수에서 여러 라인? 입력을 참으로 설정하면, ^는 현재 플랫폼의 라인 끝을 사용하여 문자열 내에 있는 각 라인의 시작과 일치됩니다. 또한, 문자 클래스의 처음에 ^을 추가하여 주어진 문자 클래스에 없는 모든 문자와 일치시킬 수 있습니다. 예를 들면, [^0-9]은 숫자가 아닌 모든 문자와 일치합니다. [^a-zA-Z0-9]는 소문자, 대문자, 숫자가 아닌 모든 문자와 일치합니다.	입력 문자열: Hello LabVIEW! 정규식: ^[^]+ 일치: Hello 이 정규식은 입력 문자열의 시작 부분에서부터 스페이스 문자를 제외하고 가능한 많은 개수의 문자와 일치합니다. 이 정규식을 사용하여 문자열의 첫번째 워드, 숫자 또는 다른 문자의 조합을 분리할 수 있습니다. 입력 문자열: Hello LabVIEW 정규식: ^LabVIEW 일치: LabVIEW 이 정규식은 여러 라인?이 참으로 설정된 경우에만 LabVIEW와 일치합니다.

특수 문자	설명	예
\$	패턴의 마지막 문자로 사용될 때 일치를 문자열의 맨 끝으로 한정합니 다. [정규식] 함수 또는 [문자열 검색과 대체] 함수에서 여러 라인? 입력을 참으로 설정하면, ^는 현재 플랫폼의 라인 끝을 사용하여 문자열 내에 있는 각 라인의 끝과 일치됩니 다.	입력 문자열: Hello LabVIEW! 정규식: [^]+\$ 일치: LabVIEW! 이 정규식은 입력 문자열의 시작 부분에서부터 스페이스 문자를 제외하고 가능한 많은 개수의 문자와 일치합니다. 이 정규식을 사용하여 문자열의 마지막 워드, 숫자 또는 다른 문자의 조합을 분리할 수 있습니다. 입력 문자열: HelloLabVIEW 정규식: Hello\$ 일치: Hello 이 정규식은 여러 라인?이 참으로 설정된 경우에만 Hello와 일치합니다.

특수 문자	설명	예
	이 리스트에 있는 백슬래쉬 부호 직후에 나오는 특수 문자의 특수한 의미를 취소하고 대신 리터럴 문자와 일치시킵니다. 다음 Escape 식은 특별한 의미를 갖습니다. • \b - 단어 경계를 나타냅니다. 단어 경계는 단어 문자인 문자 옆에 있는 단어 가 아닌 문자, 또	



팁 일치 패턴을 문자열의 시작과 끝에 한정하려면, 패턴의 첫 문자에 ^을 사용하고 패턴의 끝 문자에 \$를 사용합니다. 예를 들어, ^LabVIEW\$는 LabVIEW!나 Hello LabVIEW가 아닌 LabVIEW에서 LabVIEW와 일치합니다. 일치를 문자열의 시작과 끝에 한정하려면 전체 문자열이 일치해야 합니다.

대체 문자열용 특수 문자

[문자열 검색과 대체] 함수의 대체 문자열 입력에서 다음의 특수 문자를 사용합니다.

특수 문자	설명	예
\$n 또는 \${n}	지정된 부분 일치의 앞 또는 뒤에 지정된 문자열을 삽입합니다. [문자열 검색과 대체] 함수의 대체 문자열 입력에 역참조를 사용하는 방법에 대한 더 자세한 정보는 역참조 섹션을 참조하십시오. 정규식에 부분 일치가 9개보다 많고 10번째 이후의 부분 일치를 참조하려는 경우 \${n}을 사용하십시오. 함수는 \$ 이후에 나오는 숫자의 첫 자릿수만 읽기 때문에 \$12는 첫 번째 부분 일치만을 삽입합니다. 그러나 \${12}는 12번째 부분 일치를 삽입합니다.	입력 문자열: Welcome LabVIEW 검색 문자열: (LabVIEW) 대체 문자열: to \$1! 결과 문자열: Welcome to LabVIEW! 입력 문자열: Welcome to the LabVIEW Help! 검색 문자열: We(1)(co(m)e)((to)(t(he) (Lab)(VIE(W))) He(1p)(!) 대체 문자열: \$7\${11}ful\${12} 결과 문자열: helpful!
\\$n	대체 문자열에서 사용하는 모든 특수 문자의 해석을 취소합니다.	입력 문자열: total=123 per day 검색 문자열: (\d+)(per \w*)? 대체 문자열: \\$\$1\$2 결과 문자열: total=\$123 per day 이 예에서 리터럴 문자 \$를 삽입하려면 \\$를 사용합니다. 리터럴 \를 나타내려면 \\를 사용합니다.

역참조

역참조를 사용하여 같은 정규식에서의 이전의 일치를 참조할 수 있습니다. 역참조를 사용하면 식의 일부에서 문자 그룹을 사용하여 부분 일치를 생성하고, 이후 식의 후반부에서 그와 정확히 동일한 부분 일치를 반환할 수 있습니다.

[정규식 일치] 함수의 정규식 입력 또는 [문자열 검색과 대체] 함수의 검색 문자열 입력에 참조를 지정하려면, \1을 사용하여 첫번째 부분 일치를 참조하고, \2를 사용하여 두번째 부분 일치를 참조하는 방법을 사용합니다. 예를 들어, 다음과 같은 정규식을 생각해 봅니다:

```
(["*$"])(\w+)\1\2\1
```

첫 번째 문자 그룹에는 ", * 또는 \$와 일치하는 문자 클래스가 포함되어 있습니다. 두 번째 문자 그룹은 하나 이상의 워드 문자와 일치합니다. 첫 번째 역참조인 \1은 첫 번째 그룹인 (["*\$"])와 동일한 부분 일치를 반환합니다. 두 번째 역참조인 \2는 두 번째 그룹인 (\w+)와 동일한 부분 일치를 반환합니다. 세 번째 역참조인 \1은 첫 번째 역참조와 동일하며, 첫 번째 문자 그룹과 동일한 부분 일치를 반환합니다.

이 예는 "foo"foo", *bar*bar*, \$baz\$baz\$와 같은 문자열을 일치로 반환하지만, "foo\$foo" 또는 "foo*bar"와 같은 문자열은 일치로 반환하지 않습니다.

대체 문자열의 역참조

[문자열 검색과 대체] 함수를 정규식 모드로 사용하는 경우, 검색 문자열 입력에서 부분 일치를 참조하는 대체 문자열 입력에 역참조를 지정할 수 있습니다. \$1을 사용하여 첫 번째 부분 일치를 참조하고, \$2를 사용하여 두 번째 부분 일치를 참조하는 방법을 사용합니다. [문자열 검색과 대체] 함수의 대체 문자열 입력에서만 다음의 특수 문자를 사용할 수 있습니다. 다음의 특수 문자를 사용하여 지정된 일치 그룹 앞이나 뒤에 지정한 문자열을 삽입합니다. 다음 규칙을 고려하십시오:

입력 문자열: \$value "TRUE"TRUE" *NULL
 검색 문자열: (["*\$"])(\w+)\1\2\1
 대체 문자열: \$1\$2value\$1
 결과 문자열: \$value "TRUEvalue" *NULL

이 예에서는 대체 문자열의 \$1 역참조가 검색 문자열의 첫 번째 부분 일치를 참조합니다. 마찬가지로 \$2 역참조는 검색 문자열의 두 번째 부분 일치를 참조합니다.

문자열 분리하기

하나의 문자열을 2개의 부분 문자열로 나눕니다. 문자열을 캐리지 리턴과 같은 어떤 문자로 또는 단어나 구문으로 나눌 수 있습니다.

1. 블록 다이어그램에 [문자열 검색/분리] 함수를 추가합니다.
2. [문자열 검색/분리] 함수의 문자열 입력에 나누려는 문자열을 연결합니다.
3. [문자열 검색/분리] 함수의 검색 문자열/문자 입력에 문자열이 분리되어야 하는 위치를 텍스트로 입력합니다.

4. 오프셋에서, [문자열 검색/분리] 함수가 검색 문자열/문자의 텍스트를 검색하기 전에 건너뛴 문자열의 문자수를 입력합니다.
5. [문자열 검색/분리] 함수의 일치 구문 앞 부분 문자열 및 일치 + 문자열의 나머지 출력을 문자열 인디케이터나 입력으로 문자열을 받는 VI 또는 함수에 연결합니다.
6. VI를 실행합니다.

숫자형 문자열에서 포맷 지정자 사용하기

퍼센트 코드라고도 알려진 포맷 지정자를 사용하여, 원하는 LabVIEW의 숫자 디스플레이 방식을 지정합니다. 숫자형 프로퍼티 대화 상자 또는 포맷 문자열 프로퍼티를 사용하여 숫자 컨트롤을 포맷하는 방법을 지정합니다. 포맷 지정자는 다음 아이템을 포함합니다:

- 포맷 지정자—%로 시작하고 문자로 끝나는 문자열. 문자는 대소문자를 구별합니다. 포맷 지정자는 %와 문자 사이에 다음 요소를 포함할 수 있습니다:
 - 변경자—포맷 코드를 변경하는 기호.
 - 최소 폭—필드 폭을 지정하는 0보다 큰 수.
 - 정밀도—.으로 시작되고 그 뒤에 0보다 크거나 같은 숫자가 따라오는 문자열.
 - 유효 숫자—_로 시작되고 그 뒤에 0보다 크거나 같은 숫자가 따라오는 문자열.
 - 내장된 포맷 정보—LabVIEW가 숫자를 포맷하기 위해 사용하는 문자열. 문자열을 <와 >로 둘러쌉니다.
- 구분문자—포맷 코드 사이, 앞, 또는 뒤의 텍스트. %와 >는 두 구분문자입니다. 문자열에 % 문자를 넣으려면, %%를 사용하십시오. 문자열에 > 문자를 넣으려면, >를 사용하십시오. LabVIEW의 백슬래시 코드 해석 방법에 대한 추가적인 정보는 [백슬래시 \('\'\) 코드 디스플레이](#) 토픽을 참조하십시오.
- 리터럴 문자열—문자열에 나타내려고 하는 텍스트.



노트

하나의 포맷 지정자에서 정밀도와 유효 숫자를 함께 사용할 수 없습니다.

포맷 지정자에 대한 추가 예제는 [포맷 지정자 예제](#) 테이블을 참조하십시오.

문자열에 캐리지 리턴 입력하기

다음 단계를 따라 문자열에 캐리지 리턴을 입력합니다.

1. 문자열 컨트롤을 프런트패널에 추가합니다. 기본 모드는 [일반 디스플레이](#)입니다.
2. 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 ['\ 코드 디스플레이](#)를 선택합니다.
3. 수행 도구 또는 라벨링 도구를 사용하여, 캐리지 리턴을 입력하려는 문자열 컨트롤에 커서를 추가합니다.

4. \r을 입력합니다.

문자열에 라인 피드 입력하기

다음 단계를 따라 문자열에 라인 피드를 입력합니다.

1. 문자열 컨트롤을 프런트패널에 추가합니다. 기본 모드는 **일반 디스플레이**입니다.
2. 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 '**** 코드 디스플레이'를 선택합니다.
3. 수행 도구 또는 라벨링 도구를 사용하여, 라인 피드를 입력하려는 문자열 컨트롤에 커서를 추가합니다.
4. \n을 입력합니다. 또한, <Enter> 키를 눌러서 라인 피드를 문자열에 입력할 수 있습니다.

문자열에 탭 문자 입력하기

다음 단계를 따라 문자열에 탭 문자를 입력합니다.

1. 문자열 컨트롤을 프런트패널에 추가합니다. 기본 모드는 **일반 디스플레이**입니다.
2. 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 '**** 코드 디스플레이'를 선택합니다.
3. 수행 도구 또는 라벨링 도구를 사용하여, 탭 문자를 입력하려는 문자열 컨트롤에 커서를 추가합니다.
4. \t를 입력합니다.

문자열 컨트롤에 텍스트를 입력하거나 변경하기

다음 단계를 따라 문자열 컨트롤에 텍스트를 입력하거나 변경합니다.

1. 문자열 컨트롤을 프런트패널에 추가합니다. 기본 모드는 **일반 디스플레이**입니다.
2. 수행 도구 또는 라벨링 도구를 사용하여 커서를 문자열 컨트롤에 추가하고, 원하는 텍스트를 입력하거나 편집합니다. 텍스트가 문자열 디스플레이의 오른쪽 경계에 도달하면, 스페이스 또는 탭 문자와 같은 자연적인 분리자에서 문자열은 다음 라인으로 넘어갑니다.



노트

텍스트를 더블 클릭하여 워드를 하이라이트하거나 트리플 클릭하여 전체 문자열을 하이라이트할 수 있습니다.

또한, 문자열 컨트롤에서 여러 **키보드 바로 가기 키**를 사용할 수 있습니다.

런타임으로 콤보 박스 컨트롤에 문자열을 선택하고 입력하기

다음 방법을 사용하여 VI가 실행 모드에 있는 동안 콤보 박스 컨트롤에서 문자열을 선택하고 입력합니다:

- 컨트롤의 오른쪽에 있는 화살표 버튼을 클릭하고 나타나는 바로 가기 메뉴로부터 문자열을 선택합니다.
- 컨트롤의 오른쪽에 있는 화살표 버튼을 클릭하고 위 아래 화살표 키를 사용하여 선택하려는 문자열을 탐색합니다. 바로 가기 메뉴에서 문자열을 하이라이트한 후에, <Enter> 키를 누르거나 화살표 버튼을 클릭하여 문자열을 선택합니다.
- 텍스트 박스에 커서를 놓고, 아래 방향 화살표 키를 눌러서 바로 가기 메뉴를 디스플레이하고, 위 아래 화살표 키를 사용하여 선택하려는 문자열을 탐색합니다. 바로 가기 메뉴에서 문자열을 하이라이트한 후에, <Enter> 키를 누르거나 화살표 버튼을 클릭하여 문자열을 선택합니다.
- 컨트롤의 오른쪽에 화살표 버튼을 클릭하고 선택하려는 문자열의 시작 문자를 타입합니다. 타입하는 문자로 시작하는 컨트롤에서 LabVIEW는 처음의 가장 짧은 문자열을 하이라이트합니다. <Enter> 키를 누르거나 화살표 버튼을 클릭하여 문자열을 선택합니다. 문자열이 타입한 문자와 일치하지 않고 컨트롤이 정의되지 않은 문자열 값을 허용하지 않으면, LabVIEW는 컨트롤에서 타입한 문자를 받지 않거나 디스플레이하지 않습니다. 문자열이 타입한 문자와 일치하지 않지만 컨트롤이 정의되지 않은 문자열 값을 허용하면, <Enter> 키를 눌러서 컨트롤에 새로운 문자열 값을 입력합니다.
- 텍스트 박스에 커서를 놓고 선택하려는 문자열의 시작 문자를 타입합니다. 타입하는 문자로 시작하는 컨트롤에서 LabVIEW는 처음의 가장 짧은 문자열을 디스플레이합니다. <Enter> 키를 눌러서 문자열을 선택합니다. 문자열이 타입한 문자와 일치하지 않고 컨트롤이 정의되지 않은 문자열 값을 허용하지 않으면, LabVIEW는 컨트롤에서 타입한 문자를 받지 않거나 디스플레이하지 않습니다. 문자열이 타입한 문자와 일치하지 않지만 컨트롤이 정의되지 않은 문자열 값을 허용하면, <Enter> 키를 눌러서 컨트롤에 새로운 문자열 값을 입력합니다.
- 컨트롤이 정의되지 않은 문자열 값을 허용하면, 텍스트 박스에 커서를 놓고 입력하려는 문자열을 타입합니다. <Enter> 키를 눌러서 컨트롤에 새로운 문자열 값을 입력합니다.

콤보 박스 문자열의 사용자 값을 지정하기

콤보 박스 컨트롤을 위한 문자열의 리스트를 설정할 때, 각 문자열에 대한 사용자 값을 지정할 수 있습니다. 이때, 프런트패널 콤보 박스 컨트롤에 나타나는 문자열이 블록 다이어그램의 콤보 박스 터미널이 반환하는 문자열 값과 다르도록 합니다.

다음 단계를 따라 콤보 박스 컨트롤에서 문자열에 대해 사용자 값을 지정합니다.

1. 콤보 박스 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 아이템 편집을 선택하여 콤보 박스 프로퍼티 대화 상자의 아이템 편집 페이지를 디스플레이합니다.
2. 값을 아이템에 일치 확인란에서 확인 표시를 제거합니다.
3. 변경하려는 값을 포함하는 값 열에 있는 셀에서 더블 클릭합니다.

4. 컨트롤의 대응하는 라벨을 사용자가 선택할 때 블록 다이어그램에 사용하려는 문자열 값을 입력합니다.
5. 변경하려는 각 값에 대해서 단계 3-4를 반복합니다.
6. 확인 버튼을 클릭합니다.

또한, 문자열과 값 [] 프로퍼티를 사용하여 프로그램에서 사용자 값을 지정할 수 있습니다.

색 램프를 다이얼, 노브, 게이지에 추가하기

색 램프는 색을 사용하여 숫자값을 디스플레이합니다. 각각 숫자값과 대응하는 디스플레이 색을 갖는 최소 두 개의 임의의 마커로 구성된 색 스케일을 설정합니다. 입력값이 변하면, 디스플레이된 색도 값에 대응하는 색으로 바뀝니다. 색 램프는 게이지가 위험한 값에 도달할 때를 나타내는 경고 범위와 같이 데이터 범위를 시각적으로 나타내는데 유용합니다. 예를 들어, 색 램프를 사용하여 강도 **그래프** 및 **차트**의 색 스케일을 설정할 수 있습니다.

다음 단계를 따라 색 램프를 추가하고 설정합니다:

1. 다이얼, 노브, 또는 게이지에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목 >> 램프를 선택하여 색 램프를 추가합니다.
2. 색 램프 또는 마커에서 마우스 오른쪽 버튼을 클릭한 후 색 램프를 임의의 마커 간격으로 변환하기 위해 바로 가기 메뉴에서 마커 간격 >> 임의를 선택합니다.
3. 색 램프의 임의의 마커의 값을 변경하려면, 수행 도구를 사용하여 마커를 원하는 값까지 끌거나 라벨링 도구를 사용하여 마커의 텍스트를 하이라이트한 후 새로운 값을 입력합니다. 마커를 끌 때, 커서는 도구가 마커 위에 있는 것을 나타내기 위해서 로 바뀝니다.
4. 새 마커를 색 램프에 추가하려면, 새 마커를 추가하려는 색 램프의 위치에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 마커 추가를 선택합니다. LabVIEW는 해당 위치에 새 마커를 추가합니다. 또한, <Ctrl> 키를 누르면서 기존의 마커를 끌어서 마커를 추가하려는 곳에 놓습니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.
5. 수행 또는 위치 도구를 사용하여 색 램프 다음의 마커에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 마커 색을 선택하여 마커에 연관된 색을 변경합니다.
6. **색 선택기**가 나타나면 원하는 색을 선택합니다.
7. 위치 도구를 색 램프의 중간 지점에 나타나는 핸들 중 하나의 위로 이동하여 색 램프의 크기를 조정합니다. 핸들을 원하는 쪽으로 끕니다.



노트

미터는 기본적으로 보이는 색 램프를 가집니다.

슬라이더 또는 지침 추가하기

다음 단계를 따라 슬라이더 또는 지침을 스케일을 갖는 숫자형 객체에 추가합니다.

1. 슬라이드 또는 회전식 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 슬라이더 추가 또는 지침 추가를 선택합니다.
2. 슬라이더 또는 지침의 색을 구분하려는 경우 색칠 도구를 사용하여 슬라이더 또는 지침의 색을 변경합니다.
3. (옵션) 슬라이드 컨트롤이나 인디케이터의 경우, **채우기 옵션을 변경**하여 아래의 값까지 채우기 또는 위의 값까지 채우기를 사용합니다.

또한, 슬라이더 또는 지침을 **사용자 정의**할 수 있습니다.

숫자형 객체에 스케일을 갖는 텍스트 라벨 추가하기

슬라이드 또는 회전식 객체에 정수 값을 가지는 문자열을 연계하거나 특정 정수 값을 선택하여 사용할 수 있도록 텍스트 라벨을 추가할 수 있습니다.

다음 단계를 따라 숫자형 객체에 스케일을 갖는 텍스트 라벨을 생성합니다.

1. 슬라이드 또는 회전식 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 텍스트 라벨을 선택합니다. 기본 텍스트 라벨, 최소와 최대가 스케일에 나타납니다. 텍스트 디스플레이 또한 나타나며 텍스트 라벨을 디스플레이합니다.
2. 라벨링 도구를 사용하여 라벨을 편집합니다.
3. 텍스트 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목 >> 디지털 디스플레이를 선택하여 생성한 텍스트 라벨에 연계된 숫자 값을 디스플레이합니다. 이 값은 항상 최소값인 제로에서 시작하여 각 텍스트 라벨마다 1씩 증가합니다.
4. 다음 단계를 통해 텍스트 라벨과 스케일에 대응하는 마커를 추가합니다.
 1. 텍스트 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 항목 편집을 선택하여 슬라이드 프로퍼티 대화 상자 또는 노브 프로퍼티 대화 상자의 항목 편집 페이지를 디스플레이합니다.
 2. 항목 옆의 비어있는 셀에서 더블 클릭을 한 후, 스케일 마커 다음에 디스플레이하려는 문자열을 입력합니다. 문자열을 이미 포함하는 셀에도 커서를 놓을 수 있고, 삽입 버튼을 클릭하여 셀 위에 비어있는 행을 생성하며, 새로운 셀에 문자열을 입력할 수 있습니다.
 3. (옵션) **마커 간격이 임의**인 경우, 순차적인 값에서 확인 표시를 제거하고 사용하려는 숫자값을 입력하여 각 항목에 특정 숫자값을 지정할 수 있습니다. 기본적으로 항목 옆에 문자열을 입력하면 LabVIEW는 값 옆에 대응하는 셀에 마커의 숫자값을 추가합니다. 첫번째 항목에 대해서 0의 값을 시작으로 LabVIEW는 리스트의 항목의 순서에 따라서 순차적인 값을 할당합니다.
 4. 확인 버튼을 클릭합니다.

텍스트 디스플레이에 커서를 놓은 후 라벨을 편집하고 <Ctrl-Enter> 키를 눌러서 마커의 텍스트 라벨을 편집할 수 있습니다. (macOS) <Option-Enter> 키를 누릅니다. (Linux) <Alt-Enter> 키를 누릅니다.

또한 텍스트 라벨 [] 프로퍼티를 사용하여 프로그램에서 라벨을 편집할 수 있습니다.

텍스트 디스플레이에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **아이템 비활성화**를 선택해서 텍스트 라벨과 마커를 비활성화하여 실행시 아이템을 선택할 수 없도록 할 수 있습니다.

스케일 리미트 변경하기

슬라이드와 회전식 컨트롤 또는 인디케이터의 외부 두 마커는 스케일 리미트입니다. 스케일 리미트는 범위 리미트와 일치할 필요는 없습니다. 스케일 리미트는 컨트롤 또는 인디케이터의 범위의 부분이 될 수 있습니다. 예를 들어, 16 비트 부호있는 정수 컨트롤 또는 인디케이터의 기본 범위는 -32,768에서 32,767까지입니다. 하지만, 데이터 범위를 -1,000에서 1,000으로 설정할 수 있으며, 스케일 리미트를 0에서 500까지 설정할 수 있습니다.

다음 단계를 따라 스케일 리미트와 증가를 변경합니다.

1. 수행 도구 또는 라벨링 도구를 사용하여 슬라이드 또는 회전식 객체의 스케일상의 상위 또는 하위 리미트를 클릭합니다.
2. 변경하려는 숫자를 하이라이트한 후 새 숫자를 입력합니다.
3. <Enter> 키를 누르거나, 도구 모음의 엔터 버튼을 클릭하거나, 스케일 밖을 클릭하여 새 값을 입력합니다. 다른 리미트는 바뀌지 않으며 LabVIEW는 자동으로 증가를 다시 계산합니다.
4. 단계 1부터 3까지를 반복하여 스케일상의 중간 마커의 값을 변경합니다. LabVIEW는 스케일의 증가를 자동으로 조정합니다. 스케일의 증가가 자동으로 조정되지 않을 경우, 중간 마커가 다른 마커에 너무 가깝기 때문에 중간 마커의 값을 변경하기 이전에 **슬라이드 또는 회전식 객체의 크기를 증가**시켜야 합니다.

또한 범위 프로퍼티를 사용하여 스케일 리미트와 증가를 프로그램적으로 변경할 수 있습니다.

스크롤 막대 컨트롤의 스크롤 범위 변경하기

스크롤 막대 프로퍼티 대화 상자의 데이터 입력 페이지를 사용하여 스크롤 막대 컨트롤의 스크롤 범위를 변경할 수 있습니다. 예를 들어, 최소가 0이고 최대가 10, 페이지 크기가 2인 경우, 스크롤 범위는 0에서 8까지입니다. 스크롤 박스의 위쪽 또는 왼쪽 위치는 스크롤 막대의 값을 결정합니다.

다음 단계를 따라 스크롤 막대 컨트롤의 스크롤 범위를 설정합니다.

1. 스크롤 막대 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **데이터 입력**을 선택하여 스크롤 막대 프로퍼티 대화 상자의 데이터 입력 페이지를 디스플레이합니다.
2. 기본 리미트 사용 확인란에서 확인 표시를 제거합니다.
3. 최소값 필드에서, 낮은 값 리미트를 설정하고, 풀다운 메뉴에서 강제 변환 옵션을 선택할지 결정합니다. 최소값을 강제 변환하지 않는 경우, 감소 화살표를 클릭하여 스크롤 막대 값을 스크롤 범위 밖으로 감소시킬 수 있습니다.

4. 최대값 필드에서 높은 값 리미트를 설정하고, 풀다운 메뉴에서 강제 변환 옵션을 선택할지 결정합니다. 최대값을 강제 변환하지 않는 경우, 증가 화살표를 클릭하여 스크롤 막대 값을 스크롤 범위 밖으로 증가시킬 수 있습니다.
5. 증가 필드에서, 증가 값을 설정합니다. 증가와 감소 화살표를 클릭할 때, 스크롤 막대 값은 증가값을 더하거나 빼서 스크롤 박스를 클릭한 화살표 쪽으로 움직입니다.
6. 페이지 크기 필드에서, 페이지 크기를 설정합니다. 스크롤 박스와 화살표 사이의 공간을 클릭할 때, 페이지 크기에 의해 스크롤 막대 값이 바뀝니다.

또한 스크롤 막대 프로퍼티를 사용하여 프로그램적으로 스크롤 범위를 설정할 수 있습니다.

숫자형 컨트롤과 인디케이터가 허용하는 문자

숫자형 컨트롤과 인디케이터는 오직 다음의 문자만을 허용합니다.

문자	설명
16진수	0에서 F
8진수	0 ~ 7
2진수	0 및 1
10진수	1, 1.0, 2, 3.5 등
.	소수점
+	양수 기호
-	음수 기호
E 또는 e	과학적 또는 공학적 표기 포맷
Inf	무한대
NaN	숫자 아님 (Not a number)
/	컴퓨터의 로컬 셋팅에 따라 바뀌는 절대 시간 포맷에 사용
:	컴퓨터의 로컬 셋팅에 따라 바뀌는 절대 시간 포맷에 사용
AM, am, PM, pm	컴퓨터의 로컬 셋팅에 따라 바뀌는 절대 시간 포맷에 사용
SI 접두어	
y	yocto(10^{-24})
z	zepto(10^{-21})
a	atto(10^{-18})

문자	설명
f	femto(10^{-15})
p	pico(10^{-12})
n	nano(10^{-9})
u	micro(10^{-6})
m	milli(10^{-3})
c	centi(10^{-2})
d	deci(10^{-1})
da	deka(10^1)
h	hecto(10^2)
k	kilo(10^3)
M	mega(10^6)
G	giga(10^9)
T	tera(10^{12})
P	peta(10^{15})
E	exa(10^{18})
Z	zetta(10^{21})
Y	yotta(10^{24})

타임스탬프 컨트롤에서 시간과 날짜를 설정하기

높은 정확성을 갖는 절대 시간의 사용, 보기, 저장은 타임스탬프 컨트롤을 사용하십시오. 이 데이터 타입은 18개의 자릿수를 각각 초단위로 19개의 자리수를 초당 분할 단위로 정확하게 저장할 수 있습니다. 타임스탬프 데이터 타입은 UTC(Coordinated Universal Time: 세계시)에 값을 저장하지만, 타임스탬프 컨트롤은 현지 시간에 값을 디스플레이합니다.

숫자형 컨트롤을 사용하여 타임스탬프 값을 디스플레이할 수 있지만 숫자형 컨트롤은 상대적인 양을 가지고 있습니다. 타임스탬프 컨트롤은 절대적인 양을 가지고 있습니다.

다음 단계를 따라 타임스탬프 컨트롤에서 시간과 날짜 값을 설정하십시오.

1. 타임스탬프 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 수행 도구를 사용하여 시간/날짜 탐색 버튼을  클릭하고 시간과 날짜 설정 대화 상자를 디스플레이합니다. 또한, 타임스탬프 컨트롤에서 마우스 오른쪽 버튼

을 클릭하고 바로 가기 메뉴에서 데이터 처리>>시간과 날짜 설정을 선택하여 시간과 날짜 설정 대화 상자를 디스플레이할 수 있습니다.

3. 데이터를 위한 타임스탬프로서 시간과 날짜를 설정합니다.



노트

시간을 설정하려면, 설정하려는 분이나 초와 같은 시간 단위를 클릭합니다. LabVIEW는 시간의 선택된 단위를 하이라이트합니다. 위 화살표와 아래 화살표 키를 사용하여 알맞게 증가 또는 감소시킵니다. 또한, 선택된 시간 단위에 새 값을 입력할 수 있습니다.

또한, 타임스탬프 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 데이터 처리>>현재 시간으로 설정을 선택하여 시간과 날짜 값을 현재 시간과 날짜로 설정할 수 있습니다.

타임스탬프 컨트롤로부터 **프로퍼티 노드를 생성**하고 값 프로퍼티에 타임스탬프 상수를 연결하여 프로그램적으로 타임스탬프 컨트롤 값을 변경할 수 있습니다.



노트 타임스탬프 컨트롤과 함께 단위를 사용할 수 없습니다. 덧붙여서, National Instruments는 절대 시간으로 포맷되어 있는 숫자 컨트롤에 단위를 사용하지 않도록 권장합니다.

타임스탬프 컨트롤에서 높은 소수점 자릿수 디스플레이하기

다음 단계를 따라 **타임스탬프 컨트롤**을 위해 높은 소수점 자릿수를 디스플레이합니다.

1. 타임스탬프 컨트롤에서 마우스 오른쪽 버튼으로 클릭한 후 바로 가기 메뉴에서 디스플레이 포맷을 선택하여 타임스탬프 프로퍼티 대화 상자의 디스플레이 포맷 페이지를 디스플레이합니다.
2. 시간 포맷 섹션 맨 위에 있는 시간 타입 풀다운 메뉴에서 사용자 시간 포맷을 선택합니다.
3. HH:MM:SS 버튼을 클릭합니다.
4. 자릿수 필드에 소수점 자릿수를 입력합니다. 이 필드에 어떠한 숫자라도 넣을 수 있지만, LabVIEW는 초 이하 단위로 19 자릿수까지 정확하게 저장할 수 있습니다.
5. 확인 버튼을 클릭합니다. 타임스탬프 컨트롤은 지정한 소수점 자릿수의 숫자에 따라 업데이트됩니다. 모든 자릿수를 보기 위하여 컨트롤의 크기를 조정합니다.

시간 또는 날짜를 디스플레이하지 않으려면, 타임스탬프 프로퍼티 대화 상자의 디스플레이 포맷 페이지의 시간 타입 또는 날짜 타입 풀다운 메뉴에서 시간 사용안함 또는 날짜 사용안함을 선택합니다.

스케일에 임의의 마커 놓기

기본적으로, 스케일 마커는 균등한 간격입니다. 내부 마커를 스케일에 맞게 어디에 놓을지 정확하게 나타낼 수 있습니다. 이것은 포인트나 임계점과 같이 슬라이드에 특정한 몇 개의 포인트를 표시하는데 유용합니다.

다음 단계를 따라 특정한 포인트를 스케일에 따라 지정합니다.

1. 스케일에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 마커 간격» 임의를 선택합니다.
2. 새 마커를 추가하려는 스케일상의 위치에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 마커 추가를 선택합니다. LabVIEW는 해당 위치에 새 마커를 추가합니다. 또한, <Ctrl> 키를 누르면서 기존의 마커를 끌어서 마커를 추가하려는 곳에 놓습니다. 슬라이드 컨트롤과 인디케이터에서, 마커를 추가하려는 위치로 끌 마커 중에서 하나를 끌 수 있습니다. 회전식 컨트롤과 인디케이터에서, <Ctrl> 키를 누르고 마커를 추가하려는 위치로 끌 마커 중에서 하나를 끌어올 수 있습니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

임의의 마커를 삭제하려면, 마커에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 마커 삭제를 선택합니다.

스케일의 형을 변경하려면, 스케일에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **디스플레이 포맷**을 선택합니다.



노트

임의의 마커는 두 개의 끌 마커가 아닌 단지 내부 마커에 영향을 줍니다. 임의의 마커가 현재 범위에서 볼 수 없다면, 스케일은 일시적으로 균일한 마커로 되돌아 갑니다.

균일한 모드에서, 내부 마커를 끌어오면 마커 분포를 변경합니다. 임의의 모드에서 내부 마커를 끌어오면 다른 마커를 바꾸지 않고 단지 현재 마커를 움직입니다.

슬라이드와 회전식 컨트롤 및 인디케이터를 스케일하기

스케일 리미트를 변경할 수 있고, **텍스트 라벨**을 사용할 수 있고, 스케일 마커를 조정할 수 있고, 스케일을 가진 숫자형 객체에 선형 또는 로그 간격을 선택할 수 있습니다.

다음 단계를 따라 슬라이드 또는 회전식 객체의 스케일을 사용자 정의합니다.

1. 슬라이드 또는 회전식 객체에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 스케일을 선택하고 다음 아이템 중에서 선택합니다:
 - 마커 간격—이 옵션을 사용하여 균일하거나 임의의 스케일 마커를 선택합니다. 기본적으로, 스케일 마커는 균등하거나, 또는 균등한 간격입니다. 임의의 마커 간격을 선택하여, 스케일에 특정 내부 마커를 나타낼 수 있습니다.

다. **임의의 마커 간격**은 포인트 또는 임계점을 설정하는 것과 같이 슬라이드 또는 회전식 컨트롤에 일부 특정 포인트를 표기하는데 유용합니다.

- 마커 추가—스케일에 **마커를 추가**합니다. 마커 간격이 균일인 경우 이 옵션은 사용 가능하지 않습니다.
- 마커 삭제—스케일에서 **마커를 삭제**합니다. 마커 간격이 균일하고 스케일에 오직 두 개의 끝 마커만 포함되는 있는 경우, 이 옵션은 이용할 수 없습니다.
- 디스플레이 포맷—슬라이드 프로퍼티 대화 상자의 디스플레이 포맷 페이지를 디스플레이합니다. 디지털 객체에서와 마찬가지로 스케일을 가진 숫자형 객체의 **포맷과 정밀도**를 변경할 수 있습니다.
- 스타일—이 팔레트를 사용하여 스케일 디스플레이의 스타일을 선택합니다. 틱 마크를 가지거나 가지지 않은 스케일을, 또는 스케일 값을 가지거나 가지지 않은 스케일을 디스플레이하거나, 스케일을 모두 숨길 수 있습니다.
- 맵핑—이 옵션을 사용하여 선형이나 로그 스케일 간격을 선택합니다. 로그 스케일 간격을 선택하고 하위 스케일 리미트가 0보다 작거나 같다면, 그 리미트는 자동적으로 양의 숫자가 되고 이에 따라서 LabVIEW는 다른 마커의 값을 다시 계산합니다. 맵핑 함수를 포함하는 스케일 옵션은 데이터 입력 바로 가기 메뉴 항목을 사용하여 변경시키는 슬라이드 데이터 값의 범위에 관계 없다는 것을 유의하십시오. 데이터를 로그 값으로 제한하려면, **데이터 범위**를 변경하여 0보다 작거나 같은 값을 제거합니다.

스크롤 막대 컨트롤 사용하기

스크롤 막대 컨트롤을 사용하여 사용자 스크롤 막대를 스크롤 가능한 데이터를 갖는 컨트롤에 추가할 수 있습니다. 다음 단계를 따라 리스트박스에 스크롤 막대 컨트롤을 사용합니다.

1. 프런트패널에 리스트박스를 추가하고 **리스트박스에 항목을 입력**합니다.
2. 프런트패널에 수직 스크롤 막대를 추가합니다.
3. 리스트박스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 생성>> 프로퍼티 노드>> 최상위 행을 선택하고 블록 다이어그램에 최상위 행 프로퍼티를 추가합니다.
4. 최상위 행 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 것을 쓰기로 변경을 선택합니다.
5. 스크롤 막대 터미널을 최상위 행 프로퍼티에 연결합니다.
6. 프런트패널 스크롤 막대에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 데이터 입력을 선택하고 데이터 입력 페이지의 옵션을 설정하여 스크롤 막대를 사용자 정의합니다.
7. 연속 실행 버튼을 클릭하고 스크롤 막대 컨트롤의 값을 변경하여 리스트박스의 항목을 스크롤합니다.



노트

사용자 스크롤 막대를 사용하여 컨트롤을 스크롤하면 해당 컨트롤의 내장된 스크롤 막대는 비활성화됩니다.

또한 스크롤 막대 프로퍼티를 사용하여 프로그램적으로 스크롤 막대를 설정할 수 있습니다.

증가와 감소 버튼 사용하기

일반적으로 증가와 감소 버튼은 숫자 컨트롤에서 한 단위씩 변경합니다. 증가 또는 감소 버튼을 누르고 마우스 버튼을 계속 누르고 있는 경우, 디스플레이는 마우스 버튼을 놓을 때까지 증가하거나 감소합니다. 증가나 감소 버튼을 클릭하고, 마우스 버튼을 누르고, <Shift> 키를 누르는 경우, 증가의 크기는 점차적으로 높은 자릿수로 증가합니다. 처음에 값은 하나씩 증가하고, 그 후 10, 100 등의 단위로 증가합니다. 범위 경계에 접근하면, 증가는 자릿수대로 줄어들고, 값이 경계에 도달함에 따라 증가 속도가 느려집니다.

화살표 키를 사용하여 기본 증가 외의 자릿수로 숫자 디스플레이를 변경하려면, 수행 도구를 사용하여 삽입 포인트를 타겟 자릿수의 오른쪽에 추가합니다.

다음 단계를 따라 증가와 감소 버튼 사용을 연습합니다.

1. 숫자형 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 수행 도구를 사용하여, 증가와 감소 버튼을 클릭합니다. 값은 하나씩 증가하고 감소합니다.
3. 증가 버튼을 클릭하고 계속 누르고 있으면 반복해서 값이 증가합니다.
4. 위치 도구를 사용하여, 숫자 컨트롤을 크기 조정합니다. 컨트롤의 길이는 수직으로만 증가할 수 있습니다.
5. 다시 수행 도구를 사용하여, 증가 버튼을 누른 채로 <Shift> 키를 누릅니다. 값은 처음에 하나씩 증가하나, 그 후 10, 100 등의 단위로 증가합니다.
6. 디스플레이 윈도우의 안쪽을 더블 클릭하여 숫자 컨트롤의 값을 10으로 변경하고, 10을 입력하고 <Enter> 키를 누릅니다.
7. 증가 버튼을 두 번 클릭합니다. 값은 각 클릭 때마다 하나씩 증가하여 이제 값은 12가 됩니다.
8. 10씩 증가시키려면, 수행 도구를 사용하여 삽입 포인트를 10의 자리의 오른쪽에 추가합니다. 위 방향 화살표 키를 눌러 값을 22로 증가시킵니다. 삽입 포인트를 10 자리의 오른쪽에 놓으면 증가 값은 10으로 변경됩니다.
9. 삽입 포인트를 1자리의 오른쪽으로 이동하고 증가 버튼을 클릭합니다. 값은 23으로 증가합니다. 증가 단위는 다시 한자리가 됩니다.

숫자 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 보이는 항목 >> 증가/감소를 증가와 감소 버튼을 디스플레이하거나 숨깁니다. 또한, 증가/감소 보이 기? 프로퍼티를 사용하여 프로그램적으로 버튼을 디스플레이하거나 숨길 수 있습니다.

숫자형 데이터 타입에 대한 유효한 포맷 지정자

다음 테이블은 각 데이터 타입에 어떤 포맷 지정자가 유효한지 나타냅니다. 셀이 X를 포함하는 경우, 포맷은 해당 데이터 타입에 유효하지 않습니다. 셀이 ✓를 포함하는 경우, 포맷은 해당 데이터 타입에 유효합니다. 데이터 타입이 변경되고 현재 포맷이 새 데이터 타입에 허용되지 않거나 셀의 라벨이 ✓+로 지정된 경우, LabVIEW는 포맷을 해당 데이터 타입의 기본 포맷으로 변경합니다. 예를 들어, 배정도 부동소수 데이터 타입의 포맷을 절대

시간으로 설정하는 것은 유효하나, 배정도 부동소수 데이터 타입을 통형의 부호있는 정수 데이터 타입으로 변경하면 포맷은 10진 포맷으로 변경됩니다.

포맷 지정자	T	t	x	o	b	d	u	f	e	g	p	#	^
데이터 타입	절대 시간	상대 시간	16진수	8진수	2진수	부호있음	부호없음	부동소수	과학적 표기법	과학적 표기법 또는 부동소수 기반	SI 접두어	동밀도(이진 0 제거)	공학 표기법
타임스탬프	✓	X	X	X	X	X	X	X	X	X	X	X	X
복소수 타입(CXT, CDB, CSG)	X	X	X	X	X	✓-+	✓-+	✓	✓	✓	✓	✓	✓
실수 부동소수 타입(EXT, DBL, SGL)	✓-+	✓-+	X	X	X	✓	✓-+	✓-+	✓	✓	✓	✓	✓
정수 타입(U64, I64, U32, I32, U16, I16, U8, I8)	✓-+	✓-+	✓	✓	✓	✓	✓	✓-+	✓-+	✓-+	✓-+	✓-+	✓-+
고정 소수점 타입(FXP)	X	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

빈 경로

빈 경로는 빈 절대 경로이거나 빈 상대 경로입니다.

Windows

절대 경로는 파일 계층구조 내의 파일 위치를 가리키는 전체 주소를 포함합니다. 빈 절대 경로는 파일 계층구조의 가장 높은 포인트를 가리킵니다. 경로 컨트롤의 빈 절대 경로는 빈 문자열로 나타나며 파일 I/O 함수에 연결할 때 비어 있는 절대 경로는 컴퓨터에 연결된 드라이브의 리스트를 참조합니다.

상대 경로는 파일 계층구조 내에서 사용자의 위치에 상대적인 파일 위치의 주소를 포함합니다. 빈 상대 경로는 파일 계층구조 내에서 사용자의 위치를 가리킵니다. 이 경로는 마침표 .로 표시됩니다.

macOS 및 Linux

경로 컨트롤의 빈 절대 경로는 슬래쉬 / 로 나타나며, 파일 I/O 함수에 연결할 때 빈 경로는 루트 디렉토리를 참조합니다.

빈 상대 경로는 마침표 . 로 나타나며 파일 계층구조 내에서 사용자의 위치를 참조합니다.

기호형 경로

기호형 경로 또는 가상 경로는 LabVIEW에서 사용할 수 있는 경로의 미리 정의된 짧은 경로입니다. 기호형 경로를 사용하여 LabVIEW가 설치된 디스크 또는 로그인한 사용자 등과 같은 실행 환경에 따라 변경되는 위치를 가리킬 수 있습니다.

예를 들어 <vilib> 기호형 경로는 labview 디렉토리에서 <vi.lib> 디렉토리를 나타냅니다. 런타임 때 LabVIEW는 설치된 위치에 따라 기호형 경로 <vilib>를 C:\Program Files (x86)\National Instruments\LabVIEW 2009\vi.lib 또는 D:\Program Files\National Instruments\LabVIEW 2009\vi.lib와 같이 절대 경로로 해석합니다.

다음 테이블은 LabVIEW에서 사용할 수 있는 기호형 경로에 대한 정보를 제공합니다.

기호형 경로	설명	사용
<osdatadir>	OS에서 기본 파일 디렉토리의 경로.	이 기호형 경로는 옵션 대화 상자의 경로 페이지에서 VI 검색 경로를 설정할 때 사용합니다. 이 기호형 경로를 사용하려면 <osdatadir>:\를 입력하고 VI의 파일 이름 또는 경로 및 파일 이름을 입력합니다.
<vilib>	labview 디렉토리의 +vi.lib 디렉토리.	이 기호형 경로는 옵션 대화 상자의 경로 페이지에서 VI 검색 경로를 설정할 때 사용합니다. 이 기호형 경로를 사용하려면 <vilib>:\를 입력하고 VI의 파일 이름 또는 경로 및 파일 이름을 입력합니다.
<userlib>	labview 디렉토리의 user.lib 디렉토리.	옵션 대화 상자의 경로 페이지에 있는 VI 검색 경로 또는 VI 프로퍼티 대화 상자의 문서 페이지에 있는 도움말 경로를 설정할 때, 이 기호형 경로를 사용합니다. 이 기호형 경로를 사용하려면 <userlib>:\를 입력한 후, VI 또는 도움말 파일의 파일 이름 또는 경로 및 파일 이름을 입력합니다.
<instrlib>	labview 디렉토리의 instr.lib 디렉토리.	옵션 대화 상자의 경로 페이지에 있는 VI 검색 경로 또는 VI 프로퍼티 대화 상자의 문서 페이지에 있는 도움말 경로를 설정할 때, 이 기호형 경로를 사용합니다. 이 기호형 경로를 사용하려면 <instrlib>:\를 입력한 후, VI 또는 도움말 파일의 파일 이름 또는 경로와 파일 이름을 입력합니다.

기호형 경로	설명	사용
<helpdir>	labview 디렉토리의 help 디렉토리.	VI 프로퍼티 대화 상자의 문서 페이지에서 도움말 경로를 설정할 때, 이 기호형 경로를 사용합니다. 이 기호형 경로를 사용하려면 <helpdir>:\를 입력한 후 도움말 파일의 파일 이름 또는 경로 및 파일 이름을 입력합니다.
<resource>	labview의 resource 디렉토리이며, LabVIEW의 특정 버전을 지원하는 VI를 저장하는 디렉토리입니다.	사용자 어플리케이션에서 사용할 VI가 아니라 LabVIEW 편집기에 추가할 VI를 생성할 때 이 기호형 경로를 사용합니다. resource 디렉토리의 API는 이후의 LabVIEW 버전에서 변경될 수 있으며 이 때 하위 호환되지 않을 수도 있습니다. 이 기호형 경로를 사용하려면 <resource>:\를 입력하고 VI의 파일 이름 또는 경로 및 파일 이름을 입력합니다.
<ExtraVILib>	LabVIEW Data 디렉토리의 ExtraVILib 디렉토리.	이 기호형 경로를 사용하여 여러 LabVIEW 기능에 대한 템플릿에서 인스턴스화된 VI를 저장합니다. 예를 들면, LabVIEW는 이 기호형 경로에 채널 와이어 의 채널 인스턴스를 저장합니다. 이러한 채널 인스턴스는 채널 템플릿에서 생성되었습니다. VI를 생성하는데 사용한 LabVIEW 버전과 다른 비트와 다른 LabVIEW 버전을 사용하여 VI를 로드하는 경우, LabVIEW는 로드하는 VI가 필요로 하는 인스턴스의 복제본을 따로 생성하여 로드하는 LabVIEW 버전과 비트에서 이러한 VI를 사용할 수 있도록 합니다. 이 기호형 경로를 사용하려면 <ExtraVILib>:\를 입력하고 VI의 파일 이름 또는 경로 및 파일 이름을 입력합니다.

관련 정보

파일 경로

옵션 디스플레이

백슬래시('\') 코드 디스플레이

문자열 상수, 컨트롤, 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 '\' 코드 디스플레이를 선택하여 LabVIEW가 백슬래시(\) 뒤에 따라오는 문자를 디스플레이할 수 없는 문자의 코드로 해석하도록 합니다. 백슬래시 모드는 디버깅 VI와 디스플레이할 수 없는 문자를 인스트루먼트, 시리얼 포트, 그리고 다른 디바이스에 전송할 때 유용합니다. 다음 리스트는 LabVIEW가 어떻게 이 코드를 해석하는지를 보여줍니다.

코드	LabVIEW 해석
\00 - \FF	8 비트 문자열의 16진수 값이며 반드시 대문자여야합니다.
\b	백스페이스(\08과 동일한 ASCII BS)
\f	폼 피드(\0C와 동일한 ASCII FF)
\n	라인 피드(\0A와 동일한 ASCII LF) – [파일로 포맷] 함수는 자동으로 이 코드를 플랫폼에 의존적인 라인 끝 문자로 변환합니다.
\r	캐리지 리턴(\0D와 동일한 ASCII CR)
\t	탭(\09와 동일한 ASCII HT)
\s	스페이스(\20와 동일)
\\	백슬래쉬(\5C와 동일한 ASCII \)

16진수 문자에 대해서 대문자를 사용하고 폼 피드와 백스페이스와 같은 특수 문자에 대해서는 소문자를 사용합니다.

LabVIEW는 시퀀스 \BFare를 16진수 BF 다음에 단어 are가 뒤따라오는 것으로 해석합니다. \bFare와 \bfare는 백스페이스 다음에 단어 Fare와 fare가 붙은 것으로 해석합니다. 시퀀스 \Bfare에서, \B는 백스페이스 코드가 아니며, \Bf는 유효한 16진수 코드가 아닙니다. 이러한 경우, 백슬래쉬 뒤에 유효한 16진수 문자의 부분만 따라올 때 LabVIEW는 0이 백슬래쉬 뒤에 따라온다고 가정하므로 LabVIEW는 \B를 16진수 0B로 해석합니다. 백슬래쉬 뒤에 유효한 16진수 문자가 따라오지 않으면, LabVIEW는 백슬래쉬 문자를 무시합니다.

'\ ' 코드 디스플레이를 선택한 것과 상관없이 문자열 컨트롤에 키보드에서 캐리지 리턴과 같이 위 테이블에 나열된 몇몇 디스플레이할 수 없는 문자를 입력할 수 있습니다. 하지만, 디스플레이 윈도우가 텍스트를 포함하고 있을 때 백슬래쉬 모드를 활성화할 경우, LabVIEW는 디스플레이할 수 없는 문자의 백슬래쉬 표시와 \ 문자를 보여주기 위해서 디스플레이를 다시 그립니다.

다음 단계를 통해 '\ ' 코드 디스플레이 바로 가기 메뉴 아이템 사용을 연습합니다.

1. 문자열 컨트롤을 프런트패널 윈도우에 추가합니다. 기본 모드는 **일반 디스플레이**입니다.
2. 여러 라인의 텍스트를 디스플레이하도록 문자열 컨트롤의 **크기를 조정**하고 다음 문자열을 입력합니다. LEFT\RIGHT\3F
3. 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 '\ ' 코드 디스플레이를 선택합니다. LEFT후의 캐리지 리턴과 RIGHT 주위의 백슬래쉬 문자가 백슬래쉬 형태로 보여지므로 문자열은 다음과 같이 나타납니다. LEFT\n\RIGHT\3F
4. 프런트패널에 새 문자열 컨트롤을 추가하기 위해서 단계 1을 반복합니다.
5. 새 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 '\ ' 코드 디스플레이를 선택합니다.

6. 여러 라인의 텍스트를 디스플레이하도록 문자열 컨트롤의 **크기를 조정**하고 다음 문자열을 입력합니다. LEFT\RIGHT\3F
7. 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **일반 디스플레이**를 선택합니다. LabVIEW가 \r을 캐리지 리턴으로 해석하여 출력하기 때 문에 문자열은 다음과 같이 나타납니다. \3F는 물음표 ?의 ASCII 형입니다. LEFTRIGHT?
8. 다시 문자열 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 '\ 코드 디스플레이'를 선택합니다. 문자열은 다음과 같이 나타납니다. LEFT \nRIGHT?

인디케이터도 같은 방식으로 동작합니다. 문자열의 데이터는 한 모드에서 다른 모드로 바뀔 때 변경되지 않습니다. 오직 특정 문자의 디스플레이 표시만 변경됩니다.

문자열 컨트롤과 인디케이터 설정하기

문자열 상수, 컨트롤, 또는 인디케이터를 설정하기 위해, 객체에서 마우스 오른쪽 버튼을 클릭하고 다음의 바로 가기 메뉴 아이템으로부터 선택합니다:

일반 디스플레이

백슬래쉬('\') 코드 디스플레이

암호 디스플레이

16진수 디스플레이

한 줄로 제한

입력할 때 값 업데이트

자동 줄 바꿈 활성화

또한 문자열 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 보이는 항목 >> 디스플레이 스타일을 선택하여 디스플레이 타입을 나타내는 객체에서 문양을 디스플레이합니다.



관련 정보

[일반 디스플레이](#) (1315 페이지 참조)

[백슬래쉬\('\'\) 코드 디스플레이](#) (1311 페이지 참조)

[암호 디스플레이](#) (1315 페이지 참조)

[16진수 디스플레이](#) (1314 페이지 참조)

[한 줄로 제한](#) (1314 페이지 참조)

[입력할 때 값 업데이트](#) (1315 페이지 참조)

[자동 줄 바꿈 활성화](#) (1314 페이지 참조)

[문자열을 이용한 데이터의 그룹화](#) (410 페이지 참조)

[테이블](#) (413 페이지 참조)

자동 줄 바꿈 활성화

문자열 상수, 컨트롤, 또는 인디케이터에서 자동 줄 바꿈을 비활성화하기 위해서는 문자열에서 마우스 오른쪽 버튼을 클릭한 후 자동 줄 바꿈 활성화 바로 가기 메뉴 항목 옆의 확인 표시를 제거합니다. 자동 줄 바꿈을 비활성화하면 문자열이 라인 끝 부분에서 줄 바꿈이 됩니다. 자동 줄 바꿈을 비활성화하려는 경우, 문자열은 반드시 [일반 디스플레이](#) 모드이어야만 합니다. 자동 줄 바꿈을 비활성화한 후, 문자열 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 항목 >> 수평 스크롤 막대를 선택하여 수평 [스크롤 막대](#)를 디스플레이할 수 있습니다.

또한, 자동 줄 바꿈 활성화 프로퍼티를 사용하여 프로그램적으로 자동 줄 바꿈을 비활성화할 수 있습니다.

16진수 디스플레이

문자열 상수, 컨트롤, 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴로부터 16진수 디스플레이를 선택하여 각 문자의 ASCII 값을 문자 그 자체 대신에 16진수로 디스플레이합니다. 16진수 디스플레이 항목은 인스트루먼트를 디버깅하고 통신하는데 유용합니다.

한 줄로 제한

다음 단계를 따라 문자열 상수, 컨트롤 또는 인디케이터를 설정하여 문자열 내용을 한 줄로 제한합니다.

1. 문자열 상수, 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 한 줄로 제한을 선택합니다. 이 옵션을 사용하면 문자열 컨트롤이나 인디케이터에서 캐리지 리턴을 입력하는 것을 방지합니다.
2. 여러 줄 문자열을 복사하고 복사한 문자열을 문자열 컨트롤이나 인디케이터에 붙여넣기합니다. 여러 줄 문자열을 오직 한 줄만 입력할 수 있는 문자열 객체에 붙여넣기하면, LabVIEW는 첫번째 줄만을 붙여넣습니다.

또한, 한 줄로 제한? 프로퍼티를 사용하여 캐리지 리턴을 입력하는 것을 프로그램적으로 방지할 수 있습니다.

일반 디스플레이

문자열 상수, 컨트롤, 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 일반 디스플레이를 선택하여 디스플레이할 수 없는 문자를 제외한 모든 문자를 타이핑된 대로 디스플레이합니다. 디스플레이할 수 없는 문자는 일반적으로 상자로 나타납니다.

암호 디스플레이

문자열 상수, 컨트롤, 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 암호 디스플레이를 선택하여 스페이스를 포함하는 문자열 컨트롤에 입력하는 각 문자에 대해 별표 문자 *를 디스플레이합니다. 블록다이어그램에서 문자열 데이터를 읽을 때, 사용자가 입력한 실제 데이터를 읽습니다. 컨트롤로부터 데이터를 복사하려면, LabVIEW는 단지 * 문자를 복사합니다.

입력할 때 값 업데이트

문자열 상수, 컨트롤, 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 입력할 때 값 업데이트를 선택하여 사용자가 <Enter> 키를 누르거나 텍스트 입력을 끝낼 때까지 기다리지 않고 문자를 입력할 때 컨트롤의 값을 업데이트합니다. 이 옵션을 사용하여 입력이 올바른지 확인하거나 사용자 피드백을 줍니다.

또한, 입력할 때 업데이트? 프로퍼티를 사용하여 프로그램적으로 값을 업데이트할 수 있습니다.

라벨 붙이기

다음 라벨 타입을 사용하여 VI를 문서화합니다:

- 독립 라벨—블록다이어그램의 코드를 설명하거나 프런트패널에 사용자 설명을 추가할 때 사용합니다. 블록다이어그램에서 블록다이어그램의 객체에 독립 라벨을 첨부하여 첨부된 주석을 생성할 수 있습니다.

- **고유 라벨**—프론트패널 또는 블록다이어그램의 특정 객체에 붙입니다. 고유 라벨은 특정 객체에 속합니다. 고유 라벨을 숨길 수는 있지만, 라벨을 가진 객체에 독립적으로 복사하거나 삭제할 수는 없습니다.
- **캡션**—객체의 라벨에 포함할 수 없는 설명 및 정보를 제공합니다. 또한 라벨 대신 캡션을 사용하여 VI를 깨지 않고 VI를 현지화할 수 있습니다.
- **블리언 텍스트**—블리언 컨트롤과 인디케이터의 라벨을 생성하며, 이러한 라벨은 컨트롤과 인디케이터의 값에 따라 변합니다.

라벨을 생성한 후 편집할 수 있습니다. 또한 고유 라벨과 캡션을 이동하고 라벨의 텍스트 특성을 변경할 수 있습니다.



관련 정보

[독립 라벨 생성하기](#) (1319 페이지 참조)

[객체에 라벨 붙이기](#) (1321 페이지 참조)

[캡션 생성하기](#) (1318 페이지 참조)

[버튼의 라벨 편집하기](#) (1140 페이지 참조)

[라벨 편집하기](#) (1320 페이지 참조)

[고유 라벨과 캡션 이동하기](#) (1322 페이지 참조)

[텍스트 특성 변경하기](#) (1317 페이지 참조)

미리 지정된 위치로 라벨과 캡션 정렬하기

라벨과 캡션을 프론트패널 또는 블록다이어그램 객체를 둘러싼 미리 지정된 위치로 정렬합니다. 각 객체는 한 면에 세 개의 미리 지정된 위치를 갖습니다. [와이어 라벨](#)의 위치는 미리 정해져 있지 않습니다.



팁

새 프론트패널 및 블록다이어그램 객체의 기본 라벨 위치를 변경하려면 옵션 대화 상자의 프론트패널 및 블록다이어그램 페이지에서 기본 라벨 위치 폴다운 메뉴를 사용합니다.

다음의 단계를 따라 프론트패널 또는 블록다이어그램 객체 주위에 있는 미리 지정된 위치로 라벨 또는 캡션을 정렬합니다.

1. 프론트패널 또는 블록다이어그램 윈도우에 있는 라벨 또는 캡션을 선택합니다. 라벨 또는 캡션을 끌면, 파란색 기준선이 객체가 미리 지정된 위치로 어떻게 정렬되는지를 보여줍니다.
2. 프론트패널 또는 블록다이어그램 윈도우에 있는 라벨 또는 캡션을 원하는 위치에 놓습니다.

스페이스 바를 누르고 라벨 또는 캡션을 원하는 위치로 끌면, 미리 정해진 정렬 위치를 임시로 비활성화할 수 있습니다.

프로그램적으로 캡션 변경하기

다음 단계를 따라 프로그램에서 캡션을 변경합니다.

1. 숫자형 컨트롤을 프런트패널 윈도우에 추가합니다.
2. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 캡션을 선택합니다.
3. 블록 다이어그램상의 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성 >> 프로퍼티 노드 >> 캡션을 선택하고 나타난 리스트에서 옵션을 선택합니다.

또한 프로그램적으로 캡션을 보이고, 숨기고, 변경하기 위해서 프로퍼티 노드를 사용할 수 있습니다.

관련 정보

프로퍼티 노드

숫자형 컨트롤과 인디케이터

텍스트 특성 변경하기

LabVIEW는 **내장 폰트**를 기본 시스템 폰트로 정의합니다. VI를 플랫폼 사이에서 이동하는 경우, LabVIEW는 자동으로 내장 폰트를 업데이트하여 현재 플랫폼의 기본 시스템 폰트와 일치하도록 합니다. 사용할 수 없는 폰트를 사용하는 VI를 여는 경우, LabVIEW는 가장 비슷한 폰트로 이를 대체합니다. LabVIEW는 폰트와 비슷하게 색을 다룹니다. 컴퓨터에서 사용 가능한 색이 아닐 경우, LabVIEW는 가장 가까운 색으로 대체합니다. 또한, **시스템 색**을 사용하여 프런트패널의 모양을 VI를 실행하는 컴퓨터의 시스템 색에 맞출 수 있습니다.

다음 단계를 따라 텍스트의 폰트, 크기, 색을 변경합니다.

1. 텍스트의 폰트, 크기, 색을 변경하려는 객체를 선택합니다. 프런트패널 또는 블록 다이어그램의 모든 객체의 폰트 특성을 설정하기 위해서, 선택한 모든 객체의 선택을 해제합니다.
2. 도구 모음의 텍스트 셋팅 풀다운 메뉴에서 폰트 대화 상자를 선택하여 기본 폰트 대화 상자를 디스플레이합니다. 이 대화 상자는 폰트 스타일 대화 상자와 같은 옵션을 포함합니다.



노트

기본 폰트를 변경해도 기존 라벨의 폰트는 변경되지 않습니다. 오직 이후부터 생성하는 라벨에만 영향을 줍니다.

1. 폰트 풀다운 메뉴의 폰트를 선택합니다. LabVIEW는 컴퓨터에 현재 설치된 모든 폰트를 디스플레이합니다.

2. 화살표 버튼을 클릭한 후 풀다운 메뉴에서 크기를 선택하거나 크기 텍스트 박스에 크기를 입력하여 크기를 선택합니다.
3. 정렬 풀다운 메뉴에서 정렬 옵션을 선택합니다.
4. 색 상자를 클릭하면 나타나는 **색 선택기**에서 텍스트의 색을 선택합니다.



팁(Windows 및 Linux) 라벨을 투명하게 만들려면 색칠 도구를 사용하고 색 선택기의 T에서 마우스 오른쪽 버튼을 클릭합니다. (macOS) 라벨을 투명하게 만들려면 색칠 도구를 사용하고 색 선택기에서 <Control> 키와 함께 T를 클릭합니다.

1. 일반, 굵은체, 이탤릭체, 밑줄, 취소선, (macOS) 아웃라인, (macOS) 그림자 확인란에 확인 표시를 하거나 제거하여 텍스트를 어떻게 나타낼지를 선택합니다.
2. 수정 사항이 있는 프런트패널 텍스트가 텍스트 박스에 나타나 어떻게 나타날지를 보여줍니다.
3. 기본 패널 또는 기본 다이어그램 확인란에 확인 표시를 하여 새 텍스트 셋팅을 각각 프런트패널 또는 블록다이어그램 상의 새 라벨에 대한 기본 폰트로 사용합니다.
4. 확인 버튼을 클릭하여 이 텍스트 셋팅을 저장하거나 취소 버튼을 클릭하여 변경 사항을 저장하지 않고 기본 폰트 대화 상자를 끝냅니다.

또한 폰트 프로퍼티를 사용하여 프로그램적으로 폰트를 설정하고, 자리맞춤 프로퍼티를 사용하여 프로그램적으로 텍스트의 자리맞춤을 얻거나 설정하고, 텍스트 색 프로퍼티를 사용하여 프로그램적으로 텍스트 문자열의 전경색과 배경색을 변경할 수 있습니다.

캡션 생성하기

프런트패널 객체는 캡션도 가질 수 있습니다. 라벨 대신 캡션을 사용하여 VI를 깨지 않고 VI를 현지화할 수 있습니다. **라벨**과는 달리, 캡션은 객체의 이름에 영향을 미치지 않으며 캡션을 보다 설명적인 객체 라벨로 사용할 수 있습니다. 캡션은 프런트패널 윈도우에서만 나타납니다.

객체를 커넥터 팬 터미널에 지정할 경우, 캡션은 와이어링 도구를 사용하여 커서를 블록 다이어그램의 터미널 위로 움직일 때 팁 상자에 나타납니다. 캡션은 또한 커서를 커넥터 팬 또는 VI 아이콘 위로 움직일 때 기본 도움말 윈도우의 터미널 옆에 나타납니다.

객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 캡션을 선택하여 캡션을 보이거나 숨깁니다.



노트

캡션의 텍스트가 바뀌어도, LabVIEW는 VI 또는 그 VI의 호출자를 다시 컴파일할 필요가 없습니다.

또한 텍스트 프로퍼티를 사용하여 프로그램적으로 캡션의 텍스트를 변경하거나 캡션 프로퍼티를 사용하여 캡션을 참조할 수 있습니다.

캡션을 삭제하려면 참조를 캡션 프로퍼티에서 삭제 메소드로 연결합니다.

독립 라벨 생성하기

다음 단계를 따라 독립 라벨을 생성합니다:

1. 라벨링 도구를 사용하여 임의의 열린 공간을 클릭합니다. 자동 도구 선택이 활성화된 경우, 임의의 열린 공간을 더블 클릭합니다. 또한 컨트롤 또는 함수 팔레트의 라벨을 선택하고 각각 프런트패널 또는 블록다이어그램에 추가할 수 있습니다. 입력을 받을 준비가 된, 여백에 텍스트 커서를 가지는 작은 상자가 나타납니다.
2. 라벨에 나타내려는 텍스트를 입력합니다. 블록다이어그램의 라벨에 해쉬태그(#)를 추가하여 북마크를 생성할 수 있습니다. 프런트패널 또는 블록다이어그램에 URL 프로토콜://도메인 이름을 추가하여 인터넷 또는 네트워크상 파일로 연결된 하이퍼링크를 생성합니다. 프런트패널 라벨의 하이퍼링크를 비활성화하려면, 독립 라벨에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 하이퍼링크 활성화의 확인 표시를 제거합니다. 블록다이어그램 라벨의 하이퍼링크는 비활성화할 수 없습니다. 키보드의 <Enter> 키를 눌러 새 라인을 추가할 수 있습니다.
3. 라벨 외부로 클릭하여 편집 작업을 완료합니다. 또한, 텍스트를 입력할 때 도구 모음에 나타나는 텍스트 입력 버튼을 클릭하거나, 숫자 키패드에서 <Enter> 키를 누르거나, 키보드에서 <Ctrl-Enter> 키를 누를 수 있습니다. (macOS) <Option-Enter> 키를 누릅니다. (Linux) <Alt-Enter> 키를 누릅니다.

독립 라벨을 사용하여 VI에 주석을 추가합니다.



노트

옵션 대화 상자의 환경 페이지에서 <Enter> 키로 텍스트 입력 종료 확인란에 확인 표시를 한 경우 <Enter> 키로 편집 작업을 끝낼 수 있습니다.

첨부된 주석 생성하기

블록다이어그램에서 블록다이어그램의 객체에 독립 라벨을 첨부할 수 있습니다. 한 객체에 하나의 주석을 첨부할 수도 있고, 한 객체에 여러 개의 주석을 첨부할 수도 있습니다.

첨부된 주석을 생성하려면 독립 라벨 위로 마우스를 움직여서 오른쪽 아래 코너에 문양 이 나타나도록 합니다. 문양을 클릭한 후 블록다이어그램의 객체로 끌어 옵니다.

또한 문양을 클릭한 후 객체를 클릭하여 첨부할 수 있습니다.

주석을 분리하려면, 주석에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 객체에서 라벨 분리를 선택합니다.

관련 정보

환경 페이지

완료되지 않은 태스크 관리하기

옵션 대화 상자

도구 선택하기

라벨 편집하기

SubVI의 라벨은 편집할 수 없습니다. 그러나, **VI 설명을 생성**할 때 함수 라벨을 편집하여 블록다이어그램에서 함수의 사용을 반영할 수 있습니다. 예를 들어, [더하기] 함수의 라벨을 사용하여 어떤 값이 추가되고 왜 블록다이어그램의 해당 포인트에 추가되었는지 설명할 수 있습니다.



팁

고유 라벨은 런타임에서 편집할 수 없습니다. 관련 라벨을 프로그램적으로 편집하고 싶은 경우, **각 라벨을 캡션으로 대체**합니다. 캡션은 라벨과 똑같은 모양이며, 런타임에서 편집이 가능합니다.

다음 단계를 따라 독립 또는 고유 라벨을 편집합니다.

1. 필요한 경우, 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 **아이템 >> 라벨**을 선택하여 라벨을 디스플레이합니다. 구조, 함수, 상수와 와이어의 라벨은 기본으로 보이지 않게 설정되어 있습니다.
2. 라벨링 도구를 사용하여 라벨을 클릭하고 편집하려는 텍스트를 하이라이트합니다. 라벨을 더블 클릭하여 단어를 하이라이트하거나 라벨을 세 번 클릭하여 전체 라벨을 하이라이트합니다. **자동 도구 선택이 활성화**되어 있는 경우, 라벨을 더블 클릭하여 라벨링 도구로 스위치하여 편집하려는 텍스트를 하이라이트합니다.
3. 라벨 텍스트를 편집합니다. 키보드의 <Enter> 키를 눌러 새 라인을 추가할 수 있습니다.



노트

(Linux) 텍스트 라벨과 같이 텍스트를 포함한 컨트롤을 편집할 때, 텍스트를 먼저 복사하거나 자르지 않고 중간 마우스 버튼을 사용하여 가장 최근에 하이라이트된 텍스트를 붙여넣을 수 있습니다.

1. 라벨 외부를 클릭하여 편집 작업을 완료합니다. 또한, 도구 모음의 입력 버튼을 클릭하거나, 숫자 키패드에서 <Enter> 키를 누르거나, 키보드에서 <Ctrl-Enter> 키를 누를 수 있습니다. (macOS) <Option-Enter> 키를 누릅니다. (Linux) <Alt-Enter> 키를 누릅니다. 고유 라벨의 텍스트가 변경되면, LabVIEW는 VI를 다시 컴파일해야 하며 VI 호출자도 리컴파일해야 할 수 있습니다.



노트 옵션 대화 상자 환경 페이지의 <Enter> 키로 텍스트 입력 종료 확인란에 확인 표시를 한 경우 키보드의 <Enter> 키로 편집 작업을 완료할 수 있습니다.

객체에 라벨 붙이기

고유 라벨을 사용하여 특정 프런트패널과 블록다이어그램 객체에 주석을 추가합니다. LabVIEW에는 이름 라벨, 서브다이어그램 라벨 및 단위 라벨의 세 가지 라벨이 있습니다.

라벨을 크기 조정하고 라벨의 텍스트 특성을 변경할 수 있습니다.

특정 객체에 라벨 붙이기

이름 라벨을 사용하여 컨트롤, 인디케이터, 함수, 와이어와 같은 특정 객체를 식별합니다. 이름 라벨을 생성하려면, 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 항목 >> 라벨을 선택하여 **라벨을 편집**합니다.

구조에 라벨 붙이기

서브다이어그램 라벨을 사용하여 블록다이어그램의 구조에 포함된 코드를 설명합니다. 이벤트 구조 및 플랫폼 시퀀스 구조와 같이 여러 서브다이어그램이 있는 구조의 경우, LabVIEW는 각 서브다이어그램에 라벨을 붙입니다. 서브다이어그램 라벨을 생성하려면, 구조에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 항목 >> 서브다이어그램 라벨을 선택하여 **라벨을 편집**합니다.

숫자형 컨트롤 및 인디케이터에 측정 단위 나타내기

단위 라벨을 사용하여 숫자형 컨트롤 및 인디케이터에 측정의 물리적 단위를 나타냅니다.

라벨 및 캡션 고정하기

프런트패널 객체 주위의 여러 위치로 프런트패널 컨트롤 또는 인디케이터 라벨과 캡션을 이동시켜서, 사용자가 정의하는 사용자 인터페이스를 생성할 수 있습니다. 또한 블록다이어그램에서 터미널과 와이어 라벨을 이동시킬 수 있습니다. 라벨 또는 캡션을 원하는 위치로 옮긴 후에, 프런트패널 또는 블록다이어그램 객체에 라벨을 고정시킬 수 있습니다. 객체에 대한 라벨을 고정하여, 라벨을 움직일 때 객체 또한 움직이게 되면서 선택한 라벨의 위치가 바뀌는 것을 방지하십시오.



노트
라벨을 와이어에 고정할 수 없습니다.

다음의 단계를 따라 프런트패널 또는 블록다이어그램 객체에 라벨 또는 캡션을 고정합니다.

1. 프런트패널 또는 블록다이어그램 객체를 원하는 위치로 이동시킵니다.
2. 컨트롤, 인디케이터, 또는 터미널의 라벨에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 라벨 고정을 선택합니다.

이전 단계를 따라 바로 가기 메뉴에서 라벨 고정 확인 표시를 제거하여 라벨 또는 캡션의 고정을 해제할 수 있습니다.

또한 새 컨트롤, 인디케이터, 또는 터미널을 프런트패널이나 블록다이어그램에 추가할 때, 라벨 및 캡션이 자동으로 객체에 고정되도록 설정할 수 있습니다. 다음 단계를 따라 새 객체에 자동으로 라벨 및 캡션이 고정되도록 설정합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 프런트패널 그리고/또는 블록다이어그램을 선택합니다.
3. 기본으로 고정된 라벨 확인란에 확인 표시를 합니다.
4. 확인 버튼을 클릭하여 변경을 적용하고 옵션 대화 상자를 닫습니다.
5. 객체를 프런트패널이나 블록다이어그램에 추가합니다. 라벨이 고정되어 있습니다.

또한 고정 프로퍼티를 사용하여 프로그램적으로 라벨을 고정할 수 있습니다.

고유 라벨과 캡션 이동하기

프런트패널 객체 주위의 여러 위치로 프런트패널 컨트롤 또는 인디케이터 라벨과 캡션을 이동시켜서, 사용자가 정의하는 사용자 인터페이스를 생성할 수 있습니다. 또한 블록다이어그램에서 터미널과 와이어 라벨을 이동시킬 수 있습니다. 다음의 단계를 따라 프런트패널 또는 블록다이어그램 주위의 라벨 또는 캡션을 이동시킵니다.

1. 라벨 또는 캡션을 클릭합니다.
2. (옵션) 스페이스 바 키를 눌러서 **미리 지정된 정렬 위치**를 비활성화합니다.



노트

와이어 라벨의 위치는 미리 정해져 있지 않습니다. 고유 라벨은 와이어를 따라 아무 위치로 옮길 수 있습니다.

1. 원하는 객체 주변의 위치로 라벨을 끌어옵니다.
2. (옵션) 프런트패널 또는 블록다이어그램 객체에 **라벨 또는 캡션을 고정**시킵니다.



노트 라벨 또는 다른 객체를 컨트롤 또는 인디케이터 위에 또는 일부가 겹쳐지도록 놓으면 스크린 업데이트가 느려지고 컨트롤 또는 인디케이터가 깜박거릴 수 있습니다.

텍스트 라벨 회전하기

회전된 텍스트 라벨을 사용하여 회전되고 겹쳐진 수직 텍스트를 디스플레이할 수 있습니다. 어떤 텍스트 라벨도 회전할 수 있습니다.

텍스트 라벨을 회전하려면, 라벨에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 수직 배치를 선택하고 다음 옵션 중 하나를 선택합니다:

- 없음—라벨은 일반 텍스트로 나타납니다.
- 다층—라벨은 다층 텍스트로 나타납니다.
- 시계방향—텍스트가 시계방향으로 90° 회전되어 나타납니다. 따라서 아래에서 위로 읽는 수직 텍스트가 됩니다.
- 반시계방향—텍스트가 반시계방향으로 90° 회전되어 나타납니다. 따라서 아래에서 위로 읽는 수직 텍스트가 됩니다.

대화 상자 생성하기

[사용자에게 메시지 디스플레이] 익스프레스 VI 또는 [사용자 입력 요청] 익스프레스 VI를 사용하여 사용자 경고 또는 메시지를 포함하는 표준 대화 상자를 디스플레이하거나 사용자 이름과 암호와 같은 정보를 사용자가 입력하도록 요청합니다.



팁

대화 상자를 지나치게 많이 사용하지 않으려면, 상태 텍스트 윈도우를 사용하여 상대적으로 덜 심각한 경고나 메시지를 디스플레이합니다.

다음 단계를 따라 모든 플랫폼에서 표준 대화 상자와 비슷하게 보이는 사용자 **대화 상자**를 생성합니다.

1. 파일>>새로 만들기를 선택하여 새로 만들기 대화 상자를 디스플레이합니다. 이 대화 상자에는 내장 VI 템플릿이 나열됩니다.
2. 이벤트를 사용하는 대화 상자 템플릿을 선택하여 대화 상자를 만듭니다. 이 템플릿은 대화 상자에서 일반적으로 사용되는 프런트패널 및 블록다이어그램 객체를 포함합니다.
3. 확인을 클릭하여 새로 만들기 대화 상자를 닫고 편집할 사용자 정의 대화 상자를 엽니다.
4. VI의 프런트패널과 블록다이어그램을 만드는 동안 다음 지침을 사용하십시오.
 - 시스템 컨트롤을 사용하여 대화 상자를 디자인합니다.
 - 프런트패널 텍스트에 대화 상자 폰트를 사용합니다.
5. 프런트패널 컨트롤에 **키 조작을 지정**합니다. 확인 버튼에 <Enter> 키를, 취소 버튼에 <Escape> 키를, 도움말 버튼에 <F1> 키를 지정합니다.



노트

프런트패널이 여러 라인 문자열 컨트롤을 가지는 경우 확인 버튼에 <Enter> 키를 지정하지 마십시오.

1. 대화 상자에 나타내려는 객체만을 보이기 위해서 프런트패널 윈도우 크기를 줄입니다.
2. 프런트패널 윈도우 외형을 대화 상자로 설정하고 대화 상자 제목 표시줄의 제목을 입력합니다.

1. 파일>>VI 프로퍼티를 선택하고 VI 프로퍼티 대화 상자의 항목 풀다운 메뉴에서 윈도우 모양을 선택합니다.
2. 대화 상자 옵션을 선택합니다.
3. 윈도우 제목 섹션의 VI 이름과 동일 확인란에서 확인 표시를 제거하여 대화 상자 제목 표시줄에 나타나는 사용자 제목을 생성합니다. 확인란 옆의 필드에 대화 상자 이름을 입력합니다.
4. 확인 버튼을 클릭하여 변경을 저장하고 VI 프로퍼티 대화 상자를 닫습니다.

정렬 눈금을 사용하여 객체 정렬하기

다음 단계를 따라 객체를 놓을 때 정렬되도록 정렬 눈금을 사용합니다.

1. 필요한 경우, 다음 단계를 따라 눈금 정렬을 디스플레이하고 활성화합니다.
 1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
 2. 항목 리스트에서 프런트패널 또는 블록다이어그램을 선택합니다.
 3. 프런트패널 눈금 보이기 또는 블록다이어그램 눈금 보이기 확인란에 확인 표시를 합니다.
 4. 대응하는 눈금 정렬 활성화 확인란에 확인 표시를 합니다.
 5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.
2. 컨트롤과 함수 팔레트에서 객체를 선택합니다. 프런트패널 윈도우에서 객체를 놓기 전 객체를 움직이면, 붉은 기준선은 객체가 눈금에 어떻게 정렬되는지를 나타냅니다. 컨트롤을 끌거나, 놓거나, 크기를 조정할 때 <G> 키를 눌러서 눈금 정렬을 활성화 또는 비활성화할 수 있습니다.
3. 객체를 프런트패널이나 블록다이어그램 윈도우에 추가합니다.

편집>>패널에서 눈금 정렬 비활성화 또는 편집>>다이어그램에서 눈금 정렬 비활성화를 선택하여 눈금 정렬을 비활성화하고 보이는 눈금을 사용하여 객체를 수동으로 정렬합니다. 편집>>패널에서 눈금 정렬 활성화 또는 편집>>다이어그램에서 눈금 정렬 활성화를 선택하여 다시 눈금 정렬을 활성화합니다. <Ctrl-#> 키를 눌러서 눈금 정렬을 활성화하거나 비활성화할 수 있습니다. 프랑스어 키보드에서는, <Ctrl-~> 키를 누르십시오.

(macOS) <Command-#> 키를 누릅니다. (Linux) <Alt-#> 키를 누릅니다.

옵션 대화 상자의 프런트패널 페이지 또는 블록다이어그램 페이지에 있는 눈금 옵션을 사용하여 모든 새 VI의 눈금 옵션을 설정할 수 있습니다. 또한 VI 프로퍼티 대화 상자를 사용하여 **현재 VI의 눈금 크기를 설정**할 수 있습니다.

객체를 놓은 후 축을 따라서 객체를 정렬하려면, 도구 모음의 객체 정렬 풀다운 메뉴를 사용하거나 편집>>아이템 정렬을 선택합니다.

분리자 막대와 구획 생성하기

도구 모음이나 **상태 막대**와 같은 분리자 막대를 사용하여 프런트패널 윈도우에 전문적인 사용자 인터페이스를 만들 수 있습니다. 분리자 막대를 이미 컨트롤을 가진 프런트패

널에 추가하거나, 또는 새 VI의 프런트패널 윈도우에 추가할 수 있습니다. 다음 단계를 따라 프런트패널 윈도우에 분리자 막대를 추가하고 여러 팬을 만듭니다.

1. 새 VI를 만들거나 기존 VI를 엽니다.
2. 컨테이너 팔레트에서 수평 분리자 막대나 수직 분리자 막대를 선택합니다.
3. 분리자 막대를 원하는 프런트패널 윈도우에 추가합니다. 원하는 위치로 분리자 막대를 끌면 점선 라인이 구획상에서 분리자 막대의 위치를 나타냅니다. 점선이 붉은색이면, **정렬 눈금**이 활성화됩니다.
4. 두 개의 스크롤 가능한 구획이 프런트패널 윈도우에 나타납니다.

분리자 막대의 사용과 구획 생성에 대한 예제는 labview\examples\Controls and Indicators\Containers에 있는 Resizable Panel with Splitter Bars VI를 참조하십시오.

 예제 열기  관련 예제 찾기

상태 막대 만들기

도구 모음이나 상태 막대와 같은 분리자 막대를 이용하여 프런트패널 윈도우에 전문적인 사용자 인터페이스를 만들 수 있습니다. 다음 단계에 따라 분리자 막대를 사용하여 상태 막대를 만듭니다.

1. 새 VI를 만들거나 기존 VI를 엽니다.
2. 컨테이너 팔레트에서 수평 분리자 막대를 선택합니다.
3. 분리자 막대를 프런트패널에서 구획의 아래쪽에 추가합니다. 원하는 위치로 분리자 막대를 끌면 점선 라인이 구획상에서 분리자 막대의 위치를 나타냅니다. 점선이 붉은색이면, **정렬 눈금**이 활성화됩니다.
4. 두 개의 스크롤 가능한 구획이 프런트패널 윈도우에 나타납니다.
5. 사용하고자 하는 컨트롤을 구획의 아래쪽 상태 막대에 추가합니다.
6. 분리자 막대에서 마우스 오른쪽 버튼을 클릭하고 잠금을 선택하고, 바로 가기 메뉴에서 분리자 크기 조정 >> 분리자를 아래로 선택합니다.
7. 분리자 막대에서 마우스 오른쪽 버튼을 클릭하고 아래쪽 구획 >> 수평 스크롤 막대 >> 항상 끄기를 선택하고, 바로 가기 메뉴에서 아래쪽 구획 >> 수직 스크롤 막대 >> 항상 끄기를 선택하여 구획의 아래쪽 스크롤 막대를 숨깁니다.



노트

분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 아래쪽 구획 >> 수평 스크롤 막대 >> 항상 켜기와 아래쪽 구획 >> 수직 스크롤 막대 >> 항상 켜기를 선택하여 스크롤 막대를 디스플레이할 수 있습니다.

방금 생성한 상태 막대에 영향을 주지 않으면서 위쪽 구획에 객체를 추가할 수 있습니다.

도구 모음 만들기

도구 모음이나 상태 막대와 같은 분리자 막대를 이용하여 프런트패널 윈도우에 전문적인 사용자 인터페이스를 만들 수 있습니다. 다음 단계에 따라 분리자 막대를 사용하여 도구 모음을 만듭니다.

1. 새 VI를 만들거나 기존 VI를 엽니다.
2. 컨테이너 팔레트에서 수평 분리자 막대를 선택합니다.
3. 프런트패널에서 구획의 위쪽에 분리자 막대를 추가합니다. 원하는 위치로 분리자 막대를 끌면 점선 라인이 구획상에서 분리자 막대의 위치를 나타냅니다. 점선이 붉은색이면, **정렬 눈금**이 활성화됩니다.
4. 두 개의 스크롤 가능한 구획이 나타납니다.
5. 사용하고자 하는 컨트롤을 구획의 위쪽에 있는 상태 막대에 추가합니다.
6. 분리자 막대에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 고정과 분리자 크기 조정 >> 분리자를 위쪽으로 선택하여 프런트패널 윈도우의 크기를 조정하면 분리자 막대가 프런트패널 윈도우의 위쪽 위치에 따라 다시 위치하게 합니다.
7. 분리자 막대에서 마우스 오른쪽 버튼을 클릭하고 위쪽 구획 >> 수평 스크롤 막대 >> 항상 끄기를 선택하고, 바로 가기 메뉴에서 위쪽 구획 >> 수직 스크롤 막대 >> 항상 끄기를 선택하여 구획의 위쪽 스크롤 막대를 숨깁니다.



노트

분리자 막대에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 위쪽 구획 >> 수평 스크롤 막대 >> 항상 켜기와 위쪽 구획 >> 수직 스크롤 막대 >> 항상 켜기를 선택하여 스크롤 막대를 디스플레이할 수 있습니다.

분리자 막대 설정하기

도구 모음이나 상태 막대와 같은 분리자 막대를 이용하여 프런트패널 윈도우에 전문적인 사용자 인터페이스를 만들 수 있습니다. 분리자 막대에서 마우스 오른쪽 버튼을 누르고 바로 가기 메뉴에서 분리자 크기 조정과 같은 옵션을 선택하여 분리자 막대를 설정할 수 있습니다.

또한 분리자 프로퍼티를 사용하여 프로그램적으로 분리자 막대를 설정할 수 있습니다.

구획 설정하기

분리자 막대를 이용하여 도구 모음이나 상태 막대와 같은 전문 사용자 인터페이스를 만들 수 있습니다. 프런트패널에 분리자 막대를 추가하여 프런트패널을 구획으로 분리하는 여러개의 영역으로 나눌 수 있습니다. 각각의 구획은 고유의 프런트패널과 유사하게 동작합니다. 사용자는 각각의 구획을 개별적으로 스크롤할 수 있습니다. 각각의 구획에는 고유의 컨트롤 세트가 있습니다. 분리자 막대는 특정 구획에 있는 컨트롤을 다른 구획에 있는 컨트롤과 분리할 수 있지만 모든 컨트롤의 터미널은 동일 블록 다이어그램에 있습니다. 스크롤 막대에 마우스 오른쪽 버튼을 누르고 바로 가기 메뉴에서 옵션을 선택하여 구획을 설정할 수 있습니다.

또한 구획 프로퍼티를 사용하여 구획을 프로그램적으로 설정할 수 있습니다.

편집 모드로 런타임 프런트패널 경계 보기

LabVIEW는 편집 모드로 프런트패널을 수정하는 동안 런타임 프런트패널의 경계를 보여줄 수 있습니다.

런타임 프런트패널 경계를 보고 패널 크기를 설정하려면 다음 단계를 완료합니다.

1. 런타임 프런트패널에 보이게 할 컨트롤과 인디케이터를 찾습니다.
2. VI 프로퍼티 대화 상자의 윈도우 런타임 위치 페이지에서, 실행 시 구획을 원점으로 스크롤 확인란에 확인 표시를 합니다.
3. 위치를 중앙 또는 사용자 정의로 설정합니다.
4. 현재 패널 크기 사용 확인란에서 확인 표시를 제거합니다.
5. 폭과 높이를 설정하여 런타임 프런트패널의 크기를 설정합니다.
6. 확인을 클릭합니다. 구획 원점에 직사각형의 경계가 보이며, 이는 런타임 프런트패널 경계를 나타냅니다.



팁
구획 원점이 현재 구획 영역에서 보이지 않으면 편집 >> 구획 원점 조정을 선택하여 구획 원점을 현재 보이는 구획 영역의 왼쪽 위 코너로 조정합니다.

컨트롤과 인디케이터를 런타임 프런트패널 경계 안으로 끌어 놓습니다. VI가 실행 중일 때, 프런트패널은 경계 내의 영역만 보여줍니다.

템플릿 컨트롤 생성하기

다음 단계를 따라 템플릿 컨트롤을 생성합니다.

1. 템플릿으로 사용하려는 프런트패널 **컨트롤이나 인디케이터**에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급 >> 사용자 정의를 선택하여 **컨트롤 편집기** 윈도우를 디스플레이합니다.
2. 재사용하려는 구성요소만을 포함하기 위해서 사용자 컨트롤을 생성합니다.
3. 파일 >> 다른 이름으로 저장을 선택하여 파일 대화 상자를 디스플레이합니다.
4. 저장할 타입 풀다운 메뉴에서 템플릿 컨트롤을 선택합니다.
5. 템플릿 컨트롤의 이름을 입력한 후 확인 버튼을 클릭합니다.



주의
사용자 VI 및 다른 파일을 **LabVIEW와 함께 설치되는 .llb 파일**에 저장하지 마십시오.

최소 윈도우 크기 정의하기

다음 단계를 따라 프런트패널과 블록다이어그램의 최소 윈도우 크기를 지정합니다.

1. 파일»VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 윈도우 크기를 선택합니다.
3. 최소 패널 크기 섹션에서, 정의하려고 하는 윈도우의 최소 폭과 높이를 픽셀 단위로 입력합니다. 어느 차원에서도 윈도우를 1 픽셀보다 작게 크기 조정할 수 없습니다.



노트

단일 구획 프런트패널에서, 최소 크기는 스크롤 막대를 포함하지 않은 구획의 내용 영역을 의미합니다. 여러 구획 프런트패널 윈도우에서, 최소 크기는 모든 보이는 스크롤 막대를 포함한 전체 프런트패널을 의미합니다.

1. 확인 버튼을 클릭하여 변경을 저장합니다.

객체 삭제하기

프런트패널 객체를 삭제하기 위해, 분리자 막대보다는 위치 도구를 사용하여 **객체를 선택**하고 다음 동작 중에 하나를 수행합니다:

- 편집»잘라내기를 선택합니다.
- <Backspace> 키를 누릅니다.
- <Delete> 키를 누릅니다.



노트

이 방법으로 구조를 삭제하면 그 내용까지 삭제됩니다. **그 내용을 삭제하지 않고 구조를 제거**할 수 있습니다.

분리자 막대에서 인접하는 분리자를 제거하거나 또는 전체 분리자 계층구조를 제거할 수 있습니다. 다음 단계를 완료하여 프런트패널에서 분리자 막대를 삭제합니다.

1. 분리자 막대에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 인접하는 분리자 제거 또는 전체 분리자 계층구조 제거를 선택합니다.

선택한 분리자 막대가 상위 분리자 막대가 아니라면, LabVIEW는 바로 가기 메뉴의 전체 분리자 계층구조 제거 옵션을 비활성화합니다.

화면 해상도에 따라 윈도우 비율 유지하기

LabVIEW는 그 프런트패널 윈도우의 크기를 모니터의 해상도에 맞게 유지할 수 있습니다. VI가 윈도우 비율을 유지하도록 설정하면, 프런트패널 윈도우가 사용하는 화면의 퍼센트는 사용자 화면 해상도에 관계 없이 같습니다.

다음 단계를 따라 화면 해상도에 관하여 프런트패널 윈도우 비율을 유지합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. VI 프로퍼티 대화 상자 맨 위의 항목 풀다운 메뉴에서 윈도우 크기를 선택합니다.
3. 다른 모니터 해상도에서 윈도우의 비율 유지 대화 상자에 확인 표시를 합니다.
4. 확인 버튼을 클릭하여 변경을 저장합니다.

이 옵션을 선택하여 윈도우 비율을 유지합니다. 프런트패널 객체의 비율을 유지하기 위해서, 윈도우가 크기 조정될 때 프런트패널의 모든 객체 스케일 확인란에 확인 표시를 놓아야 합니다.

또한, 같은 비율 유지 프로퍼티를 사용하여 화면 해상도와 관계된 프런트패널 윈도우 비율을 프로그램적으로 유지합니다.

객체 움직이기

다음 단계를 따라 객체를 움직입니다.

1. 위치 도구를 사용하여 **객체를 선택**합니다.
2. 새 위치로 객체를 끌어옵니다. 다음의 방법을 사용하여 움직임의 방향이나 진행을 제한할 수 있습니다:
 - 객체 움직임을 수평이나 수직 방향으로 제한하기 위해, <Shift> 키를 누르고 객체를 끕니다. 움직임이 처음 객체를 움직이는 방향에 따라 제한됩니다.
 - 선택된 객체를 작고 정확하게 증가하도록 움직이기 위해, 객체를 움직이려는 각 픽셀에 한번씩 적절한 화살표 키를 누릅니다. 화살표 키를 잡고 동작을 반복시킵니다. 화살표 키를 따라 <Shift> 키를 눌러서 객체를 더 빠르게 움직입니다.



노트

정렬 눈금이 활성화되면, 화살표 키를 사용하여 객체를 움직이지 않는 한, 객체를 눈금 라인이나 점을 따라서만 움직일 수 있습니다. 화살표 키와 함께 <Shift> 키를 눌러서 객체를 눈금 라인이나 점에 따라 정렬합니다. 객체가 이미 정렬되어 있다면, 화살표 키와 함께 <Shift> 키를 눌러서 객체를 한 눈금만큼 움직입니다.

이동 작업을 취소하려면, <Esc> 키를 누릅니다. 그런다음 마우스 버튼을 놓습니다. 객체가 그 원래 위치에 그대로 있습니다.



노트 또한, 위치 도구는 객체의 크기를 다시 조정합니다. 객체의 크기를 조정하지 않으려면, 움직이는 객체의 중앙에 커서를 놓습니다. 객체의 코너를 클릭하여 그 **객체의 크기를 조정**합니다.

객체를 놓은 후 축을 따라서 객체를 정렬하려면, 도구 모음의 객체 정렬 풀다운 메뉴를 사용하거나 편집>아이템 정렬을 선택합니다.

라벨 크기 조정하기

위치 도구를 사용하여 라벨의 크기를 조정합니다. 일반적으로 라벨은 자동 크기 조정합니다. 즉, 텍스트 사각형은 입력하는 텍스트에 맞춰 자동으로 크기를 조정합니다. 라벨 텍스트는 캐리지 리턴을 입력하거나 라벨 박스의 크기를 조정하지 않는 이상 한 줄로 유지됩니다.

라벨에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 텍스트 크기에 맞춤을 선택하여 자동 크기 조정을 토글합니다.

라벨의 크기를 조정할 때, LabVIEW는 자동으로 자동 크기 조정을 끕니다.

라벨의 크기를 조정하는 것과 같은 방법으로 문자열 상수의 크기를 조정할 수 있습니다.

또한, 텍스트 크기에 맞춤? 프로퍼티를 사용하여 프로그램적으로 텍스트 객체의 크기가 포함된 텍스트에 따라 변경되는지 여부를 설정합니다.

여러 객체를 같은 크기로 크기 조정하기

객체 크기 조정 풀다운 메뉴를 사용하여 여러 프런트패널 객체를 같은 크기로 조정합니다. 모든 선택된 객체를 최대 또는 최소 객체의 폭과 높이로 크기 조정할 수 있으며, 모든 선택된 객체를 특정한 픽셀 크기로 조정할 수 있습니다.



노트

프런트패널에 분리자 막대를 추가하고 구획을 생성하면, <Shift> 키를 누르고 크기를 조정하려는 객체를 선택하여 여러 구획에 있는 객체들을 선택할 수 있습니다.

다음 단계를 따라 여러 프런트패널 객체의 크기를 조정합니다.

1. 같은 크기로 조정하려는 객체를 선택합니다.
2. 프런트패널 도구 모음의 **객체 간격 조절**과 **순서 재설정** 풀다운 메뉴 사이에 위치한 객체 크기 조정 풀다운 메뉴를 선택합니다.
3. 다음 옵션 중 선택합니다.
 - 최대 폭—선택된 객체의 폭을 가장 넓은 객체 폭으로 크기 조정합니다.
 - 최대 높이—선택된 객체의 높이를 가장 높은 객체 높이로 크기 조정합니다.

- 최대 폭과 높이—선택된 객체의 폭을 가장 넓은 객체 폭으로, 선택된 객체의 높이를 가장 높은 객체 높이로 크기 조정합니다.
- 최소 폭—선택된 객체의 폭을 가장 좁은 객체 폭으로 크기 조정합니다.
- 최소 높이—선택된 객체의 높이를 가장 낮은 객체 높이로 크기 조정합니다.
- 최소 폭과 높이—선택된 객체의 폭을 가장 좁은 객체 폭으로, 선택된 객체의 높이를 가장 낮은 객체 높이로 크기 조정합니다.
- 폭과 높이 설정—선택된 객체를 특정한 픽셀 크기로 조정하는데 사용할 수 있는 객체 크기 조정 대화 상자를 디스플레이합니다.

디지털 숫자형 컨트롤과 인디케이터 같은 일부 객체는 크기 조정할 때 수평 또는 수직으로만 크기가 조정됩니다. 노브와 같은 다른 객체는 크기 조정할 때 같은 비율을 유지합니다. 예를 들어, 크기를 조정하려는 높이의 선택된 객체가 디지털 숫자 컨트롤을 포함하는 경우, LabVIEW는 수직으로 크기를 변경할 수 있는 객체의 높이만을 변경하고 디지털 숫자 컨트롤의 높이는 변경하지 않습니다.

객체 크기 조정하기

다음 단계를 따라 대부분의 프런트패널 객체의 크기를 조정합니다.

1. 위치 도구를 객체 위로 이동합니다. 크기 조정 핸들  또는  이 객체의 크기를 조정할 수 있는 포인트에 나타납니다. 객체의 크기를 조정할 수 없는 경우, 크기 조정 핸들이 나타나지 않습니다.
2. 커서를 크기 조정 핸들 위로 이동하여 커서를 크기 조정 커서  로 변경합니다.
3. 경계가 원하는 객체 크기를 표시할 때까지 크기 조정 핸들을 클릭하고 끕니다.
4. 마우스 버튼을 놓습니다. 객체는 새 크기로 다시 나타납니다.

수직 또는 수평으로 제한하거나 객체의 현재 비례를 유지하려면, <Shift> 키를 누르면서 크기 조정 핸들을 클릭하고 끕니다.

중심점 주위로 객체의 크기를 조정하려면, <Ctrl> 키를 누르면서 크기 조정 핸들을 클릭하여 끕니다.

수직 또는 수평으로 제한하거나 객체의 현재 비율을 유지하면서 중심 포인트 주변에서 객체의 크기를 조정하려면, <Ctrl-Shift> 키를 누르면서 크기 조정 핸들을 클릭하고 끕니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

프런트패널에 객체를 추가할 때 객체의 크기를 조정하려면, 객체를 클릭하고 마우스 버튼을 누른 채 경계가 원하는 객체 크기를 표시할 때까지 마우스를 끕니다.



노트

정렬 눈금이 활성화된 경우, 눈금의 크기 단위로만 객체의 크기를 조정할 수 있습니다. 정렬 눈금을 일시적으로 비활성화하려면, 프런트패널에 객체를 추가하기 전에 <G>를 누릅니다.

다음 단계에 따라 크기 조정 작업을 취소합니다.

1. 경계가 사라질 때까지 크기 조정 핸들을 윈도우 밖으로 계속 끕니다.
2. 마우스 버튼을 놓습니다. 객체는 원래 크기로 유지됩니다.



팁 또한 <Esc> 키를 눌러서 크기 조정 작업을 취소할 수 있습니다.

틀 크기 프로퍼티를 사용하여, 프로그램적으로 스케일하면서 숫자형 컨트롤과 인디케이터의 크기를 재조정할 수 있습니다.

또한, 여러 프론트패널 객체의 크기를 조정하여 같은 크기로 할 수 있습니다.

객체 선택하기

이동하기, 복사하기, 삭제하기 등과 같은 여러 편집 작업 시, 객체를 선택해야 합니다. 선택되지 못한 객체를 클릭하거나 열린 공간을 클릭하여 현재 선택한 모든 것을 선택해제합니다. 프론트패널 객체와 블록다이어그램 객체를 동시에 선택할 수 없습니다.

다음 단계를 따라 객체를 선택합니다.

1. 화살표의 포인트가 선택하려는 객체 위에 있을 때까지 위치 도구를 움직입니다. 자동 도구 선택이 활성화되어 있으면, <Shift> 키를 눌러서 일시적으로 위치 도구로 전환할 수 있습니다.
2. 마우스 버튼을 클릭합니다. 선택표시를 나타내기 위해 움직이는 점선으로 된 아웃라인선이 객체를 하이라이트합니다.

또한, 객체 주위에 선택을 위한 사각형을 생성할 수 있습니다.

다음 단계를 따라 선택 사각형을 생성합니다.

1. 위치 도구를 사용하여 프론트패널 윈도우나 블록다이어그램에 윈도우에서 빈 공간을 클릭합니다.
2. 선택을 위한 사각형이 선택하려는 모든 객체를 포함하고 건드릴 때까지 대각선으로 끌어옵니다. 객체를 선택하면, 선택한 부분 전체가 회색 사각형으로 표시되고 선택한 객체가 하이라이트됩니다. 구조를 선택하면 구조의 배경색이 어둡게 변해서 구조가 선택되었음을 나타냅니다.



노트

기본적으로, 객체를 사각형으로 둘러싸서 선택하는 경우, 구조나 와이어 선분을 선택하려면 구조 전체를 선택하거나 와이어 선분의 중간 지점을 포함해서 선택해야 합니다. 사각형으로 선택할 때 스페이스바를 누르면, 사각형에 닿는 객체가 모두 선택됩니다. 스페이스바를 다시 누르면 기본 선택 방식으로 되돌아갑니다.

<Shift> 키를 누르고 객체를 클릭하여 추가적인 객체를 선택하거나 객체의 선택을 해제합니다.

자동 도구 선택이 활성화된 경우, 커서가 가리키는 객체의 선택을 해제하여 위치 도구를 다른 도구로 전환해야만 합니다.

하나의 프런트패널 객체를 스케일에 맞게 설정하기

프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 크기 조정에 따라 객체 스케일 조정을 선택하여 구획이 크기 조정될 때 객체가 자동적으로 스케일되도록 설정합니다.

하나의 프런트패널 객체를 자동적으로 스케일하도록 지정한 후에, 회색 라인이 일부 프런트패널 영역을 아웃라인으로 잡습니다. 그 영역은 스케일하려는 객체를 기준으로 하여 다른 프런트패널 객체의 위치를 정의합니다. 구획의 크기를 조정할 때, 자동적으로 스케일하도록 설정한 객체는 크기를 재조정하고 그 원래의 위치를 중심으로 위치를 재조정합니다. VI를 실행할 때 회색 라인이 사라집니다.

객체가 구획과 함께 스케일되지 않도록 설정하려면, 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 크기 조정에 따라 객체 스케일 조정을 선택하여 확인 표시를 제거합니다.

또한, 스케일링 모드 설정 메소드를 사용하여 단일 프런트패널 객체를 프로그램적으로 스케일할 수 있습니다.

프런트패널의 모든 객체를 스케일에 맞게 설정하기

다음 단계를 따라 프런트패널 윈도우의 크기를 재조정할 때 모든 프런트패널 객체를 자동적으로 스케일에 맞도록 설정합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 윈도우 크기를 선택합니다.
3. 윈도우가 크기 조정될 때 프런트패널의 모든 객체 스케일 확인란에서 확인 표시를 합니다.



노트

LabVIEW는 프런트패널에 **분리자 막대를 추가**할 때 이 옵션을 비활성화합니다. 구획의 스크롤 막대에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **구획 크기 조정>>구획에 따라 모든 객체 스케일 조정**을 선택하여 구획을 개별적으로 설정해야 합니다.

1. 확인 버튼을 클릭하여 변경을 저장합니다. 변경을 저장한 후에, 프런트패널의 크기를 변경하면 모든 프런트패널 객체가 자체의 크기를 재조정하고 원래 위치에 따라 위치를 변경합니다.

윈도우가 크기 조정될 때 프런트패널의 모든 객체 스케일을 선택한 후 하나의 객체가 스케일되기로 설정되면, 대화 상자가 나타나 하나의 특정 객체나 모든 프런트패널 객체를 위한 오토스케일링 셋팅 사이에서 선택하도록 입력요청합니다.

또한, 스케일링 모드 설정 메소드를 사용하여 모든 프런트패널 객체를 스케일에 맞도록 프로그램적으로 설정할 수 있습니다.

단일 프런트패널 객체를 구획에 맞게 피팅하기

컨트롤을 구획에 맞춤 옵션을 사용하여, 컨트롤 크기가 구획 크기와 같도록 설정합니다. 구획의 크기가 조정되면, 컨트롤은 자동적으로 구획과 같은 크기가 되고 스케일됩니다. 또한, 구획에 있는 스크롤 막대가 사라집니다.

1. 객체에서 마우스 오른쪽 버튼을 클릭한 후 **바로 가기 메뉴**에서 컨트롤을 구획에 맞춤을 선택합니다.
2. 객체는 구획에 맞도록 크기가 조정됩니다.

또한 구획의 크기가 조정되는 동안 컨트롤이 자동으로 스케일되도록 설정할 수 있습니다. **분리자 막대**에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 구획 크기 조정 >> 크기 조정에 따라 객체 스케일 조정을 선택합니다. 구획에서 스케일하는 아이템은 모두 사용자가 구획 크기 조정을 완료했을 때가 아닌, 사용자가 구획의 크기를 조정하는 동안에 스케일됩니다.

프런트패널에서 장식 사용하기

장식 팔레트에 위치한 장식을 사용하여 상자, 라인 그리고 화살표와 함께 프런트패널 객체를 그룹으로 모으거나 분리합니다. 이 객체들은 단지 장식을 위한 것이지 데이터를 디스플레이하지 않습니다.

프런트패널에 장식을 추가한 후에, 순서 재설정 풀다운 메뉴를 사용하여 각 객체 위에 쌓인 **객체의 순서를 재설정**합니다.

또한, 블록 다이어그램의 장식을 사용할 수 있습니다.

타입 정의와 엄격한 타입 정의 생성하기

타입 정의 또는 엄격한 타입 정의는 사용자 컨트롤 또는 인디케이터로, 사용자 컨트롤 또는 인디케이터를 위해 저장된 파일에 링크됩니다. 사용자 컨트롤이나 인디케이터를 타입 정의로 저장한 경우, 타입 정의에서 데이터 타입을 변경하면 이를 사용하는 모든 VI의 모든 인스턴스의 타입 정의가 영향을 받습니다. 사용자 컨트롤이나 인디케이터를 엄격한 타입 정의로 저장한 경우, 엄격한 타입 정의에서 데이터 타입에서 외관을 변경하면 엄격한 타입 정의의 모든 프런트패널 인스턴스가 영향을 받습니다. 그러나, 엄격한 타입 정의를 변경하면, 상수로 **블록 다이어그램에 놓은** 인스턴스가 링크가 엄격하지 않은 것처럼 작동하며, LabVIEW는 데이터 타입 변경에서만 상수를 업데이트합니다.

다음 단계를 따라 타입 정의 또는 엄격한 타입 정의를 생성합니다.

1. 타입 정의로 만들려는 컨트롤, 인디케이터 또는 상수에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의 만들기를 선택합니다.

2. 타입 정의의 인스턴스에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 타입 정의 열기를 선택하여 **컨트롤 편집기** 윈도우를 디스플레이합니다.
3. (옵션) 엄격한 타입 정의를 생성하려면, 도구 모음의 컨트롤 타입 폴다운 메뉴에서 엄격한 타입 정의를 선택합니다.
4. 컨트롤 또는 인디케이터에 원하는 변화를 만듭니다. 컨트롤 또는 인디케이터의 크기, 색, 그리고 원소의 상대적인 위치 그리고 컨트롤 또는 인디케이터에 반입하는 이미지를 변경할 수 있습니다.
5. 파일>>변경된 사항 적용을 선택하여 컨트롤이나 인디케이터에 변경된 사항을 적용합니다. 변경된 사항 적용 메뉴 아이템은 컨트롤 또는 인디케이터를 변경한 이후에만 가능합니다. 컨트롤 또는 인디케이터를 변경한 후 파일>>변경된 사항 적용을 선택하지 않고 컨트롤 편집기 윈도우를 닫으려 할 경우, LabVIEW는 변경을 저장하려는지 물어보는 메시지를 디스플레이합니다.
6. 파일>>저장을 선택하여 타입 정의 또는 엄격한 타입 정의로 사용자 컨트롤 또는 인디케이터를 저장합니다. 타입 정의 또는 엄격한 타입 정의로 디렉토리 또는 **LLB**에 저장할 수 있습니다.

저장된 타입 정의와 저장된 엄격한 타입 정의 편집하기

다음 단계를 따라 저장된 **타입 정의**와 저장된 **엄격한 타입 정의**를 편집합니다.

1. 파일>>열기를 선택하여 파일 대화 상자를 디스플레이합니다.
2. 편집하려는 타입 정의 또는 엄격한 타입 정의를 탐색합니다.
3. 타입 정의 또는 엄격한 타입 정의를 선택하고 타입 정의 또는 엄격한 타입 정의를 더블 클릭을 하거나 열기 버튼을 클릭하여, **컨트롤 편집기** 윈도우에 타입 정의 또는 엄격한 타입 정의를 디스플레이합니다.
4. 타입 정의 또는 엄격한 타입 정의를 원하는 대로 변경합니다. 타입 정의의 데이터 타입에 대한 변경만 타입 정의의 인스턴스에 영향을 줍니다. 데이터 타입 변경, 외관 변경, 또는 링 컨트롤에서 아이템 이름을 변경하는 것과 같이 엄격한 타입 정의에 대한 대부분의 변경은 엄격한 타입 정의의 모든 인스턴스에 영향을 줍니다.
5. 파일>>저장을 선택하여 타입 정의 또는 엄격한 타입 정의를 저장합니다.

수동으로 타입 정의 인스턴스 업데이트하기

LabVIEW가 자동으로 타입 정의 인스턴스를 업데이트할 수 없는 경우, LabVIEW는 해당 인스턴스를 결정되지 않은 상태에 놓습니다. 결정되지 않은 인스턴스는 비활성화되고, 이러한 인스턴스를 포함하는 VI의 실행 버튼은 깨져서 나타납니다.

다음 단계를 완료하여 데이터 매핑 차이를 해결하고, 결정되지 않은 타입 정의 인스턴스를 수동으로 업데이트합니다:

1. 결정되지 않은 타입 정의 인스턴스에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 타입 정의로부터 검토 및 업데이트를 선택하여 타입 정의로부터 검토 및 업데이트 대화 상자를 엽니다.

- 인스턴스 리스트박스에서 인스턴스를 선택하여 해당 인스턴스에 대한 이전 기본값과 지정될 새 기본값을 디스플레이합니다.
- 새 기본값의 기본값을 검토하고 올바르지 않은 값을 모두 수정합니다. LabVIEW는 LabVIEW 맵핑 처리 과정에서 손실되거나 잘못 놓여진 데이터가 있는 컨트롤이나 상수를 하이라이트합니다.
- 승인 버튼을 클릭하여 인스턴스 리스트박스에서 현재 인스턴스를 승인된 것으로 표시합니다.



노트

변경 사항을 승인한 후에도 기본값을 보고 변경할 수 있습니다. LabVIEW는 사용자가 변경 사항을 승인한 후에 다시 값을 변경하면, 새 기본값에서 승인된 인스턴스를 컨트롤 또는 상수에 마지막으로 나타난 값으로 업데이트합니다.

- 타입 정의로부터 업데이트하려는 각 인스턴스에 대해 단계 2부터 단계 4까지 반복합니다.

노트 또한 모두 승인 버튼을 클릭하여 각 인스턴스를 개별적으로 일일이 검토하지 않고 인스턴스 리스트박스에 나열된 모든 인스턴스에 대한 모든 변경사항을 승인할 수 있습니다.

- 변경된 사항 적용 버튼을 클릭하여 승인한 인스턴스의 현재값과 기본값을 업데이트하고 대화 상자를 닫습니다. 승인하지 않은 인스턴스는 사용자가 승인할 때까지 또는 타입 정의에서 연결을 해제할 때까지 결정되지 않은 상태로 남아있습니다.

또한 타입 정의로부터 업데이트 메소드를 사용하여 프로그램적으로 모든 타입 정의 인스턴스 업데이트를 승인하고 적용할 수 있습니다. 그러나, LabVIEW가 프로그램적으로 업데이트하는 인스턴스에서는 기본값이 손실되거나 올바르게 보존되지 않을 수 있습니다. 인스턴스에 사용자 기본값이 있고 타입 정의를 크게 변경할 계획이 있다면, 타입 정의의 인스턴스를 프로그램적으로 업데이트하지 마십시오.

타입 정의 또는 엄격한 타입 정의의 링크 재설정하기

컨트롤 또는 인디케이터와 **타입 정의 또는 엄격한 타입 정의** 사이의 링크를 재설정하려는 경우, 컨트롤 또는 인디케이터를 타입 정의 또는 엄격한 타입 정의의 인스턴스로 대체해야 합니다.

다음 단계를 따라 컨트롤 또는 인디케이터를 타입 정의 또는 엄격한 타입 정의로 대체합니다.

- 대체하려고 하는 컨트롤 또는 인디케이터에서 마우스 오른쪽 버튼을 클릭합니다.
- 바로 가기 메뉴에서 대체를 선택하여, 컨트롤 팔레트의 임시 버전을 디스플레이합니다.
- 컨트롤 선택 아이콘을 클릭하여 파일 대화 상자를 디스플레이합니다.
- 대체하려고 하는 타입 정의 또는 엄격한 타입 정의를 탐색합니다.
- 타입 정의 또는 엄격한 타입 정의를 선택하고, 타입 정의 또는 엄격한 타입 정의에서 더블 클릭을 하거나 확인 버튼을 클릭하여, 프런트패널 컨트롤 또는 인디케이터를 교체합니다.

SubVI와 익스프레스 VI

SubVI와 익스프레스 VI를 아이콘 또는 확장 가능 노드로 디스플레이하기

VI와 익스프레스 VI를 아이콘 또는 확장 가능 노드로 나타낼 수 있습니다. 확장이 가능한 노드는 색이 있는 배경 안에 놓인 아이콘으로 나타납니다. 확장 가능 노드로 디스플레이 되는 경우, SubVI는 노란색 배경 안의 아이콘으로 나타납니다, 익스프레스 VI는 파란색 배경 안의 아이콘으로 나타납니다. 블록다이어그램의 공간을 효율적으로 사용하려면 아이콘을 사용하십시오. 확장 가능한 노드를 사용하면 블록다이어그램에서 와이어 연결과 문서화가 용이합니다. 블록다이어그램에서 기본 설정으로 SubVI는 아이콘으로 나타나고, 익스프레스 VI는 확장 가능한 노드로 나타납니다.



노트

SubVI 또는 익스프레스 VI를 확장 가능 노드로 디스플레이하는 경우, 해당 노드에 대한 **데이터베이스 접근을 활성화**할 수 없습니다.

다음 단계를 따라 SubVI 또는 익스프레스 VI를 아이콘으로 디스플레이합니다.

1. SubVI 또는 익스프레스 VI에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 아이콘으로 보기를 선택합니다. SubVI 또는 익스프레스 VI가 아이콘으로 나타납니다.

아이콘으로 보기 옵션에서 확인 표시를 삭제하여 SubVI와 익스프레스 VI를 확장 가능 노드로 디스플레이합니다.



노트 블록다이어그램에 추가한 새 SubVI만이 확장 가능 노드로 나타납니다. 아이콘으로 디스플레이할 수 있는 모든 기존의 SubVI를 확장 가능 노드로 변경할 수 있는 것은 아닙니다. 기본적으로 블록다이어그램에 익스프레스 VI를 아이콘으로 추가할 수 없습니다.

확장 가능 SubVI 또는 익스프레스 VI는 **크기 조정**할 수 있습니다. 확장 가능 subVI 또는 익스프레스 VI를 크기 조정할 때, subVI 또는 익스프레스 VI의 입력과 출력 터미널은 아이콘 아래에 나타납니다.

또한, SubVI로 사용될 때 터미널 보이기 프로퍼티를 사용하여 프로그램적으로 확장 가능 노드로 SubVI를 디스플레이할 수 있습니다.

입력과 출력을 삽입하고 제거하기

입력과 출력 터미널을 SubVI나 익스프레스 VI의 확장 가능한 필드에 삽입할 수 있습니다. 삽입할 수 있는 터미널은 SubVI나 익스프레스 VI가 포함하는 입력과 출력에 달려 있습니다. 예를 들어, 먼저 SubVI의 커넥터 팬에 **프론트패널 컨트롤 또는 인디케이터를 할당**해야 SubVI의 확장된 필드에 그 컨트롤 또는 인디케이터를 삽입할 수 있습니다.

다음 단계를 따라 SubVI나 익스프레스 VI에서 입력이나 출력을 삽입합니다.

1. **확장 가능 노드**로서 SubVI나 익스프레스 VI를 디스플레이합니다
2. SubVI나 익스프레스 VI의 확장된 필드의 입력이나 출력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 입력/출력 삽입을 선택합니다. 새로운 입력 또는 출력 터미널이 마우스 오른쪽 버튼을 클릭한 입력 또는 출력 위에 나타납니다.

입력 또는 출력을 제거하기 위해, 제거하려는 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 입력 제거 또는 출력 제거를 선택합니다.

익스프레스 VI

기존 VI로부터 익스프레스 VI 생성하기

다음 단계에 따라 기존의 표준 VI에서 **익스프레스 VI**를 생성합니다.

1. 도구 » 고급 » 익스프레스 VI 생성 또는 편집을 선택하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다.
2. 새로 만들기 버튼을 클릭하여 익스프레스 VI 생성자 마법사의 초기화 페이지를 디스플레이합니다.
3. VI에서 생성 옵션을 선택합니다.
4. 다음 버튼을 클릭합니다.
5. 활성화되는 파일 대화 상자에서, 새 익스프레스 VI의 기반이 될 표준 VI를 찾고 확인 버튼을 클릭합니다. VI 변환 페이지가 익스프레스 VI 생성자 마법사에 나타납니다.
6. VI 터미널이 **설정 가능 또는 확장 가능**할지를 결정한 후 해당 확인란에 확인 표시를 하거나 제거합니다.
7. 다음 버튼을 클릭하여 익스프레스 VI 생성자 마법사의 익스프레스 VI 설정 페이지를 디스플레이합니다.
8. 새 익스프레스 VI의 이름을 익스프레스 VI 이름 텍스트 박스에 입력합니다.
9. 새 익스프레스 VI의 위치를 익스프레스 VI 경로 텍스트 박스에서 지정합니다. 기본 디렉토리는 labview\user.lib_express입니다. 이 디렉토리에 익스프레스 VI를 저장하는 경우, 익스프레스 사용자 라이브러리 팔레트와 익스프레스 VI 생성 또는 편집 대화 상자에 나타납니다.



노트

(macOS 및 Linux) 사용자가 생성하는 익스프레스 VI의 기본 위치는 LabVIEW 기본 데이터 디렉토리입니다. user.lib 디렉토리에 대해 쓰기 권한이 없는 사용자도 있기 때문입니다.

10. 마침 버튼을 클릭하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다. 익스프레스 VI 트리에 새 익스프레스 VI가 나타납니다.
11. 익스프레스 VI 리스트에서 익스프레스 VI 아이템을 확장하여, **소스 VI**와 **설정 대화 상자 VI**를 디스플레이합니다.

12. 소스 VI를 열어서 완성합니다. 확장 가능으로 설정된 원본 VI의 모든 컨트롤 및 인디케이터, 커넥터 팬, VI 아이콘이 모두 소스 VI에 포함되어 있을 것입니다. 또한, 소스 VI의 VI 아이콘은 익스프레스 VI의 아이콘으로도 나타납니다.
13. 설정 대화 상자 VI를 열어서 **프런트패널**과 **블록다이어그램**을 완성합니다.
14. VI를 저장하고 닫습니다.

기존 익스프레스 VI로부터 익스프레스 VI 생성하기

다음 단계에 따라 기존의 익스프레스 VI에서 **익스프레스 VI**를 생성합니다.

1. 도구 >> 고급 >> 익스프레스 VI 생성 또는 편집을 선택하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다.
2. 새로 만들기 버튼을 클릭하여 익스프레스 VI 생성자 마법사의 초기화 페이지를 디스플레이합니다.
3. 익스프레스 VI에서 생성 옵션을 선택합니다.
4. 다음 버튼을 클릭하여 익스프레스 VI 복사 페이지를 익스프레스 VI 생성자 마법사에서 디스플레이합니다.
5. 익스프레스 VI를 선택하거나 익스프레스 VI 리스트에 있는 탐색 아이템을 클릭하여, 새 익스프레스 VI의 기본이 될 익스프레스 VI를 찾습니다.
6. 다음 버튼을 클릭하여 익스프레스 VI 생성자 마법사의 익스프레스 VI 설정 페이지를 디스플레이합니다.
7. 새 익스프레스 VI의 이름을 익스프레스 VI 이름 텍스트 박스에 입력합니다.
8. 새 익스프레스 VI의 위치를 익스프레스 VI 경로 텍스트 박스에서 지정합니다. 기본 디렉토리는 labview\user.lib_express입니다. 이 디렉토리에 익스프레스 VI를 저장하는 경우, 익스프레스 사용자 라이브러리 팔레트와 익스프레스 VI 생성 또는 편집 대화 상자에 나타납니다.



노트

(macOS 및 Linux) 사용자가 생성하는 익스프레스 VI의 기본 위치는 LabVIEW 기본 데이터 디렉토리입니다. user.lib 디렉토리에 대해 쓰기 권한이 없는 사용자도 있기 때문입니다.

1. 마침 버튼을 클릭하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다. 익스프레스 VI 리스트에 새 익스프레스 VI가 나타납니다.
2. 익스프레스 VI 리스트에서 익스프레스 VI 아이템을 확장하여, **소스 VI**와 **설정 대화 상자 VI**를 디스플레이합니다.
3. 소스 VI를 열어서 완성합니다.
4. 설정 대화 상자 VI를 열어서 **프런트패널**과 **블록다이어그램**을 완성합니다.
5. VI를 저장하고 닫습니다.

새 VI로부터 익스프레스 VI 생성하기

다음 단계에 따라 처음부터 **익스프레스 VI**를 생성합니다.

1. 도구 » 고급 » 익스프레스 VI 생성 또는 편집을 선택하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다.
2. 새로 만들기 버튼을 클릭하여 익스프레스 VI 생성자 마법사의 초기화 페이지를 디스플레이합니다.
3. 새 VI에서 생성 옵션을 선택합니다.
4. 다음 버튼을 클릭하여 익스프레스 VI 생성자 마법사의 익스프레스 VI 설정 페이지를 디스플레이합니다.
5. 새 익스프레스 VI의 이름을 익스프레스 VI 이름 텍스트 박스에 입력합니다.
6. 새 익스프레스 VI의 위치를 익스프레스 VI 경로 텍스트 박스에서 지정합니다. 기본 디렉토리는 `labview\user.lib\_express`입니다. 이 디렉토리에 익스프레스 VI를 저장하는 경우, 익스프레스 사용자 라이브러리 팔레트와 익스프레스 VI 생성 또는 편집 대화 상자에 나타납니다.



노트

(macOS 및 Linux) 사용자가 생성하는 익스프레스 VI의 기본 위치는 LabVIEW 기본 데이터 디렉토리입니다. `user.lib` 디렉토리에 대해 쓰기 권한이 없는 사용자도 있기 때문입니다.

7. 마침 버튼을 클릭하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다. 익스프레스 VI 리스트에 새 익스프레스 VI가 나타납니다.
8. 익스프레스 VI 리스트에서 익스프레스 VI 아이템을 확장하여, **소스 VI**와 **설정 대화 상자 VI**를 디스플레이합니다.
9. 소스 VI를 열어서 완성합니다.
10. 설정 대화 상자 VI를 열어서 **프런트패널**과 **블록 다이어그램**을 완성합니다.
11. VI를 저장하고 닫습니다.

설정 대화 상자 VI 만들기

익스프레스 VI 생성자 마법사로 새 **익스프레스 VI**를 생성하고 **소스 VI**를 설계한 후, **설정 대화 상자 VI**의 **프런트패널**을 설계하여 사용자 인터페이스를 완성할 수 있고, 그 다음에 **제공된 블록 다이어그램 템플릿**을 수정하여 익스프레스 VI의 런타임 동작을 설정할 수 있습니다.

다음 단계를 완료하여 설정 대화 상자 VI의 프런트패널을 만듭니다. 또한, 사용자 인터페이스를 만들 때 **LabVIEW Style Checklist** 및 기타 Development Guidelines를 사용하십시오.

1. 익스프레스 VI 생성 또는 편집 대화 상자에서 설정 대화 상자 VI를 엽니다.

2. 확인, 취소, 도움말 버튼은 제외하고, 프런트패널의 주요 선택 프레임에서 설정 대화 상자 안에 보이도록 만들고 싶은 컨트롤 및 인디케이터를 놓습니다.



노트

표준 VI로부터 익스프레스 VI를 생성한 경우, 사용자가 표준 VI에서 **설정 가능**으로 선택한 컨트롤과 인디케이터는 자동으로 프런트패널에 나타납니다.

3. 오직 주요 선택 프레임 및 확인, 취소, 도움말 버튼만 디스플레이되도록 프런트패널의 크기를 조정합니다. VI를 저장한 이후에는 설정 대화 상자가 이 셋팅을 유지하게 됩니다.
4. 설정 대화 상자 VI를 저장합니다.

다음 단계를 완료하여 설정 대화 상자 VI의 블록다이어그램을 만듭니다.

1. 블록다이어그램의 초기화 섹션에 있는 컨트롤 데이터 배열 안에서 익스프레스 VI에 대한 설정 가능한 파라미터를 지정합니다. 컨트롤 데이터 배열의 각 원소는 3개의 필드를 갖고 있습니다.
 1. 첫번째 필드에서, 컨트롤 타입은 프런트패널 컨트롤, 상수, 익스프레스 VI 제목, 다이나믹 데이터 타입 컨트롤 등과 같이 소스 VI에 있는 컨트롤의 타입을 지정합니다.
 2. 두번째 필드에서, 소스 VI 컨트롤 라벨에는 소스 VI의 프런트컨트롤에 보이는 것과 같은 컨트롤의 라벨을 입력합니다.

노트 소스 VI에 있는 컨트롤 라벨에서 보이는 것과 똑같이 대소문자를 구분하여 문자열을 정확하게 입력해야 합니다. 정확성을 높이려면, 소스 VI의 프런트패널 라벨을 복사하고 붙여넣습니다.

3. 세번째 필드에서, 설정 대화 상자 컨트롤 라벨에는 설정 대화 상자 VI의 프런트컨트롤에 보이는 것과 같은 컨트롤의 라벨을 입력합니다. 비어있는 문자열은 해당 라벨이 소스 VI의 라벨과 동일하다는 표시입니다.
2. 블록다이어그램의 주요 While 루프 섹션을 설정하여, 그래프 업데이트하기 또는 라디오 버튼 사용하기 등의 사용자 인터페이스 피드백을 처리합니다. 이 섹션의 이벤트 구조는 값 변경과 같은 사용자 이벤트와 직접 연계된 모든 코드를 위한 것입니다.
3. 블록다이어그램의 재설정 섹션에 있는 확장할 터미널 배열 안에서 익스프레스 VI에 대한 **설정 가능 터미널**을 지정합니다.
4. 설정 대화 상자 VI를 저장하고 닫습니다.



노트 익스프레스 VI에서 설정 대화 상자 VI를 시작하지 않고 자체적으로 실행하면 에러 메시지가 반환됩니다. 익스프레스 VI의 일부로서 VI를 실행하지 않고 있기 때문에 발생하는 에러인데, 이 에러를 무시해도 됩니다.

기본 도움말 윈도우에서 설정 셋팅 디스플레이하기

도움말 생성 VI를 사용하여 기본 도움말 윈도우에 **익스프레스 VI의 현재 설정을 디스플레이**합니다. 도움말 생성 VI는 블록다이어그램에서 익스프레스 VI 위에 마우스가 지나갈 때 기본 도움말 윈도우에 디스플레이하려는 컨트롤의 값을 지정합니다. 또한 컨트롤 이름과 같은 텍스트를 지정하여, 기본 도움말 윈도우에서 컨트롤 값 앞에 나오도록 만들 수

있습니다. 도움말 생성 VI는 익스프레스 VI 생성자 마법사를 통해 **생성**한 모든 익스프레스 VI의 **설정 대화 상자 VI**에서 SubVI로 나타납니다. 이 SubVI의 이름은 genHelp ExpressVIName입니다.

기본 도움말 윈도우가 자동적으로 업데이트되어 익스프레스 VI의 현재 설정이 디스플레이 되도록 다음 단계에 따라 도움말 생성 VI를 편집합니다.

1. 설정 대화 상자 **블록다이어그램**의 재설정 섹션에서 도움말 생성 SubVI를 더블 클릭하여 엽니다.
2. 도움말 생성 VI의 블록다이어그램에서, 기본 도움말 윈도우에 있는 첫 번째 컨트롤의 값 앞에 놓으려는 텍스트를 기본 도움말 이름 상수에 입력합니다. 예를 들어, 첫 번째 컨트롤의 이름을 입력하고 뒤에 콜론과 한칸 띄기를 할 수 있습니다.
3. 컨트롤 이름 상수에 값을 디스플레이하고 싶은 첫 번째 컨트롤의 이름을 입력합니다. 상수에서 다른 텍스트는 모두 삭제합니다.



노트

대소문자를 구분하여 컨트롤 데이터 배열에서 보이는 것과 똑같이 문자열을 입력해야 합니다. 정확성을 높이려면, 프런트패널 라벨에서 복사하고 붙여넣습니다.

4. 열거형 컨트롤일 경우, ex_PPGetProp SubVI의 열거형 & T/F 문자열 입력에서 상수를 생성하고 컨트롤에 나타나는 순서와 똑같은 순서로 열거형 문자열을 입력합니다.
5. 두 번째 컨트롤에 대한 설정을 디스플레이하려는 경우, ex_PPGetProp SubVI를 복사하여 원본 SubVI 오른쪽에 놓습니다. 그렇지 않으면, 단계 14로 건너뛰십시오.
6. SubVI 복사본의 출력 클러스터 입력에 출력 클러스터 컨트롤을 연결합니다.
7. 원본 SubVI에서, 첫 번째 컨트롤의 데이터 타입에 대응하는 출력을 SubVI 복사본의 이전 문자열 입력에 연결합니다. 예를 들어, 첫 번째 컨트롤이 열거형인 경우, 원본 SubVI의 열거형 결과 출력을 SubVI 복사본의 이전 문자열 입력에 연결합니다.

노트 첫 번째 컨트롤이 배정도 부동소수 데이터 타입이고 컨트롤의 값을 직접적으로 디스플레이하려는 경우, 원본 SubVI의 dbl 문자열 출력을 SubVI 복사본의 이전 문자열 입력에 연결합니다. 그러나, 값을 기본 도움말 윈도우에 값을 디스플레이하기 이전에 값을 포맷할 수도 있습니다. 이러한 경우, 예를 들어 [문자열로 포맷] 함수를 사용하여, 원본 SubVI의 dbl 출력을 포맷하고, 그 결과로 나오는 문자열을 SubVI 복사본의 이전 문자열 입력에 연결하십시오.

8. SubVI 복사본의 접두어 입력에 상수를 생성합니다.
9. 기본 도움말 윈도우 상에서 두 번째 컨트롤의 값 앞에 나오게 하려는 텍스트를 접두어 상수에 입력합니다.
10. SubVI 복사본의 컨트롤: 입력에 상수를 생성합니다.
11. 컨트롤: 상수에 값을 디스플레이하고 싶은 두 번째 컨트롤의 이름을 입력합니다.
12. 열거형 상수일 경우, 4번째 단계를 반복합니다.
13. 기본 도움말 윈도우에 값을 디스플레이하려는 컨트롤 모두에 대해 각각 5번째 단계에서 12번째 단계까지 반복합니다.
14. 마지막 ex_PPGetProp subVI에서, 마지막 컨트롤의 데이터 타입에 대응하는 출력을 도움말 설명 인디케이터에 연결합니다.
15. 도움말 생성 VI와 설정 대화 상자 VI를 저장하고 닫습니다.

블록다이어그램에 익스프레스 VI를 놓으면, 사용자가 설정 대화 상자를 닫을 때마다 기본 도움말 윈도우는 가장 최근의 컨트롤 값을 업데이트합니다.

익스프레스 VI 검증하기

검증 테스트를 실행하여 사용자가 생성하는 익스프레스 VI에서 흔히 발생하는 문제를 찾을 수 있습니다. 다음 단계를 완료하여 익스프레스 VI에 대한 검증 테스트를 실행합니다.

1. 도구 » 고급 » 익스프레스 VI 생성 또는 편집을 선택하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다.
2. 익스프레스 VI 리스트에서 검증하려는 익스프레스 VI를 선택합니다.
3. 검증 버튼을 클릭하여 검증 대화 상자를 디스플레이합니다. 검증 대화 상자에는 익스프레스 VI에 대한 테스트 결과 리스트가 디스플레이됩니다. 초록색 확인 표시는 익스프레스 VI가 해당 테스트를 통과했음을 나타냅니다. 빨간색 확인 표시는 익스프레스 VI가 해당 테스트를 통과하지 못했음을 나타냅니다.
4. 특정 테스트 이름을 클릭하여 해당 테스트에 대한 설명을 확인할 수 있습니다.
5. 확인 버튼을 클릭하여 익스프레스 VI 생성 또는 편집 대화 상자로 되돌아갑니다.

익스프레스 VI 인스턴스에 대한 사용자 제목 옵션 추가하기

익스프레스 VI의 각 인스턴스는 서로 다른 이름을 갖습니다. LabVIEW는 연속적인 숫자를 사용하여 익스프레스 VI 각 인스턴스에 고유한 이름을 붙입니다. 블록다이어그램에서 확장 가능한 노드에서 익스프레스 VI의 이름을 더블 클릭하여 해당 인스턴스의 이름을 편집합니다. 또한 익스프레스 VI의 인스턴스에 대해 사용자 정의 제목을 추가할 수 있는 옵션을 설정 가능한 대화 상자에 생성할 수 있습니다.

다음 단계를 완료하여 익스프레스 VI 인스턴스에 사용자 정의 제목을 추가하는 옵션을 생성합니다.

1. 익스프레스 VI의 설정 대화 상자 VI를 엽니다.
2. 프런트패널 주요 선택 프레임에 시스템 문자열 컨트롤을 놓습니다. 컨트롤에 익스프레스 VI 이름이라는 라벨을 붙입니다.
3. 블록다이어그램의 초기화 섹션에서, 컨트롤 데이터 배열의 원소 중에서 사용하지 않는 첫 번째 원소를 찾고 컨트롤 타입으로 제목을 선택합니다.
4. 익스프레스 VI 이름이라고 해당 원소에 대한 소스 VI 컨트롤 라벨 컨트롤에 입력합니다.



노트

프런트패널에 있는 문자열 컨트롤 라벨에서 보이는 것과 똑같이 대소문자를 구분하여, 문자열을 정확하게 입력해야 합니다. 정확성을 높이려면, 프런트패널 라벨에서 복사하고 붙여넣습니다.

5. 설정 대화 상자 VI를 저장하고 닫습니다.
6. 익스프레스 VI의 복사본을 블록다이어그램에 놓습니다.
7. 설정 대화 상자에서, 원하는 제목을 익스프레스 VI 이름 컨트롤에 입력합니다.

- 확인 버튼을 클릭하여 설정 대화 상자를 닫습니다. 익스프레스 VI에 새로 만든 제목이 보입니다. 익스프레스 VI를 편집하여 **현재 설정 셋팅을 디스플레이**하면, 기본 도움말 윈도우에도 새 제목이 나타납니다.

익스프레스 VI 이름 변경 및 이동하기

익스프레스 VI 구성요소는 특정 경로 및 명명 규칙을 통해 연계되어 있으므로, 수동으로 구성요소의 이름을 변경하거나 구성요소를 옮기면 익스프레스 VI가 깨집니다. 익스프레스 VI의 이름열 변경하거나 익스프레스 VI를 옮기려면 익스프레스 VI 생성 또는 편집 대화 상자를 사용해야 합니다.

다음 단계를 완료하여 익스프레스 VI의 이름을 변경하거나, 같은 컴퓨터 상의 다른 위치로 익스프레스 VI를 옮깁니다.

- 도구 >> 고급 >> 익스프레스 VI 생성 또는 편집을 선택하여 익스프레스 VI 생성 또는 편집 대화 상자를 디스플레이합니다.
- 이름을 변경하거나 익스프레스 VI 리스트에서 옮기려는 익스프레스 VI를 선택합니다.



노트

익스프레스 VI의 이름을 변경할 때, LabVIEW는 **소스 VI**와 **설정 대화 상자 VI**의 이름을 모두 바꿉니다.

- 이름 바꾸기 버튼을 클릭하여 익스프레스 VI 이름 바꾸기 대화 상자를 디스플레이합니다.
 - 익스프레스 VI의 이름을 바꾸려는 경우, 새 익스프레스 VI 이름 텍스트 박스에 새 이름을 입력합니다. 기본적으로, LabVIEW는 원본 익스프레스 VI 이름 이 번호를 순차적으로 추가합니다.
 - 익스프레스 VI를 옮기려는 경우, 익스프레스 VI의 원래 이름을 새 익스프레스 VI 이름 텍스트 박스에 입력하고 상위 디렉토리 텍스트 박스에 새 위치를 지정합니다.
- 확인 버튼을 클릭하여 익스프레스 VI 생성 또는 편집 대화 상자로 되돌아갑니다. 익스프레스 VI 리스트에 이름이 바뀐 익스프레스 VI가 나타납니다. 익스프레스 VI를 옮기는 경우, 업데이트된 경로가 경로 텍스트 박스에 나타납니다.

데이터를 다이나믹 데이터 타입으로 변환하기

다음 단계를 따라 숫자, 불리언, 웨이브폼, 배열 데이터를 익스프레스 VI에서 사용하기 위해서 다이나믹 데이터로 변환합니다.

- 블록 다이어그램에 [다이나믹 데이터로 변환] 익스프레스 VI를 추가합니다.
- 활성화되는 설정 대화 상자에서 어떤 데이터를 다이나믹 데이터로 변환시키고 싶은지를 지정하고 확인 버튼을 클릭합니다.
- 변환하려는 데이터를 다이나믹 데이터로 변환 익스프레스 VI의 입력에 연결합니다.

4. 다이나믹 데이터로 변환 익스프레스 VI의 출력을 다이나믹 데이터를 받는 다른 익스프레스 VI, VI, 또는 함수에 연결합니다.



노트

익스프레스 VI로부터 변환된 VI와 같은 몇몇 VI, [신호 병합]과 [신호 분리] 함수와 같은 함수는 다이나믹 데이터를 입력으로 받습니다.

다이나믹 데이터를 다른 데이터 타입으로 변환하기

다음 단계를 따라 다이나믹 데이터 타입을 다른 VI와 함수에서 사용하기 위해서 숫자, 웨이브폼, 배열 데이터 타입으로 변환합니다.

1. 블록 다이어그램에 [다이나믹 데이터로부터 변환] 익스프레스 VI를 추가합니다.
2. 활성화되는 설정 대화 상자에서 어떤 데이터로 다이나믹 데이터를 변환시키고 싶은지를 지정하고 확인 버튼을 클릭합니다.
3. 다이나믹 데이터가 반환하는 출력을 [다이나믹 데이터로부터 변환] 익스프레스 VI의 다이나믹 데이터 타입 입력에 연결합니다.



노트

대부분의 익스프레스 VI는 다이나믹 데이터를 출력으로 반환합니다.

1. [다이나믹 데이터로부터 변환] 익스프레스 VI의 출력을 그 결과적인 데이터 타입을 받는 VI, 함수, 또는 인디케이터에 연결합니다.

센서 맵핑 익스프레스 VI를 사용하여 데이터 맵핑하기

독립 센서 및 DAQmx 채널을 사용하여 현실 세계의 데이터를 3D 모델에 맵핑합니다. 센서 맵핑 익스프레스 VI를 사용하여 센서 설정 대화 상자를 통해 설정하는 센서를 나타내는 데이터 값 배열을 와이어로 연결합니다. DAQmx 채널을 나타내도록 센서를 설정하거나 익스프레스 VI에 연결한 데이터를 독립 센서가 나타내도록 생성할 수 있습니다. DAQmx 채널의 사용에 대한 추가적인 정보는 NI-VISA 도움말을 참조하십시오.

다음 단계에 따라 센서 맵핑 익스프레스 VI를 사용하여 3D 모델로 현실 세계의 데이터를 맵핑합니다.

1. [센서 맵핑] 익스프레스 VI를 블록 다이어그램에 추가합니다.
2. 센서 설정 대화 상자에서 모델 로드 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다.
 1. 로드하려는 3D 모델을 탐색합니다.

2. 확인 버튼을 클릭하여 모델을 선택하고 파일 대화 상자를 닫습니다.
LabVIEW는 모델 경로 필드에 3D 모델의 경로를 디스플레이합니다.



노트

LabVIEW는 3D 모델에 대해 VRML(Virtual Reality Modeling Language) 및 STL(stereolithography) 파일 포맷을 지원합니다.

3. (옵션) DAQmx 태스크 이름 풀다운 메뉴에서 DAQmx 태스크를 선택하십시오.
LabVIEW는 채널 리스트를 DAQmx 태스크의 채널로 채웁니다.

노트 DAQmx 태스크 이름 옵션은 사용자가 NI-DAQmx를 설치하기 이전에는 나타나지 않습니다.

4. 리스트에서 3D 모델로 채널을 끌어와서 센서를 추가합니다. DAQ 채널로부터 센서를 생성할 때 센서는 채널 데이터가 아니라, 채널 이름을 나타냅니다. 또한, 모델 상에서 어디서나 마우스 오른쪽 버튼을 클릭하여 독립 센서를 추가할 수 있습니다. 시연지기 3D 모델에 추가하는 센서는, 센서가 나타내는 데이터 값에 따라, 센서 포인트에 있는 3D 모델의 색상에 영향을 미칩니다.
5. (옵션) 그리기 스타일 옵션을 설정하여 3D 모델의 그리기 스타일을 변경합니다. 이 옵션을 사용하여 모델의 포인트, 선, 정점을 보고 센서가 정확한 위치에 놓여졌는지 점검합니다.
6. (옵션) 센서 크기 옵션을 증가 또는 감소시켜서 모델에 있는 센서의 크기를 재조정합니다. 이 옵션을 사용하여 센서가 정확한 위치에 놓여졌는지 점검합니다.
7. 색 램프 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴 항목을 사용하여 모양, 크기, 색, 색의 개수를 사용자 정의합니다. 또한, 컨트롤을 사용하여 들어오는 데이터의 최소값 및 최대값 범위를 설정할 수 있습니다. 색 맵 최대값 또는 최소값을 하이라이트한 후, 새 값을 입력하고 <Enter> 키를 눌러서 값의 범위를 변경합니다. 값 범위를 설정한 후, 최대 및 최소 범위를 벗어난 색을 클릭하여 사용자가 설정한 값 범위 밖에 있는 값을 표시할 색을 설정합니다.
8. 확인 버튼을 클릭하여 설정을 저장하고 대화 상자를 닫습니다.
9. **데이터 값의 배열을 생성**하고 배열을 익스프레스 VI의 센서 값 입력에 연결합니다. 배열 위치는 센서 설정 대화 상자에서 사용자가 설정한 센서의 순서에 대응합니다.

노트 사용자가 와이어로 연결한 배열이 포함하는 값의 개수가 모델에 추가한 센서의 개수보다 적은 경우, LabVIEW는 최대 범위를 벗어난 색으로 나머지 센서를 맵핑합니다.

10. 3D 그림 컨트롤을 프런트패널 윈도우에 추가합니다.
11. 블록 다이어그램에서, [센서 맵핑] 익스프레스 VI의 장면 참조 출력 출력을 3D 그림 컨트롤 터미널에 연결합니다.
12. 3D 그림 컨트롤 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 생성>>인보크 노드>>자동 포커스를 선택하여 인보크 노드를 생성합니다. 3D 그림 컨트롤 터미널의 오른쪽에 노드를 추가합니다. 자동 포커스 메소드는 3D 모델에 맞춰서 카메라의 위치가 자동으로 정해지도록 만듭니다.
13. 블록 다이어그램에서 **모든 에러 입력과 에러 출력 터미널**을 연결합니다.
14. VI를 실행합니다. LabVIEW는 센서 데이터가 있는 3D 모델을 3D 그림 컨트롤에 디스플레이합니다.

센서 설정 대화 상자에서, 모델을 클릭하고 끌어오는 동안 <Shift> 키를 눌러서 줌 인 및 줌 아웃할 수 있습니다. 모델을 클릭하고 끌어오는 동안 <Ctrl> 키를 눌러서 디스플레이

를 팬합니다. (macOS) <Command>-<Shift>를 클릭합니다. (Linux) <Alt>-<Shift>를 누른 채로 클릭합니다.

센서를 삭제하려면, 채널 리스트에서 센서를 선택하고 센서 삭제 버튼을 클릭하여 모델 및 채널 리스트에서 해당 센서를 제거합니다. 센서가 DAQmx 채널인 경우, LabVIEW는 센서를 모델에서 제거하지만 채널 리스트에서는 제거하지 않습니다.

SubVI

블록다이어그램에 SubVI 놓기

다음 단계를 따라 블록다이어그램에 subVI를 놓습니다.

1. 윈도우>>블록다이어그램 보이기를 선택하여 새 VI 또는 기존 VI의 블록다이어그램을 디스플레이합니다.
2. 필요한 경우, 보기>>함수 팔레트를 선택하여 함수 팔레트를 디스플레이합니다.
3. 함수 팔레트상의 VI 선택 아이콘이나 텍스트를 클릭합니다.
4. SubVI로 사용하려는 VI를 탐색하여 더블 클릭한 후, 블록다이어그램에 놓습니다.
5. SubVI 터미널을 블록다이어그램의 다른 노드에 연결합니다.



노트

SubVI에 커넥터 팬이 없는 경우, 이 SubVI를 다른 노드에 연결할 수 없습니다.

SubVI의 프런트패널을 디스플레이하기 위해서 subVI 노드를 더블 클릭할 수 있습니다.

또한 다른 열린 VI의 블록다이어그램에 열린 VI를 놓을 수 있습니다. 위치 도구를 사용하여 subVI로 사용하려는 VI의 프런트패널 또는 블록다이어그램의 우측 상단 모서리의 아이콘을 클릭한 후, 다른 VI의 블록다이어그램까지 끌어서 놓습니다.



팁(Windows 및 macOS) VI나 컨트롤을 파일 시스템에서 LabVIEW 블록다이어그램으로 끌어서 그 VI에 대한 SubVI 호출을 생성합니다.

LabVIEW 프로젝트를 사용하는 경우, 다른 열린 VI의 블록다이어그램에 **프로젝트 탐색기** 윈도우로부터의 VI를 놓을 수 있습니다. SubVI로 사용하려는 VI를 프로젝트 탐색기 윈도우에서 선택하여 다른 VI의 블록다이어그램으로 끌어 이동합니다.

SubVI 노드를 SubVI 콘텐츠로 대체하기

SubVI에서 마우스 오른쪽 버튼을 클릭한 후 SubVI 콘텐츠로 대체를 선택하면, 호출 블록다이어그램을 편집할 때 SubVI의 내용을 삽입할 수 있습니다. SubVI 콘텐츠로 대체 대화 상자가 나타나면, 다음 옵션을 설정합니다.

- 호출 다이어그램에 SubVI 내용이 포함되도록 공간 만들기—SubVI 내용이 포함되도록 블록다이어그램을 조정할 것인지 지정합니다.
- SubVI 내용을 시퀀스 구조 안에 놓기—SubVI 내용을 **다중 시퀀스** 구조에 놓을 것인지 지정합니다.



노트

SubVI가 구조 안에 인디케이터를 포함하는 경우 SubVI 콘텐츠로 대체 메뉴 아이콘을 사용할 수 없습니다.

SubVI 열기 및 편집하기

SubVI를 열고, 편집하려면, 수행 도구 또는 위치 도구를 사용하여 블록다이어그램에 있는 SubVI를 더블 클릭합니다. LabVIEW는 SubVI의 프런트패널을 디스플레이합니다. <Ctrl> 키를 누르고 수행 도구나 위치 도구를 사용하여 블록다이어그램에서 SubVI를 더블 클릭하면, SubVI의 블록다이어그램과 프런트패널을 디스플레이할 수 있습니다. (macOS) <Option> 키를 누릅니다. (Linux) <Alt> 키를 누릅니다.

선택에 의해 SubVI 생성하기

위치 도구를 사용하여 재사용하려는 블록다이어그램 부분을 선택하고 편집> SubVI 생성을 선택하여 VI의 일부를 SubVI로 변환합니다. 새로운 SubVI의 아이콘이 블록다이어그램에서 선택한 부분을 대체합니다. LabVIEW가 새로운 SubVI에 대한 컨트롤과 인디케이터를 만들고 SubVI를 기존 와이어에 연결합니다.

이렇게 일부 선택한 후 SubVI를 생성하면 편리하지만, 이런 경우 VI의 논리적인 계층구조를 생성하려면 신중한 계획이 요구됩니다. 어떤 객체를 선택할 것인지 고려해야 하고, 원래 VI나 새로 생성된 VI의 기능이 바뀌는 경우는 피해야 합니다.

VI를 선택하여 SubVI를 생성할 경우 다음 유의사항을 고려합니다.

- 선택에 의한 SubVI 생성은 선택된 객체를 지우고 이를 SubVI로 대체하는 것과 같습니다. 두 가지 상황에서, LabVIEW는 다음을 수행합니다.
 - LabVIEW는 원래 VI에서 선택한 부분에 포함되는 블록다이어그램 터미널을 삭제하지 않습니다. 컨트롤과 인디케이터는 원래 VI의 프런트패널에 남아 있고 새 SubVI에 터미널이 연결됩니다.
 - 선택 대상에서 프로퍼티 노드 또는 로컬 변수를 포함하는 각 프런트패널 객체에 대해, LabVIEW는 컨트롤 참조를 원래의 블록다이어그램에 추가하고 SubVI를 참조와 연결합니다. SubVI에서, LabVIEW는 참조를 프로퍼티 노드에 연결합니다.
- SubVI를 생성할 때 28개의 객체를 초과하여 선택하지 않습니다. 왜냐하면 커넥터 팬의 **연결 최대 개수**가 28이기 때문입니다.
- **블록다이어그램에 사이클** 생성을 피합니다.
- 선택할 때 **블록다이어그램 터미널을 포함한 구조**를 포함하는 것은 피합니다.

사이클을 생성하는 SubVI 선택 피하기

선택에 의해 subVI를 생성할 때 블록 다이어그램에서 사이클 생성을 피합니다. 사이클은 데이터 흐름이 subVI의 출력에서 시작되고 subVI의 입력에서 끝날 때 발생합니다.

선택을 만드는 동안 사이클을 인식하는 것은 어렵지만, LabVIEW는 그것을 인식합니다. LabVIEW가 사이클을 인식하면, 왜 subVI를 생성할 수 없는지를 설명하는 대화 상자를 디스플레이합니다. 그러나, 새롭고 분리된 VI에 선택 대상을 복사할 수 있습니다.

구조 안에 터미널을 포함하는 SubVI 선택 피하기

선택으로부터 SubVI를 생성할 때, 블록 다이어그램 터미널을 포함하는 구조를 선택에 포함시키는 것을 피하십시오. 선택으로부터 SubVI를 생성하는 경우, LabVIEW는 잠재적인 문제에 대해서 경고를 주는 대화 상자를 디스플레이하고 변환을 계속할지 또는 취소할지를 물어봅니다.

블록 다이어그램 터미널을 포함하는 For 루프 또는 While 루프로부터 SubVI를 생성하는 경우, 원래 VI의 기능은 바뀔 수도 있습니다. 터미널은 원래 블록 다이어그램에 남겨지고 터미널이 새 subVI에 연결되기 때문에, SubVI는 SubVI 내부 루프의 각 반복마다 터미널의 값을 업데이트하지 않습니다. 터미널의 값은 루프의 모든 반복이 실행되고 SubVI가 실행을 끝낸 이후에만 업데이트됩니다.

또한, 출력 터미널을 포함하는 케이스 구조로부터 SubVI를 생성하는 경우, 원래 VI의 기능은 바뀔 수 있습니다. 출력 터미널은 원래 블록 다이어그램에 남겨지고 인디케이터가 새 SubVI에 연결되기 때문에, LabVIEW는 항상 SubVI가 실행된 후에 인디케이터에 값을 씁니다. SubVI를 생성하기 이전에, LabVIEW는 오직 케이스가 실행된 인디케이터를 포함할 때에만 인디케이터에 값을 씁니다. 케이스 구조로부터 SubVI를 생성하는 경우, 반드시 모든 케이스의 인디케이터에 값을 전달하도록 SubVI를 편집해야 합니다.

SubVI가 터미널을 요구하는 객체를 28개 이상 선택하지 않기

선택해서 subVI를 생성할 때 터미널이 필요한 객체를 28개 이상 선택하지 마십시오. 왜냐하면 커넥터 팬의 최대 연결 개수가 28개이기 때문입니다. 각 블록 다이어그램 터미널, 프로퍼티 노드를 가진 각 프런트패널 객체, 선택한 특정한 로컬 변수는 커넥터 팬에서 터미널이 필요합니다. 터미널이 필요한 객체를 28개 이상 선택하는 경우, LabVIEW는 왜 subVI를 생성할 수 없는지 설명하는 대화 상자를 디스플레이합니다.

최대 터미널 수를 초과하지 않으려면, 블록 다이어그램의 작은 섹션을 선택하거나 일부 객체를 클러스터로 묶으십시오.



관련 정보

[선택에 의해 SubVI 생성하기](#) (1348 페이지 참조)

[커넥터 팬 패턴을 선택하기](#) (1353 페이지 참조)

[클러스터 컨트롤과 인디케이터 생성하기](#) (1062 페이지 참조)

터미널

컨트롤과 인디케이터에 터미널 지정하기

커넥터 팬에 사용할 **패턴을 선택**한 후, 각 커넥터 팬 터미널에 프런트패널 컨트롤 또는 인디케이터를 지정할 수 있습니다. 커넥터 팬에 컨트롤과 인디케이터를 지정할 때, 복잡하거나 혼동을 줄 수 있는 와이어 패턴을 피하기 위해 왼쪽에 입력을, 오른쪽에 출력을 놓습니다.

다음 단계를 따라 커넥터 팬의 터미널을 컨트롤과 인디케이터에 지정합니다.

1. 커넥터 팬에 지정하려는 컨트롤과 인디케이터의 개수에 충분한 **패턴을 선택**했는지 확인합니다.
2. 커넥터 팬의 터미널을 클릭합니다. 도구는 자동적으로 와이어링 도구로 바뀌며 터미널은 검정색으로 변합니다.



노트

와이어링 도구를 사용하여 커넥터 팬의 터미널을 프런트패널 컨트롤 및 인디케이터에 지정하지만, 실제 와이어가 그려지는 것은 아닙니다.

1. 터미널에 지정하려는 프런트패널 컨트롤 또는 인디케이터를 클릭합니다. 선택 표시가 객체를 나타내고, 터미널은 터미널을 연결한 것을 나타내기 위해서 컨트롤의 데이터 타입 색으로 바뀝니다.



노트 커넥터 팬 터미널이 흰색으로 바뀌면 연결이 안 된 것입니다. 커넥터 팬 터미널이 적절한 데이터 타입 색으로 바뀔 때까지 2 단계와 3 단계를 반복합니다.

1. 터미널에 지정하려는 각 컨트롤과 인디케이터에 대해서 2 단계와 3 단계를 반복합니다. 터미널에 지정된 컨트롤 또는 인디케이터를 변경하려는 경우, 반드시 먼저 **연결을 삭제**한 후 터미널에 다른 컨트롤 또는 인디케이터를 지정하기 위해서 2 단계와 3 단계를 반복합니다.
2. 필요한 경우, **각 터미널 연결을 확인**합니다.

자동 도구 선택이 비활성화되어 있을 경우, 와이어링 도구를 사용하여 컨트롤 또는 인디케이터를 먼저 선택한 후 터미널을 선택합니다.

터미널을 **필수, 권장, 옵션**으로 지정할 수 있습니다.



팁 (Windows) 두 개의 터미널의 위치를 바꾸려면 <Ctrl> 키를 누른 상태에서 위치 도구를 사용하여 위치를 바꾸려는 터미널을 선택합니다. 이 키보드 바로 가기에서 <Ctrl> 키는 (macOS)의 <Option> 키나 (Linux)의 <Alt> 키와 같습니다.

터미널 연결 확인하기

어떤 컨트롤 또는 인디케이터가 커넥터 팬 터미널에 할당되었는지를 확인하기 위해서는, 커넥터 팬의 터미널을 클릭하십시오. 선택 표시가 지정된 객체에 나타납니다.

또한, 와이어링 도구를 사용하여 컨트롤 또는 인디케이터를 클릭할 수 있습니다. 커넥터 팬에서 지정된 터미널의 색은 어두워집니다.



관련 정보

[컨트롤과 인디케이터에 터미널 지정하기](#) (1350 페이지 참조)

커넥터 팬에서 터미널 연결 삭제하기

[터미널과 해당 컨트롤 또는 인디케이터 사이의 연결](#)을 개별적으로 또는 한 번에 모두 삭제할 수 있습니다.

터미널 연결을 개별적으로 삭제하려면, 연결 끊기를 원하는 커넥터 팬의 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 이 터미널 연결 끊기를 선택합니다. 터미널은 하얗게 되어 더이상 연결이 존재하지 않음을 가리킵니다.

커넥터 팬에서 모든 연결을 삭제하려면, 커넥터 팬의 어디에서나 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 터미널 연결 끊기를 선택합니다.

터미널의 연결을 끊는 것과 커넥터 팬 패턴에서 터미널을 제거하는 것과는 다릅니다. 이 터미널 연결 끊기는 연결을 삭제하지만, 터미널 제거는 터미널을 패턴에서 제거합니다.

실행 중 SubVI 프런트패널 열기

하나의 프런트패널로는 다양한 옵션 또는 디스플레이를 나타내기에 부족한 경우가 있습니다. 이 문제를 해결하려면, VI를 정리하여 최상위 VI가 상위 레벨 옵션을 나타내고 subVI가 관련된 옵션을 나타내도록 합니다.



팁

또한 [탭 컨트롤](#)을 사용하여 프런트패널을 더 사용하기 편리하게 만들 수 있습니다.

LabVIEW가 subVI를 호출할 때, 통상적으로 subVI의 프런트패널은 표시되지 않고 실행만 됩니다. SubVI의 한 인스턴스가 호출될 때 해당 프런트패널을 열도록 하려는 경우, SubVI 노드 설정 대화 상자를 사용합니다.

SubVI의 모든 인스턴스가 호출될 때 해당 프런트패널을 열도록 하려는 경우, VI 프로퍼티 대화 상자를 사용합니다.

단일 인스턴스

다음 단계를 따라 SubVI의 하나의 인스턴스가 호출될 때 프런트패널을 열도록 합니다.

1. 블록다이어그램에 **subVI**를 놓습니다.
2. SubVI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 SubVI 노드 설정을 선택합니다.
3. 호출 시 프런트패널 보이기와 원래 닫혀있으면 실행 후 닫기 확인란에 확인 표시를 합니다.
4. 확인 버튼을 클릭합니다.
5. 최상위 레벨 VI를 실행합니다. 최상위 레벨 VI가 subVI를 호출할 때 해당 프런트패널이 나타납니다.

모든 인스턴스

다음 단계를 따라 SubVI의 모든 인스턴스가 호출될 때 프런트패널을 열도록 합니다.

1. SubVI를 더블 클릭하고 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 윈도우 모양을 선택합니다.
3. 사용자 정의 버튼을 클릭하고 호출 시 프런트패널 보이기와 원래 닫혀있으면 실행 후 닫기 확인란에 확인 표시를 합니다.
4. 확인 버튼을 클릭합니다.
5. SubVI를 저장하고 닫습니다.
6. 최상위 레벨 VI를 실행합니다. SubVI의 모든 인스턴스가 호출될 때 프런트패널을 엽니다.

또한, 호출 시 프런트패널 보이기와 로드 시 프런트패널 보이기 프로퍼티를 사용하여 프로그램적으로 프런트패널을 디스플레이합니다.

SubVI의 현재 인스턴스 결정하기

다음 단계를 따라 현재 sbVI의 인스턴스를 결정합니다.

1. **SubVI의 실행을 일시 정지합니다.**
2. 도구 모음을 나타내기 위해 풀다운 메뉴에서 호출 리스트를 선택합니다. 이 메뉴는 최상위 VI부터 아래의 subVI까지 호출자의 체인 리스트를 만들 수 있습니다. 이 리스트는 VI가 현재 실행 중인지 여부와 상관없이 모든 호출 VI를 나열하는 보기>>관계 탐색>>이 VI의 호출자를 선택할 때 볼 수 있는 리스트와 같지 않습니다.
3. 호출 리스트 메뉴에서 VI를 선택하여 그 블록다이어그램과 현재 subVI의 인스턴스를 하이라이트합니다.

또한, [호출 체인] 함수를 사용하여 프로그램적으로 호출자의 체인 리스트를 만들 수 있습니다.

커넥터 팬 패턴을 선택하기

VI를 subVI로 사용하려면, **커넥터 팬**을 만들어야 합니다. 다음 단계를 따라 VI를 위한 커넥터 팬을 선택합니다.

1. 기본 커넥터 팬 패턴은 $4 \times 2 \times 2 \times 4$ 입니다. 다른 패턴을 선택하려면, 커넥터 팬에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 패턴을 선택합니다.



노트

터미널 개수가 필요한 것보다 더 많은 커넥터 팬 패턴을 선택합니다. VI에 여러분의 터미널을 포함시키면 VI에 추가 커넥터를 더할 수 있습니다.

2. 커넥터 팬 패턴을 선택한 후에, 터미널을 추가하거나 제거하거나 회전하여 VI에 맞도록 그 패턴을 사용자 정의할 수 있습니다.
 - 패턴에 터미널을 추가하기 위해, 터미널을 추가하려는 곳에 커서를 놓고, 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 터미널 추가를 선택합니다.
 - 패턴에서 기존의 터미널을 제거하기 위해, 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 터미널 제거를 선택합니다.
 - 커넥터 팬 패턴의 공간 배치를 변경하려면, 커넥터 팬에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 수평으로 뒤집기, 수직으로 뒤집기 또는 90도 회전하기를 선택합니다.
3. 각 커넥터 팬 터미널에 **프런트패널 컨트롤이나 인디케이터를 할당**합니다.
4. 다른 블록 다이어그램에 subVI로서 이 VI를 놓으면, subVI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 SubVI에 다시 링크를 선택하여 변경한 VI의 커넥터 팬에 그 subVI를 다시 링크해야 합니다. 그렇지 않은 경우, 이 subVI를 가지고 있는 VI는 깨지고 실행되지 않을 것입니다.

National Instruments는 사용하는 터미널이 16개가 넘지 않을 것을 권장합니다. 터미널 개수가 16개가 넘는 커넥터 팬은 와이어로 연결하기 어렵습니다. 더 많은 데이터를 전달할 필요가 있는 경우에는, **클러스터**를 사용하십시오.

필수, 권장, 옵션 입력과 출력 설정하기

어떤 입력과 출력이 필수, 권장, 옵션인지를 지정하여, 사용자가 SubVI의 터미널의 연결을 잊지 않도록 도와줄 수 있습니다.

터미널 입력에서, 필수는 subVI를 블록 다이어그램에 놓을 때 필수 입력을 연결하지 않으면 깨진다는 것을 의미합니다. 그러나, 터미널 출력은 필수로 설정할 수 없습니다. 터미널 입력과 출력에서, 권장과 옵션은 SubVI를 블록 다이어그램에 놓고 권장하거나 옵션인 터미널을 연결하지 않아도 실행할 수 있다는 것을 의미합니다. 이 터미널을 연결하지 않아도 VI는 경고를 생성하지 않습니다.

다음 단계를 따라 터미널을 필수, 권장, 옵션으로 설정합니다.

1. 커넥터 팬의 터미널에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 연결을 선택합니다.
2. 필수, 권장, 옵션 중 하나를 선택합니다.

LabVIEW 설정을 통해 커넥터 팬에서 설정하는 모든 새로운 터미널을 권장이 아닌 필수로 설정할 수 있습니다. 도구 >> 옵션 >> 프런트패널을 선택한 후 커넥터 팬 터미널 기본을 필수로 확인란에 확인 표시를 합니다. 이 옵션은 또한 편집 >> SubVI 생성 메뉴 항목을 사용한 연결에 적용됩니다.

터미널에 하나의 컨트롤이나 인디케이터만을 연결할 수 있습니다.

관련 정보

[블록다이어그램 객체](#)

[컨트롤과 인디케이터에 터미널 지정하기](#)

[터미널 연결 확인하기](#)

[커넥터 팬에서 터미널 연결 삭제하기](#)

아이콘

아이콘 생성하기

아이콘을 생성하는 가장 기본적인 방법은 아이콘 편집기 대화 상자를 사용하는 것입니다. 또한, 파일 시스템에 있는 그래픽을 끌어다 놓거나 VI 클래스에 있는 VI 아이콘 메소드를 사용하여, 아이콘을 생성할 수 있습니다.



관련 정보

[아이콘 사용하기](#) (141 페이지 참조)

[VI와 그림, 텍스트 끌어서 놓기](#) (877 페이지 참조)

라이브러리 아이콘 생성하기

아이콘을 생성하여 그래픽적으로 [프로젝트 라이브러리](#), statechart, [클래스](#), 또는 [x컨트롤](#)을 나타낼 수 있습니다. 아이콘 편집기 대화 상자를 사용하여 아이콘을 생성하거나 편집합니다.

LabVIEW는 라이브러리 아이콘을 해당 라이브러리 안에 있는 모든 객체의 아이콘에 적용합니다. National Instruments는 [라이브러리 아이콘에서 오직 배너만을 생성할 것](#)을 권장합니다. 그러면, 라이브러리 안에 있는 모든 객체의 아이콘은 이 배너를 포함하게 되며, [아이콘을 수정](#)하여 특정 객체에 대한 정보를 제공할 수 있습니다.

프로젝트 라이브러리 아이콘 생성하기

다음 단계에 따라 새 프로젝트 라이브러리를 위한 아이콘을 생성합니다.

1. **프로젝트 탐색기** 윈도우에 있는 프로젝트 라이브러리 아이템에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **프로퍼티**를 선택하여 프로젝트 라이브러리 프로퍼티 대화 상자를 디스플레이합니다. LabVIEW는 새 프로젝트 라이브러리의 기본 아이콘으로 배너를 생성합니다. 일반 셋팅 페이지의 VI 아이콘 템플릿 섹션에서 기본 프로젝트 라이브러리 아이콘을 확인할 수 있습니다.
2. 일반 셋팅 페이지에서 아이콘 편집 버튼을 클릭하여 아이콘 편집기 대화 상자를 디스플레이합니다.
3. (선택 옵션) 아이콘 편집기 대화 상자 오른쪽에 있는 채우기 도구를 사용하여, 아이콘의 배너를 다른 색으로 채울 수 있습니다.
4. 아이콘 배너에 텍스트를 입력하려면, 텍스트 도구를 사용합니다. 텍스트가 활성화 되어 있는 동안에는, 화살표 키를 눌러서 텍스트를 이동할 수 있습니다.
5. (옵션) 파일»다른 이름으로 저장»템플릿을 선택하여 다른 이름으로 아이콘 저장 대화 상자를 디스플레이하고, 추후에 사용할 수 있도록 아이콘을 템플릿으로 저장할 수 있습니다. LabVIEW는 아이콘 템플릿을 256 색 .png 파일로 저장합니다.
6. 확인 버튼을 클릭하여 아이콘 정보를 프로젝트 라이브러리에 저장하고 아이콘 편집기 대화 상자를 닫습니다. 프로젝트 라이브러리 프로퍼티 대화 상자의 일반 셋팅 페이지에서 256 색 아이콘이 생성한 아이콘을 디스플레이합니다.
7. 확인 버튼을 클릭하여 프로젝트 라이브러리 프로퍼티 대화 상자를 닫습니다.



팁

프로젝트 라이브러리 프로퍼티 대화 상자의 일반 셋팅 페이지에 있는 아이콘을 VI에 적용 버튼을 사용하면 기존의 프로젝트 라이브러리 아이콘을 자동으로 프로젝트 라이브러리에 있는 VI에 적용할 수 있습니다.

프로젝트 라이브러리에 있는 VI의 아이콘 생성하기

다음 단계에 따라 새 프로젝트 라이브러리에서 새 VI를 위한 아이콘을 생성합니다.

1. 프로젝트 탐색기 윈도우에 있는 프로젝트 라이브러리 아이템에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 **새로 만들기»VI**를 선택하여 새 VI를 생성하고 엽니다. 이때 프론트패널 윈도우의 오른쪽 위 모서리에 있는 VI 아이콘 자리에 프로젝트 라이브러리 아이콘이 포함됩니다.
2. VI 아이콘에서 더블 클릭하여 아이콘 편집기 대화 상자를 엽니다.
3. 레이어»레이어 페이지 보이기를 선택하여 레이어 페이지를 디스플레이합니다. 이 페이지에서 다음의 아이템을 발견할 수 있습니다:

- 프로젝트 라이브러리 아이콘은 NI_Library라고 부르는 **사용자 레이어**로 나타납니다.



노트

LabVIEW는 NI_Library 사용자 레이어의 불투명도 또는 보이는 정도를 사용자가 변경해도, 다음에 사용자가 프로젝트 라이브러리 아이콘을 수정할 때 사용자 레이어를 이전 상태로 되돌립니다. 영구적으로 NI_Library 사용자 레이어의 불투명도나 보이는 정도를 변경하고 싶은 경우, 프로젝트 라이브러리 아이콘에서 직접 변경하십시오.

- NI_IconNumber 이라고 부르는 사용자 레이어는 최대 9 개까지, LabVIEW를 실행한 이후로 사용자가 열어본 새 VI의 개수를 숫자로 표시합니다.

노트 이 숫자 표시를 비활성화하려면 도구 >> 옵션 >> 프런트패널을 선택한 후 새 VI의 아이콘에 숫자(1-9) 사용 확인란에서 확인 표시를 제거합니다.

- VI 아이콘 이라고 부르는 사용자 레이어는 아이콘에 대한 1 픽셀의 검정색 프레임을 포함합니다.
 - VI 아이콘은 경계가 있는 하얀색 템플릿을 사용하는데, 레이어 페이지의 아이콘 템플릿 섹션에서 확인할 수 있습니다.
- (선택 옵션) NI_IconNumber 사용자 레이어의 미리보기를 클릭하고 <Delete> 키를 누르면, 레이어를 삭제할 수 있습니다.
 - 아이콘 텍스트 페이지에서, 아이콘 안에 최대 4 줄까지 아이콘 텍스트를 입력할 수 있습니다. 사용자가 텍스트의 폰트, 정렬, 크기, 색을 설정할 수 있습니다. 텍스트 수직으로 가운데 정렬 확인란에 확인 표시를 하는 경우, 아이콘 편집기 대화 상자는 아이콘 안에서 아이콘 텍스트를 수직으로 가운데 정렬합니다.
 - 문양 페이지에서 문양을 끌어서 미리보기 영역에 놓습니다. 문양을 움직이면서 <F> 키 또는 <R> 키를 누르면 각각 문양을 수평으로 뒤집거나 시계방향으로 회전할 수 있습니다. 또한 문양을 더블 클릭하면 아이콘의 왼쪽 위 코너에 문양을 놓을 수 있습니다. 항목별 또는 키워드별로 문양을 볼 수 있습니다.
 - 문양을 움직일 때 이동 도구를 사용합니다. 각 문양은 별도의 레이어에 놓여지며, 따라서 개별적으로 움직입니다. 한 문양을 선택하면 아이콘의 나머지 부분이 어두워져서, 현재 선택해서 이동시키고 있는 문양을 쉽게 구분할 수 있습니다.
 - 아이콘 편집기 대화 상자의 오른쪽에 있는 편집 도구를 사용하여 필요한 경우 다시 아이콘을 편집할 수 있습니다. 아이콘 편집기 대화 상자는 매번 편집 도구를 열고 사용할 때마다 새로운 사용자 레이어를 생성합니다. 레이어 >> 새 레이어 생성을 선택하면, 편집 도구를 연속적으로 사용하여 일련의 편집 작업을 수행하는 도중에 새 사용자 레이어를 생성할 수 있습니다.

노트 아이콘 편집기 대화 상자의 오른쪽에 있는 편집 도구를 사용하여 아이콘 템플릿이나 아이콘 텍스트를 변경할 수 없습니다. 템플릿 페이지와 아이콘 텍스트 페이지를 사용하여 아이콘 템플릿과 아이콘 텍스트를 각각 수정하십시오.

- 레이어 페이지에서 아이콘 레이어의 이름, 불투명도, 보이는 정도, 순서를 설정할 수 있습니다.
- 확인 버튼을 클릭하여 VI에 아이콘 정보를 저장하고 아이콘 편집기 대화 상자를 닫습니다.

VI 아이콘 생성하기

아이콘을 생성하여 VI 또는 **사용자 컨트롤**을 그래픽으로 나타낼 수 있습니다. 아이콘 편집기 대화 상자를 사용하여 아이콘을 생성하거나 편집합니다.

배너를 사용하여 관련 VI를 식별할 수 있습니다. National Instruments는 **배너를 생성하고 템플릿으로 저장**하는 것을 권장합니다. 그러면 관련 VI 아이콘에서 이 템플릿을 사용할 수 있고, **이 VI 아이콘을 수정하여** 특정 VI에 대한 정보를 제공할 수 있습니다.

라이브러리에 있는 객체에 대한 아이콘을 생성하고 싶은 경우, 아이콘 템플릿을 사용하는 대신에 **라이브러리 아이콘을 생성**하십시오. LabVIEW는 라이브러리 아이콘을 해당 라이브러리 안에 있는 모든 객체의 아이콘에 적용합니다.

배너를 템플릿으로 저장하기

다음 단계에 따라 배너를 VI의 아이콘 템플릿으로 저장합니다.

1. 프런트패널이나 블록 다이어그램 윈도우의 오른쪽 위 코너에 있는 아이콘을 더블 클릭하거나 아이콘에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 아이콘 편집을 선택하여 아이콘 편집기 대화 상자를 디스플레이합니다.
2. <Ctrl-A> 키를 누르면 아이콘의 모든 **사용자 레이어**를 선택할 수 있고, <Delete> 키를 누르면 선택된 것을 삭제할 수 있습니다. 기본 아이콘은 VI 아이콘이라고 불리는 1 개의 사용자 레이어입니다.
3. 템플릿 페이지의 VI >> 프레임워크 항목에서 `_blank.png` 아이콘 템플릿을 선택합니다. 항목별 또는 키워드별로 템플릿을 볼 수 있습니다.
4. 아이콘 편집기 대화 상자 오른쪽에 있는 채우기 도구를 사용하여, 아이콘의 배너를 색으로 채웁니다.
5. 아이콘 배너에 텍스트를 입력하려면, 텍스트 도구를 사용합니다. 텍스트가 활성화 되어 있는 동안에는, 화살표 키를 눌러서 텍스트를 이동할 수 있습니다.
6. 파일 >> 다른 이름으로 저장 >> 템플릿을 선택하여 다른 이름으로 아이콘 저장 대화 상자를 디스플레이하고, 추후에 사용할 수 있도록 아이콘을 템플릿으로 저장합니다. LabVIEW는 아이콘 템플릿을 256 색 .png 파일로 저장합니다.

템플릿을 통해 VI 아이콘 생성하기

다음 단계에 따라 사용자가 생성한 템플릿을 활용하여 VI 아이콘을 만듭니다.

1. <Ctrl-A> 키를 누르면 아이콘의 모든 **사용자 레이어**를 선택할 수 있고, <Delete> 키를 누르면 선택된 것을 삭제할 수 있습니다.
2. 템플릿 페이지에서 사용자가 생성한 템플릿을 선택합니다. 항목별 또는 키워드별로 템플릿을 볼 수 있습니다.
3. 아이콘 텍스트 페이지에서, 아이콘 안에 최대 4 줄까지 아이콘 텍스트를 입력할 수 있습니다. 사용자가 텍스트의 폰트, 정렬, 크기, 색을 설정할 수 있습니다. 텍스트 수직으로 가운데 정렬 확인란에 확인 표시를 하는 경우, 아이콘 편집기 대화 상자는 아이콘 안에서 아이콘 텍스트를 수직으로 가운데 정렬합니다.

4. 문양 페이지에서 문양을 끌어서 미리보기 영역에 놓습니다. 문양을 움직이면서 <F> 키 또는 <R> 키를 누르면 각각 문양을 수평으로 뒤집거나 시계방향으로 회전할 수 있습니다. 또한 문양을 더블 클릭하면 아이콘의 왼쪽 위 코너에 문양을 놓을 수 있습니다. 항목별 또는 키워드별로 문양을 볼 수 있습니다.
5. 문양을 옮길 때 이동 도구를 사용합니다. 각 문양은 별도의 레이어에 놓여지며, 따라서 개별적으로 움직입니다. 한 문양을 선택하면 아이콘의 나머지 부분이 어두워져서, 현재 선택해서 이동시키고 있는 문양을 쉽게 구분할 수 있습니다.
6. 아이콘 편집기 대화 상자의 오른쪽에 있는 편집 도구를 사용하여 필요한 경우 다시 아이콘을 편집할 수 있습니다. 아이콘 편집기 대화 상자는 매번 편집 도구를 열고 사용할 때마다 새로운 사용자 레이어를 생성합니다. 레이어»새 레이어 생성을 선택하면, 편집 도구를 연속적으로 사용하여 일련의 편집 작업을 수행하는 도중에 새 사용자 레이어를 생성할 수 있습니다.



노트

아이콘 편집기 대화 상자의 오른쪽에 있는 편집 도구를 사용하여 아이콘 템플릿이나 아이콘 텍스트를 변경할 수 없습니다. 템플릿 페이지와 아이콘 텍스트 페이지를 사용하여 아이콘 템플릿과 아이콘 텍스트를 각각 수정하십시오.

7. (옵션) 레이어»레이어 페이지 보이기를 선택하여 레이어 페이지를 디스플레이합니다. 이 페이지에서 아이콘 레이어의 이름, 불투명도 및 레이어의 순서를 설정할 수 있습니다.
8. 확인 버튼을 클릭하여 VI에 아이콘 정보를 저장하고 아이콘 편집기 대화 상자를 닫습니다.

또한, **파일 시스템 상에 있는 그래픽을 끌어와서** 프런트패널 윈도우의 오른쪽 상단 모서리에 놓고 이 그래픽을 VI 아이콘으로 사용할 수 있습니다. 끌어와서 사용할 수 있는 파일에는 .png, .bmp, .jpg 파일이 있습니다.



노트 파일 시스템에서 그래픽을 끌어와서 아이콘을 변경하는 경우, LabVIEW는 이 그래픽에 대해 VI 아이콘이라는 **사용자 레이어**를 생성하고, 기존에 있던 다른 아이콘 정보를 모두 아이콘 편집기 대화 상자에서 제거합니다.

또한, VI 아이콘:파일로부터 설정 메소드를 사용하여 이미지를 기반으로 VI 아이콘을 프로그램적으로 설정할 수 있습니다. LabVIEW는 이 이미지에 대해 VI 아이콘이라는 사용자 레이어를 생성하고, 기존에 있던 다른 아이콘 정보를 모두 아이콘 편집기 대화 상자에서 제거합니다.

사용자 모양 아이콘 생성하기

전체 32×32 픽셀 아이콘 영역보다 작은 사용자 정의 모양의 아이콘을 생성할 수 있습니다. 다음 단계에 따라 새 VI를 위한 사용자 모양 아이콘을 생성합니다.

1. 프런트패널 윈도우 또는 블록 다이어그램 윈도우의 오른쪽 위 모서리에 있는 아이콘에서 더블 클릭하여, 아이콘 편집기 대화 상자를 디스플레이합니다.
2. <Ctrl-A> 키를 누르면 아이콘의 모든 **사용자 레이어**를 선택할 수 있고, <Delete> 키를 누르면 선택된 것을 삭제할 수 있습니다. 기본 아이콘은 VI 아이콘이라고 불리는 1개의 사용자 레이어입니다.

3. 폐쇄된 모양의 경계를 가진 **아이콘을 생성**합니다. 예를 들어, 타원형 도구를 사용하여 닫힌 타원형을 그린 후에 타원형 안에만 문양이나 텍스트를 추가합니다.
4. 확인 버튼을 클릭하여 VI에 아이콘 정보를 저장하고 아이콘 편집기 대화 상자를 닫습니다.

블록다이어그램에서 사용자 모양 아이콘을 선택하면, 선택 표시가 아이콘의 각각의 그래픽 원소 주위에 나타납니다. 예를 들어, 아이콘에 경계선이 없으나 서로 겹치지 않는 원과 사각형으로 구성되어 있는 경우를 가정해볼 수 있습니다. 블록다이어그램에서 이 아이콘을 선택하면, 원과 사각형 주변 모두에 선택 표시가 나타납니다.

수동으로 아이콘 라이브러리와 동기화하기

아이콘 편집기 대화 상자에서 도구 >> ni.com 아이콘 라이브러리에 동기화를 선택하면 LabVIEW는 ni.com의 아이콘 라이브러리에서 가장 최신 문양을 다운로드합니다. LabVIEW가 National Instruments 웹 서버에 접속할 수 없는 경우, LabVIEW는 수동으로 동기화 대화 상자를 디스플레이합니다. 이 대화 상자를 사용하여 아이콘 라이브러리와 직접 동기화할 수 있습니다.

다음 단계를 따라 수동으로 동기화 대화 상자가 나타나면 아이콘 라이브러리와 수동으로 동기화합니다.

1. ni.com의 아이콘 라이브러리에서 Icon_Library.zip 파일을 쉽게 접근할 수 있는 위치로 다운로드합니다.
2. 아이콘 라이브러리 XML 파일을 쉽게 접근할 수 있는 위치에 복사하고 저장합니다. 아이콘 라이브러리 XML 파일은 Icon_Library.zip 파일에서 파일의 구조를 결정합니다.
3. 수동으로 동기화 대화 상자에서, 아이콘 라이브러리 압축(Zip) 파일 필드에 Icon_Library.zip 파일에 대한 경로를 입력합니다.
4. 아이콘 라이브러리 XML 파일 필드에 아이콘 라이브러리 XML 파일에 대한 경로를 입력합니다.
5. 확인 버튼을 클릭합니다. LabVIEW는 아이콘 라이브러리에 동기화 대화 상자를 .zip 파일로부터 모든 아이콘과 XML 파일이 정의하는 구조로 채웁니다.

메뉴

메뉴 사용자 정의하기

메뉴 편집기 대화 상자를 사용하여 VI마다 사용자 정의 메뉴를 생성할 수 있고, VI에 메뉴 모음 보이기 또는 숨기기 설정을 할 수 있습니다. 또한 기존의 런타임 메뉴(.rtm) 파일을 VI와 연결할 수 있습니다.



노트

사용자 메뉴는 VI가 실행되는 동안에만 나타납니다.

VI를 편집할 때 정적으로 사용자 메뉴를 만들거나 또는 기본 LabVIEW 메뉴를 수정할 수 있으며, 또는 VI를 실행할 때 프로그램적으로 사용자 메뉴를 만들거나 또는 기본 LabVIEW 메뉴를 수정할 수 있습니다.



관련 정보

[메뉴 모음 보이기와 숨기기](#) (1364 페이지 참조)

[기존 메뉴 결합하기](#) (1360 페이지 참조)

[정적으로 메뉴 사용자 정의하기](#) (1362 페이지 참조)

[메뉴를 프로그램적으로 사용자 정의하기](#) (1360 페이지 참조)

기존 메뉴 결합하기

기존 런타임 메뉴 .rtm 파일을 하나 이상의 VI와 함께 사용할 수 있습니다.

다음 단계를 통해 기존 .rtm 파일을 VI에 결합합니다.

1. 편집>>런타임 메뉴를 선택하여 메뉴 편집기 대화 상자를 디스플레이합니다.
2. 메뉴 편집기 대화 상자에서 파일>>열기를 선택합니다.
3. VI에 결합하려는 메뉴 파일을 탐색한 후 확인 버튼을 클릭합니다. 메뉴 파일에 대한 경로가 메뉴 편집기 대화 상자의 제목 표시줄에 디스플레이됩니다.
4. 메뉴 편집기 대화 상자를 닫습니다.
5. 런타임 메뉴를 선택된 메뉴 파일로 변경할지 여부를 묻으면, 예 버튼을 클릭합니다.

다음 단계를 따라 런타임 메뉴를 VI에서 분리합니다.

1. 편집>>런타임 메뉴를 선택하여 메뉴 편집기 대화 상자를 디스플레이합니다.
2. 메뉴 타입 풀다운 메뉴에서 기본을 선택합니다. 파일 경로는 더 이상 메뉴 편집기 대화 상자의 제목 표시줄에 나타나지 않습니다.
3. 메뉴 편집기 대화 상자를 닫습니다.
4. 런타임 메뉴를 기본 메뉴로 변경할지 여부를 묻으면 예 버튼을 클릭합니다.

메뉴를 [정적으로](#) 또는 [프로그램식으로](#) 사용자 정의할 수 있습니다.

메뉴를 프로그램적으로 사용자 정의하기

다음 단계를 따라 메뉴를 프로그램적으로 사용자 정의합니다.

1. 편집>>런타임 메뉴를 선택하여 메뉴 편집기 대화 상자를 디스플레이합니다.
2. 메뉴 타입 풀다운 메뉴에서 사용자를 선택합니다. 계층구조 리스트박스의 물음표는 아직 이름이 없는 새 아이টে임을 생성하고 있음을 나타냅니다.

3. 파일>>저장을 선택하고 사용자 메뉴의 이름을 정합니다. 선택한 곳 어디에서나 메뉴 파일을 저장할 수 있습니다.
4. 메뉴 편집기 대화 상자를 닫습니다.
5. 블록다이어그램에 [현재 VI의 메뉴 모음] 함수를 추가합니다.
6. 블록다이어그램에 [메뉴 아이템 삽입] 함수를 추가합니다.
7. [현재 VI의 메뉴 모음] 함수를 [메뉴 아이템 삽입] 함수의 메뉴 참조 입력에 연결합니다.
8. [메뉴 아이템 삽입] 함수의 아이템 이름 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 배열 상수가 나타납니다.
9. 배열의 0 슬롯에 파일, 1 슬롯에 편집을 입력합니다.
10. [메뉴 아이템 삽입] 함수의 아이템 태그 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 배열 상수가 나타납니다.
11. 다시, 배열의 0 슬롯에 파일, 1 슬롯에 편집을 입력합니다. 메뉴 아이템 태그는 메뉴 아이템 이름과 같은 이름을 가질 수 있습니다.
12. 블록다이어그램에 While 루프를 추가하고, While 루프가 메뉴 함수를 둘러싸도록 합니다.
13. [기다림 (ms)] 함수를 While 루프 안에 추가하여, While 루프의 실행을 지연시킵니다.



노트

[기다림(ms)] 함수의 대기 시간 (ms) 입력에 값을 연결해야 VI가 실행됩니다.

1. 프런트패널 윈도우로 스위치하여 연속 실행 버튼을 클릭합니다. 파일 및 편집 메뉴가 프런트패널 윈도우에 나타납니다. 그러나, 거기에는 서브메뉴 아이템이 없습니다.

기존 파일 및 편집 메뉴에 서브메뉴 아이템 추가하기

다음 단계를 따라 기존의 파일과 편집 메뉴에 서브메뉴 아이템을 추가합니다.

1. 블록다이어그램에서 또다른 [메뉴 아이템 삽입] 함수를 첫번째 [메뉴 아이템 삽입] 함수의 오른쪽에 추가합니다.
2. 첫번째 [메뉴 아이템 삽입] 함수의 메뉴 참조 출력과 에러 출력 출력을 두번째 [메뉴 아이템 삽입] 함수의 메뉴 참조와 에러 입력 입력에 연결합니다.
3. 블록다이어그램의 **모든 에러 입력과 에러 출력 터미널을 연결**합니다.
4. 두번째 [메뉴 아이템 삽입] 함수의 메뉴 태그 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 문자열 상수가 나타납니다.
5. 문자열에 파일을 입력하여 9 단계에서 생성한 파일 메뉴에 아이템을 추가합니다.
6. 두번째 [메뉴 아이템 삽입] 함수의 아이템 이름 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 배열 상수가 나타납니다.
7. 0 슬롯에 메뉴1, 1 슬롯에 메뉴2를 입력합니다. 아이템 태그 입력을 위해 이 단계를 반복합니다.

8. 프론트패널 윈도우로 전환하여 연속 실행 버튼을 클릭합니다. 프론트패널 윈도우는 파일 메뉴와 편집 메뉴, 그리고 파일 메뉴의 서브아이템으로 메뉴1과 메뉴2를 포함합니다.
9. **케이스 구조 이용**을 통해 메뉴 아이템을 실행합니다.



노트 또한 [메뉴 아이템 삽입] 함수의 입력으로 **어플리케이션 아이템 태그**를 제공하여 기본 LabVIEW 메뉴 아이템에 추가할 수 있습니다.

프로그램적으로 메뉴를 사용자 정의하는 예제는 labview\examples\Dialog and User Interface\Menu\Customizing the Menubar\Customizing the Menubar.lvproj를 참조하십시오.



예제 열기



관련 예제 찾기

정적으로 메뉴 사용자 정의하기

다음 단계를 따라 메뉴를 정적으로 사용자 정의합니다.

1. 편집>>런타임 메뉴를 선택하여 메뉴 편집기 대화 상자를 디스플레이합니다.
2. 메뉴 타입 풀다운 메뉴에서 사용자를 선택합니다.
3. 아이템 타입 풀다운 메뉴에서 사용자 아이템을 선택합니다. LabVIEW 기본 메뉴 아이템을 사용자 메뉴에 추가하려면, 어플리케이션 아이템을 선택하고 원하는 LabVIEW 메뉴 아이템을 선택합니다.



노트

LabVIEW 기본 **키보드 바로 가기 키**를 사용자 메뉴에 사용하려면, 사용자 메뉴에 반드시 그에 대응하는 기본 메뉴 아이템이 추가되어야 합니다. 그러나, 객체 잘라내기, 객체 복사하기, 객체 붙여넣기, 기본 도움말 윈도우 디스플레이하기, 기본 도움말 윈도우 잠그기, LabVIEW 도움말 디스플레이하기와 같은 기본적인 키보드 바로 가기 키는 사용자 메뉴에 기본적으로 포함됩니다.

1. 아이템 이름 텍스트 박스에 메뉴 아이템 이름을 입력합니다. 메뉴 아이템 이름이 미리보기 필드와 아이템 태그 필드에 나타납니다. 태그 이름을 수정하여 메뉴 아이템에 유일한 식별자를 줄 수 있습니다. 블록 다이어그램은 프로그램적으로 메뉴 아이템을 식별하기 위해 이 태그 이름을 사용합니다. 메뉴 아이템 태그는 메뉴 아이템 이름과 같은 이름을 가질 수 있습니다. (Windows) 사용자가 <Alt> 키와 함께 어떤 문자 키를 눌러서 키보드를 통해 메뉴를 열도록 하게 하려면, 아이템 이름 텍스트 박스에서 그 문자 앞에 밑줄 _ 을 입력합니다. 아이템 이름이 2개의 밑줄 __로 시작하는 경우, LabVIEW는 메뉴에 해당 아이템 이름이 나타날 때 앞에 나오는 밑줄을 제거하지만 사용자가 입력한 다른 밑줄은 모두 포함시킵니다. 메뉴 아이템이 밑줄을 포함하고 있지만, 어떤 문자에도 밑줄일 쳐지지 않기를 원하는 경우에 이 기능을 사용하십시오.

노트 아이템 이름의 시작 부분에 2 개의 밑줄을 입력하는 경우, LabVIEW는 <Alt> 키에 바로 가기 키를 할당할 수 없게 되지만 사용자는 계속 바로 가기 텍스트 박스에 있는 아이템 이름에 대해 바로 가기 키를 할당 할 수 있습니다.

2. 활성화 확인란에 확인 표시를 하면 사용자는 아이템을 사용할 수 있습니다. 확인 표시를 제거하여 메뉴 아이템을 잘 안보이게 만들면, 사용자는 아이템을 사용할 수 없습니다.
3. 확인됨 확인란에 확인 표시를 하면 메뉴 아이템 옆에 확인 표시가 나타나게 됩니다. 확인 표시를 하면 메뉴 아이템에서 사용자가 확인 표시를 제거할 때까지 연속 실행되는 동작을 유효하게 합니다.
4. 바로 가기 텍스트 박스에 바로 가기 키를 생성합니다. <Ctrl> 키를 조합키로 사용하지 않고 <F1>에서 <F24>까지의 기능 키를 사용할 수 있습니다. <Ctrl> 메뉴 키는 다른 키와 같이 사용해야 합니다. (macOS) <Command> 키를 사용합니다. (Linux) <Alt> 키를 누릅니다.



노트 LabVIEW는 비활성화된 메뉴 아이템의 키보드 바로 가기 키에 대해 응답하지 않습니다. (macOS) 기능 키를 키보드 바로 가기 키로 사용하는 것은 기능 키가 Exposé에서 바로 가기 키로 할당되지 않은 경우에만 가능합니다.

1. 도구 모음에서 파란색의 더하기 버튼  을 클릭하여 사용자 메뉴에 더 많은 아이템을 추가합니다. 빨간색 x 버튼  을 클릭하여 아이템을 삭제합니다.
2. 파일 >> 다른 이름으로 저장을 선택하여 메뉴 파일을 저장합니다. 나타난 대화 상자에 메뉴의 이름을 입력하고 확인 버튼을 클릭합니다. VI가 로드될 때, 사용자 정의된 메뉴가 자동으로 로드됩니다.
3. **케이스 구조**를 이용하여 각 메뉴 아이템을 실행합니다.

메뉴 편집기 대화 상자에서 메뉴 계층구조는 도구 모음의 화살표 버튼을 클릭하거나, 편집 메뉴의 계층구조 조작 옵션을 사용하거나, 끌어서 놓기를 통해 배치할 수 있습니다. 서브메뉴 문양을 클릭하거나 편집 메뉴의 확장 또는 축소 옵션에서 선택을 이용하여 계층구조를 확장하거나 축소할 수 있습니다. 분리자를 추가하려면, 아이템 타입 풀다운 메뉴의 분리자를 선택합니다.

메뉴 선택 처리하기

메뉴를 **정적으로** 또는 **프로그램적으로** 사용자 정의한 후에 사용자 메뉴에서 각 메뉴 아이템을 실행하는 케이스 구조를 생성해야 합니다.

다음 단계를 따라 사용자 메뉴에서 생성한 메뉴 아이템을 처리합니다.

1. 블록 다이어그램에 While 루프를 추가하고 사각형 영역을 끌어옵니다.
2. While 루프 밖에 [현재 VI의 메뉴 모음] 함수를 추가합니다.
3. While 루프 안에 [메뉴 선택 얻기] 함수를 추가합니다.
4. [현재 VI의 메뉴 모음] 함수를 [메뉴 선택 얻기] 함수의 메뉴 참조 입력에 연결합니다.
5. [메뉴 선택 얻기] 함수의 ms 타임아웃 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성 >> 상수를 선택합니다. 200의 값을 가진 숫자형 상수가 나타납니다.

6. While 루프 안에 케이스 구조를 추가하고 그 While 루프 안에서 사각형 영역을 끌어서 늘립니다.
7. [메뉴 선택 얻기] 함수의 아이템 태그 출력을 케이스 구조의 경계의 선택자 터미널에 연결합니다.
8. [메뉴 선택 얻기] 함수의 메뉴 참조 출력을 케이스 구조의 경계에 연결합니다.
9. 사용자 메뉴에서 생성한 각 메뉴 아이템에 대해 **케이스를 생성**합니다.
10. 이 VI를 실행하여 사용자 메뉴를 봅니다.

메뉴 선택을 다루는 예제는 labview\examples\Dialog and User Interface\Menu\Customizing the Menubar\Customizing the Menubar.lvproj를 참조하십시오.

 예제 열기  관련 예제 찾기

메뉴 모음 보이기와 숨기기

다음 단계를 따라 메뉴 모음을 보이고 숨깁니다.

1. 파일»VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 윈도우 모양을 선택합니다.
3. 사용자 정의 버튼을 클릭하고 메뉴 모음 보이기 확인란에 확인 표시를 하거나 제거합니다.
4. 변경을 저장하기 위해 확인 버튼을 두 번 클릭합니다.

어플리케이션 아이템 보기

어플리케이션 아이템 태그를 어디서 보는지 알면 프로그램적으로 메뉴를 수정할 때 더 쉽게 이 아이템을 호출할 수 있습니다. 어플리케이션 아이템 태그를 VI의 런타임 메뉴 또는 런타임 바로 가기 메뉴에 사용할 수 있습니다.

다음 단계를 따라 어플리케이션 아이템 태그를 봅니다.

1. 프런트패널 윈도우에서 편집»런타임 메뉴를 선택하여 메뉴 편집기 대화 상자를 디스플레이합니다. 또한, 컨트롤이나 인디케이터에서 마우스 오른쪽 버튼을 클릭하고 고급»런타임 바로 가기 메뉴»편집을 선택하여 바로 가기 메뉴 편집기 대화 상자를 디스플레이할 수 있습니다.
2. 메뉴 타입 풀다운 메뉴에서 기본을 선택합니다.
3. 메뉴 계층구조 리스트박스에서, 아이템 태그를 보려는 메뉴 아이템을 선택합니다. 이것은 오른쪽에 나타납니다.

런타임 바로 가기 메뉴 사용자 정의하기

사용자 인터페이스에서 컨트롤의 런타임 바로 가기 메뉴를 사용자 정의할 수 있습니다. 바로 가기 메뉴를 사용자 정의하려면, 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>런타임 바로 가기 메뉴>>편집을 선택합니다.

또한, VI 또는 개별 컨트롤 및 인디케이터에 대한 런타임 바로 가기 메뉴를 비활성화할 수 있습니다.



관련 정보

[컨트롤의 런타임 바로 가기 메뉴를 정적으로 사용자 정의하기](#) (1367 페이지 참조)

[런타임 바로 가기 메뉴 비활성화하기](#) (1368 페이지 참조)

기존 런타임 바로 가기 메뉴 결합하기

기존 런타임 바로 가기 메뉴 .rtm 파일을 하나 이상의 컨트롤과 함께 사용할 수 있습니다.

다음 단계를 통해 기존 .rtm 파일을 컨트롤에 결합합니다.

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>런타임 바로 가기 메뉴>>편집을 선택하여 바로 가기 메뉴 편집기 대화 상자를 디스플레이합니다.
2. 바로 가기 메뉴 편집기 대화 상자에서 파일>>열기를 선택합니다.
3. 컨트롤에 결합하려는 바로 가기 메뉴 파일을 탐색한 후 확인 버튼을 클릭합니다. 메뉴 파일에 대한 경로가 바로 가기 메뉴 편집기 대화 상자의 제목 표시줄에 디스플레이됩니다.
4. 바로 가기 메뉴 편집기 대화 상자를 닫습니다.
5. 런타임 바로 가기 메뉴를 선택된 메뉴 파일로 변경할지 여부를 묻으면, 예 버튼을 클릭합니다.

다음 단계를 따라 바로 가기 런타임 메뉴를 컨트롤에서 분리합니다.

1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>런타임 바로 가기 메뉴>>편집을 선택하여 바로 가기 메뉴 편집기 대화 상자를 디스플레이합니다.
2. 메뉴 타입 풀다운 메뉴에서 기본을 선택합니다. 파일 경로는 더 이상 바로 가기 메뉴 편집기 대화 상자의 제목 표시줄에 나타나지 않습니다.
3. 바로 가기 메뉴 편집기 대화 상자를 닫습니다.
4. 런타임 바로 가기 메뉴를 기본 메뉴로 변경할지 여부를 묻으면 예 버튼을 클릭합니다.

런타임 바로 가기 메뉴를 **정적으로** 또는 **프로그램식으로** 사용자 정의할 수 있습니다.

프론트패널의 런타임 바로 가기 메뉴를 프로그램적으로 사용자 정의하기

다음 단계를 따라 VI의 프론트패널에 대한 바로 가기 메뉴를 프로그램적으로 사용자 정의합니다.

1. 블록다이어그램에 이벤트 구조를 추가하고, 블록다이어그램에서 영역을 끌어서 넓힙니다.
2. 이벤트 구조에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 케이스에 의해 핸들되는 이벤트 편집을 선택합니다.
3. 이벤트 편집 대화 상자에서, 이벤트 소스 리스트에서 구획을 선택한 후 이벤트 리스트에서 바로 가기 메뉴 활성화?를 선택합니다.



노트

LabVIEW는 첫번째 구획을 구획으로 라벨합니다. LabVIEW는 분리자 막대를 프론트패널에 추가하여 생성한 그 다음 구획을 구획X라고 라벨합니다. 여기서 X는 구획을 생성한 순서를 말합니다. 구획에 있는 스크롤 막대에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 보이는 아이템>>라벨을 선택하여 구획의 라벨을 볼 수 있습니다.

4. 확인 버튼을 클릭하여 대화 상자를 닫습니다.
5. 블록다이어그램에서 바로 가기 메뉴 활성화? 이벤트 구조 안에 [메뉴 아이템 삽입] 함수를 추가합니다.
6. [메뉴 아이템 삽입] 함수의 아이템 이름 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>상수를 선택합니다. 배열 상수가 나타납니다.
7. 문자열 배열 상수의 원소 0에 이 어플리케이션에 대해서 입력하고 원소 1에 도움말을 입력합니다.
8. MenuRef 이벤트 데이터 필드를 [메뉴 아이템 삽입] 함수의 메뉴 참조 입력에 연결합니다.
9. 연속 실행 버튼을 클릭합니다.
10. 프론트패널 윈도우 안에서 마우스 오른쪽 버튼을 클릭합니다. 바로 가기 메뉴는 이 어플리케이션에 대해서 아이템과 도움말 아이템을 포함합니다.
11. **바로 가기 메뉴 선택 이벤트를 설정**하여 아이템의 선택을 처리합니다.

컨트롤의 런타임 바로 가기 메뉴를 프로그램적으로 사용자 정의하기

다음 단계를 따라 컨트롤에 대한 메뉴를 프로그램적으로 사용자 정의합니다.

1. 컨트롤을 프론트패널에 추가합니다. 이 예제는 숫자 컨트롤을 사용합니다.
2. 블록다이어그램에 이벤트 구조를 추가합니다. 숫자 컨트롤을 이벤트 구조에 포함시킬 필요는 없습니다.
3. 이벤트 구조에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 케이스에 의해 핸들되는 이벤트 편집을 선택합니다.

- 이벤트 편집 대화 상자에서, 이벤트 소스 리스트에서 숫자를 선택한 후 이벤트 리스트에서 바로 가기 메뉴 활성화?를 선택합니다.
- 확인 버튼을 클릭하여 대화 상자를 닫습니다.
- 블록다이어그램에서 바로 가기 메뉴 활성화? 이벤트 구조 안에 [메뉴 아이템 삽입] 함수를 추가합니다.
- [메뉴 아이템 삽입] 함수의 아이템 이름 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>> 상수를 선택합니다. 배열 상수가 나타납니다.
- 문자열 배열 상수의 원소 0에 도움말을 원소 1에 복사를 입력합니다.
- MenuRef 이벤트 데이터 필드를 [메뉴 아이템 삽입] 함수의 메뉴 참조 입력에 연결합니다.



팁

기본적인 바로 가기 아이템이 나타나지 않기를 원하는 경우, [메뉴 아이템 삭제] 함수를 사용합니다. 기본 바로 가기 메뉴 아이템을 삭제하려면, MenuRef 이벤트 데이터 필드를 [메뉴 아이템 삭제] 함수의 메뉴 참조 입력에 연결한 후 메뉴 참조 출력 출력을 [메뉴 아이템 삽입] 함수의 메뉴 참조 입력에 연결합니다.

- 연속 실행 버튼을 클릭합니다.
- 프론트패널로 전환하여 숫자형 컨트롤에서 마우스 오른쪽 버튼을 클릭합니다. 바로 가기 메뉴는 도움말 아이템과 복사 아이템을 포함합니다.
- 바로 가기 메뉴 선택 이벤트를 설정**하여 아이템의 선택을 처리합니다.



노트

또한, [메뉴 아이템 삽입] 함수의 입력으로 어플리케이션 아이템 태그를 제공하여 기본적인 바로 가기 메뉴 아이템을 추가할 수 있습니다.

컨트롤의 런타임 바로 가기 메뉴를 정적으로 사용자 정의하기

다음 단계를 따라 컨트롤의 런타임 바로 가기 메뉴를 정적으로 사용자 정의합니다.

- 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>런타임 바로 가기 메뉴>>편집을 선택하여 바로 가기 메뉴 편집기 대화 상자를 디스플레이합니다.



노트

배열 셀에 추가한 사용자 정의 런타임 바로 가기 메뉴를 포함하는 컨트롤 또는 인디케이터는 배열의 바로 가기 메뉴를 사용합니다.

- 대화 상자의 메뉴 타입 풀다운 메뉴에서 사용자를 선택합니다.
- 아이템 타입 풀다운 메뉴에서 사용자 아이템을 선택합니다. LabVIEW 기본 메뉴 아이템을 사용자 런타임 바로 가기 메뉴에 추가하려면, 어플리케이션 아이템을 선택하고 원하는 LabVIEW 바로 가기 메뉴 아이템을 선택합니다.

- 아이템 이름 텍스트 박스에 이 아이템에 사용하려는 텍스트를 입력합니다. 같은 텍스트가 아이템 태그 필드에 나타납니다. 태그 이름을 수정하여 바로 가기 메뉴 아이템이 유일한 식별자인지를 제공할 수 있습니다. 블록 다이어그램은 프로그램적으로 바로 가기 메뉴 아이템을 식별하기 위해 이 태그 이름을 사용합니다. 바로 가기 메뉴 아이템 태그는 바로 가기 메뉴 아이템 이름과 같은 이름을 가질 수 있습니다.
- 도구 모음의 파란색 더하기 버튼  을 클릭하여 바로 가기 메뉴에 더 많은 아이템을 추가합니다. 빨간색 버튼을  클릭하여 아이템을 삭제합니다.
- 파일>>저장을 선택합니다. 나타난 대화 상자에서, 바로 가기 메뉴를 컨트롤과 함께 저장할 것인지 또는 바로 가기 메뉴를 파일로 저장할지를 선택합니다. 파일로 저장을 선택할 경우, 바로 가기 메뉴에 이름을 입력하고 저장할 디스크의 위치를 묻는 대화 상자가 나타납니다. VI가 로드될 때, 사용자 정의된 바로 가기 메뉴가 자동으로 로드됩니다.
- 바로 가기 메뉴 선택(사용자) 이벤트를 설정**하여 아이템의 선택 대상을 처리합니다.

대화 상자의 바로 가기 메뉴 계층구조는 도구 모음의 화살표 버튼을 클릭하거나, 편집 메뉴의 계층구조 조작 옵션을 사용하거나, 끌어서 놓기를 사용하여 정렬할 수 있습니다. 서브메뉴 문양을 클릭하거나 편집 메뉴의 확장 또는 축소 옵션에서 선택을 이용하여 계층구조를 확장하거나 축소할 수 있습니다. 분리자를 추가하려면, 아이템 타입 풀다운 메뉴의 분리자를 선택합니다.

런타임 바로 가기 메뉴 비활성화하기

다음 단계를 따라 VI에 대한 모든 런타임 바로 가기 메뉴를 비활성화해서 사용자가 객체에서 마우스 오른쪽 버튼을 클릭한 후 모양을 변경할 수 없도록 합니다.

- 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
- 항목 풀다운 메뉴에서 윈도우 모양을 선택합니다.
- 사용자 정의 버튼을 클릭하여 윈도우 모양 사용자 정의 대화 상자를 디스플레이합니다.
- 기본 런타임 바로 가기 메뉴 허용 확인란에서 확인 표시를 제거합니다.
- 변경을 저장하기 위해 확인 버튼을 두 번 클릭합니다.

개별 컨트롤과 인디케이터에 대한 런타임 바로 가기 메뉴를 비활성화하기 위해서는 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 고급>>런타임 바로 가기 메뉴>>비활성화를 선택합니다.

런타임 바로 가기 메뉴 선택 처리하기

런타임 바로 가기 메뉴를 정적 또는 프로그램적으로 사용자 정의한 후에는 반드시 이벤트 구조가 사용자 메뉴에 있는 각 메뉴 아이템을 처리할 수 있도록 설정해야 합니다.

다음 단계를 따라 사용자 메뉴에서 생성한 메뉴 아이템을 처리합니다.

1. 블록다이어그램에 이벤트 구조를 추가합니다.
2. 이벤트 구조에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 이 케이스에 의해 핸들되는 이벤트 편집을 선택합니다.
3. 이벤트 편집 대화 상자의 이벤트 소스 리스트에서 컨트롤 또는 구획을 선택하고 이벤트 리스트에서 바로 가기 메뉴 선택(사용자)를 선택합니다.
4. 확인 버튼을 클릭하여 대화 상자를 닫습니다.
5. 블록다이어그램에 이벤트 구조를 둘러싸는 While 루프를 추가합니다.
6. 케이스 구조를 이벤트 구조의 바로 가기 메뉴 선택(사용자) 케이스 내부에 추가합니다.
7. 아이탬태그 이벤트 데이터 필드를 케이스 구조 경계의 선택자 터미널에 연결합니다.
8. 케이스 선택자 라벨에 바로 가기 메뉴 아이탬의 태그 이름을 입력합니다.
9. 생성한 각 바로 가기 메뉴 아이탬에 대해 케이스를 생성합니다. 예를 들어, 도움말 아이탬이 있을 경우, [온라인 도움말 컨트롤] 함수를 사용하여, 사용자가 바로 가기 메뉴에서 도움말을 선택하면 도움말 파일이 열리도록 할 수 있습니다.



관련 정보

[컨트롤의 런타임 바로 가기 메뉴를 정적으로 사용자 정의하기](#) (1367 페이지 참조)

[컨트롤의 런타임 바로 가기 메뉴를 프로그램적으로 사용자 정의하기](#) (1366 페이지 참조)

[각 케이스 구조 서브다이어그램의 값 입력하기](#) (1245 페이지 참조)

[서브다이어그램 추가 및 복제하기](#) (1241 페이지 참조)

문제 해결

프로브

사용자 프로브를 사용할 때의 유의점과 권장사항

다음 리스트는 사용자 프로브를 생성하고 사용할 때 알아야 할 유의점과 권장사항들을 설명합니다.

- 사용자 프로브는 데이터가 와이어를 통해서 흐를 때에만 실행됩니다. 즉, 사용자 프로브의 컨트롤을 언제나 동작시킬 수 있지만, 변경된 컨트롤 값에 대한 프로그램에서의 응답은 프로브가 와이어로부터 데이터를 받을 때에만 발생합니다.
- VI와 유사한 방식으로 사용자 프로브를 디버깅할 수 있습니다. 하지만, 프로브는 자신의 블록다이어그램뿐만 아니라 그 SubVI의 블록다이어그램도 프로브할 수 없습니다. 프로브를 디버깅할 때, **일반 프로브**를 사용합니다.
- 프로브를 labview\user.lib_probes\default나 **기본 데이터**\Probes 디렉토리에 저장하십시오. labview\user.lib_probes\default 디렉토리에 저장된 프로브는 컴퓨터에서 원격으로 LabVIEW를 실행하는 누구에게나 사

용가능합니다. 이 디렉토리는 퍼블릭 프로브에 유용합니다. 기본 데이터 디렉토리가 컴퓨터에 로그인한 사용자에게 따라서 변경되기 때문에, National Instruments는 이 디렉토리에 개인적인 프로브를 저장하는 것을 권장합니다.

- 주요 블록 다이어그램은 프로브가 실행을 끝낼 때까지 일시 정지합니다. 루프 또는 이벤트 구조는 해당 원소가 프로브가 실행을 끝내는 시간을 증가시킬 수 있으므로 사용자 프로브에서 사용하는 것을 피합니다.
- 사용자 프로브에 초기화되지 않은 시프트 레지스터를 사용할 때에는 주의합니다. VI가 재호출 VI가 아닌 경우라도, VI가 프로브로 사용되면, 초기화되지 않은 모든 시프트레지스터는 **재호출** VI와 같이 작동합니다. 각 프로브는 루프의 마지막 값을 유지하며 동일한 프로브의 다른 인스턴스와 이 값을 공유하지 않습니다.

사용자 프로브 생성하기

더 많은 컨트롤로 LabVIEW가 어떻게 와이어를 통해서 데이터 흐름을 조사하는지를 알려는 경우 프로브를 생성합니다. 새 프로브를 생성할 때, 프로브의 데이터 타입은 마우스 오른쪽 버튼으로 클릭한 와이어의 데이터 타입과 같습니다. 사용자 정의 프로브를 생성하기 전에 **유의점 및 권장사항**을 참조하십시오.

다음 단계를 따라 사용자 프로브를 생성합니다.



노트

LabVIEW **클래스** 또는 **인터페이스** 데이터 타입의 사용자 프로브를 생성하는 경우, 생성하는 사용자 프로브가 클래스 또는 인터페이스의 멤버 VI인지 여부에 따라 다음에 나열되는 단계 중 일부가 다를 수 있습니다. 변경사항이 있으면 해당 부분에서 노트 처리됩니다. 클래스의 멤버 VI인 사용자 프로브는 멤버 프로브이며 클래스의 프라이빗 데이터에 직접 접근할 수 있습니다. 멤버 프로브는 모든 프라이빗 클래스 데이터 부분을 표시할 수 있습니다. 멤버가 아닌 사용자 프로브는 클래스의 퍼블릭 인터페이스를 통해 접근 가능한 정보만을 디스플레이할 수 있습니다.

1. 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 사용자 정의 프로브 >> 새로 만들기를 선택하여 새 프로브 생성 대화 상자를 디스플레이합니다.
2. 새 프로브를 생성할 것인지 또는 기존 프로브에 기초하여 프로브를 생성할 것인지를 선택합니다.
 - 기존의 프로브에서 프로브 생성을 선택하여 기존의 프로브를 기준으로 프로브를 생성합니다. 기존 프로브는 LabVIEW가 제공하는 프로브일 수도 있고, 사용자가 새 프로브 생성 대화 상자를 사용하여 생성한 다른 프로브일 수도 있습니다.
 1. 다음 버튼을 클릭하고 새 프로브가 기초로 할 기존 프로브를 선택합니다. LabVIEW는 labview\vi.lib_probes 디렉토리에서 **제공되는 프로브** 모두와 사용가능한 모든 사용자 프로브를 디스플레이합니다.
 2. 다음 버튼을 클릭합니다.
 3. 파일 이름, 윈도우 제목, 바로 가기 메뉴 이름, 프로브를 저장하려는 디렉토리를 입력합니다. 디렉토리를 지정하지 않는 경우, LabVIEW는 **기본 데이터 디렉토리**의 프로브 서브디렉토리에 사용자 프로브를 저장합니다. 사용자 프로브를 특정 데이터 타입의 기본 프로브로 설정하고 싶으면 labview\user.lib_probes\default 디

렉토리에 프로브를 저장하십시오. LabVIEW는 설치가 제거되거나 다시 설치될 때 기존 파일을 덮어쓰기 때문에 labview\vi.lib_probes 디렉토리에 기존 파일로 프로브를 저장하지 않습니다.

4. 저장 버튼을 클릭합니다. LabVIEW는 사용자가 선택한 기존 프로브와 일치하는 프로브를 엽니다.

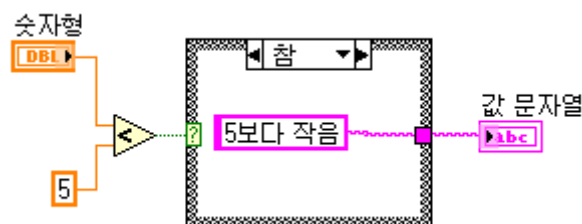
노트 LabVIEW 클래스나 인터페이스의 멤버가 아닌 프로브를 생성하는 경우, 위의 설명을 따르십시오. LabVIEW 클래스 또는 인터페이스의 멤버 프로브를 생성하는 경우, 사용자 프로브를 클래스나 인터페이스의 다른 멤버 vi를 저장한 서브디렉토리에 저장하여, 클래스 또는 인터페이스의 아이템이 같이 있어 쉽게 배포되도록 할 수 있습니다.

- 새 프로브 생성을 선택하여 새로운 프로브를 생성합니다.

1. 다음 버튼을 클릭합니다.
2. 파일 이름, 윈도우 제목, 바로 가기 메뉴 이름, 프로브를 저장하려는 디렉토리를 입력합니다.
3. 저장 버튼을 클릭합니다. LabVIEW는 마우스 오른쪽 버튼을 클릭한 와이어의 데이터 타입인 컨트롤과 조건적인 브레이크포인트와 같이 동작하는 불리언 인디케이터를 포함하는 프로브를 엽니다. 또한 프로브는 [문자열 프로브로] 함수와 문자열 인디케이터를 포함하고 있으며, 이를 통해 LabVIEW가 프로브 관찰 윈도우의 값 옆에 디스플레이하는 값을 설정할 수 있습니다. 프로브가 깨지므로, 컨트롤 또는 인디케이터를 삭제하거나 커넥터 팬을 변경하지 마십시오.
4. (옵션) [문자열 프로브로] 함수의 데이터 값 입력에 어떤 데이터 타입이든지 와이어로 연결하면, 프로브 관찰 윈도우의 값 옆에서 해당 데이터를 문자열로 볼 수 있습니다. 이 함수는 데이터를 분해해서 값 문자열 인디케이터가 값 옆에 디스플레이할 수 있는 문자열로 변환시킵니다. 또한, [문자열 프로브로] 함수는 공간에 맞도록 데이터를 압축하고 정리합니다. 이 함수를 사용하여 복소수 데이터 타입으로 부터 연계된 정보를 추출할 수 있습니다.

노트 LabVIEW는 컨트롤을 [문자열 프로브로] 함수에 자동으로 연결합니다. [문자열 프로브로] 함수를 제거하여 LabVIEW가 값 옆에 디스플레이할 값을 사용자가 지정할 수 있습니다.

5. (옵션) LabVIEW가 디스플레이할 문자열을 프로브 관찰 윈도우의 값 옆에서 지정하려는 경우, 다음 블록 다이어그램에서 보이는 것처럼 해당 문자열을 값 문자열 인디케이터에 와이어로 연결합니다.



앞의 그림은 숫자형 데이터를 포함하는 와이어의 사용자 프로브를 보여줍니다. **케이스 구조**를 사용하여 LabVIEW가 와이어를 통해 전

달되는 데이터를 디스플레이하는 기본 설정 대신에 5보다 큰 숫자 또는 5보다 작은 숫자를 디스플레이하도록 만들 수 있습니다.

6. 프로브 사용자 지정을 끝나치면, 저장하고 닫습니다. 프로브를 편집하려는 경우, 반드시 프로브를 저장한 디렉토리에서 열어야만 합니다.



노트 LabVIEW 클래스 또는 인터페이스의 데이터 타입의 프로브를 생성하는 경우, 기본적으로 새 프로브 생성 대화 상자는 새 프로브를 LabVIEW 클래스나 인터페이스의 멤버로 만들지 않습니다. 프로브를 클래스 또는 인터페이스의 멤버 프로브로 하려면, **멤버 프로브를 클래스 또는 인터페이스에 추가해야 합니다..**

생성한 프로브를 사용하기 위해서는, 와이어에서 다시 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로브를 선택합니다. 프로브를 생성하면 프로브는 해당 데이터 타입의 기본 프로브가 되며, LabVIEW는 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로브를 선택할 때 그 프로브를 로드합니다. 데이터 타입의 기본 프로브를 변경하기 위해서는, 해당 데이터 타입의 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 사용자 프로브를 선택하고 바로 가기 메뉴에서 제공된, 사용자, 또는 일반 프로브를 선택합니다.



노트 LabVIEW 클래스 또는 인터페이스에 프로브를 생성한 후 LabVIEW 클래스나 인터페이스를 다른 컴퓨터로 옮기거나 현재 컴퓨터에서 다시 설치하는 경우, 해당 클래스나 인터페이스의 사용자가 사용할 수 있는 기본 프로브는 일반 프로브입니다. 사용자 프로브는 컴퓨터에서 사용할 때까지 어떤 데이터 타입의 기본 프로브도 아닙니다. 그러나 멤버 프로브를 생성하는 경우, 클래스/인터페이스 프로퍼티 대화 상자를 사용하여 **사용자 프로브를 클래스나 인터페이스의 초기 기본 프로브로 설정**할 수 있습니다. 따라서 해당 클래스 또는 인터페이스를 새로운 컴퓨터에 배포할 때, 클래스나 인터페이스의 사용자가 사용할 수 있는 기본 프로브는 일반적인 프로브가 아니라 사용자 프로브입니다.

기존 프로브를 편집하려면, 프로브 VI를 열거나 새 프로브 생성 대화 상자를 사용하여 기존 프로브에 기초하여 새 프로브를 생성합니다.

VI 실행 후 프로브 생성하기

VI를 설정하여 와이어 값을 유지하게 함으로써 블록 다이어그램에 **프로브를 생성**할 때 프로브가 마지막 VI 실행에서 와이어를 따라 흐른 데이터를 디스플레이하도록 할 수 있습니다. VI 실행 후에 프로브를 사용하는 것은 블록 다이어그램이 복잡하고 VI를 디버그하려는 경우 유용합니다.

다음 단계를 따라 VI 실행 후 프로브를 생성합니다.

1. 블록 다이어그램 도구 모음의 와이어 값 유지 버튼  을 클릭하여 프로브와 함께 사용할 와이어의 값을 유지합니다.
2. VI를 실행합니다. VI가 반드시 한 번 이상 실행되어야 생성한 프로브로 즉시 와이어 값을 얻을 수 있습니다.
3. VI를 정지합니다.

4. 와이어에서 마우스 오른쪽 버튼을 클릭하고 프로브를 선택하여 와이어에 프로브를 놓습니다. 와이어를 통과한 마지막 데이터가 프로브 관찰 윈도우의 서브패널 안에 나타나는 것을 확인할 수 있습니다.

각 실행에서 와이어의 값을 저장하지 않도록 하려면 와이어의 값을 유지하지 않음 버튼  을 아무때나 클릭합니다.



노트

와이어의 값을 유지하지 않음 옵션을 선택하면 메모리 요구량을 줄이고 약간의 성능 개선을 도모할 수 있습니다.

인디케이터를 사용하여 프로브된 데이터 보기

사용자 프로브를 생성하거나 **제공된 프로브** 중 하나를 사용하는 것 외에, 인디케이터를 사용하여 와이어를 통과하는 데이터 포인트를 볼 수 있습니다.

다음 단계를 따라 인디케이터를 사용하여 프로브된 데이터를 봅니다.

1. 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 사용자 프로브 >> 컨트롤을 선택합니다.
2. 컨트롤 팔레트에서 인디케이터를 선택하여 데이터를 디스플레이합니다. 예를 들어, 차트를 사용하여 숫자 데이터를 볼 수 있습니다. 또한, 컨트롤 선택을 클릭하고 컴퓨터 또는 서버의 공유 디렉토리에 저장된 **사용자 컨트롤 또는 타입 정의**를 선택하거나 새 프로브 생성 대화 상자를 사용하여 만든 사용자 컨트롤을 탐색합니다. 선택한 인디케이터의 데이터 타입은 오른쪽 마우스로 클릭한 와이어의 데이터 타입과 일치해야 합니다.

제공된 프로브 사용하기

제공된 프로브를 사용하여 와이어를 통해 흐르는 데이터에 대한 포괄적 정보를 디스플레이합니다. 또한 제공된 프로브를 사용하여 와이어를 통과하는 데이터에 기초하여 응답할 수 있습니다.

다음 단계를 따라 제공된 프로브를 사용합니다.

1. 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 사용자 프로브를 선택합니다.
2. 다음 제공된 프로브 중 선택합니다. 오른쪽 마우스로 클릭한 와이어의 데이터 타입과 일치하는 모든 제공된 프로브가 바로 가기 메뉴의 최상위에 나타납니다.
 - .NET 참조 번호 프로브 — .NET 참조 번호, 객체의 타입, 객체의 여러 코드, 참조의 16진수 값에 의해 나타낸 객체에서 ToString() 메소드의 호출 결과를 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.
 - 아날로그 웨이브폼 프로브(DBL)—아날로그 데이터의 그래프; 프로브가 데이터, x값, 시간을 받은 때를 지정하는 타임스탬프가 있는 테이블; 웨이브폼 데이터의 속성을 디스플레이하는 테이블을 디스플레이합니다.

- 조건적 불리언 프로브—불리언 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 배정도 배열 프로브—배정도 부동소수 컨트롤의 배열을 모니터링합니다. 데이터 탭은 입력 배열의 크기를 디스플레이합니다. 조건적 브레이크 포인트를 설정하기 위해서 조건 페이지를 사용합니다. 배열의 크기 또는 배열 내의 원소를 기준으로 조건을 설정할 수 있습니다. 여러 조건을 선택하는 경우, LabVIEW는 그 조건 중 어떤 것이라도 참이면 브레이크 포인트를 실행합니다.
- 조건적 배정도 프로브—배정도 부동소수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다. 여러 조건을 선택하는 경우, LabVIEW는 그 조건 중 어떤 것이라도 참이면 브레이크 포인트를 실행합니다.
- 조건적 에러 프로브—에러 상태, 코드 그리고 설명을 디스플레이합니다. 조건적 브레이크 포인트가 에러, 경고 또는 특정 에러 코드에서 일시 정지하도록 허용합니다.
- 조건적 부호있는 8 프로브—32비트 부호있는 정수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 부호있는 16 프로브—16비트 부호있는 정수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 부호있는 32 배열 프로브—조건적 배정도형 배열 프로브와 비슷하며, 이 프로브는 32비트 부호있는 정수 컨트롤의 배열을 모니터링합니다. 데이터 페이지는 입력 배열의 크기를 디스플레이합니다. 조건적 브레이크 포인트를 설정하기 위해서 조건 페이지를 사용합니다. 배열의 크기 또는 배열 내의 원소를 기준으로 조건을 설정할 수 있습니다. 여러 조건을 선택하는 경우, LabVIEW는 그 조건 중 어떤 것이라도 참이면 브레이크 포인트를 실행합니다.
- 조건적 부호있는 32 프로브—32비트 부호있는 정수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 부호있는 64 배열 프로브—조건적 배정도형 배열 프로브와 비슷하며, 이 프로브는 64비트 부호있는 정수 컨트롤의 배열을 모니터링합니다. 데이터 페이지는 입력 배열의 크기를 디스플레이합니다. 조건적 브레이크 포인트를 설정하기 위해서 조건 페이지를 사용합니다. 배열의 크기 또는 배열 내의 원소를 기준으로 조건을 설정할 수 있습니다. 여러 조건을 선택하는 경우, LabVIEW는 그 조건 중 어떤 것이라도 참이면 브레이크 포인트를 실행합니다.
- 조건적 부호있는 64 프로브—64비트 부호있는 정수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 불리언 프로브—문자열 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 부호없는 8 프로브—8비트 부호없는 정수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 부호없는 16 프로브—16비트 부호없는 정수 데이터를 디스플레이하고 조건적 브레이크 포인트를 허용합니다.
- 조건적 부호없는 32 배열 프로브—조건적 배정도형 배열 프로브와 비슷하며, 이 프로브는 32비트 부호없는 정수 컨트롤의 배열을 모니터링합니다. 데이터 페이지는 입력 배열의 크기를 디스플레이합니다. 조건적 브레이크 포인트를 설정하기 위해서 조건 페이지를 사용합니다. 배열의 크기 또는 배열 내의 원소를 기준으로 조건을 설정할 수 있습니다. 여러 조건을 선택하는 경우, LabVIEW는 그 조건 중 어떤 것이라도 참이면 브레이크 포인트를 실행합니다.

- 조건적 부호없는 32 프로브—32비트 부호없는 정수 데이터를 디스플레이하고 조건적 브레이크포인트를 허용합니다.
- 조건적 부호없는 64 배열 프로브—조건적 배정도형 배열 프로브와 비슷하며, 이 프로브는 64비트 부호없는 정수 컨트롤의 배열을 모니터합니다. 데이터 페이지는 입력 배열의 크기를 디스플레이합니다. 조건적 브레이크포인트를 설정하기 위해서 조건 페이지를 사용합니다. 배열의 크기 또는 배열 내의 원소를 기준으로 조건을 설정할 수 있습니다. 여러 조건을 선택하는 경우, LabVIEW는 그 조건 중 어떤 것이라도 참이면 브레이크포인트를 실행합니다.
- 조건적 부호없는 64 프로브—64비트 부호없는 정수 데이터를 디스플레이하고 조건적 브레이크포인트를 허용합니다.
- 컨트롤 참조 번호 프로브—클래스 이름, 컨트롤 라벨, 참조의 16진수 값, 소유자의 VI 이름 그리고 VI의 경로를 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 일시 정지하기 위해서 이 프로브를 사용할 수 있습니다.
- 기본 그림 프로브—그림을 디스플레이합니다.
- 기본 픽스맵 프로브—픽스맵을 디스플레이합니다.
- 디지털 웨이브폼 프로브—디지털 데이터의 그래프; 프로브가 데이터, x값, 시간을 받은 때를 지정하는 타임스탬프가 있는 테이블; 웨이브폼 데이터의 속성을 디스플레이하는 테이블을 디스플레이합니다.
- 다이나믹 데이터 타입 프로브—각 웨이브폼의 속성을 디스플레이하고 데이터를 그래프로 표시합니다.
- 히스토리-볼리언 — 와이어를 통해 흐르는 볼리언 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - CDB — 와이어를 통해 흐르는 복소수 배정도 부동소수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - CSG — 와이어를 통해 흐르는 복소수 단정도 부동소수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - CXT — 와이어를 통해 흐르는 복소수 확장형 부동소수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - DBL — 와이어를 통해 흐르는 배정도 부동소수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - EXT — 와이어를 통해 흐르는 확장형 부동소수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - I8 — 와이어를 통해 흐르는 8비트 부호 있는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - I16 — 와이어를 통해 흐르는 16비트 부호 있는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - I32 — 와이어를 통해 흐르는 32비트 부호 있는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - I64 — 와이어를 통해 흐르는 64비트 부호 있는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - 경로 — 와이어를 통해 흐르는 경로 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - SGL — 와이어를 통해 흐르는 단정도 부동소수 데이터의 히스토리를 디스플레이합니다.

- 히스토리 - 문자열 — 와이어를 통해 흐르는 문자열 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - U8 — 와이어를 통해 흐르는 8비트 부호 없는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - U16 — 와이어를 통해 흐르는 16비트 부호 없는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - U32 — 와이어를 통해 흐르는 32비트 부호 없는 정수 데이터의 히스토리를 디스플레이합니다.
- 히스토리 - U64 — 와이어를 통해 흐르는 64비트 부호 없는 정수 데이터의 히스토리를 디스플레이합니다.
- 그림 프로브—그림을 디스플레이하고 이미지의 높이와 폭을 포함합니다.
- 픽스맵 프로브—그림을 디스플레이하고 입력 그림의 폭, 높이, 이미지 색 농도에 관한 정보를 포함합니다.
- 랑데부 참조 번호 프로브—랑데부 이름과 크기, 랑데부 수집을 기다리고 있는 VI 또는 함수의 숫자, 그리고 참조의 16진수 값을 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.
- 세마포어 참조 번호 프로브—세마포어 이름과 크기, 세마포어 얻기를 기다리고 있는 VI 또는 함수의 숫자, 그리고 참조의 16진수 값을 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.
- 문자열 알림자 참조 번호 프로브—알림자 이름, 마지막 알림, 알림자를 기다리고 있는 VI 또는 함수의 숫자, 그리고 참조의 16진수 값을 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.
- 문자열 큐 참조 번호 프로브—큐 이름, 큐의 원소, 큐 안의 원소 개수, 삽입 대기 중인 아이템 개수, 제거 대기 중인 아이템 개수 그리고 참조의 16진수 값을 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.
- 실행 하이라이트 토글 — VI가 실행 중일 때 지정된 지점에서 실행 하이라이트를 토글합니다. 이 프로브는 RT 타겟에서 지원되지 않습니다.
- 반투명한 볼리언 프로브-볼리언 데이터를 와이어에 디스플레이합니다.
- VI 참조 번호(패널 이미지) 프로브—VI 이름, VI 경로, VI 프론트패널 이미지, 참조의 16진수 값을 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.
- VI 참조 번호 프로브—VI 이름, VI 경로, 참조의 16진수 값을 디스플레이합니다. 값이 유효하지 않은 참조 번호인 경우 이 프로브를 사용하여 브레이크포인트를 설정할 수 있습니다.

프로브를 선택하면 프로브는 해당 데이터 타입의 기본 프로브가 되며, LabVIEW는 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로브를 선택할 때 그 프로브를 로드합니다.

제공된 프로브를 기반으로 새 프로브를 생성하고 사용자 프로브 바로 가기 메뉴에 추가할 수 있습니다.

일반 프로브 사용하기

일반 프로브를 사용하여 와이어를 통과하는 데이터를 봅니다. 일반 프로브는 데이터를 디스플레이합니다. 일반 프로브를 설정하여 데이터에 응답할 수 없습니다. 와이어에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 사용자 프로브 >> 일반 프로브를 선택하여 일반 프로브를 사용합니다.

VI와 유사한 방식으로 사용자 프로브를 디버깅할 수 있습니다. 하지만, 프로브는 자신의 블록다이어그램뿐만 아니라 그 SubVI의 블록다이어그램도 프로브할 수 없습니다. 프로브를 디버깅하려는 경우, 일반 프로브를 사용합니다.

공급된 프로브 또는 **사용자 프로브**를 사용하여 와이어를 통과하는 데이터에 응답합니다.

LabVIEW 클래스에서 일반 프로브 사용하기

LabVIEW 클래스 와이어에 일반 프로브를 사용할 수 있습니다. LabVIEW 클래스 와이어의 일반 프로브는 실행 중인 클래스 이름, 프라이빗 클래스 데이터, 그리고 모든 직상위 클래스의 프라이빗 클래스 데이터를 보여줍니다. 잠금 상태의 LabVIEW 클래스에서 일반 프로브를 사용하면, 오직 실행 중인 클래스의 이름과 클래스 계층구조 내에서 잠금해제 상태의 모든 프라이빗 데이터를 프로브에서 볼 수 있습니다.

프로브 도구 사용하기

프로브 도구를 사용하여 VI가 실행될 때 와이어의 중간값을 체크합니다. 여러 개의 연속적인 작업으로 구성된 복잡한 블록다이어그램의 경우, 어느 동작에서 잘못된 데이터가 반환되는지 알아보기 위해 프로브 도구를 사용합니다. 프로브 도구를 **실행 하이라이트**, **단계별 실행**, **브레이크포인트**와 함께 사용하여 데이터가 부정확한지 여부와 어느 데이터가 부정확한지 결정합니다. 데이터가 있는 경우, 실행 하이라이트나 단계별 실행을 사용 중일 때, 또는 브레이크포인트에서 일시 정지했을 때 즉시 프로브가 업데이트됩니다. 또한 단계별 실행 또는 브레이크포인트로 인해 노드에서 일시 정지 상태일 때, 방금 실행된 와이어를 프로브하면 해당 와이어를 따라 진행 중인 값을 볼 수 있습니다.

사용자 프로브를 생성하거나, **제공된 프로브**를 사용하거나, **프로브된 데이터를 볼 수 있는 인디케이터**를 사용할 수 있습니다.

다음 단계에 따라 프로브 도구 를 사용합니다.

1. (옵션) 마지막 VI 실행에서 와이어를 따라 흐른 데이터를 디스플레이할 프로브를 원하는 경우, 블록다이어그램에서 **와이어 값 유지** 옵션을 사용합니다.
2. 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로브를 선택하거나 바로 가기 메뉴에서 사용자 프로브와 프로브를 선택합니다. 제공된 프로브를 선택하지 않거나 **사용자 프로브를 생성**하지 않은 경우, LabVIEW는 labview\vi.lib_probes\default와 labview\user.lib_probes\default 디렉토리를 검색하고 오른쪽 마우스로 클릭한 와이어의 데이터 타입과 일치하는 프로브를 찾습니다. 사용자 프로브 또는 기본 프로브를 선택하면, 그 프로브는 해당 데이터 타입의 기본 프로브가 됩니다. 프로브 관찰 윈도우가 나타납니다. LabVIEW는 자동으로 프로브 관찰 윈도우에 프로브에 번호를 매기고 나열하며, 사

용자가 클릭한 와이어의 문양에 이 프로브 번호를 디스플레이합니다. 또한, 프로브 도구를 사용하여 와이어를 클릭하거나 보기» 프로브 관찰 윈도우를 선택하여 프로브 관찰 윈도우를 디스플레이할 수 있습니다.

3. (선택 옵션) 데이터가 변경되리라 예상되는 VI의 다른 와이어와 객체에는 더 많은 프로브를 놓습니다. 필요한 경우 방해가 되지 않도록 프로브 관찰 윈도우를 밖으로 옮깁니다.
4. VI를 실행합니다. 프로브 관찰 윈도우의 오른쪽에 있는 서브패널은 와이어를 통과하는 데이터를 디스플레이합니다. 서브패널에서 보고 싶은 데이터를 포함하는 프로브를 선택합니다. 별도의 프로브 윈도우에서 프로브를 열려면, 프로브 관찰 윈도우에 있는 해당 프로브에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 윈도우 열기를 선택합니다. 또한, 프로브를 선택한 후 프로브 관찰 윈도우 도구 모음에서 별도의 프로브 윈도우 열기를 클릭해도 됩니다.
5. (선택 옵션) 블록 다이어그램에서 프로브된 와이어의 값을 보여주는 서브패널이나 윈도우의 위치를 찾으려면, 프로브와 연계된 와이어에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로브 찾기를 선택합니다. 서브패널이나 프로브 윈도우가 빨간색 경계로 하이라이트됩니다.
6. 서브패널이나 프로브 윈도우에 나타나는 데이터를 예상하던 데이터와 비교합니다.
7. 프로브를 닫으려면, 프로브 관찰 윈도우에 있는 프로브 또는 프로브가 속해 있는 VI에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 제거를 선택합니다. 또한, 프로브 관찰 윈도우에서 프로브를 선택한 후, 프로브 관찰 윈도우 도구 모음의 선택한 프로브 제거를 클릭해도 됩니다. 1 개 이상의 프로브를 선택하거나 프로브가 속해 있는 VI를 선택하여, 해당 VI에 있는 프로브를 모두 제거할 수 있습니다. 사용자가 프로브 관찰 윈도우나 블록 다이어그램을 닫으면, 모든 프로브가 자동으로 닫힙니다.

프로브 관찰 윈도우를 사용하여 데이터를 변경할 수는 없습니다. 옵션으로 브레이크포인트로 작동하는 사용자 프로브를 제외하면, 프로브는 VI가 실행하는 방식에 영향을 미치지 않습니다.



노트

사용자 프로브 사용 방법에 대해 사용자에게 도움을 주려는 경우, 도움말 버튼을 프로브에 놓지 않습니다. LabVIEW는 데이터가 와이어를 따라 흐를 때만 프로브를 실행하기 때문입니다. 그 대신, 프로브의 프론트패널에 **텍스트를 추가**하거나 각 프론트패널 컨트롤에 **설명과 팁 정보를 추가**하여 도움을 줍니다. 전체 프로브에 대한 도움말을 제공하려면 컨트롤을 탭 컨트롤이나 클러스터 안에 놓고 탭 컨트롤이나 클러스터에 대한 설명과 팁 정보를 추가할 것을 고려하십시오.

프로브 관찰 윈도우 또는 프로브 윈도우의 프로브 디스플레이 서브패널에 있는 데이터에서 마우스 오른쪽 버튼을 클릭할 때 나타나는 바로 가기 메뉴를 통해 다음의 3가지 옵션을 사용할 수 있습니다.

- 기본값으로 다시 초기화—프로브 윈도우가 디스플레이하는 값을 기본값으로 리셋합니다.
- 데이터 복사—같은 VI 또는 다른 VI에 있는 다른 컨트롤에 붙여넣을 데이터를 복사합니다. 또한, 프로브 관찰 윈도우에 나열된 프로브에서 마우스 오른쪽 버튼을 클릭할 때 나타나는 바로 가기 메뉴에서도 이 옵션을 사용할 수 있습니다.
- 와이어 찾기—관련된 와이어를 찾습니다. LabVIEW는 해당 와이어를 포함하는 블록 다이어그램을 가져오고 와이어를 하이라이트합니다. 또한, 프로브 관찰 윈도우

에 나열된 프로브에서 마우스 오른쪽 버튼을 클릭할 때 나타나는 바로 가기 메뉴에서도 이 옵션을 사용할 수 있습니다.

브레이크포인트 관리하기

브레이크포인트 도구를 사용하여 VI, 노드, 와이어에 브레이크포인트를 놓고 그 위치에서 실행을 일시 중지할 수 있습니다. 와이어에 브레이크포인트를 설정하는 경우, 데이터가 와이어를 통과하면 실행이 일시 중지되고 일시 중지 버튼이 빨간 색이 됩니다. 블록 다이어그램에 브레이크포인트를 놓으면, 블록 다이어그램에 있는 모든 노드가 실행된 다음에 실행이 일시 중지됩니다. 블록 다이어그램의 경계가 빨간 색으로 변하고 깜박거리면서 브레이크포인트가 놓여졌다는 것을 알립니다.

브레이크포인트에서 VI가 일시 중지하면, LabVIEW는 블록 다이어그램을 앞으로 가져와서 선택 표시를 사용하여 브레이크포인트를 포함하는 노드나 와이어를 **하이라이트** 합니다. 기존의 브레이크포인트 위로 커서를 이동시키면, 브레이크포인트 도구 커서 안의 검은 부분이 흰색으로 나타납니다.

브레이크포인트 설정하기

브레이크포인트 도구를 사용하여 VI, 노드, 와이어에 브레이크포인트를 놓고 그 위치에서 실행을 일시 중지할 수 있습니다.

1. 브레이크포인트 도구 를 사용하여 실행을 일시 중지하려는 VI, 노드, 또는 와이어를 클릭합니다. 또한 VI, 노드, 와이어에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 브레이크포인트 >> 브레이크포인트 설정을 선택해도 됩니다.
1. VI를 실행합니다. 실행 도중 브레이크포인트에 도달하면, VI는 멈추고 일시 중지 버튼은 빨간색으로 나타납니다. 또한, VI의 배경과 경계가 깜박이기 시작합니다. 이 경우 다음과 같은 작업이 가능합니다.
 - 단계별 실행 버튼을 사용한 **단계별 실행**.
 - VI 실행 이전에 와이어에 놓은 **프로브**에서 중간값을 확인합니다.
 - **와이어 값 유지** 옵션을 활성화한 경우, VI를 실행한 후에 놓은 프로브에서 중간값 확인.
 - 프런트패널 컨트롤의 값을 변경합니다.
 - **호출 리스트** 폴다운 메뉴를 확인하여 브레이크포인트에서 정지한 VI를 호출하는 VI의 리스트를 볼 수 있습니다.
 - 다음 브레이크포인트 또는 VI가 실행을 마칠 때까지 계속 실행하기 위한 일시 중지 버튼 클릭.

브레이크포인트 비활성화 및 활성화하기

브레이크포인트를 비활성화하여 LabVIEW가 브레이크포인트를 만나도 VI가 계속 실행될 수 있도록 만드려면, 브레이크포인트가 있는 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 브레이크포인트 >> 브레이크포인트 비활성화를 선택합니다. 이전에 비활성화된 브레이크포인트를 활성화하려면, 해당 블록 다이어그램 객체에서 마우스 오

른쪽 버튼을 클릭하고 바로 가기 메뉴에서 브레이크포인트»브레이크포인트 활성화를 선택합니다. 또한 브레이크포인트 관리자 윈도우를 사용하여 브레이크포인트를 개별적으로 또는 한 번에 전체적으로 활성화 및 비활성화할 수 있습니다.

브레이크포인트 제거하기

브레이크포인트 도구를 사용하여 기존 브레이크포인트를 클릭해서 제거합니다. 또한 위치 도구를 사용하여 브레이크포인트에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 브레이크포인트»브레이크포인트 지우기를 선택하여 브레이크포인트를 제거할 수 있습니다. VI 계층구조에서 모든 브레이크포인트를 제거하려면, 편집»계층구조에서 브레이크포인트 제거를 선택합니다. 또한 브레이크포인트 관리자 윈도우를 사용하여 VI 계층구조에 있는 모든 브레이크포인트를 제거할 수 있습니다.

다이나믹하게 호출된 VI의 브레이크포인트 및 [정적 VI 참조] 함수를 사용하여 참조된 VI의 브레이크포인트는 수동으로 제거해야 합니다.

VI 계층구조에 있는 모든 브레이크포인트를 찾고 관리하려면, 보기»브레이크포인트 관리자를 선택합니다. 리스트박스에서 임의의 아이템을 더블 클릭하여 블록 다이어그램의 브레이크포인트를 하이라이트합니다.



노트 사용자가 암호를 모르는 경우에는 암호로 보호된 블록 다이어그램에 있는 브레이크포인트를 볼 수 없지만 브레이크포인트를 활성화, 비활성화 및 제거할 수는 있습니다. VI를 실행한 다음에는, 브레이크포인트를 변경하기 이전 상태로 VI가 되돌아갑니다.

에러 알림 컨트롤하기

예외 컨트롤을 사용하여 일반적인 에러를 에러 아님으로 취급하거나 경고를 에러로 취급합니다.

다음 단계를 따라 기존 에러를 취소하거나 경고를 에러로 상향 조정합니다.

1. 블록 다이어그램에 [일반 에러 핸들러] VI를 추가합니다.
2. [예외 동작] 입력에서 마우스 오른쪽 버튼을 클릭한 후 생성»컨트롤을 선택합니다. 컨트롤 아이콘이 나타납니다.
3. [예외 동작] 컨트롤을 더블 클릭하여 프런트패널을 엽니다. [예외 동작]을 일치시 에러 취소로 설정하여 에러를 취소하거나, 일치시 에러 설정으로 설정하여 경고를 에러로 취급합니다.
4. 블록 다이어그램의 [예외 코드] 입력에서 마우스 오른쪽 버튼을 클릭한 후 생성»컨트롤을 선택합니다. 컨트롤 아이콘이 나타납니다.
5. [예외 코드] 컨트롤을 더블 클릭하여 프런트패널을 엽니다. [예외 코드]를 취소하려는 에러 코드와 일치하도록 설정하거나, 에러로 취급하려는 경고의 에러 입력의 코드 값으로 설정합니다.
6. (옵션) 에러 또는 경고가 발생할 때 일반적으로 나타나는 에러 소스의 정확한 문자열 값을 알고 있으면, 블록 다이어그램의 [예외 소스] 입력에서 마우스 오른쪽 버튼을 클릭한 후 생성»컨트롤을 선택합니다. [예외 소스] 컨트롤을 더블 클릭하여

프론트패널을 엽니다. [예외 소스] 텍스트 박스에 정확한 문자열 값을 입력합니다.

7. 파일 » 다른 이름으로 저장을 선택하여 파일 대화 상자를 디스플레이합니다. VI를 저장하려는 라이브러리를 탐색합니다. 이 VI의 이름을 입력한 후 저장 버튼을 클릭합니다.

사용자 에러 코드 정의하기

LabVIEW는 XML 기반의 텍스트 파일로 내장된 LabVIEW 에러 코드와 드라이버, 툴킷, 모듈에 대한 에러 코드를 설치합니다. LabVIEW에 내장된 에러 코드에는 특정 에러 코드 범위가 주어져 있습니다. 그러나 사용자는 다음 범위에서 사용자 에러 코드를 정의할 수 있습니다:

- -8999에서 -8000
- 5000 ~ 9999
- 500000 ~ 599999

다음의 방법으로 사용자 에러 코드를 정의할 수 있습니다:

- 어플리케이션 전체에서 사용하고, 어플리케이션 또는 공유 라이브러리에서 배포할 여러 개의 에러 코드를 생성하려면, 에러 코드 편집기 대화 상자를 사용합니다.
- 어플리케이션 전체에 걸쳐 사용되지 않거나, 어플리케이션 또는 공유 라이브러리에서 배포하지 않아도 되는 사용자 에러 코드를 생성하려면, [일반 에러 핸들러] VI를 사용합니다.
- 어플리케이션에서 한 개의 특정한 인스턴스에 사용할 한 개의 사용자 에러 코드를 생성하려면, 에러 링을 사용합니다.



관련 정보

[어플리케이션을 통해 배포할 사용자 에러 코드 정의하기](#) (1382 페이지 참조)

[\[일반 에러 핸들러\] VI를 사용하여 사용자 에러 코드 정의하기](#) (1384 페이지 참조)

[에러 링에서 사용자 에러 코드 정의하기](#) (1381 페이지 참조)

에러 링에서 사용자 에러 코드 정의하기

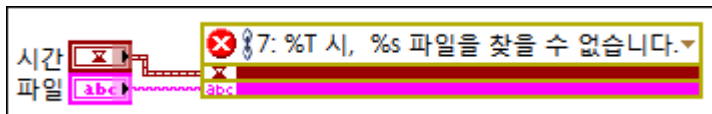
한 개의 에러 코드를 정의하거나 한 개의 내장 에러 코드를 덮어쓰려고 할 때는 사용자 에러 코드를 생성하는 것이 유용합니다. 에러 링에서 여러 사용자 에러를 사용하려면, 에러 코드 편집기 대화 상자를 사용합니다. 다음 단계를 완료하여 에러 링에서 1회만 사용할 사용자 에러를 생성합니다.

1. 블록 다이어그램에 에러 링을 추가합니다.
2. 드롭다운 화살표(▼)를 선택하여 에러 선택 대화 상자를 디스플레이합니다.
3. 에러 코드 범위 드롭다운 리스트박스에서 사용자 에러 코드를 선택합니다.

4. 에러 코드 필드에서 에러 코드를 지정합니다.

- 사용자 에러 메시지를 정의하려면, 사용자 에러 메시징으로 별도로 예약되어 있는 -8999 ~ -8000, 5000 ~ 9999 또는 500,000 ~ 599,999까지의 에러 코드 범위에 해당하는 에러 코드를 입력합니다.
- 이미 존재하는 에러 코드 메시지를 덮어 쓰려면, 기존 에러 코드를 입력합니다. 기존에 정의된 에러 코드에 사용자 에러 메시지를 생성하는 것은, 일반 에러이지만 해당 에러가 발생하는 특정 경우에 관한 정보를 제공하는 에러 코드를 반환하고자 하는 경우에 유용합니다. 예를 들어 에러 코드 7, "파일을 찾을 수 없습니다"를 사용하는 경우, 찾을 수 없는 관련 파일에 대한 설명으로 대체할 수 있습니다.

5. 에러 설명(옵션) 필드에 설명을 입력합니다. 설명 필드는 [문자열로 포맷] 함수에 서와 같이 %s 및 %T 같은 동일한 포맷 지정자를 허용합니다. 예를 들어, 사용자 에러 메시지 "%T 시, %s 파일을 찾을 수 없습니다"의 경우, 설정된 에러 링은 다음 그림과 같이 각각 타임스탬프와 문자열에 대한 입력을 포함하도록 변경됩니다.



6. 라디오 버튼을 사용하여 에러 링이 에러 또는 경고를 반환하도록 설정합니다.
7. 에러 클러스터에 모든 호출 체인 정보를 포함하지 않으려면 호출 체인 포함 확인란에서 확인 표시를 제거합니다. 호출 체인을 포함하는 경우, 에러 설명은 현재 VI에서 최상위 VI까지의 호출 체인을 나타내며, 이렇게 하면 VI 계층구조의 어느 곳에서 에러가 발생했는지 찾는 데 유용합니다. (Real-Time Module) 리얼타임 어플리케이션에서 발생하는 지터를 줄이기 위해 호출 체인을 제외합니다.
8. 확인을 클릭합니다.

에러 링을 설정한 후, 대화 상자로 돌아가지 않고 설정을 조정할 수 있습니다.

- 에러 링의 아이콘 두 개를 클릭하여 각각 에러/경고 셋팅과 호출 체인 포함/제외 셋팅을 토글합니다.
- 에러 링에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 에러 생성, 경고 생성, 호출 체인 포함 또는 호출 체인 제외를 선택하여 관련 셋팅을 설정합니다.
- 에러 링에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 에러 설명 텍스트를 선택하여 에러 설명을 보이거나 숨깁니다.

어플리케이션을 통해 배포할 사용자 에러 코드 정의하기

다음 단계를 따라 에러 코드 편집기 대화 상자를 사용하여 사용자 에러 코드를 정의합니다. 사용자 에러 코드 파일을 생성한 후, 에러 선택 대화 상자에서 에러를 선택하고, 어플리케이션 또는 공유 라이브러리에서 **파일을 배포**할 수 있습니다.

1. 도구 >> 고급 >> 에러 코드 편집을 선택하여 에러 코드 편집기를 시작합니다.
2. 이때 나타나는 입력 요청에서, 새로 만들기 버튼을 클릭하여 새 에러 코드 파일을 생성하거나 기존 버튼을 클릭하여 기존 에러 코드 파일을 탐색합니다.
3. 이 파일에 대한 설명 텍스트 박스에서 에러 코드 파일에 대한 주석을 입력합니다.

4. 에러 코드를 클릭하여 코드를 편집하고, 텍스트 열 아래 공간을 선택하여 해당 에러에 대한 텍스트를 추가합니다. 에러 코드 추가 버튼을 클릭하여 다른 에러 코드를 에러 코드 파일에 추가합니다.
5. 에러 코드와 설명을 편집하려면, 변경하려는 에러 코드나 텍스트를 선택한 후 편집하여 입력합니다. 에러 코드 정렬 버튼을 선택하여 에러 코드를 오름차순으로 정렬합니다. 에러 코드를 삭제하려면, 해당 에러 코드를 선택한 후 선택한 에러 코드 삭제 버튼을 클릭합니다.
6. 편집이 끝나면, 저장 버튼을 클릭하여 LabVIEW\user.lib\errors 디렉토리에 에러 코드 파일을 저장합니다. 에러 코드 파일을 반드시 xxx-errors.txt 포맷으로 저장해야 하며, 이때 xxx는 사용자가 입력하는 이름입니다.

또한, XML을 기반으로 한 텍스트 파일을 생성하여 같은 범위에서 사용자 에러 코드를 정의할 수 있습니다. xxx-errors.txt 형태로 반드시 텍스트 파일을 저장해야 하며, 이때 xxx는 사용자가 입력하는 이름입니다. xxx-errors.txt 파일은 반드시 대문자 사용, 공백 등과 같은 다음 문법을 정확히 지켜야 합니다. 이탤릭체 텍스트를 제공합니다.



노트

사용자가 직접 LabVIEW\user.lib 디렉토리 안에 errors 폴더를 생성해야 사용자 에러 코드 파일들을 정리할 수 있습니다.

```
<?xml version="1.0" encoding="Windows-1252"?> <nidocument>
<nicomment> 이 파일은 내 VI에 대한 사용자 에러를 설명합니다.<nifamily
familyname=" Your Custom Errors " displayname=" Your Custom
Errors "><nifamily><nicomment> <nierror code=" 5000 "> 메모리가 꽉
찼습니다. 더 많은 메모리를 추가합니다.<nierror> <nierror code=" 5001 "> 유
효하지 않은 이름. Enter a new name.<nierror> <nidocument>
```

<nicomment><nicomment> 태그 사이에 고유의 주석을 추가할 수 있습니다. 각각의 <nierror> 태그에서, 에러 코드 숫자를 반드시 정의해야 합니다. <nierror><nierror> 태그 사이에 에러 코드 메시지를 정의합니다. <nifamily><nifamily> 태그는 옵션입니다. 에러 선택 대화 상자를 사용하는 경우, 이 태그 안에 있는 텍스트는 에러 항목으로 나타납니다.

에러 코드 텍스트 파일의 변경은 다음에 LabVIEW를 시작할 때 영향을 줍니다.

텍스트 파일에서 사용자 에러 코드 배포하기

독립 어플리케이션 또는 공유 라이브러리의 일부로 배포하려는 VI가 사용자가 정의한 **XML 기반 텍스트 파일의 에러 코드**를 사용하도록 하려면, 해당 파일을 어플리케이션 또는 공유 라이브러리와 함께 배포해야 에러 핸들링 VI가 그 파일을 사용할 수 있습니다.

사용자 에러 코드를 포함하려는 경우, 에러 코드 파일을 어플리케이션 프로퍼티 대화 상자의 소스 파일 페이지 또는 공유 라이브러리 프로퍼티 대화 상자의 소스 파일 페이지의 항상 포함 리스트박스에 추가합니다.

또한, LabVIEW 에러 코드를 포함시킬 수 있습니다. 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서, 에러 코드 파일 복사 확인란에 확인 표시를 하여 LabVIEW 에러 코드 파일을 어플리케이션에 포함합니다. 공유 라이브러리 프로퍼티 대화 상자의 고급 페이지

에서, 에러 코드 파일 복사 확인란에 확인 표시를 하여 LabVIEW 에러 코드 파일을 공유 라이브러리에 포함합니다.

어플리케이션 빌더를 사용하여 설치 프로그램을 생성하는 경우, 설치 프로그램 프로퍼티 대화 상자의 고급 페이지에서 사용자 에러 코드 파일 설치 확인란에 확인 표시를 하여 LabVIEW 에러 코드 파일을 포함시킬 수 있습니다. 설치 프로그램 빌드는 `labview\project\errors`와 `labview\user.lib\errors`의 모든 에러 코드 파일을 포함하며 이 파일을 `Shared\LabVIEW Run-Time\x.x\errors` 디렉토리에 설치합니다. 여기서, `x.x`는 LabVIEW 버전입니다.



노트

사용자가 직접 `LabVIEW\user.lib` 디렉토리 안에 `errors` 폴더를 생성해야 사용자 에러 코드 파일들을 정리할 수 있습니다.

[일반 에러 핸들러] VI를 사용하여 사용자 에러 코드 정의하기

다음 단계를 따라 [일반 에러 핸들러] VI를 사용하여 사용자 에러 코드를 정의합니다.

1. 블록 다이어그램에 [일반 에러 핸들러] VI를 추가합니다.
2. [사용자 정의 코드] 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 상수를 선택합니다. 배열이 나타납니다.
3. 숫자형 상수를 더블 클릭하고 배열에 -8999에서 -8000, 5000에서 9999 또는 500,000에서 599,999의 범위에 있는 숫자를 입력합니다. 예를 들면, 5008을 입력합니다.
4. [사용자 정의 설명] 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 상수를 선택합니다. 다른 배열이 나타납니다.
5. 문자열 상수를 더블 클릭하고 사용자 정의 설명 배열에 설명을 입력합니다. 예를 들면, 이 메시지를 무시하십시오을 입력합니다.
6. 에러 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 상수를 선택합니다. 불리언 상수와 숫자형 상수, 배열을 포함하는 클러스터가 나타납니다.
7. 수행 도구를 사용하여 불리언 컨트롤을 참으로 만듭니다.
8. 숫자형 상수를 더블 클릭한 후 [사용자 정의 코드] 상수에 있는 것과 같은 숫자를 입력합니다.



노트

6에서 8까지의 단계를 대체하는 방법으로, [에러 코드를 에러 클러스터로] VI에 특정 에러 코드가 있는 숫자형 상수 값을 연결할 수 있습니다. [에러 코드를 에러 클러스터로] VI의 에러 출력을 [일반 에러 핸들러] VI의 에러 입력에 연결합니다.

1. VI를 실행합니다. 에러 대화 상자가 나타나고 사용자 정의 에러 코드 및 관련 설명을 디스플레이합니다. 이 에러 대화 상자에서 계속 버튼을 클릭합니다.
2. 일반 에러 핸들러 SubVI를 더블 클릭하고 프런트패널을 봅니다. 사용자 에러 코드로 정의한 VI로 전환하고 이 VI를 다시 실행합니다. 계속 버튼을 다시 클릭하여 [일

반 에러 핸들러] VI의 프론트패널에 있는 메시지 텍스트 박스에도 이 에러 대화 상자와 똑같은 사용자 에러 코드 및 설명이 디스플레이되는 것을 확인하십시오.

또한, **XML 기반의 텍스트 파일 생성하기**로 사용자 에러 코드를 정의할 수 있으며 텍스트 파일에 에러 코드와 메시지를 추가할 수 있습니다.

에러 코드 설명 찾기

LabVIEW는 XML 기반의 텍스트 파일로 내장된 LabVIEW 에러 코드와 드라이버, 툴킷, 모듈에 대한 에러 코드를 설치합니다.

[일반 에러 핸들러] 또는 [단순 에러 핸들러] VI를 사용하여 에러 코드의 설명을 볼 수 있습니다.

- 일반 에러 핸들러
- 단순 에러 핸들러

또한, 도움말 >> 에러 설명을 선택하여 에러 설명 대화 상자에 에러 코드에 대한 설명을 디스플레이할 수 있습니다. 코드 필드 또는 (에러 코드 숫자가 16진수 포맷인 경우) 16진 코드 필드에 에러 코드 숫자를 입력하십시오. <Enter> 키를 누르거나 대화 상자에서 클릭하여 설명 필드에 에러 코드에 대한 설명을 디스플레이합니다. 이 설명은 이 도움말 파일에 나타나는 것과 같은 설명입니다. 또한 **에러 클러스터 리포트를 생성**하여 내장된 에러 코드에 대한 설명을 볼 수 있습니다.

VI가 설명을 반환하지 않는다면, 에러 코드 파일은 이동, 삭제, 손상되었을 것입니다. 이 문제는 보통 **사용자 정의 에러 코드**, 어플리케이션, 또는 공유 라이브러리, 또는 LabVIEW 6.1 이전에 배포된 드라이버를 사용할 때 발생합니다.

정의되지 않은 에러 코드

정의되지 않은 에러 코드는 많은 이유들 때문에 발생할 수 있습니다. 다음 보기는 정의되지 않은 에러를 식별하고 처리하기 위한 제안입니다.

- 정의되지 않은 에러는 에러 코드에 대한 설명이 없을 때 발생할 수 있습니다. 직접 **사용자 에러 코드를 정의 및 생성** 하실 수 있습니다. National Instruments의 정의되지 않은 에러 코드와 관련해 기술 지원을 문의하실 수 있습니다.
- 사용자 정의 코드는 에러코드가 아닌 번호를 에러 입력에 와이어 했을 때 발생할 수도 있습니다.
- 추가적으로, 에러가 OS 또는 **ActiveX**와 같은 타사에서 제공하는 객체에 관련되어 있을 때 정의되지 않은 에러가 발생할 수 있습니다. 타사의 객체에서 발생한 에러일 경우 정의되지 않은 에러코드의 설명을 웹에서 검색하실 수 있습니다. 숫자형 에러코드나 16진수에 대한 에러코드는 그에 해당하는 형의 숫자나 16진수로 검색

하실 수 있습니다. 예를 들면, 에러코드 12341234나 **16진수 디스플레이**인 0xBEEDF00F로 에러코드를 검색하실 수 있습니다.



팁

ActiveX 에러 코드는 Microsoft의 winerror.h에 문서화되어 있습니다.

어플리케이션 또는 공유 라이브러리

어플리케이션 또는 공유 라이브러리를 개발하려면, 반드시 어플리케이션 또는 공유 라이브러리에서 **관련된 모든 에러 코드 텍스트 파일을 배포**해야 합니다.

이전 또는 타사 드라이버

LabVIEW 6.1 이전에 배포된 드라이버를 사용하려면, 모든 드라이버 에러 코드를 포함하고 있는 error.11b의 버전을 설치합니다. 드라이버 에러 코드를 설명하기 위해 드라이버는 에러 코드 텍스트 파일을 설치하지 않습니다.

LabVIEW 6.1 설치 이후에 이전 드라이버를 설치하면, 반드시 LabVIEW 6.1 또는 LabVIEW 7.0에서 설치된 error.11b의 버전 대신에 드라이버의 error.11b 버전으로 대체해야 합니다.

에러 클러스터 리포트 생성하기

다음 단계를 따라 에러 클러스터가 에러를 어떻게 리포트하는지를 살펴봅니다.

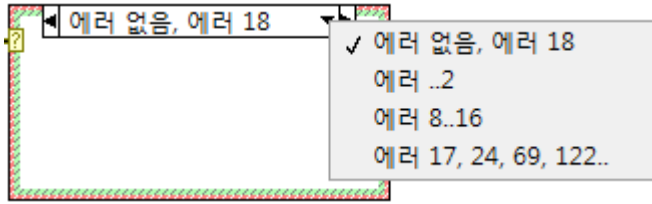
1. 에러 입력 또는 에러 출력 프런트패널 컨트롤의 상태 버튼을 클릭합니다. 상태 버튼이 녹색 확인 표시에서 붉은색 X로 바뀝니다.
2. 상태 버튼 또는 클러스터 경계에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 에러 설명을 선택합니다. 에러 설명 대화 상자는 에러에 대한 정보를 포함합니다. VI가 에러가 아닌 **경고**를 포함하는 경우, 바로 가기 메뉴는 경고 설명 옵션을 포함합니다.

케이스 구조로 에러 핸들링하기

다음 단계를 따라 **케이스** 구조를 사용하여 에러를 핸들링합니다.

1. 에러 클러스터 터미널을 선택합니다.
2. 에러 클러스터를 케이스 구조의 선택자 터미널에 와이어로 연결합니다. 케이스 선택자 라벨은 두 케이스를 모두 디스플레이하고(에러 및 에러 없음), 케이스 구조의 경계는 케이스에 따라 색이 바뀝니다(에러의 경우 빨강, 에러 없음의 경우 녹색). 케이스 구조는 모든 에러의 에러 서브다이어그램을 실행합니다.
3. (옵션) 케이스 구조가 특정 에러 또는 에러 리스트에 대해 특정 서브다이어그램을 실행하도록 설정하려면 케이스를 더 추가하고 **선택자 라벨에 에러 값이나 범위를**

입력합니다. 에러 없음 케이스에도 에러 값을 추가할 수 있습니다. 선택자 라벨이 에러 코드를 포함하는 경우 에러 없음 케이스의 경계는 녹색과 빨간색입니다.



노트

에러 케이스의 선택자 라벨을 사용자 정의하지 않고 에러 케이스를 더 추가하면 에러 케이스가 기본으로 변경되고, 이는 기존 에러 케이스로 처리되지 않는 나머지 모든 에러를 처리합니다.

4. (옵션) 구조 경계에서 마우스 오른쪽 버튼을 클릭한 후 기수를 선택하여 선택자 라벨에서 에러 코드의 디스플레이 포맷을 설정합니다. 디스플레이 포맷이 10진수이면, 포맷을 나타내는 문양이 선택자 라벨의 왼쪽에 나타납니다.
5. 에러 케이스에 코드를 추가하여 여러 다른 케이스에 대해 동작을 정의합니다.

루프의 조건 터미널로 에러 핸들링하기

에러가 조건 터미널로 전달되면 While 루프나 For 루프의 반복을 컨트롤할 수 있습니다. 다음 단계에 따라 조건 터미널을 사용하여 에러를 핸들링합니다.

1. 에러 클러스터 터미널을 선택합니다.
2. While 루프나 For 루프의 조건 터미널에 에러 클러스터를 와이어로 연결합니다.



노트

에러 클러스터를 조건 터미널에 연결하면 에러 클러스터 상태 파라미터의 참(TRUE) 또는 거짓(FALSE) 값만 터미널에 전달됩니다. 그리고 바로 가기 메뉴 아이템도 참인 경우 정지, 참인 경우 계속에서 에러 발생 시 정지, 에러 발생 시 계속으로 변경됩니다.

에러 링에서 내장 에러 선택하기

다음 단계를 완료하여 에러 링에서 내장 에러를 선택합니다.

1. 블록 다이어그램에 에러 링을 추가합니다.
2. 드롭다운 화살표(▼)를 선택하여 에러 선택 대화 상자를 디스플레이합니다.
3. 에러 코드 범위 드롭다운 리스트박스에서, 사용하고자 하는 에러 코드 항목을 지정합니다. 또한 이 대화 상자에서 1회만 사용할 수 있는 **사용자 에러 메시지를 생성**할 수도 있습니다.
4. 리스트가 나타나면 에러를 선택합니다.
5. 라디오 버튼을 사용하여 에러 링이 에러 또는 경고를 반환하도록 설정합니다.

6. 에러 클러스터에 모든 호출 체인 정보를 포함하지 않으려면 호출 체인 포함 확인란에서 확인 표시를 제거합니다. 호출 체인을 포함하는 경우, 에러 설명은 현재 VI에서 최상위 VI까지의 호출 체인을 나타내며, 이렇게 하면 VI 계층구조의 어느 곳에서 에러가 발생했는지 찾는 데 유용합니다. (Real-Time Module) 리얼타임 어플리케이션에서 발생하는 지터를 줄이기 위해 호출 체인을 제외합니다.
7. 확인을 클릭합니다.

에러 링을 설정한 후, 대화 상자로 돌아가지 않고 설정을 조정할 수 있습니다.

- 에러 링의 아이콘 두 개를 클릭하여 각각 에러/경고 셋팅과 호출 체인 포함/제외 셋팅을 토글합니다.
- 에러 링에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 에러 생성, 경고 생성, 호출 체인 포함 또는 호출 체인 제외를 선택하여 관련 셋팅을 설정합니다.
- 에러 링에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템 >> 에러 설명 텍스트를 선택하여 에러 설명을 보이거나 숨깁니다.

VI에서 에러 I/O 설정하기

대부분의 VI와 함수는 에러 I/O 정보를 제공합니다. 이 정보는 런타임에 발생하는 블록 다이어그램 상의 에러의 위치를 식별하는 데 유용합니다.

다음 단계를 따라 VI에서 에러 I/O를 설정합니다.

1. SubVI 또는 함수의 에러 입력 터미널에서 오른쪽 클릭하고 바로 가기 메뉴에서 생성 >> 컨트롤을 선택하여 에러 클러스터를 생성합니다. 배열, 행렬 & 클러스터 및 클래식 배열, 행렬 & 클러스터 팔레트에서 에러 클러스터 컨트롤을 찾을 수도 있습니다.
2. 첫번째 SubVI 또는 함수에서, 에러 출력을 다음 SubVI 또는 함수의 에러 입력에 연결합니다.
3. 데이터 흐름의 마지막 subVI 또는 함수에 도달할 때까지 이어지는 subVI와 함수에 대해 2번째 단계를 반복합니다.
4. 마지막 함수 또는 SubVI에서는, 에러 출력 파라미터에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성 >> 인디케이터를 선택하여 에러 인디케이터를 생성합니다.
5. VI의 **커넥터 팬에서**, 에러 입력 컨트롤을 터미널에 연결하고 에러 출력 인디케이터를 다른 터미널에 연결합니다.



노트

커넥터 팬의 왼쪽 아래 터미널에 에러 입력을, 오른쪽 아래 터미널에 에러 출력을 연결하는 것이 LabVIEW 표준입니다.

일부 VI와 함수는 에러 클러스터 대신에 숫자 에러 코드와 함께 에러를 반환합니다. VI가 에러 또는 에러 코드 출력을 에러 출력 대신에 사용하는 경우, 이 VI에 [에러 핸들러] VI를 연결하십시오.

에러 클러스터 사용하기

사용하거나 만드는 각 VI에서 에러 입력과 에러 출력 클러스터를 사용하여 VI를 통해 에러 정보를 전달합니다. 대부분의 경우, 소스 클러스터 원소는 또한 어디서 에러가 발생했는지 식별합니다. 에러 입력이 에러를 감지하는 경우, VI는 에러 출력에 에러 정보를 반환하고 계속 실행되지 않습니다. 기본적으로 LabVIEW는 VI를 실행할 때 에러가 발생하면 실행을 정지하고, 에러가 발생한 SubVI 또는 함수를 하이라이트하고, 에러 대화 상자를 열어 자동으로 에러를 처리합니다. **자동 에러 핸들링을 비활성화**할 수 있습니다.

다음 단계를 따라 VI에서 에러 클러스터를 사용하는 방법의 예제를 보십시오.

1. 블록다이어그램에 [리포트 생성] VI를 추가한 후에 더블 클릭하여 엽니다.
2. 윈도우 » 블록다이어그램 보이기를 선택하여 블록다이어그램을 디스플레이합니다.
3. 기본 도움말 윈도우를 사용하거나 **VI 계층구조** 윈도우를 사용하여 리포트 생성 생성 VI를 찾습니다.
4. 리포트 생성 생성 VI를 더블 클릭하여 VI를 엽니다. 프런트패널에는 에러 입력 및 에러 출력 클러스터가 포함됩니다. 블록다이어그램에서, 이 클러스터는 케이스 구조에 외부적으로 연결됩니다.

에러가 발생하는 경우, 케이스 구조는 이 VI에 도달하기 전에 실행됩니다. VI는 주요 코드를 실행하지 않고 대신 에러 정보를 직접 케이스 구조를 통하여 에러 출력 클러스터로 전달합니다. 에러가 발생하지 않는 경우, VI는 새 리포트를 생성합니다.

어플리케이션과 공유 라이브러리 디버그하기

어플리케이션 빌더로 생성한 독립 어플리케이션과 공유 라이브러리를 디버그할 수 있습니다.



노트

LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. LabVIEW Base Development System 또는 Full Development System을 사용하는 경우, National Instruments 웹사이트를 방문하여 어플리케이션 빌더를 별도로 구매할 수 있습니다.

다음 설명을 따라 독립 어플리케이션 또는 공유 라이브러리를 디버그합니다.



노트 어플리케이션과 공유 라이브러리를 디버그할 때 [VI 참조 열기] 함수가 생성하는 재호출 패널 또는 LabVIEW에서 빌드된 공유 라이브러리에 대한 입력 포인트인 재호출 패널을 디버그할 수 없습니다. 또한 다이어그램 비활성화 구조 안에 있는 SubVI를 디버그할 수 없습니다. 사용자가 어플리케이션을 디버그하는 동안 다이어그램 비활성화 구조 안에 있는 SubVI는 물음표로 나타납니다. 하지만 다이어그램 비활성화 구조 안에 있는 코드가 실행되지 않기 때문에, 이는 디버깅에 영향을 미치지 않습니다.

1. 어플리케이션 또는 공유 라이브러리를 만들기 전에, 빌드 스펙에서 디버깅을 활성화해야 합니다. 어플리케이션의 경우, 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서 디버깅 활성화 확인란에 확인 표시를 하십시오. 공유 라이브러리의 경우, 공유 라이브러리 프로퍼티 대화 상자의 고급 페이지에서 디버깅 활성화 확인란에 확인 표시를 하십시오. 디버깅을 활성화하면 VI의 블록 다이어그램을 빌드 안에 보존하여 디버깅을 수행할 수 있습니다.
2. 디버깅 도구를 실행할 때까지 어플리케이션 또는 공유 라이브러리가 실행을 대기하도록 하려는 경우, 시작할 때 디버거 기다림 확인란에 확인 표시를 합니다.
3. 어플리케이션 또는 공유 라이브러리를 만듭니다.
4. 빌드된 어플리케이션을 실행하거나 LabVIEW가 아닌 다른 프로그램으로 생성한 공유 라이브러리를 호출합니다.
5. 프로젝트 탐색기 윈도우에서, 수행 >> 어플리케이션 또는 공유 라이브러리 디버거를 선택하여 어플리케이션 또는 공유 라이브러리 디버거 대화 상자를 디스플레이합니다. 대화 상자는 디버깅이 활성화된 어플리케이션과 공유 라이브러리의 리스트를 디스플레이합니다.
6. 디버깅하려는 어플리케이션 또는 공유 라이브러리가 다른 컴퓨터에서 실행 중인 경우, 컴퓨터 이름 또는 IP 주소 텍스트 박스에 컴퓨터 이름을 입력합니다. 새로 고침 버튼을 클릭하여 원격 컴퓨터에서 디버깅이 활성화된 어플리케이션 및 공유 라이브러리의 목록을 확인합니다.
7. 디버깅하려는 어플리케이션 또는 공유 라이브러리를 선택합니다.
8. 연결 버튼을 클릭하여 디버깅을 수행합니다. LabVIEW가 어플리케이션 또는 공유 라이브러리를 다운로드하고, 디버깅을 시작할 수 있도록 시작 VI의 프런트패널이 나타납니다. 시작할 때 디버거 기다림을 활성화하는 경우, 어플리케이션이나 공유 라이브러리를 실행하려면 실행 버튼을 클릭해야 합니다.
9. 시작 VI의 블록 다이어그램을 사용하여 어플리케이션 또는 공유 라이브러리를 디버깅합니다. 문제를 찾아내기 위해 프로브, 브레이크포인트, 기타 **디버깅 기술**을 사용할 수 있습니다. 이후 디버깅 과정에서 찾은 문제를 정정할 수 있습니다.



노트 디버깅하려는 어플리케이션 또는 공유 라이브러리로 만들어지지 않은 공유 라이브러리에서 라이브러리 함수 호출 노드를 사용하는 사용자 프로브를 사용하여 디버깅하는 경우, 공유 라이브러리를 디버깅하려는 어플리케이션 또는 공유 라이브러리와 같은 디렉토리에 놓아야 합니다.

1. 디버깅이 끝나면 시작 VI를 닫습니다. 시작 VI를 닫으면 원격으로 컨트롤되는 어플리케이션 또는 공유 라이브러리도 닫힙니다. 시작 VI를 닫지 않고 원격으로 컨트롤되는 어플리케이션 또는 공유 라이브러리를 닫으려는 경우, 시작 VI에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 원격 디버깅 >> 디버깅 세션 종료를 선택합니다.

다음과 같은 경우 디버깅 가능한 어플리케이션 또는 런타임 라이브러리가 발견되지 않았습니다 에러가 발생할 수 있습니다.

- 디버깅 가능한 어플리케이션이나 공유 라이브러리가 로드되지 않았거나, 로드되어 있던 디버깅 가능 공유 라이브러리가 어플리케이션에서 취소되었습니다.
- 어플리케이션 빌더에 의해 생성된 공유 라이브러리 또는 어플리케이션의 설정 ini 파일이 공유 라이브러리나 어플리케이션과 함께 배포되지 않았습니다.

어플리케이션 만들기 및 배포하기 홈

빌드 과정 문제 해결하기

다음에 따라 LabVIEW에서 **어플리케이션을 만들고 배포**할 때 일반적으로 발생할 수 있는 2개의 문제를 수정하십시오.

프로젝트에 있는 VI 중 하나의 옆에 [충돌]이 나타나서, 프로젝트로 어플리케이션을 빌드할 수 없습니다.

프로젝트에 충돌 사항이 있으면 어플리케이션 빌더가 어플리케이션 만들기를 허용하지 않습니다. LabVIEW 프로젝트의 충돌은 LabVIEW가 이미 프로젝트에 있는 아이템과 정규화된 이름이 같은 VI를 로드하려할 때 발생하는 잠재적 **상호 링크**입니다. 충돌을 수정하려면, 프로젝트 >> 충돌 해결을 선택합니다.

어플리케이션을 만드는 중 빌드 과정에 실패했지만, 원인을 식별하거나 문제를 수정하는 법을 알 수 없습니다.

다음의 두 가지 방법을 사용하여 이 문제를 해결할 수 있습니다. 일부 상황에서는 두 방법을 모두 사용해야 할 필요가 있습니다.

1. VI를 다시 컴파일합니다.
 1. **VI 서버**, 참조에 의한 호출 노드 또는 비동기화 호출 시작 노드를 사용하여,ダイナ믹하게 로드하려는 VI와 빌드에 사용하려는 최상위 VI 중 하나를 엽니다.
 2. 열린 VI에서, 실행 버튼을 클릭하는 동안 <Ctrl-Shift> 키를 눌러서 메모리의 모든 VI를 다시 컴파일합니다. (macOS) <Option-Shift> 키를 누릅니다. (Linux) <Meta-Shift> 키를 누르십시오.
 3. 파일 >> 모두 저장을 선택하여 메모리의 모든 VI를 저장합니다.
2. 어플리케이션 빌더 캐시를 지웁니다.
3. 어플리케이션의 빌드 스펙 이름에서 마우스 오른쪽 버튼을 클릭한 후 빌드를 선택하여 어플리케이션을 다시 만듭니다.

LabVIEW에서 문서화하기

VI 문서화하기

LabVIEW를 사용하여 완성된 VI를 문서화하고 VI의 사용자를 위한 설명을 생성할 수 있습니다.

문서는 LabVIEW에서 보거나 인쇄하거나, HTML, RTF, text 파일로 저장하여 볼 수 있습니다. HTML과 RTF 파일은 대부분의 워드 프로세싱 어플리케이션으로 반입할 수 있으며, 컴파일된 도움말 파일을 생성할 때 사용할 수 있습니다.

VI에 효과적인 문서를 생성하기 위해서, 객체 설명과 VI 설명을 생성합니다.



관련 정보

- [VI, 템플릿, 객체를 위한 문서 인쇄하기](#) (1399 페이지 참조)
- [VI, 템플릿, 객체를 위한 HTML 문서 생성하기](#) (1394 페이지 참조)
- [VI, 템플릿, 객체를 위한 RTF 문서 생성하기](#) (1395 페이지 참조)
- [VI, 템플릿, 객체를 위한 텍스트 문서 생성하기](#) (1395 페이지 참조)
- [컴파일된 도움말 파일 생성하기](#) (1396 페이지 참조)
- [VI 설명 생성하기와 편집하기](#) (1392 페이지 참조)
- [객체 설명과 팁 상자 생성하기](#) (1393 페이지 참조)

북마크 관리자 생성하기

LabVIEW는 labview\resource\dialog\BookmarkManager\managers 디렉토리에 북마크 관리자를 설치합니다. LabVIEW와 함께 설치되는 북마크 관리자는 매우 기본적인 것이며 사용자의 모든 요구 사항 또는 용도를 모두 충족시키지 못할 수 있습니다. 따라서, 사용자는 어플리케이션의 필요에 따라 고유한 북마크 관리자를 생성할 수 있습니다. 다음 단계를 따라 북마크 관리자를 생성합니다.

1. labview\resource\dialog\BookmarkManager\managers 디렉토리에 있는 Default 폴더를 복사합니다. 이 소스를 북마크 관리자를 개발하는 출발점으로 사용합니다.
2. 해당 폴더를 동일한 디렉토리에 붙여 넣고 Default라는 이름을 다른 이름으로 변경합니다.
3. managers 디렉토리의 최상위 레벨에서 Default.txt 파일을 찾아서, 폴더의 새 이름과 일치하도록 이름을 변경합니다.
4. *.txt 파일의 복사본을 열어서 Default라는 단어를 새 북마크 관리자의 이름으로 변경합니다.
5. 사용자의 필요에 맞도록 새 폴더에서 파일을 편집합니다.

managers 디렉토리에 하나 이상의 폴더 및 텍스트 파일이 있는 경우, LabVIEW는 북마크 관리자 윈도우가 시작될 때 원하는 관리자를 선택하도록 입력 요청합니다. 다시 보이지 않음 확인란에 확인 표시를 한 경우, LabVIEW는 마지막으로 선택한 관리자를 실행합니다. 새 북마크 관리자를 추가하는 경우, LabVIEW는 확인란 셋팅을 되돌리고 다시 북마크 관리자 선택 대화 상자를 디스플레이합니다.

VI 설명 생성하기와 편집하기

VI의 목적을 설명하고 VI의 사용법을 알리기 위해 VI에 대한 설명을 생성합니다. 커서를 VI 아이콘과 생성한 VI 문서 위로 움직일 때 기본 도움말 윈도우에 VI 설명이 나타납니다.

다음 단계를 따라 VI 설명을 생성하고 편집합니다.

1. 파일>>VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 문서를 선택합니다.
3. VI 설명 텍스트 박스에서 VI에 대한 설명을 입력하거나 편집합니다. 설명의 **텍스트를 포맷**하여 기본 도움말 윈도우 내에서 굵은체로 보이도록 할 수 있습니다.
4. (옵션) HTML 파일 또는 컴파일된 도움말 파일을 **VI와 연결**합니다.
5. 확인 버튼을 클릭하여 변경을 저장합니다.



노트

호출하는 VI 블록 다이어그램에서 SubVI에 대한 설명을 편집할 수 없습니다. SubVI에 대한 설명을 편집하려면, **SubVI를 열고**, SubVI에서 파일>>VI 프로퍼티를 선택한 후, 항목 풀다운 메뉴에서 문서를 선택합니다.

또한, VI 설명 프로퍼티를 사용하여 VI 설명을 프로그램적으로 생성하고 편집할 수 있습니다.

객체 설명과 팁 상자 생성하기

컨트롤과 인디케이터 같은 객체의 설명을 생성하는 것은 객체의 목적을 설명하고 객체의 사용에 대한 지침을 사용자에게 주기 위한 것입니다. 커서를 객체와 생성한 VI 문서쪽으로 움직일 때 기본 도움말 윈도우에 나타나는 객체의 설명입니다. 팁 상자는 실행 중인 VI에서 마우스가 객체 위에 위치했을 때 나타나는 간략한 설명문입니다.

다음 단계를 따라 객체에 대한 설명과 팁 상자를 입력합니다.

1. 객체를 프런트패널이나 블록 다이어그램에 추가합니다.
2. **객체에 라벨**을 붙입니다.
3. 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 설명과 팁을 선택하여 설명과 팁 대화 상자를 디스플레이합니다.
4. 설명 텍스트 박스에 객체의 설명을 입력합니다. 설명 텍스트 박스는 객체에 입력한 라벨을 사용합니다. 작업에서 팁 상자를 위해 설명 정보를 입력할 수 없습니다. 설명에 **굵은 포맷 추가**를 할 수 있습니다.



노트

함수 팔레트에 위치하는 VI 또는 함수에 대한 설명을 입력할 수 있지만 설명과 팁 대화 상자의 설명을 볼 수만 있습니다. 이 설명은 기본 도움말 윈도우에서는 보이지 않습니다.

1. 객체에서 팁 상자는 팁 텍스트 박스에 최대 255개의 문자까지 입력할 수 있습니다. 팁 텍스트 박스를 사용하여 객체에 입력한 라벨에 사용하는 것에 주의합니다.



노트 팁 텍스트 박스는 컨트롤, 인디케이터, 상수에서만 이용 가능합니다. 팁 정보를 VI나 함수에 입력할 수 없습니다.

1. 팁 상자와 설명을 저장하기 위해 확인 버튼을 클릭합니다.

또한, 설명과 팁 상자 프로퍼티를 사용하여 설명과 팁 정보를 프로그램적으로 변경할 수 있습니다.

기본 도움말 윈도우의 텍스트 포맷하기

기본 도움말 윈도우의 텍스트를 포맷하여 VI 설명, 팔레트 설명, 컨트롤과 인디케이터 설명이 굵은체로 나타나도록 할 수 있습니다. 설명에서 굵은체로 포맷하려는 텍스트 주변에 와 태그를 사용합니다.

캐리지 리턴을 기본 도움말 윈도우에 디스플레이하려는 경우, 반드시 두 개의 캐리지 리턴으로 문장을 분리해야만 합니다.



관련 정보

[VI 설명 생성하기와 편집하기](#) (1392 페이지 참조)

[객체 설명과 팁 상자 생성하기](#) (1393 페이지 참조)

VI, 템플릿, 객체를 위한 HTML 문서 생성하기

HTML 파일은 대부분의 워드 프로세싱 어플리케이션으로 반입할 수 있으며, [컴파일된 도움말 파일을 생성](#)할 때 사용할 수 있습니다.

다음 단계를 따라 VI, 템플릿, 또는 객체를 위한 HTML 문서를 생성합니다.

1. VI, 컨트롤, 글로벌 변수, 또는 템플릿 파일을 열고 파일>>인쇄를 선택하여 인쇄 대화 상자를 디스플레이합니다.
2. 인쇄 대화 상자의 VI 선택 페이지에서, 현재 VI에 대한 HTML 문서를 생성할 것인지 아니면 여러 VI에 대한 HTML 문서를 생성할 것인지 선택하고 다음 버튼을 클릭합니다. 여러 VI를 선택하는 경우, VI 선택 페이지에서 HTML 문서를 생성하려는 파일을 선택하고 다음 버튼을 클릭합니다.
3. 인쇄 내용 페이지에서 문서에 포함하려는 구성요소를 선택하고 다음 버튼을 클릭합니다. VI 문서를 선택한 경우, VI 문서 페이지에서 포함하려는 구성요소를 선택하고 다음 버튼을 클릭합니다.
4. 대상 페이지에서 HTML 파일을 선택하고 다음 버튼을 클릭하여 HTML 페이지를 디스플레이합니다.
5. 이미지를 위한 옵션을 설정합니다.
6. 저장 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다. 대화 상자는 HTML과 [이미지 파일](#)의 이름, 그리고 저장하려고 하는 파일의 디렉토리 선택을 요청합니다. 모든 HTML 문서의 기본 이름은 VI와 같은 이름을 가지는 .html 확장자입니다.

또한 [VI 패널 또는 문서 인쇄] VI를 사용하거나 인쇄:VI를 HTML로 메소드를 사용하여 프로그램적으로 HTML 포맷의 VI 문서를 생성할 수 있습니다.

(Statechart) Statechart에 대한 HTML 문서를 생성할 수 없습니다. 그러나 Statechart에 대한 XML 문서를 생성하여 이 XML 파일을 웹 브라우저에서 볼 수 있습니다.

VI, 템플릿, 객체를 위한 RTF 문서 생성하기

서식 있는 텍스트 포맷(RTF) 파일은 대부분의 워드 프로세싱 프로그램에서 읽을 수 있으며, **컴파일된 도움말 파일을 생성**할 때 사용할 수 있습니다.

다음 단계를 따라 VI, 템플릿, 또는 객체를 위한 RTF 문서를 생성합니다.

1. VI, 컨트롤, 글로벌 변수, 또는 템플릿 파일을 열고 **파일 >> 인쇄**를 선택하여 인쇄 대화 상자를 디스플레이합니다.
2. 인쇄 대화 상자의 VI 선택 페이지에서, 현재 VI에 대한 HTML 문서를 생성할 것인지 아니면 여러 VI에 대한 RTF 문서를 생성할 것인지 선택하고 다음 버튼을 클릭합니다. 여러 VI를 선택하는 경우, VI 선택 페이지에서 RTF 문서를 생성하려는 파일을 선택하고 다음 버튼을 클릭합니다. 그렇지 않다면, 다음 단계로 계속 갑니다.
3. 인쇄 내용 페이지에서 문서에 포함하려는 구성요소를 선택하고 다음 버튼을 클릭합니다. VI 문서를 선택한 경우, VI 문서 페이지에서 문서에 포함하려는 구성요소를 선택하고 다음 버튼을 클릭합니다. 그렇지 않다면, 다음 단계로 계속 갑니다.
4. 대상 페이지에서 리치 텍스트 포맷(RTF) 파일을 선택하고 다음 버튼을 클릭합니다.
5. 컴파일된 도움말 파일에 RTF 문서를 사용하려면 RTF 페이지에서 도움말 컴파일러 소스 확인란에 확인 표시를 합니다. 이 확인란에 확인 표시를 하면, LabVIEW는 RTF 파일에 내장되어 있는 것 대신에 **외부 BMP 파일** 이미지로 저장합니다.
6. 이미지의 색 농도를 선택합니다.
7. 저장 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다. 대화 상자는 파일의 이름, 그리고 저장하려고 하는 파일의 디렉토리 선택을 요청합니다. 모든 RTF 문서의 기본 이름은 VI와 같은 이름을 가지는 .rtf 확장자입니다.

또한, 인쇄:VI를 RTF로 메소드를 이용하여 RTF 포맷 안의 VI 문서를 프로그램적으로 생성할 수 있습니다.

VI, 템플릿, 객체를 위한 텍스트 문서 생성하기

다음 단계를 따라 VI, 템플릿, 또는 객체를 위한 텍스트 문서를 생성합니다.



노트

이 메소드는 VI의 설명과 개정 히스토리와 같은 문서의 텍스트 부분만 만듭니다. 그래픽을 포함하지 않습니다.

1. VI, 컨트롤, 글로벌 변수, 또는 템플릿 파일을 열고 **파일 >> 인쇄**를 선택하여 인쇄 대화 상자를 디스플레이합니다.
2. 인쇄 대화 상자의 VI 선택 페이지에서, 현재 VI에 대한 텍스트 문서를 생성할 것인지 아니면 여러 VI에 대한 텍스트 문서를 생성할 것인지 선택합니다. 여러 VI를 선택

택한 경우, 다음 버튼을 클릭하여 VI 선택 페이지를 디스플레이하고 생성하려는 텍스트 문서에 필요한 파일을 선택합니다.

3. 다음 버튼을 클릭하여 인쇄 내용 페이지를 디스플레이합니다.
4. 문서에 포함할 구성요소를 선택합니다. VI 문서를 선택한 경우, 다음 버튼을 클릭하여 VI 문서 페이지를 디스플레이하고 문서에 포함하려는 구성요소를 선택합니다.
5. 다음 버튼을 클릭하여 대상 페이지를 디스플레이합니다.
6. 일반 텍스트 파일을 선택하고 다음 버튼을 클릭하여 일반 텍스트 페이지를 디스플레이합니다.
7. 라인당 텍스트의 한계 확인란에 확인표시를 하여 텍스트의 각 줄에 표시될 캐릭터의 수를 지정합니다.
8. 저장 버튼을 클릭하여 파일 대화 상자를 디스플레이합니다. 대화 상자는 텍스트 파일의 이름, 그리고 저장하려고 하는 파일의 디렉토리 선택을 요청합니다. 모든 텍스트 문서의 기본 이름은 VI와 같은 이름을 가지는 .txt 확장자입니다.

또한, 인쇄:VI를 텍스트로 메소드를 이용하여 텍스트 포맷에서 VI 문서를 프로그램적으로 생성할 수 있습니다.

컴파일된 도움말 파일 생성하기

LabVIEW가 생성한 HTML 또는 RTF 파일로 컴파일된 사용자 정의 도움말 파일을 생성할 수 있습니다.

다음 단계를 따라 컴파일된 도움말 파일을 생성합니다.

1. HTML 또는 RTF 파일을 VI 문서에서 생성합니다:
 - (Windows) 컴파일된 HTML 도움말 파일 또는 (macOS) Apple 도움말을 만들려면 **HTML 파일을 생성**합니다.
 - (Windows) WinHelp 또는 (Linux) HyperHelp 파일을 만들려면 **RTF 파일을 생성**합니다.
2. 소스 HTML 또는 RTF 문서를 한 파일로 컴파일합니다. 다음의 제 3의 유틸리티 중 하나를 이용하여 문서를 컴파일할 수 있으며 여러 유틸리티가 사용 가능합니다:
 - (Windows) Microsoft HTML Help Workshop을 사용하여 HTML 도움말 파일을 컴파일할 수 있으며 Microsoft Help Workshop을 사용하여 WinHelp 파일을 컴파일할 수 있습니다.
 - (macOS) Apple Help를 사용하여 HTML 파일을 도움말 시스템으로 볼 수 있습니다.
 - (Linux) Bristol Technology 사의 HyperHelp 컴파일러로 QuickHelp 파일을 컴파일할 수 있습니다.
3. **VI와 연결** 컴파일된 도움말 파일을 문서화할 수 있습니다.

여러 플랫폼에서 HTML 기반의 도움말 파일을 생성하려면, Windows의 경우 컴파일된 HTML 도움말 파일을 사용하고, macOS 또는 Linux에서는 컴파일 되지 않은 HTML 파일을 사용합니다. 여러 플랫폼에서 RTF 기반의 도움말 파일을 생성하기 위해서는, 반드시 각 플랫폼에서 각각 도움말 파일을 컴파일해야 합니다.

VI를 사용자 문서에 링크하기

VI를 웹 기반 도움말 또는 로컬 도움말 파일에 링크하여 기본 도움말 윈도우에 사용자 문서를 제공합니다. (Windows) 사용자 문서에는 텍스트 파일을 제외한 모든 타입의 파일을 사용할 수 있습니다. (macOS와 Linux) 사용자 문서에 HTML(.html 및 .htm) 파일과 PDF 파일만 사용할 수 있습니다.



노트

(Linux) 사용자 문서에 PDF 파일을 사용하려면, 반드시 다음 중 하나의 PDF 뷰어가 설치되어 있어야 합니다:

- Acroread
- Xpdf
- KPDF
- GGV

다음 단계를 완료하여 VI에 사용자 도움말 파일을 링크합니다:

1. 파일»VI 프로퍼티를 선택하여 VI 프로퍼티 대화 상자를 디스플레이합니다.
2. 항목 풀다운 메뉴에서 문서를 선택합니다.
3. 드롭 다운 메뉴에서 로컬 도움말 파일 또는 웹 기반 도움말을 선택합니다.
4. 단계 3에서 로컬 도움말 파일을 선택한 경우, VI를 로컬 도움말 파일에 링크하려면 다음 단계를 완료하십시오.
 1. 탐색 버튼을 클릭합니다.
 2. 링크하려는 사용자 도움말 파일을 탐색한 후 확인 버튼을 누릅니다. 파일 경로가 도움말 경로 텍스트 박스에 나타납니다.
 3. 도움말 경로가 .htm 또는 .html 파일에 대한 경로 또는 **기호형 경로**를 포함할 경우, LabVIEW는 도움말 태그 텍스트 박스를 무시합니다. 도움말 경로가 컴파일된 도움말 파일 .chm 또는 .hlp에 대한 경로를 포함할 경우, 도움말 태그 텍스트 박스를 사용하여 VI를 해당 도움말 파일의 특정 주제에 링크합니다.
 - .chm 파일의 경우, .htm 또는 .html 파일 이름을 입력하거나 도움말 태그 텍스트 박스에 인덱스 키워드를 입력할 수 있습니다. .htm 또는 .html 파일 이름은 반드시 HTML 도움말 프로젝트의 컴파일되지 않은 개별 HTML 파일의 이름과 일치해야만 합니다. HTML 파일 안에서 북마크를 링크하려면, 파일 이름 끝에 #를 추가하고 뒤에 북마크 이름을 넣습니다. 도움말 태그 텍스트 박스에 색인 키워드를 입력하는 경우, 입력한 색인 키워드는 HTML 도움말 프로젝트 색인의 키워드와 일치해야 합니다. Windows의 .chm 파일 및 macOS와 Linux의 컴파일되지 않은 HTML 파일에 링크를 하려면, 도움말 태그 텍스트 박스에 HTML 파일 이름을 입력합니다. (Windows) LabVIEW는 .chm 파일 내에 있는 HTML 파일에 링크를 겁니다. (macOS와 Linux) LabVIEW는 HTML 파일에 링크를 걸며, 이때 HTML 파일은 반드시 help\html\help.chm 디렉토리에 있어야 합니다. 여기서 help.chm은 4단계에서 도움말 경로 텍스트 박스에 입력한 .chm 파일 이름입니다.

- .hlp 파일에서는 도움말 태그 텍스트 박스에 인덱스 키워드를 입력할 수 있습니다. 인덱스 키워드는 반드시 WinHelp 또는 HyperHelp 프로젝트의 인덱스에 있는 키워드와 일치해야 합니다.

도움말 팔레트에 있는 도움말 함수 또는 문서 경로와 문서 태그 프로퍼티를 사용하여 VI를 프로그램적으로 사용자 도움말 파일에 링크할 수 있습니다.

5. VI를 웹 기반 도움말 파일에 링크하도록 선택하는 경우, 도움말 URL 텍스트 박스에 웹 기반 도움말 파일의 URL을 입력합니다. 지정한 파일이 VI의 기본 도움말에 있는 상세 도움말 링크와 상세 도움말 버튼으로 링크됩니다. 또한 웹 URL 사용과 문서 웹 URL 프로퍼티를 사용하여 프로그램적으로 VI를 웹 기반 문서에 링크할 수 있습니다.
6. 확인 버튼을 클릭하여 변경을 저장합니다. 커서를 VI 아이콘 위로 움직일 때, 파란색 상세 도움말 링크가 기본 도움말 윈도우에 나타나고 상세 도움말 버튼이 활성화됩니다.
7. 파란색 링크 또는 기본 도움말 윈도우의 상세 도움말 버튼을 클릭하여 HTML 또는 컴파일된 도움말 파일에 대한 VI의 링크를 테스트합니다. 또한, VI를 블록 다이어그램에 놓고 VI 아이콘에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 도움말을 선택할 수 있습니다.



노트 HTML 파일 또는 컴파일된 도움말 파일에 VI를 링크하고 이 VI를 사용하여 독립 어플리케이션을 만드는 경우, 어플리케이션을 만들 때 반드시 이 파일을 소스 파일로 포함해야만 합니다.

LabVIEW가 생성한 이미지 파일의 명명 규칙

외부 그래픽을 포함하는 **HTML** 또는 **RTF 문서를 생성**할 때, LabVIEW는 컨트롤과 인디케이터 데이터 타입 터미널과 다른 VI 부분을 일관된 이름을 갖는 그래픽 파일로 저장합니다.

LabVIEW는 다음의 정보를 사용하여 컨트롤과 인디케이터를 저장합니다:

- 첫번째 문자는 터미널이 컨트롤 또는 인디케이터인지에 따라 각각 c 또는 i입니다.
- 터미널이 배열을 나타내는 경우, 첫번째 문자 다음에 2d와 같은 차원이 옵니다.
- 차원 뒤에 데이터 타입을 나타내는 짧은 이름이 옵니다.
- 터미널의 데이터 타입이 LabVIEW 클래스이면, 데이터 타입의 이름이 LabVIEW 클래스의 정규화된 이름을 포함합니다.

다음 테이블은 컨트롤과 인디케이터의 예제 이미지 파일 이름을 나열합니다.

이름	터미널
ci32.jpg	컨트롤, 32비트 부호있는 정수
i2di32.jpg	인디케이터, 32비트 부호있는 정수의 2D 배열

이름	터미널
c3dstr.jpg	컨트롤, 문자열의 3D 배열
cLibrary_Child_1lvclass.jpg	컨트롤, Library.lvlib의 Child 1.lvclass

LabVIEW가 프런트패널과 같이 VI의 다른 부분에 대한 이미지 파일을 생성하는 경우, LabVIEW는 해당 VI의 정규화된 이름을 이미지 파일 이름에 포함시킵니다. 예를 들어, Test.vi가 라이브러리에 있으면, LabVIEW는 이 라이브러리를 이미지 파일 이름에 제곱합니다. 예를 들어, JPEG 그래픽 포맷을 선택하여 Test.vi라는 이름의 VI를 저장하는 경우, LabVIEW는 이 그래픽을 JPEG 파일로 저장합니다.

다음 테이블은 다른 VI 객체의 예제 이미지 파일 이름을 나열합니다.

VI 원소	JPEG 파일 이름
아이콘	Testi.jpg
커넥터 팬	Testc.jpg
프런트패널	Testp.jpg
라이브러리에 있는 VI의 프런트패널	Library_lvlib_Parent_lvclass_Testp.jpg
블록다이어그램	Testd.jpg
라이브러리에 있는 VI의 블록다이어그램	Library_lvlib_Parent_lvclass_Testd.jpg
VI 계층구조 윈도우	Testh.jpg
SubVI 아이콘	subVI_name i.jpg

VI, 템플릿, 객체를 위한 문서 인쇄하기

다음 단계를 따라 VI, 템플릿, 객체를 위한 문서를 인쇄합니다.

1. VI, 컨트롤, 글로벌 변수, 또는 템플릿 파일을 열고 파일>>인쇄를 선택하여 인쇄 대화 상자를 디스플레이합니다.
2. 인쇄 대화 상자의 VI 선택 페이지에서, 현재 VI에 대한 문서를 인쇄할 것인지 아니면 여러 VI에 문서를 인쇄할 것인지 선택합니다. 여러 VI를 선택한 경우, 다음 버튼을 클릭하여 VI 선택 페이지를 디스플레이하고 인쇄하려는 문서에 필요한 파일을 선택합니다.
3. 다음 버튼을 클릭하여 인쇄 내용 페이지를 디스플레이합니다.
4. 문서에 포함할 구성요소를 선택합니다. VI 문서를 선택한 경우, 다음 버튼을 클릭하여 VI 문서 페이지를 디스플레이하고 문서에 포함하려는 구성요소를 선택합니다.
5. 다음 버튼을 클릭하여 대상 페이지를 디스플레이합니다.

6. 프린터를 선택하고 다음 버튼을 클릭하여 페이지 설정 페이지를 디스플레이합니다.
7. 옵션을 선택하여 문서의 모양을 설정하고 다음 버튼을 클릭하여 프린터 페이지를 디스플레이합니다.
8. 이용할 수 있는 인쇄 옵션을 선택합니다.
9. 인쇄 버튼을 클릭하여 시스템의 기본 프린터에서 문서를 인쇄합니다. 시스템 대화 상자가 나타나면, 문서를 인쇄하기 전에 더 많은 인쇄 옵션을 설정할 수 있습니다.

또한 [VI 패널 또는 문서 인쇄] VI를 사용하거나 인쇄:VI를 프린터로 메소드를 사용하여 프로그램적으로 VI 문서를 프린터에 보낼 수 있습니다.

여러 어플리케이션 인스턴스에서 VI를 위한 문서 인쇄하기

VI는 사용자가 **여러 어플리케이션 인스턴스에서 편집**한 경우 다르게 나타날 수 있습니다.

다음 단계를 따라 여러 어플리케이션 인스턴스에서 VI를 위한 문서를 인쇄합니다.

1. 현재 어플리케이션 인스턴스에서, 문서를 인쇄하려는 VI를 열고 파일>>인쇄를 선택하여 인쇄 대화 상자를 디스플레이합니다.
2. 인쇄 대화 상자를 사용하여 프린터로 VI의 **문서를 인쇄**하거나 **HTML, RTF, 또는 텍스트 문서를 생성**합니다.
3. 두번째 어플리케이션 인스턴스에서, 문서를 인쇄하려는 해당 VI를 열고 파일>>인쇄를 선택하여 인쇄 대화 상자를 디스플레이합니다.
4. 인쇄 대화 상자를 사용하여 프린터로 VI의 문서를 인쇄하거나 HTML, RTF, 또는 텍스트 문서를 생성합니다.
5. 위의 두 단계를 문서를 인쇄하기 위해 연 VI의 모든 어플리케이션 인스턴스에 반복합니다.

HTML 문서 생성하기 및 프런트패널 이미지 내장하기

다음 단계를 따라 HTML 문서를 생성하고, VI나 독립 어플리케이션 프런트패널의 정적인 혹은 동적인 이미지를 내장합니다. 또한, 프런트패널의 이미지를 **기존의 HTML 문서**에 넣을 수 있습니다.



노트

HTML문서에 어플리케이션의 프런트패널을 임베드하는 경우, 해당 어플리케이션에서 **웹 서버 활성화**를 위해 임베드된 VI의 프런트패널을 사용합니다.

1. 클라이언트에게 보여주고자 하는 프런트패널을 엽니다.
2. 도구>>웹 출판 도구를 선택하여 웹 출판 도구 대화 상자를 디스플레이합니다.
3. VI 이름 필드에서 VI 이름을 선택하거나 VI 이름 풀다운 메뉴에서 탐색을 선택하여 VI를 찾습니다.

노트 웹 출판 도구는 VI를 열려는 **어플리케이션 인스턴스**에서 HTML 파일을 생성합니다. 다른 어플리케이션 인스턴스에서 VI를 닫고 다시 열려는 경우 클라이언트는 VI를 볼 수 없습니다.

4. 웹 브라우저에서 프런트패널이 어떻게 나타날지 설정합니다.
 1. 보기 모드 섹션에서, 프런트패널 내장, 스냅샷 찍기, 모니터링 중 원하는 것을 선택합니다.
 2. 내장된 패널 이미지 주위에 경계선을 놓으려면 경계 보이기 확인란에 확인 표시를 합니다.
 3. 내장된 프런트패널이 곧바로 컨트롤을 요청할 수 있도록 하려면 연결될 때 컨트롤 요청 확인란에 확인 표시를 합니다.
5. 다음 버튼을 클릭하여 HTML 출력 옵션을 디스플레이합니다. 문서 제목, 머리글, 바닥글 텍스트 박스에 텍스트를 입력합니다.
6. 기본 브라우저에서 문서를 미리 보려면, 웹 서버 시작 버튼을 클릭한 후 브라우저에서 미리보기 버튼을 클릭합니다. 웹 서버 시작 버튼이 비활성화된 경우, 웹 서버가 이미 시작된 상태입니다.
7. 다음 버튼을 클릭하여 파일 저장 옵션을 디스플레이합니다. HTML 파일과 HTML 파일 이름을 저장하려는 디렉토리의 경로를 확인하거나 편집합니다. 원격 컴퓨터에서 문서를 보고자 한다면, 일반적으로 labview\www인 **웹 서버 루트 디렉토리**에 HTML 문서를 저장합니다.
8. HTML 문서에 제목, 텍스트, VI 프런트패널 이미지를 저장하기 위해 디스크에 저장 버튼을 클릭합니다. 문서를 웹 서버 루트 디렉토리에 저장한 경우, 문서 URL 대화 상자는 웹 페이지의 URL을 디스플레이합니다.
 1. 연결 버튼을 클릭하여 HTML 문서를 보거나, 또는 확인 버튼을 클릭하여 웹 출판 도구 대화 상자로 돌아갑니다.
 2. 완료 버튼을 클릭하여 웹 출판 도구에서 나옵니다.
9. 보기 모드 섹션의 임베디드를 선택한 경우, 클라이언트를 통해 프런트패널을 보려고 하는 VI가 서버 컴퓨터의 메모리 안에 있도록 확인합니다.

웹 출판 도구에서 웹 서버를 시작했고 웹 서버를 닫아야 할 경우, 다음 단계를 따릅니다.

1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 옵션 대화 상자의 항목 리스트에서 웹 서버를 선택합니다.
3. 리모트 패널 서버 섹션의 리모트 패널 서버 활성화 확인란에서 확인 표시를 제거합니다.
4. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

LabVIEW를 설치하지 않은 클라이언트가 원격으로 프런트패널을 보고 컨트롤하려는 경우, 반드시 **LabVIEW 런타임 엔진을 설치**한 다음 **원격으로 프런트패널을 확인**해야 합니다.

모든 HTML 편집기에서 웹 출판 도구로 생성된 HTML 파일을 열 수 있으며 HTML을 사용자 정의할 수 있습니다. 프런트패널 이미지를 내장하는 코드 섹션을 찾아서 **이미지에 대한 속성을 지정**합니다.

기존의 HTML 문서에 프런트패널 이미지 내장하기

National Instruments는 웹 출판 도구를 사용하여 **HTML 문서를 생성하고 프런트패널의 정적 또는 애니메이션 이미지를 포함**시키거나 HTML 문서에 프런트패널을 포함시켜 클라이언트 컴퓨터가 **원격으로 프런트패널을 보고 제어**할 수 있게 할 것을 권장합니다.

하지만, HTML 문서를 이미 가지고 있는 경우, 다음 단계를 따라 기존의 HTML 문서에 프런트패널의 정적 이미지를 내장합니다. HTML 문서를 **웹 서버 루트 디렉토리**에 보관합니다. VI를 HTML 문서에 내장하는 예제는 **웹 서버 구문 예제**를 참조하십시오.

1. 다음 HTML 코드를 기존의 HTML 문서에 복사합니다:

```
<IMG SRC="http://web.server.addr/.snap?VI_Name[&type=png]
[&depth=24][&quality=80][&compression=-1][&refresh=60]
[&full=off]">
```

2. web.server.addr를 LabVIEW 웹 서버의 주소로 대체합니다.

3. VI_Name을 VI의 이름으로 대체합니다.

물음표는 VI 이름에서 URL을 분리합니다. 대괄호 []의 파라미터는 옵션입니다.

4. (옵션) 다음 파라미터를 추가하여 이미지의 속성을 지정할 수 있습니다.

파라미터

설명

보여줄 VI 프런트패널의 이름. VI_Name은 독립 어플리케이션의 최상위 VI입니다. 표준 URL 명명 규칙에 따라 VI 이름을 인코딩합니다. 예를 들어, 알파벳이 아닌 문자는 %xx로 대체하며, 이때 xx는 문자의 16진 값입니다. 또한 더하기 + 부호를 사용하여 공백을 대체할 수 있습니다.

PGN 또는 JPEG의 이미지 타입. type이 지정되지 않은 경우, 기본은 png입니다. 브라우저가 PNG를 지원하지 않는 경우, 기본은 jpeg입니다.

이미지의 색의 농도. 농도는 1, 4, 8, 또는 24 비트가 될 수 있습니다. 농도가 지정되지 않은 경우, 기본은 모니터의 색 농도입니다.

JPEG 프런트패널 이미지의 이미지 품질과 메모리 크기. 이 파라미터는 타입이 jpeg일 때에만 적용 가능합니다. 품질은 0과 100 사이가 될 수 있습니다. 품질이 지정되지 않은 경우, 기본은 80입니다.

압축 레벨은 PNG 이미지를 압축하는데 사용됩니다. 이 파라미터는 타입이 png일 때에만 적용 가능합니다. 압축은 -1과 9 사이가 될 수 있으며, 이때 0은 압축 없음을 나타냅니다. 1은 압축이 적으나 속도가 빠름을 나타내고, 9는 압축이 크나 속도가 느림을 나타냅니다. 3부터 6까지의 압축 레벨은 9 레벨만큼의 성능을 보이지만 9 레벨보다 적은 계산을 수행합니다. 압축이 지정되지 않은 경우, 기본은 -1이며, 이는 최적으로 조화된 속도와 압축을 사용합니다.

캐쉬된 이미지의 초 단위의 최대 기간. 캐쉬된 이미지가 새로 고침 초보다 오래된 경우, 새로운 이미지가 생성됩니다.

파라미터

설명

윈도우에 전체 컨트롤의 이미지를 반환할 것인지 또는 단지 보이는 부분만을 반환할 것인지를 지정합니다. 모든 컨트롤을 나타내려면 전체를 on으로 설정하고 윈도우 내용을 나타내려면 off로 설정합니다. 전체가 지정되지 않은 경우, 기본은 off입니다.

프런트패널 이미지를 보기 위해서, 웹 브라우저의 URL에 서버의 IP 주소 또는 컴퓨터 이름을 입력하고 그 뒤에 HTML 파일의 이름을 입력합니다. 예를 들어 http://ipaddress/filename.html입니다.



노트

웹에 VI 프런트패널을 디스플레이하려면, VI가 반드시 컴퓨터의 메모리에 있어야 합니다.

기존의 HTML 문서에 VI 내장하기

National Instruments는 사용자가 웹 출판 도구를 사용하여 **HTML 문서를 생성하고 VI를 임베드**함으로써 클라이언트 컴퓨터가 웹 브라우저에서 VI의 프런트패널을 보고 제어할 수 있게 하기를 권장합니다.

하지만, 이미 HTML 문서를 가지고 있는 경우, 다음 구문을 사용하여 VI를 HTML 문서에 내장하여 클라이언트 컴퓨터가 웹 브라우저에서 VI의 프런트패널을 보고 제어할 수 있도록 합니다. HTML 문서를 **웹 서버 루트 디렉토리**에 보관합니다. VI를 HTML 문서에 내장하는 예제는 **웹 서버 구문 예제**를 참조하십시오.

클라이언트와 서버 컴퓨터는 반드시 LabVIEW의 동일한 버전의 리모트 패널 프로토콜로 실행해야 합니다. 리모트 프런트패널을 보고 컨트롤하기 위해서 브라우저를 사용하는 경우, 반드시 서버 컴퓨터의 LabVIEW 버전과 호환되는 **LabVIEW 런타임 엔진 버전을 사용**해야 합니다.



노트

다른 버전의 LabVIEW 브라우저 플러그인은 동시에 수행되지만 서로 다른 리모트 프런트패널 서버와 통신합니다. 각 브라우저 플러그인은 같은 버전의 LabVIEW로 개발한 VI를 디스플레이할 수 있습니다. HTML의 OBJECT/EMBED 태그는 브라우저가 어느 플러그인을 로드할 것인지 결정합니다. LabVIEW ActiveX 컨트롤의 경우, OBJECT 태그에 지정한 CLASSID가 어느 플러그인을 로드할 것인지 결정합니다. 각 버전에 따른 플러그인의 CLASSID는 서로 다릅니다.

```
<OBJECT ID="LabVIEWControl" CLASSID="CLSID:A40B0AD4-
B50E-4E58-8A1D-8544233807B0" WIDTH= x HEIGHT= x CODEBASE="ftp://ftp.ni.com/
support/labview/runtime/windows/ 15.0 /LVRTE 15.0 min.exe"> <PARAM
name="LVFPPVINAME" value=" VI_NAME.vi "> <EMBED SRC=".LV_FrontPanelProtocol.
rpvi150 " LVFPPVINAME=" VI_NAME.vi " TYPE="application/x- labviewrpvi150 " WIDTH= x
HEIGHT= x PLUGINSOURCE="http://digital.ni.com/express.nsf/bycode/exck2m"></EMBED>
</OBJECT>
```

여러 다른 버전의 LabVIEW에서 사용되는 구문에 대한 더 자세한 정보는 다음 테이블을 참조하십시오. 버전 간 변경사항은 위의 예에서 굵은체로 하이라이트되어 있습니다.

LabVIEW 버전	코드 베이스	임베드 소스	임베드 타입
2015	15.0	rpvi150	labviewrpvi150
2014	14.0	rpvi140	labviewrpvi140
2013	13.0	rpvi130	labviewrpvi130



노트 클라이언트가 LabVIEW ActiveX 컨트롤을 사용하여 VI에 접근하는 경우 오직 <OBJECT> 태그 정보만 포함하면 됩니다. 하지만, 다양한 설정을 가지는 모든 잠재적인 클라이언트를 가장 잘 지원하기 위해서는, <OBJECT>와 <EMBED> 태그를 모두 포함해야 합니다. 이것은 모든 클라이언트가 브라우저를 사용해서 원격으로 VI에 접근할 수 있도록 합니다.

다음 리스트는 반드시 설정해야만 하는 파라미터를 설명합니다.

파라미터	설명
CODEBASE	LabVIEW ActiveX 컨트롤을 사용하여 VI에 접근하는 클라이언트에 대한, .exe 파일의 이름을 포함하는 LabVIEW 런타임 엔진 설치 프로그램의 위치. 이전 예제에서 제공된 주소는 현재 설치 프로그램 버전의 위치를 포함합니다.
VI_NAME	Example.vi와 같은 VI의 이름. VI_NAME은 VI가 비록 서브디렉토리 또는 .llb 파일 내에 위치하더라도 경로 또는 디렉토리 이름을 포함하지 않습니다.
PLUGINSOURCE	Netscape 플러그인을 사용하여 VI에 접근하는 클라이언트에 대해서, 클라이언트가 LabVIEW 런타임 엔진 설치 프로그램을 다운로드 받을 수 있는 URL. 이전 예제에서 제공된 주소는 현재 설치 프로그램 버전의 위치를 포함합니다.
(옵션) WIDTH	픽셀 단위의 프런트패널의 폭.
(옵션) HEIGHT	픽셀 단위의 프런트패널의 높이.

LabVIEW에서 인쇄하기

VI 인쇄물 향상시키기

다음 기술을 사용하여 VI 또는 프런트패널 인쇄물의 모양을 향상시킵니다:

- 색칠 도구와 투명을 사용하여 인쇄물에서 나타내지 않으려는 프런트패널 컨트롤의 일부를 숨깁니다. 예를 들어, 투명을 사용하여 컨트롤의 테두리의 모양을 단순화할 수 있습니다.
- 장식 팔레트의 객체를 사용하여 프런트패널의 섹션을 하이라이트합니다. 예를 들어, 상자로 컨트롤의 섹션을 둘러싸서 사용자 인터페이스의 나머지에서부터 컨트롤을 두드러지게 할 수 있습니다.
- 구획 배경, 링 컨트롤의 아이템, 다른 프런트패널 객체 부분과 같은 다른 어플리케이션에서 그래픽을 사용하여 인쇄물을 향상시킵니다.



관련 정보

[투명한 객체 생성하기](#) (992 페이지 참조)

[프런트패널에서 장식 사용하기](#) (1334 페이지 참조)

[프런트패널 설정하기](#) (548 페이지 참조)

독립적인 프런트패널 객체의 이미지 생성하기

이미지 얻기 메소드를 이용하여 단일 프런트패널 객체의 이미지를 생성합니다. [컨트롤 이미지를 리포트에 추가] VI를 이용하여 리포트에 프런트패널 객체의 이미지를 추가합니다.

다음 단계를 따라 이미지 얻기 메소드를 이용하여 단일 프런트패널 객체의 이미지를 생성합니다.

1. 블록 다이어그램에서, 이미지를 생성하려는 컨트롤 또는 인디케이터의 터미널에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성»인보크 노드»이미지 얻기를 선택하여 인보크 노드를 생성합니다.
2. 이미지 농도 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»상수를 선택하여 숫자형 상수를 생성합니다.
3. 숫자형 상수에서 1, 4, 8, 또는 24를 입력하고 <Enter>를 눌러서 이미지의 색 농도를 설정합니다.

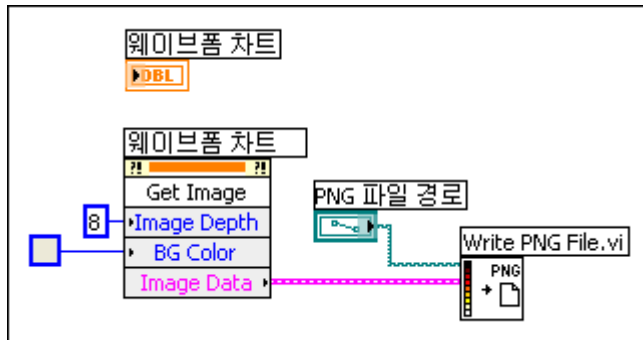
값	정의
1비트, 흑백 이미지	
4비트, 16색 이미지	
8비트, 256색 이미지	
24비트, 트루 컬러 이미지	

4. (옵션) 색 상자 상수 또는 숫자로 된 색 값을 배경색 입력에 연결하여 이미지의 배경색을 지정합니다.
5. 다음 VI 중에서 하나를 블록 다이어그램에 추가합니다:
 - JPEG 포맷으로 이미지를 저장하려면 [JPEG 파일 쓰기] VI를 추가합니다.
 - PNG 포맷으로 이미지를 저장하려면 [PNG 파일 쓰기] VI를 추가합니다.
 - BMP 포맷으로 이미지를 저장하려면 [BMP 파일 쓰기] VI를 추가합니다.

6. 인보크 노드의 이미지 데이터 출력을 [그래픽 포맷] VI의 이미지 데이터 입력에 연결합니다.
7. [그래픽 포맷] VI의 포맷 파일의 경로 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>>컨트롤을 선택합니다. 이때 포맷은 이미지의 그래픽 포맷입니다.
8. 포맷 파일의 경로 프런트패널 컨트롤에서 이미지의 파일 이름과 이미지를 저장하려는 디렉토리를 포함하는 경로를 입력합니다.

VI를 실행하면 입력한 디렉토리에 이미지를 저장합니다.

다음 블록다이어그램은 PNG 포맷으로 웨이브폼 차트의 이미지를 저장합니다.



또한, **인보크 노드를 포함하는 subVI를 생성**할 수 있으며 subVI에 컨트롤 참조를 연결합니다.

VI 실행 후 VI의 프런트패널 인쇄하기

수행>>완료 시 인쇄를 선택하여 실행을 끝낼 때 VI의 프런트패널을 인쇄합니다.

또한, 다음 단계를 따라 VI 실행 후에 VI의 프런트패널을 인쇄합니다.

1. 파일>>VI 프로퍼티를 선택합니다.
2. 항목 풀다운 메뉴에서 인쇄 옵션을 선택합니다.
3. VI가 실행을 완료할 때마다 자동으로 프런트패널 인쇄 확인란에 확인 표시를 합니다.
4. 확인 버튼을 클릭합니다.

이 옵션을 선택하여 **VI를 인쇄**하는 것은 프런트패널이 활성 윈도우일 때 파일>>윈도우 인쇄를 선택하는 것과 유사합니다.

이 VI를 SubVI로 사용할 경우, LabVIEW는 subVI의 실행이 완료되고 subVI가 호출차로로 복귀하기 전에 인쇄합니다.



팁

이름, 날짜, 페이지 번호를 포함하지 않은 프런트패널을 인쇄하려면 파일>>VI 프로퍼티를 선택하고 풀다운 메뉴에서 인쇄 옵션 프로퍼티를 선택하여 인쇄 헤더 확인란에서 확인 표시를 제거합니다.

프로그램적으로 VI 인쇄하기

다음의 방법 중 하나를 사용하여 파일>>윈도우 인쇄와 파일>>인쇄를 선택하면 나타나는 대화 상자에서 대화식보다는 프로그램식으로 VI를 인쇄합니다.

- VI가 매번 실행을 마칠 때마다 프런트패널을 자동적으로 인쇄하도록 VI를 설정합니다.
- VI를 인쇄할 subVI를 생성합니다.
- [리포트 생성] VI를 사용하여 리포트를 인쇄하거나 VI 문서 또는 VI가 반환하는 데이터를 포함하는 HTML 리포트를 저장합니다.
- VI 클래스의 인쇄 메소드를 사용하여 프로그램적으로 VI 윈도우를 인쇄하거나 VI 문서를 인쇄하거나 이를 HTML, RTF, 또는 텍스트 파일로 저장합니다.



노트

독립 어플리케이션에서 VI 문서를 인쇄할 경우, 프런트패널만 인쇄할 수 있습니다.

VI를 프로그램적으로 인쇄할 때, LabVIEW는 현재 VI를 위한 인쇄 옵션을 사용합니다. 예를 들면, 파일>>인쇄를 선택하여 VI를 마지막으로 인쇄했다면, LabVIEW는 VI를 프로그램적으로 인쇄할 때 인쇄 대화 상자에서 설정한 모든 옵션을 사용합니다. 어플리케이션 클래스나 VI 클래스에서 인쇄 프로퍼티를 설정하면, LabVIEW는 현재 VI의 인쇄 옵션을 변경하고 프로그램적으로 설정한 옵션을 사용합니다.



관련 정보

[VI 실행 후 VI의 프런트패널 인쇄하기](#) (1406 페이지 참조)

[SubVI를 사용하여 상위 레벨 VI의 데이터 인쇄하기](#) (1408 페이지 참조)

프린터 에러 문제 해결하기

다음 기술을 사용하여 LabVIEW로부터 인쇄할 때 직면할 수 있는 에러에 대한 해법을 찾을 수 있습니다.

- 다른 인쇄 방법을 사용합니다. 옵션 대화 상자의 인쇄 페이지에서, 사용하는 OS에 따라 인쇄 방법을 표준, 포스트스크립트 또는 비트맵으로 선택할 수 있습니다.
- 프린터 드라이버 정보를 확인합니다. 프린터 드라이버에서 에러 때문에 인쇄할 때 문제가 발생할 수 있습니다. 프린터 드라이버의 최신 버전은 프린터 제조업체에 문의하십시오.

- 다른 컴퓨터나 다른 프린터를 사용합니다.
- 프린터가 설정한 페이지 여백을 허용하는지 확인합니다. 올바르지 않은 페이지 여백은 출력을 만들어내기 전에 프린트 큐에서 프린트 작업이 사라지는 것과 같은 문제를 발생시킬 수 있습니다.
- 비디오 셋팅을 변경하고 비디오 드라이버를 확인합니다. 화면 해상도와 색 농도 셋팅이 프린터 작업에 영향을 줄 수 있습니다. 비디오 드라이버의 최신 버전은 그래픽 카드의 제조업체에 문의하십시오.
- (macOS) LabVIEW에 더 많은 메모리를 할당합니다.
- **HTML**이나 **RTF** 문서를 생성한 후 웹 브라우저나 워드 프로세싱 어플리케이션을 사용하여 문서를 인쇄합니다.
- 리포트 생성 VI를 사용하여 리포트를 프로그램적으로 생성하고, 다루고, 인쇄할 수 있습니다.

라벨과 프런트패널 컨트롤의 텍스트가 VI를 인쇄할 때 잘리면, 프린터의 폰트가 모니터의 폰트 크기와 일치하지 않습니다. 다음 기술 중의 하나를 사용하여 VI 출력에서 컨트롤과 라벨의 텍스트가 잘리는 것을 막을 수 있습니다.

- 프린터가 포스트스크립트 언어를 지원한다면 포스트스크립트 프린터 드라이버를 사용합니다.
- 라벨과 프런트패널 컨트롤의 크기를 증가시켜서 폰트가 일치하지 않아서 텍스트가 확장될 때 컨트롤이나 라벨 경계 안에 여전히 들어가도록 합니다.
- 모니터와 프린터에서 같은 크기를 가지는 폰트를 찾습니다. 많은 프린터 드라이버가 이 과정을 지원하기 위해 폰트 대체 알고리즘을 가지고 있습니다.
- (Windows) 비트맵 인쇄를 사용하여 모니터에서 보는 것과 출력이 정확히 일치하도록 만듭니다.

National Instruments에 버그 리포트를 제출하는 방법을 포함한 문제해결 인쇄 에러에 대한 추가적인 정보는 기술지원 데이터베이스를 참조하십시오.

SubVI를 사용하여 상위 레벨 VI의 데이터 인쇄하기

버튼을 클릭하거나 테스트 실패와 같은 특정한 조건이 발생하는 경우에만 **프로그램식으로 VI를 인쇄**하는 것이 좋습니다. 또한, 인쇄의 포맷을 더 많이 컨트롤하거나 컨트롤의 일부분만 인쇄하려 할 수도 있습니다. 이런 경우, **완료 시 인쇄로 설정된** SubVI를 사용할 수 있습니다.

다음 단계를 따라 subVI를 사용하여 상위 VI로부터 데이터를 인쇄합니다.

1. SubVI를 생성합니다.
2. LabVIEW를 사용하여 인쇄하려는 방식으로 **프런트패널을 포맷**합니다.
3. SubVI에서 수행>>완료 시 인쇄를 선택합니다.
4. 인쇄하고자 할 때, subVI를 호출하고 subVI로 인쇄하고자 하는 데이터를 연결합니다.

LabVIEW의 리포트

리포트 생성하기

[리포트 생성] VI를 사용하여 리포트를 인쇄하거나 VI 문서 또는 VI가 반환하는 데이터를 포함하는 HTML 리포트를 저장합니다. [VI 패널 또는 문서 인쇄] VI를 사용하여 VI 문서를 포함하는 기본 리포트를 생성합니다. [간단한 텍스트 리포트 생성] VI를 사용하여 VI가 반환하는 데이터를 포함하는 기본 리포트를 생성합니다. 나머지 리포트 생성 VI를 사용하여 더욱 복잡한 리포트를 생성합니다.

리포트 익스프레스 VI를 사용하여 VI 문서, VI가 반환한 데이터, 그리고 만든이, 회사, 페이지 수 같은 리포트 프로퍼티를 포함하는 미리 포맷된 리포트를 생성합니다.

리포트 생성을 위한 데이터 포매팅

리포트 생성 VI를 사용하여 LabVIEW에서 정보를 디스플레이하고 포맷할 수 있습니다. [테이블을 리포트에 추가] VI로 2D 숫자 또는 문자열 배열로 데이터를 포맷합니다. 문자열 함수를 사용하여 데이터를 텍스트로 변환하여 리포트에 추가할 수 있습니다. [이미지를 리포트에 추가] VI를 사용하여 리포트에 데이터 그래프 그림을 포함시킵니다.

리포트에 토큰 사용하기

토큰은 자동으로 정보를 생성하는 문자열이며, 리포트의 머릿글과 바닥에 유용합니다. 토큰을 사용하여 페이지의 번호를 매기고 리포트에 타임스탬프를 놓을 수 있습니다. 토큰을 [리포트 생성] VI 또는 문자열을 받는 [리포트] 익스프레스 VI의 모든 파라미터와 함께 사용할 수 있습니다. 토큰은 표준과 HTML 리포트와 함께만 사용할 수 있습니다.

예를 들어, [간단한 텍스트 리포트 생성] VI를 사용하는 경우, 바닥글 문자열 중 하나에 토큰 <page>를 입력하면 리포트의 모든 페이지의 바닥글에 페이지 번호가 추가됩니다.

다음 리스트는 LabVIEW에서 사용 가능한 토큰을 나타냅니다.

토큰	설명
<tab>	(Windows) HTML 리포트에서 LabVIEW는 이 토큰을 로 해석합니다.
<page>	현재 페이지 번호. LabVIEW는 HTML 리포트에서 이 토큰을 무시합니다.
<pages>	전체 페이지 수. LabVIEW는 HTML 리포트에서 이 토큰을 무시합니다.
<pagenofm>	리포트에서 전체 페이지 수 중에서 현재 페이지 수. LabVIEW는 HTML 리포트에서 이 토큰을 무시합니다. 예제: 30 중에서 7
<shortdate>	xx / xx / xx 형식의 현재 날짜. 예: 10/5/98

토큰	설명
<longdate>	요일, 월일, 연도 형식의 현재 날짜. 월, 일, 연도의 순서는 OS의 날짜 설정을 기본으로 합니다. 예제: 월요일, 10월 5일, 1998
<time>	시:분:초 AM/PM 형식의 현재 시간. 사용자 컴퓨터의 시계 설정을 기본으로 합니다. 예제: 1:58:22 PM
<newline>	리포트에 라인 구분을 삽입합니다.

새 VI 반출 리스트 엔트리 추가하기

다음 단계를 따라 VI 반출 리스트에 새 엔트리를 추가합니다.

1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. VI 반출 리스트 섹션의 추가 버튼을 클릭하여 새 VI를 추가하고 새 엔트리를 생성합니다.
4. VI 반출 리스트 텍스트 박스에 VI 또는 디렉토리 경로의 이름을 입력합니다. 또한, VI 이름이나 디렉토리 경로에 **와일드 카드를 사용**할 수 있습니다.
5. 접근 허용 또는 접근 거부 버튼을 클릭하여 선택된 VI에 대한 접근을 각각 허용하거나 거부합니다. 접근을 허용할 때 녹색 확인 표시가 아이템의 왼쪽에 나타나고, 접근을 거부할 때 붉은색 X가 나타납니다. 엔트리에 녹색 확인 표시나 붉은색 X마크가 그 이름 옆에 있지 않을 때, 엔트리에 대한 구문이 맞지 않습니다. 엔트리를 편집하거나 제거 버튼을 클릭하여 엔트리를 삭제합니다.
6. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

기존 VI 반출 리스트 엔트리 편집하기

다음 단계를 따라 VI 반출 리스트에서 VI 이름 또는 디렉토리 경로를 편집합니다.

1. 도구>>옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. VI 반출 리스트에서 수정하려는 VI 이름 또는 디렉토리 경로를 선택합니다. 이름 또는 경로가 텍스트 박스에 나타납니다.
4. 텍스트 박스에서 이름 또는 경로를 편집하거나 접근 허용 또는 접근 거부 버튼을 클릭합니다. 또한, VI 이름이나 디렉토리 경로에 **와일드 카드를 사용**할 수 있습니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

VI 반출 리스트에서 VI 삭제하기

다음 단계를 따라 VI 반출 리스트에서 VI 이름 또는 디렉토리 경로를 삭제합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. VI 반출 섹션의 VI 반출 리스트에서 제거하려는 VI 이름이나 디렉토리 경로를 선택합니다.
4. 제거 버튼을 클릭합니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

새 머신 접근 리스트 엔트리 추가하기

다음 단계를 따라 머신 접근 리스트에 새 엔트리를 추가합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 옵션 대화 상자의 항목 리스트로부터 VI 서버를 선택합니다.
3. 머신 접근 섹션의 추가 버튼을 클릭하여 새 엔트리를 생성합니다.
4. 머신 이름/주소 텍스트 박스에 유효한 브라우저 주소를 입력합니다.
mydomain.com과 같은 도메인 이름이나 130.164.123.123과 같은 IP 주소를 입력합니다. 또한, 브라우저 주소에 **와일드 카드를 사용**할 수 있습니다.



노트

DNS 서버에 접근할 수 없는 컴퓨터에서 VI 서버가 실행될 경우, 머신 접근 리스트에 도메인 이름 엔트리를 사용하지 마십시오. 도메인 이름 또는 IP 주소 분석 요청이 실패하면 컴퓨터가 느려집니다.

1. 접근 허용 또는 접근 거부 버튼을 클릭하여 선택된 아이템에 대한 접근을 각각 허용하거나 거부합니다. 접근을 허용할 때 녹색 확인 표시가 아이템의 왼쪽에 나타나고, 접근을 거부할 때 붉은색 X가 나타납니다. 엔트리에 녹색 확인 표시나 붉은색 X마크가 그 이름 옆에 있지 않을 때, 엔트리에 대한 구문이 맞지 않습니다. 엔트리를 편집하거나 제거 버튼을 클릭하여 엔트리를 삭제합니다.
2. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

기존의 머신 접근 리스트 엔트리 편집하기

다음 단계를 따라 머신 접근 리스트에서 브라우저 주소를 편집합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. 머신 접근 섹션의 머신 접근 리스트에서 수정하려는 주소를 선택합니다. 주소가 텍스트 박스에 나타납니다.
4. 텍스트 박스의 주소를 편집하거나 접근 허용 또는 접근 거부 버튼을 클릭합니다. 또한, 브라우저 주소에 **와일드 카드를 사용**할 수 있습니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

머신 접근 리스트 엔트리 제거하기

다음 단계를 따라 머신 접근 리스트에서 브라우저 주소를 삭제합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. 머신 접근 섹션의 머신 접근 리스트에서 제거하려는 주소를 선택합니다.
4. 제거 버튼을 클릭합니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

VI 서버를 위한 머신 접근 설정하기

옵션 대화 상자의 VI 서버 페이지에 있는 프로토콜 섹션에서, TCP/IP 확인란에 확인 표시를 하여 TCP/IP에 대한 서버 지원을 활성화할 수 있습니다. 머신 접근 섹션을 사용하여 컴퓨터 상에 있는 VI에 접근할 수 있는 IP 주소를 설정합니다. 개별 IP 주소에 접근을 허용하고 거부하는 머신 접근 리스트를 생성합니다. 또한, 서버:TCP/IP 접근 리스트 프로퍼티를 사용하여 VI 서버에 접근할 수 있는 머신의 TCP/IP 주소를 프로그램적으로 나열할 수 있습니다.

원격 LabVIEW 어플리케이션이 VI 서버와의 연결을 시도할 때, VI 서버는 IP 주소를 머신 접근 리스트의 엔트리와 비교해서 주소를 부여해야 할 것인지 결정합니다. 머신 접근 리스트의 엔트리가 IP 주소와 일치하면, VI 서버는 어떻게 **엔트리를 설정**할 것인지를 기반으로 접근을 허용하거나 거부합니다. 엔트리가 IP 주소와 일치하지 않으면, VI 서버는 접근을 거부합니다. 새로운 머신 접근 리스트 엔트리를 **추가**하거나, 기존 엔트리를 **편집**하거나, 또는 엔트리를 **제거**할 때 엔트리에 올바른 **구분, 순서, 와일드카드**를 사용합니다.

브라우저 접근 리스트 엔트리 제거하기

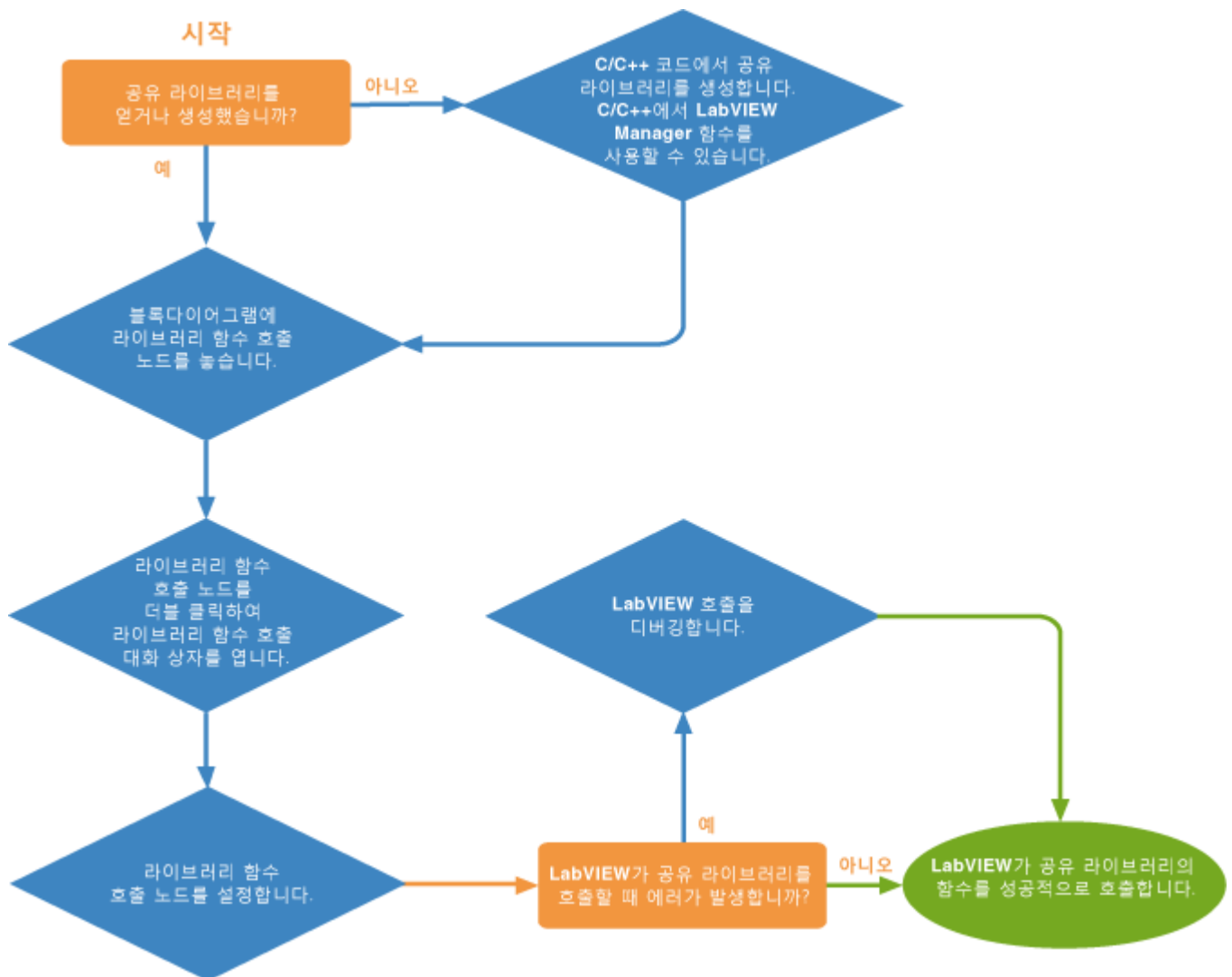
다음 단계를 따라 브라우저 접근 리스트에서 브라우저 주소를 삭제합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 웹 서버를 선택합니다.
3. 브라우저 접근 섹션의 브라우저 접근 리스트에서 삭제하려는 주소를 선택합니다.
4. 제거 버튼을 클릭합니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

LabVIEW에서 공유 라이브러리 호출하기

라이브러리 함수 호출 노드를 사용하여 LabVIEW 대부분의 표준 공유 라이브러리를 호출합니다. [오토메이션 열기] 함수를 [프로퍼티 노드] 및 [인보크 노드]와 함께 사용하여 **ActiveX** 객체를 포함하는 공유 라이브러리를 호출할 수 있습니다.

다음은 LabVIEW의 공유 라이브러리 호출에 대한 일반적인 작업 흐름도입니다. 각 단계에 대한 상세 정보를 확인하려면 파란색 마름모꼴 상자를 클릭하십시오.



(Windows) 관련된 토픽을 보려면 이 윈도우의 맨 위에 있는 도구 모음에서 왼쪽 그림과 같은 위치 보기 버튼을 클릭합니다. LabVIEW 도움말의 내용 탭에서 이 토픽이 하이라이트되어 사용자가 관련 토픽을 탐색할 수 있게 합니다.

관련 개념

공유 라이브러리 호출하기

Configuring the Call Library Function Node

Using Arrays and Strings in the Call Library Function Node

공유 라이브러리 및 공유 라이브러리 호출 디버깅하기

공유 라이브러리 반입하기

Using LabVIEW Manager Functions in Shared Libraries

Memory Manager

관련 사용법

패턴화된 문자열로 함수 사용하기



관련 정보

[ActiveX 컨테이너 개요](#) (1440 페이지 참조)

[Configuring the Call Library Function Node](#) (1569 페이지 참조)

[Debugging Shared Libraries and Calls to Shared Libraries](#) (1594 페이지 참조)

[Calling Shared Libraries](#) (1568 페이지 참조)

[Configuring the Call Library Function Node](#) (1569 페이지 참조)

[Using Arrays and Strings in the Call Library Function Node](#) (1599 페이지 참조)

[Debugging Shared Libraries and Calls to Shared Libraries](#) (1594 페이지 참조)

[Importing Shared Libraries](#) (1603 페이지 참조)

[Using LabVIEW Manager Functions in Shared Libraries](#) (1608 페이지 참조)

[Memory Manager](#) (1620 페이지 참조)

[패턴화된 문자열로 함수 사용하기](#) (1425 페이지 참조)

[ActiveX 컨테이너 개요](#) (1440 페이지 참조)

[Configuring the Call Library Function Node](#) (1569 페이지 참조)

[Debugging Shared Libraries and Calls to Shared Libraries](#) (1594 페이지 참조)

[Calling Shared Libraries](#) (1568 페이지 참조)

[Configuring the Call Library Function Node](#) (1569 페이지 참조)

[Using Arrays and Strings in the Call Library Function Node](#) (1599 페이지 참조)

[Debugging Shared Libraries and Calls to Shared Libraries](#) (1594 페이지 참조)

[Importing Shared Libraries](#) (1603 페이지 참조)

[Using LabVIEW Manager Functions in Shared Libraries](#) (1608 페이지 참조)

[Memory Manager](#) (1620 페이지 참조)

[패턴화된 문자열로 함수 사용하기](#) (1425 페이지 참조)

함수 원형 만들기

텍스트 기반 프로그래밍 언어로부터 공유 라이브러리를 만드려면, LabVIEW에서 원형을 만든 후 코드의 세부사항을 모두 완성해야 합니다. LabVIEW가 이 함수 원형을 생성하도록 허용하면, 공유 라이브러리의 기본 코드 구문이 유효함을 확인하는데 도움이 됩니다. 생성하는 원형 소스 파일은 .c 파일이며 전달하려는 파라미터에 대한 C 선언을 포함합니다.

다음 단계를 따라 원형 소스 파일 myshared.c를 만듭니다.

1. 새 VI를 열고 이 VI를 배열 평균으로 저장합니다. 이 함수는 값 배열의 평균을 계산합니다.
2. 블록 다이어그램에 [라이브러리 함수 호출 노드]를 추가합니다.
3. [라이브러리 함수 호출 노드]에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 설정을 선택하여 라이브러리 함수 호출 대화 상자를 엽니다.
4. 라이브러리 이름 또는 경로 컨트롤을 비워 둡니다.



노트

라이브러리 이름 또는 경로 컨트롤을 사용하여 [라이브러리 함수 호출 노드]가 **호출**하는 공유 라이브러리를 지정합니다.

1. 다음 일반 스펙을 입력합니다:
 1. 함수 이름 필드에 avg_num을 입력합니다.
 2. 호출 형식 필드의 C 컨트롤을 선택합니다.
2. 다음 스펙을 사용하여 반환값을 정의합니다:
 1. 라이브러리 함수 호출 대화 상자의 파라미터 탭을 탐색합니다.
 2. 반환 타입의 이름 컨트롤의 기본값을 보다 설명적인 이름인 에러로 변경합니다.
 3. 타입 풀다운 메뉴에서 숫자형을 선택합니다.
 4. 데이터 타입 풀다운 메뉴에서 부호있는 32 비트 정수를 선택합니다.
3. 다음 스펙을 사용하여 a 파라미터를 정의합니다:
 1. 파라미터 리스트 오른쪽에 파라미터 추가 버튼을 클릭합니다.
 2. 이름 필드의 arg1 기본값을 a로 대체합니다.
 3. 타입 풀다운 메뉴에서 배열을 선택합니다.
 4. 데이터 타입 풀다운 메뉴에서 4 바이트 단정도를 선택합니다.
 5. 배열 포맷 풀다운 메뉴에서 배열 데이터 포인터를 선택합니다.
4. 다음 스펙을 사용하여 크기 파라미터를 정의합니다:
 1. 파라미터 리스트 오른쪽에 파라미터 추가 버튼을 클릭합니다.
 2. 이름 필드의 arg2 기본값을 크기로 대체합니다.
 3. 타입 풀다운 메뉴에서 숫자형을 선택합니다.
 4. 데이터 타입 풀다운 메뉴에서 부호있는 32 비트 정수를 선택합니다.

5. 전달 풀다운 메뉴에서 값을 선택합니다.
5. 다음 스펙을 사용하여 avg 파라미터를 정의합니다:
 1. 파라미터 리스트 오른쪽에 파라미터 추가 버튼을 클릭합니다.
 2. 파라미터 필드의 arg3 기본값을 avg로 대체합니다.
 3. 타입 풀다운 메뉴에서 숫자형을 선택합니다.
 4. 데이터 타입 풀다운 메뉴에서 4 바이트 단정도를 선택합니다.
 5. 전달 풀다운 메뉴에서 값의 포인터를 선택합니다.
6. 함수 원형 필드가 다음과 같이 반환값과 세 파라미터를 올바른 순서로 디스플레이하는지 확인합니다. `int32_t avg_num(float *a, int32_t size, float *avg);`



노트 함수 원형 필드에서 볼 수 있는 구문은 기술적으로 정확합니다. 하지만 다음 섹션에서 [라이브러리 함수 호출 노드]가 생성하는 .c 파일이 보다 정확합니다. 왜냐하면 첫 번째 파라미터가 `float a[]`로 나타나기 때문입니다.

1. 확인 버튼을 클릭하여 셋팅을 저장하고 대화 상자를 닫습니다.
2. [라이브러리 함수 호출 노드] 아이콘이 셋팅을 반영하기 위해서 업데이트됩니다.
3. [라이브러리 함수 호출 노드]에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 C 파일 생성을 선택합니다.
4. 파일을 `myshared.c`로 저장합니다.



노트 이 예제에서는 .c 소스 파일을 사용합니다. C++ 라이브러리를 사용하여 작업할 때, 소스 파일의 확장자를 .cpp로 변경합니다.

C++ 이름 디코레이션

헤더 파일에서 C++ 컴파일러 함수 반출 지정자(C++ compiler function export directive)인 `extern "C"{}`를 사용하여, 이름 변환(name mangling)이라는 프로세스를 실행함으로써 C++ 컴파일러가 반출된 함수 이름을 처리할 때 플랫폼과 관계없이 실행되도록 합니다. 다음에 보이는 예제 코드를 참조하십시오:

```
extern "C" { int32_t MyDLLFunction(int32_t nInput, uint32_t
nOutput, void *arg1); }

int32_t MyDLLFunction(int32_t nInput, uint32_t nOutput, void
*arg1)

{

/* Insert Code Here */

}
```



노트 함수의 C++ 데코레이션을 비활성화한 경우, 컴파일러는 함수의 다형성 버전을 생성할 수 없습니다.

함수 원형을 만든 후, **.c 파일을 완성**합니다.

.C 파일 완성하기

텍스트 기반의 프로그래밍 언어로부터 공유 라이브러리를 만들기 위해 **함수 원형을 만든** 이후에, .c 파일을 완성합니다.

라이브러리 함수 호출 노드는 myshared.c에 다음 소스 코드 뼈대를 생성합니다.

```
/* Call Library source File */

#include "extcode.h"

int32_t avg_num(float a[], int32_t size, float *avg);
int32_t avg_num(float a[], int32_t size, float *avg)
{
    /* Insert code here */
}

/* Insert code here */ 공간을 다음 함수 코드로 대체하며 괄호 내에 코드가 들어가는지를 확인합니다.

int i;

float sum = 0;

if(a != NULL)
{
    for(i=0; i < size; i++)
        sum = sum + a[i];
}

else

return (1);
```

```
*avg = sum / size;

return (0);
```

요구되는 라이브러리

이 간단한 예제는 몇몇 단순한 데이터 타입에 대해 `extcode.h` 헤더 파일을 필요로 합니다. `extcode.h`는 로우 레벨 바이트 조작에서부터 데이터 정렬 루틴과 메모리 관리까지 간단하고 복잡한 연산을 수행하는 LabVIEW Manager 함수들의 세트에 대한 접근을 제공합니다. 보다 복잡한 공유 라이브러리를 만들 때, 반드시 모든 관련된 라이브러리에 대한 헤더 파일을 포함해야만 합니다. 예를 들어, Windows 공유 라이브러리 프로젝트는 `windows.h`를 포함할 필요가 있습니다.

공유 라이브러리 내에서 LabVIEW Manager 함수를 사용하려면, `labviewv.lib` 라이브러리 파일을 컴파일된 프로젝트에 포함시켜야 합니다: 이 파일은 `labview\cintools` 디렉토리에 있습니다.

다음 작업을 수행하려면 LabVIEW Manager 함수가 필요합니다:

- LabVIEW의 라이브러리에 전달하거나 전달받는 배열, 문자열, 다른 데이터 구조의 할당, 해제, 크기 조정.
- LabVIEW 경로 데이터 타입을 사용하는 작업.
- 라이브러리 내에서 파일 참조 번호를 사용하는 작업.
- [지원 관리자] 함수 사용.

관리자 함수에 대한 추가적인 정보는 [LabVIEW Manager] 함수를 참조하십시오.

.c 파일을 완성한 후에는, 외부 IDE에 [라이브러리 프로젝트를 만듭니다](#).

공유 라이브러리 파일에서 함수 반입하기

공유 라이브러리 반입 마법사를 사용하여, Windows .dll 파일, macOS .framework 파일 또는 Linux .so 파일 형태의 함수에 대한 래퍼 VI로 구성된 LabVIEW 프로젝트 라이브러리를 생성하거나 업데이트할 수 있습니다. 마법사가 분석할 공유 라이브러리 파일과 헤더 .h 파일의 이름을 제공해야 합니다. 공유 라이브러리 반입 마법사는 C/C++ 구문과 호환 가능하며 C 또는 표준 `_stdcall` [호출 형식](#)을 사용하는 헤더 파일만을 지원합니다.

다음의 단계를 따라 공유 라이브러리 파일에서 함수를 위한 래퍼 VI를 생성하거나 업데이트합니다.

1. 도구 >> 반입 >> 공유 라이브러리를 선택하여 공유 라이브러리 반입 마법사를 실행합니다. 생성 또는 업데이트 모드 지정 페이지가 나타납니다.
2. 이전에 반입하지 않은 공유 라이브러리에서 래퍼 VI를 생성하려면, 공유 라이브러리를 위한 VI 생성을 선택합니다. 이전에 반입하지 않은 공유 라이브러리에서 래

퍼 VI를 업데이트하려면, 공유 라이브러리를 위한 VI 업데이트를 선택하고 리스트에서 해당 라이브러리를 선택합니다.



노트

공유 라이브러리를 위한 VI 업데이트 옵션을 선택하면, 마법사는 이전에 생성된 프로젝트 라이브러리 파일과 해당 프로젝트 라이브러리 파일에 있는 기존 VI를 덮어씁니다. 프로젝트 라이브러리 파일에서 생성된 VI를 다시 반입하지 않고자 하는 경우, VI는 디렉토리에서 그대로 남아있습니다. 마법사는 특정 공유 라이브러리의 각 개별 함수에 대한 가장 최신의 셋팅을 유지합니다. 예를 들어, 세 개의 함수가 있는 공유 라이브러리가 있는 경우, 두번째 함수만을 업데이트할 수 있습니다. 다음에 공유 라이브러리 파일에서 마법사를 실행하면, 첫번째와 세번째 함수에 대해서는 원래의 셋팅을 유지하고, 두번째 함수에 대해서는 새로운 셋팅을 유지합니다.

1. 다음 버튼을 클릭하여 마법사의 공유 라이브러리와 헤더 파일 선택 페이지를 디스플레이합니다.
2. 공유 라이브러리 파일 텍스트 박스에 반입하려는 공유 라이브러리 파일에 대한 경로를 입력하거나, 탐색 버튼을 클릭하여 해당 파일을 탐색하고 선택합니다.
3. 공유 라이브러리가 RT 타겟에 있는 경우처럼, 로컬 컴퓨터에 없는 공유 라이브러리에 대한 VI를 생성하려면, 공유 라이브러리 파일이 로컬 컴퓨터에 없음 확인란에 확인 표시를 하고 공유 라이브러리 파일 텍스트 박스에 공유 라이브러리 파일의 파일 이름을 입력합니다. 경로를 공유 라이브러리 파일에 입력할 필요가 없습니다.



노트 파일 이름을 지정하고 경로를 지정하지 않을 경우, 공유 라이브러리 파일은 반드시 래퍼 VI에 대한 대상 디렉토리에 있어야 하고 또는 시스템 검색 경로를 통해 사용 가능해야 합니다.

1. 헤더(.h) 파일 텍스트 박스에 공유 라이브러리와 함께 사용할 헤더 파일의 경로를 입력하고 탐색 버튼을 클릭하여 해당 파일을 탐색하고 선택합니다. 공유 라이브러리 반입 마법사는 C/C++ 구문과 호환 가능하며 C 또는 표준 (WINAPI) 호출 형식을 사용하는 헤더 파일만을 지원합니다.
2. 다음 버튼을 클릭하여 포함 경로와 선행 처리기 정의 설정 페이지를 디스플레이하고, 헤더 파일을 분석할 때 마법사가 사용할 포함 경로나 선행 처리기 정의 모두를 입력합니다.
3. 다음 버튼을 클릭하고 변환할 함수 선택 페이지를 디스플레이합니다. 대화 상자가 나타나서 LabVIEW가 헤더 파일을 분석하고 있음을 알려줍니다.
4. 변환할 함수 선택 페이지는 공유 라이브러리에서 찾은 모든 함수를 디스플레이합니다. 마법사가 함수를 반입하지 못하는 경우, 리스트에서 반입되지 않은 함수 이름 옆에 검정색 x 문양이 나타납니다. 해당 함수를 선택하여 에러에 대한 가능한 원인을 디스플레이합니다.



노트 공유 라이브러리 파일이 로컬 컴퓨터에 없음 옵션을 선택한 경우, 이 페이지는 헤더 파일에서 선언된 함수만을 나열합니다. 공유 라이브러리 파일이 로컬 컴퓨터에 없음 옵션을 선택하지 않은 경우, 이 페이지는 헤더 파일에서 선언되지 않은 공유 라이브러리 파일에서 참조된 파일도 나열합니다.

1. (옵션) 마법사가 하나 이상의 함수를 반입할 수 없는 경우, 이전 버튼을 클릭하여 기존 포함 경로 또는 선행 처리기 정의를 추가, 제거 또는 업데이트합니다. 에러가 포괄적일 경우, 마법사를 취소하고 각 헤더 파일을 편집하여 파일이 양호한지를 확인한 후에 공유 라이브러리 반입 마법사를 사용하여 래퍼 VI를 생성합니다.
2. 변환할 함수 선택 페이지에서, 각 함수 옆의 확인란에 확인 표시를 하여 변환하려는 함수를 선택합니다. 리스트에서 모든 함수에 대한 래퍼 VI를 생성하려면 모두 확인 버튼을 클릭합니다.
3. 다음 버튼을 클릭하여 프로젝트 라이브러리 셋팅 설정 페이지를 디스플레이합니다.
4. 프로젝트 라이브러리 이름(.lvlib) 텍스트 박스에서 생성하려는 프로젝트 라이브러리 파일의 이름을 입력합니다. LabVIEW는 생성된 VI를 프로젝트 라이브러리 파일 내에 자동적으로 포함하며 파일에 대한 .lvlib 확장을 사용합니다.
5. 마법사가 프로젝트 라이브러리 경로 텍스트 박스에서 생성한 파일을 저장할 디렉토리 경로를 입력합니다. 또한 탐색 버튼을 클릭하여 디렉토리를 탐색하고 선택할 수 있습니다.
6. (옵션) 원래의 공유 라이브러리 파일의 복사본을 대상 디렉토리에 저장하고자 하는 경우, 공유 라이브러리 파일을 대상 디렉토리에 복사 확인란에 확인 표시를 합니다.



노트 마법사는 사용자가 지정한 공유 라이브러리만을 복사합니다. 종속 공유 라이브러리 파일은 복사하지 않습니다. 생성된 VI가 깨지지 않도록 보장하려면, 모든 종속 공유 라이브러리 파일을 대상 폴더에 수동으로 복사하거나 시스템 검색 경로에 추가합니다.

1. 다음 버튼을 클릭하여 에러 핸들링 모드 선택 페이지를 디스플레이합니다.
2. 에러 핸들링 모드 폴다운 메뉴에서 대부분의 래퍼 VI에서 사용할 에러 핸들링 모드를 선택합니다. 예제 블록 다이어그램 필드는 마법사가 래퍼 VI에서 생성하는 기본적인 블록 다이어그램을 보여줍니다. 여러 에러 핸들링 모드를 선택하려면, 마법사를 다시 실행하고, 공유 라이브러리를 위한 VI 업데이트 옵션을 선택하여, 다른 에러 핸들링 모드를 사용하고자 하는 VI만을 다시 생성할 수 있습니다.
3. 다음 버튼을 클릭하여 VI와 컨트롤 설정 페이지를 디스플레이합니다.
4. 생성하고자 하는 VI 및 컨트롤의 설정을 검토합니다. 셋팅 및 이름과 설명 탭에서 옵션을 사용하여, 특정 VI를 편집하고 생성하려는 대부분의 VI 정보를 컨트롤할 수 있습니다. 아이콘은 각 컨트롤의 데이터 타입을 나타냅니다. 느낌표  문양이 LabVIEW가 처리할 수 없는 데이터 타입 옆에 나타납니다. LabVIEW가 지원하지 않는 데이터 타입에서, 마법사를 완료한 후 **사용자 컨트롤을 생성**하고 컨트롤을 생성된 라이브러리에 추가합니다.
5. 다음 버튼을 클릭하여 생성 요약 페이지를 디스플레이합니다.
6. 요약 생성을 검토합니다. 변경을 하려면 뒤로 버튼을 클릭하여 마법사에서 탐색합니다.
7. 생성 요약 페이지에서, 다음 버튼을 클릭하여 특정 공유 라이브러리에서 선택한 함수의 래퍼 VI를 생성합니다. 생성 진행 페이지는 래퍼 VI를 생성할 때 마법사의 진행 정도를 보여줍니다. 생성이 완료되면, 종료 페이지가 나타납니다.
8. 마법사가 닫힐 때 생성된 라이브러리 파일을 열려면, 생성된 라이브러리 열기 확인란에 확인 표시를 합니다.
9. 마법사가 닫힐 때 리포트 생성을 보려면, 리포트 보기 확인란에 확인 표시를 합니다. 이 옵션을 선택하면 리포트가 기본 웹 브라우저에서 열립니다.

10. 마침 버튼을 클릭하여 어플리케이션을 닫습니다.
11. 프로젝트 라이브러리 파일과 생성된 래퍼 VI를 검토합니다. 개별 에러 핸들링 VI를 호출하는 에러 핸들링 모드 VI를 선택하였을 경우 반드시 에러 VI를 열어서 수정해야 합니다.

예제: 공유 라이브러리 파일에서 함수 반입하기를 참고하여 예제 공유 라이브러리 파일 ImportSharedLibraryExample.dll에 포함된 함수에 기반한 VI의 라이브러리를 생성하거나 업데이트할 수 있습니다.

예제: 공유 라이브러리 파일에서 함수 반입하기

다음의 단계를 따라 공유 라이브러리 반입 마법사를 사용하여 DLL 정규식에 있는 함수를 기반으로 하는 VI의 라이브러리를 생성하고 업데이트합니다. 공유 라이브러리 파일에 있는 함수에서 래퍼 VI를 생성하거나 업데이트하려면 **공유 라이브러리 파일에서 함수 반입하기**를 참조합니다.

1. 도구 >> 반입 >> 공유 라이브러리를 선택하여 공유 라이브러리 반입 마법사를 실행합니다. 생성 또는 업데이트 모드 지정 페이지가 나타납니다.
2. 공유 라이브러리를 위한 VI 생성을 선택하고, 다음 버튼을 클릭하여 마법사의 공유 라이브러리와 헤더 파일 선택 페이지를 디스플레이합니다.
3. 공유 라이브러리 파일 텍스트 박스 옆의 탐색 버튼을 클릭하여 labview\examples\dll\ImportSharedLibraryExample\ImportSharedLibraryExample.dll을 탐색 및 선택합니다. 이 파일을 선택하면, LabVIEW는 labview\examples\dll\ImportSharedLibraryExample\ImportSharedLibraryExample.h를 기본 헤더 파일로 지정합니다.



노트

ImportSharedLibraryExample.dll은 macOS에서 지원되지 않습니다.

4. 공유 라이브러리 파일이 로컬 컴퓨터에 없음 확인란에서 확인 표시가 나타나면, 이 표시를 제거하십시오.
5. 다음 버튼을 클릭하여 포함 경로와 선행 처리기 정의 설정 페이지를 디스플레이합니다. 이 DLL에서 다른 포함 경로나 선행 처리기 정의를 추가하지 않아도 됩니다.
6. 다음 버튼을 클릭하고 변환할 함수 선택 페이지를 디스플레이합니다. 대화 상자가 나타나서 LabVIEW가 지정된 헤더 파일을 분석하고 있음을 알려줍니다.
7. 각 함수 옆의 확인란에 확인 표시를 하여 래퍼 VI를 생성하려는 함수를 선택합니다. 기본적으로 나열된 두 개의 함수 모두 확인 표시를 포함합니다. LabVIEW는 선택된 각 함수에 대한 VI를 생성합니다.
8. 다음 버튼을 클릭하여 프로젝트 라이브러리 셋팅 설정 페이지를 디스플레이합니다.
9. 기본적으로, ImportSharedLibraryExample은 프로젝트 라이브러리 이름 (.lvlib) 필드에 나타납니다. LabVIEW는 생성된 VI 모두를 프로젝트 라이브러리 파일 내에 자동적으로 포함하며 이에 .lvlib 파일 확장자를 사용합니다. 또한 기본적으로, labview\user.lib\ImportSharedLibraryExample은 프로젝

트 라이브러리 경로 필드에 나타나며, 공유 라이브러리 파일을 대상 디렉토리에 복사합니다. 확인란은 확인 표시를 포함합니다.

10. 다음 버튼을 클릭하여 에러 핸들링 모드 선택 페이지를 디스플레이합니다.
11. 에러 핸들링 모드 폴다운 메뉴에서 함수는 에러 코드/상태를 반환을 선택합니다. 에러가 발생하면, 이 에러 상태 설명을 사용하여 LabVIEW에서 **에러 클러스터** 정보를 형성할 수 있습니다.
12. 다음 버튼을 클릭하여 VI와 컨트롤 설정 페이지를 디스플레이합니다.
13. 생성하고자 하는 VI 및 컨트롤의 설정을 검토합니다. 왼쪽에서 VI 또는 컨트롤을 선택한 후 셋팅과 이름과 설명 탭의 필드를 사용하여 특정 정보를 확인하고 편집합니다. 기본 설정 셋팅을 다음과 같이 변경합니다:
 - DWTest_Change_Info()
 1. name에 대한 포인터를 초기화하는 기본 문자열 텍스트 박스를 Old로 변경합니다.
 - DWTest_Copy()
 1. outputEnum1의 입력/출력을 출력으로 변경합니다.
 2. outputEnum2의 입력/출력을 출력으로 변경합니다.
 3. outputRunTimeSizeArray의 입력/출력을 출력으로 변경합니다.
 4. outputBool의 입력/출력을 출력으로 변경합니다.
 5. outputString의 입력/출력을 출력으로 변경합니다.

팁 VI와 컨트롤 이름을 더 설명적이거나 더 쉽게 읽을 수 있도록 변경할 수 있습니다. 유사하게도, LabVIEW의 컨트롤 이름은 더욱 쉽게 읽을 수 있도록 여백을 포함할 수 있습니다.

14. 다음 버튼을 클릭하여 생성 요약 페이지를 디스플레이하고, 생성 요약을 확인하여 VI를 생성할 준비가 되었는지를 판단합니다. 변경을 하려면 뒤로 버튼을 클릭합니다.
15. 다음 버튼을 클릭하여 특정 공유 라이브러리에서 선택한 함수의 래퍼 VI를 생성합니다. 생성 진행 페이지는 래퍼 VI를 생성할 때 마법사의 진행 정도를 보여줍니다. 생성이 완료되면, 종료 페이지가 나타납니다.
16. 생성된 라이브러리 열기 및 리포트 보기 확인란에 확인 표시를 하고 종료 버튼을 클릭하여, 마법사를 닫고, 프로젝트 라이브러리 파일을 열고, 기본 브라우저에서 리포트를 봅니다.
17. 리포트를 검토합니다. 이 리포트는 어떤 에러나 경고가 없었음을 보여줍니다.
18. 프로젝트 라이브러리 파일과 안에 생성된 VI를 검토합니다. VI를 일차적으로 생성한 후, 블록 다이어그램 또는 각 VI의 프런트패널 내에 있는 VI를 개별적으로 변경할 수 있습니다. 이후 마법사를 재실행하여 라이브러리 안의 VI를 업데이트할 수 있습니다.
19. ImportSharedLibraryExample.lvlib 프로젝트 라이브러리 파일을 닫습니다.

사용자 컨트롤로 생성된 프로젝트 라이브러리 VI 업데이트

생성된 라이브러리는 완성 상태이며 그 상태로 사용할 수 있습니다. 그러나, 라이브러리 내의 여러 VI를 변경하고 싶을 수도 있습니다. 이렇게 하려면, 마법사를 다시 시작하고 생성 또는 업데이트 모드 지정 페이지에서 공유 라이브러리를 위한 VI 업데이트를 선택합니다. 이 옵션을 사용하면, 이전에 생성한 특정 VI를 업데이트하고 덮어쓸 수 있습니다. 다음 단계를 따라 사용자 컨트롤을 사용하여

ImportSharedLibraryExample.lvlib에 있는 VI의 서브셋을 업데이트합니다.

1. 도구 >> 반입 >> 공유 라이브러리를 선택하여 공유 라이브러리 반입 마법사를 실행합니다. 생성 또는 업데이트 모드 지정 페이지가 나타납니다.
2. 공유 라이브러리를 위한 VI 업데이트를 선택하고 프로젝트 리스트에서 ImportSharedLibraryExample.lvlib를 선택합니다.
3. 다음 버튼을 클릭하여 마법사의 공유 라이브러리와 헤더 파일 선택 페이지를 디스플레이합니다. 공유 라이브러리 파일 텍스트 박스는 labview\examples\dll\ImportSharedLibraryExample\ImportSharedLibraryExample.dll 경로를, 헤더(.h) 파일 텍스트 박스는 labview\examples\dll\ImportSharedLibraryExample\ImportSharedLibraryExample.h 경로를 기본 헤더 파일로 포함합니다.

노트 ImportSharedLibraryExample.dll은 macOS에서 지원되지 않습니다.

4. 다음 버튼을 클릭하여 포함 경로와 선행 처리기 정의 설정 페이지를 디스플레이합니다. 이 DLL에서 다른 포함 경로나 선행 처리기 정의를 추가하지 않아도 됩니다.
5. 다음 버튼을 클릭하고 변환할 함수 선택 페이지를 디스플레이합니다. 대화 상자가 나타나서 LabVIEW가 지정된 헤더 파일을 분석하고 있음을 알려줍니다.
6. 각 함수 옆의 확인란에 확인 표시를 하여 변환하려는 함수를 선택합니다. 기본적으로 나열된 두 개의 함수 모두 확인 표시를 포함합니다. 선택해제 선택 버튼을 클릭하여 확인 표시를 제거합니다.
7. DWTest_Copy() 옆의 확인란에 확인 표시를 해서 이를 업데이트할 함수로 선택합니다.
8. 다음 버튼을 클릭하여 프로젝트 라이브러리 셋팅 설정 페이지를 디스플레이합니다. 이 페이지의 셋팅은 변경하지 않아도 됩니다.
9. 다음 버튼을 클릭하여 에러 핸들링 모드 선택 페이지를 디스플레이합니다. LabVIEW는 특정한 파일을 덮어쓸지 확인하기 위해서 입력 요청합니다. 확인 버튼을 클릭합니다.
10. 에러 핸들링 모드 선택 페이지의 에러 핸들링 모드 풀다운 메뉴에서 함수는 에러 코드/상태를 반환을 선택합니다.
11. 다음 버튼을 클릭하여 VI와 컨트롤 설정 페이지를 디스플레이합니다.
12. 기본 설정 셋팅을 다음과 같이 변경합니다:

- DWTest_Copy()
 1. outputStruct의 컨트롤 타입을 사용자 컨트롤로 변경합니다.
 2. 사용자 컨트롤에 대한 경로 필드 옆의 탐색 버튼을 클릭하여 사용자 컨트롤인 labview\examples\dll\ImportSharedLibraryExample\CustomDefinedControl.cti를 탐색합니다.

13. 다음 버튼을 클릭하여 생성 요약 페이지를 디스플레이합니다.
14. 다음 버튼을 클릭한 후 해당 VI를 업데이트합니다.
15. 종료 페이지에서, 생성된 라이브러리 열기 옵션을 선택하고 종료 버튼을 클릭하여 마법사를 닫고 프로젝트 라이브러리 파일을 엽니다.
16. ImportSharedLibraryExample.lvlib 프로젝트 라이브러리 파일을 닫습니다.

공유 라이브러리 반입 마법사의 문제 해결하기

다음은 **공유 라이브러리 반입 마법사를 사용**과 관련된 일반적인 문제입니다. 제안된 방법으로도 해결되지 않으면, **공유 라이브러리를 디버그**하십시오.

마법사가 함수를 찾을 수 없습니다.

공유 라이브러리 반입 마법사의 변환할 함수 선택 페이지에서 함수 리스트는 지정한 공유 라이브러리 파일의 모든 함수를 나열합니다. 마법사가 헤더 파일을 분석한 후 어떤 함수도 나타나지 않으면, 공유 라이브러리 파일에 대한 경로가 정확한지 라이브러리에 함수가 있는지 확인하십시오.

마법사가 반입할 수 없는 함수를 찾았습니다

공유 라이브러리 반입 마법사는 지정한 공유 라이브러리 파일의 모든 함수를 나열하지만, 마법사가 헤더 파일에서 정확하게 분석할 수 있는 함수만을 반입할 수 있습니다. 변환할 함수 선택 페이지의 함수 리스트에서 함수를 선택할 때 나타나는 경고로 가능한 문제를 미리 확인할 수 있습니다. 헤더 파일이 모든 필요한 함수 선언을 포함하는지 확인합니다. 각 함수 선언은 **호출 형식**, 반환값의 데이터 타입, 정의된 파라미터를 포함해야 합니다. 헤더 파일을 수정하기 위해, 파일을 직접 편집하거나, 마법사의 포함 경로와 선행 처리기 정의 설정 페이지를 사용하여 기호 정의를 추가할 수 있습니다.

헤더 파일로부터의 다음 예제 코드는 사용자가 WIN32를 정의하는 경우 오직 functionA만을 컴파일합니다.

```
#ifdef WIN32 int functionA(int para); #endif
```

LabVIEW가 functionA를 반입하려 하는데 헤더 파일이 WIN32을 정의하지 않은 경우, 직접 헤더 파일을 편집하여 WIN32정의하거나, 공유 라이브러리 반입 마법사에서 WIN32을 정의할 수 있습니다. 마법사에서 WIN32를 정의하려면, 포함 경로와 선행 처리기 정의 설정 페이지의 선행 처리기 정의 텍스트 박스에 WIN32를 입력합니다.

다음 예제 코드는 헤더 파일의 다른 함수를 보여줍니다. 이 상황에서, 사용자가 NIAPI를 정의하지 않았으므로 LabVIEW는 기호를 인식하지 않습니다:

```
NIAPI int functionA(intpara);
```

LabVIEW가 이 함수를 반입하기를 원한다면, 선행 처리기 정의 텍스트 박스에 NIAPI=_stdcall를 추가하여 NIAPI를 정의합니다.

마법사가 선언된 함수를 찾을 수 없습니다.

함수 리스트는 공유 라이브러리 파일에서 나타나는 함수를 보여줍니다. 헤더 파일이 공유 라이브러리 파일에 없는 함수에 대한 선언을 포함하는 경우, 마법사는 이 같은 함수를 나열하지 않습니다. 공유 라이브러리 파일에 존재하는지 여부에 관계없이 헤더 파일에서 선언된 파일 모두를 보려면, 마법사의 공유 라이브러리와 헤더 파일 선택 페이지에서 공유 라이브러리 파일이 로컬 컴퓨터에 없음의 확인란에 확인 표시를 합니다. 그러면 변환할 함수 선택 페이지로 돌아올 때 마법사가 헤더 파일에서 선언된 모든 함수를 나열합니다. 래퍼 VI를 생성하지 전 반드시 이 확인란에서 확인 표시를 제거하십시오.

마법사가 하위 헤더 파일을 찾을 수 없습니다

지정한 헤더 파일이 하위 헤더 파일을 포함하는 경우, 마법사의 포함 경로와 선행 처리기 정의 설정 페이지의 포함 경로 텍스트 박스에서 반드시 하위 파일에 대한 경로를 제공해야 합니다. 하위 헤더 파일에 대한 포함 경로를 지정하지 않으면, 마법사가 헤더 파일을 정확하게 분석할 수 없습니다.

패턴화된 문자열로 함수 사용하기

[패턴화된 문자열로] 함수는 LabVIEW 데이터를 받아서 문자열로 변환합니다. 다양한 통신 함수 또는 I/O 함수와 결합되어 사용될 때 이 문자열은 파일 또는 데이터베이스로 저장되거나 다른 컴퓨터로 전송될 수 있습니다.

[패턴화된 문자열로]에 의해 생성된 문자열은 LabVIEW 문자열입니다. [패턴화된 문자열로] 함수의 앞에 배열이나 문자열 크기 포함? 입력이 참인 경우, LabVIEW 문자열은 32비트 부호있는 정수 데이터 타입인 4 바이트 숫자를 문자열 시작 부분에 가지며 이는 문자열의 길이를 지정합니다. 문자열의 길이를 지정하는 것은 LabVIEW 문자열이 ASCII 문자 제로(0)와 같은 NULL 문자를 포함할 수 있도록 합니다. LabVIEW 문자열이 외부 코드로 전달되어 C 문자열로 사용될 때, C 문자열이 첫번째 NULL 문자에서 종료되는 것으로 해석되기 때문에 문자열에 내장된 NULL 문자는 문제를 일으킬 수 있습니다.

LabVIEW 데이터의 패턴화된 형태를 C 코드에 전달하기 위해서는 패턴화된 LabVIEW 데이터 형태를 2진 문자열 포맷에서 문자나 숫자의 문자열 포맷으로 변환합니다. 예를 들어, 문자열이 다음 5개의 문자로 구성되었다고 가정합니다.

* 문자 35 (#)

* 문자 65 (A)

* 문자 0 (NULL)

* 문자 50 (2)

* 문자 107 (k)

다음 단계를 따라 이전 5개 문자를 2진 문자열 포맷에서 문자와 숫자 문자열 포맷으로 변환합니다.

1. 5개 문자의 10진수 값을 16진수 값으로 변환합니다.

35d = 0x23

65d = 0x41

0d = 0x00

50d = 0x32

107d = 0x6B

2. 16진수 값에 대한 실제 문자나 숫자를 적고 마지막에 단일 NULL 값을 포함합니다.

* 문자 50 (2)

* 문자 51 (3)

* 문자 52 (4)

* 문자 49 (1)

* 문자 48 (0)

* 문자 48 (0)

* 문자 51 (3)

* 문자 50 (2)

* 문자 54 (6)

* 문자 66 (B)

* 문자 0 (NULL)

2진 문자열 포맷에서 문자나 숫자 문자열 포맷으로 변환하면 문자열의 크기가 두배가 됩니다. 하지만, 문자나 숫자 포맷으로 변환하면 LabVIEW 문자열을 C 문자열로 대체해야만 하는 환경에서 문자열을 사용할 때 문자열의 정보를 유지할 수 있습니다.

점으로 표시된 프로퍼티와 메소드 생성하기

점으로 표시된 프로퍼티와 메소드는 참조를 반환하는 프로퍼티가 다른 프로퍼티 또는 메소드와 연계되도록 합니다. 점으로 표시된 프로퍼티와 메소드는 참조를 반환하는 단계별 여러 프로퍼티 노드의 단축 버전입니다. 이 때 프로퍼티에서 반환된 참조는 다른 프로퍼티 또는 메소드의 참조 입력에 연결됩니다. 점으로 표시된 프로퍼티나 점으로 표시된 메소드는 점으로 표시된 프로퍼티에서 반환된 중간 참조를 자동으로 닫습니다. 최종 프로퍼티가 참조를 반환하는 경우, 해당 참조는 자동으로 닫히지 않습니다.

클래스 탐색기 윈도우를 사용하여 점으로 표시된 프로퍼티와 메소드를 생성할 수 있습니다. 점으로 표시할 수 있는 프로퍼티는 참조를 반환하며 프로퍼티와 메소드 리스트의 프로퍼티 이름 옆에 파란색 문양을 가지고 있습니다.

다음 단계에 따라, 점으로 표시된 프로퍼티 또는 메소드를 생성합니다.

1. 점으로 표시된 프로퍼티를 더블 클릭하여, 프로퍼티와 메소드 리스트를 적합한 클래스로 업데이트합니다. 점으로 표시된 프로퍼티는 선택된 프로퍼티 또는 메소드 상에 HTML 링크로 나타납니다. 다른 프로퍼티나 메소드를 선택하는 경우, 이는 점으로 표시된 프로퍼티에 연결되고 상자에 나타납니다.



노트

현재 타입 라이브러리 밖의 클래스에 참조를 반환하는 .NET 또는 ActiveX 프로퍼티는 점으로 표시된 프로퍼티로 만들 수 없습니다. 클래스 탐색기 윈도우에서 프로퍼티를 볼 때, 프로퍼티와 메소드 리스트에서 프로퍼티 이름 옆에는 파란 문양이 나타나지 않습니다.

1. (옵션) 상자 안의 HTML 링크를 클릭하여 점으로 표시된 프로퍼티를 선택한 위치를 찾아서 되돌아가고, 클래스 탐색기를 선택된 아이템으로 업데이트할 수 있습니다.

프로퍼티와 메소드 생성하기

프로퍼티는 VI, 객체 또는 어플리케이션의 속성입니다. 메소드는 VI, 객체, 또는 어플리케이션에서 동작을 수행합니다. 프로퍼티 노드를 사용하여 특정 VI, 객체 또는 어플리케이션의 프로퍼티를 얻거나 설정합니다. 인보크 노드를 사용하여 특정 VI, 객체 또는 어플리케이션에서 동작이나 메소드를 수행합니다.

동일한 VI, 객체 또는 어플리케이션에 대해 여러 개의 프로퍼티와 메소드를 생성할 수 있습니다.

다음 단계에 따라 프로퍼티 노드 또는 인보크 노드를 사용하여 프로퍼티 또는 메소드를 생성하거나, 또는 **다른 방법**을 사용하여 프로퍼티나 메소드를 생성합니다.

1. 프로퍼티 또는 메소드를 생성하려는 해당 객체를 결정합니다. 다음 중 한 가지 방법으로 해당 객체의 참조를 얻습니다:
 - VI—현재 VI에 대해 프로퍼티나 메소드를 생성하려는 경우, 참조를 얻을 필요가 없습니다. 다른 VI에 대해 프로퍼티나 메소드를 생성하려는 경우, [VI 참조 열기] 함수를 블록 다이어그램에 추가합니다. VI 참조 열기
 - 프런트패널 객체—객체의 블록 다이어그램 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 참조를 선택하여 객체의 참조를 생성합니다.
 - 어플리케이션—현재 어플리케이션 인스턴스에 대해 프로퍼티나 메소드를 생성하려는 경우, 참조를 얻을 필요가 없습니다. 다른 어플리케이션 인스턴스에 대해 프로퍼티나 메소드를 생성하려는 경우, [어플리케이션 참조 열기] 함수를 블록 다이어그램에 추가합니다. 어플리케이션 참조 열기
2. 프로퍼티와 메소드 중 무엇을 생성할 것인지 결정합니다.

- 프로퍼티를 생성하려면, 프로퍼티 노드를 블록다이어그램에 추가합니다. 프로퍼티 노드
 - 메소드를 생성하려면, 인보크 노드를 블록다이어그램에 추가합니다. 인보크 노드
3. 객체의 참조를 노드의 참조 입력에 연결합니다. 노드가 자동으로 변경되어 노드에 연결된 객체의 클래스를 보여줍니다. 객체의 서로 다른 클래스에는 서로 다른 프로퍼티와 메소드가 사용 가능합니다.



노트

현재 어플리케이션이나 VI에 대해 프로퍼티나 메소드를 생성하려는 경우, LabVIEW가 기본 설정으로 현재 어플리케이션이나 VI를 사용하기 때문에, 데이터러를 노드의 참조 입력에 연결할 필요가 없습니다. 그러나 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 클래스 선택을 선택하여 어플리케이션 또는 VI 클래스를 직접 선택해야 합니다.

4. 사용할 특정 프로퍼티나 메소드를 선택하려면, 노드의 흰색 영역을 클릭한 후 나타나는 계층구조 메뉴에서 프로퍼티나 메소드를 선택합니다.
5. (옵션) 프로퍼티 노드를 사용하면 여러 프로퍼티를 같은 노드에서 얻거나 설정할 수 있습니다. 같은 프로퍼티 노드에 프로퍼티를 추가하려면, 노드의 흰색 영역에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 원소 추가를 선택하거나, 또는 위치 도구를 사용하여 **노드 크기 조정**을 합니다.
6. (옵션) 프로퍼티를 읽기와 쓰기 사이에서 변경하려면, 프로퍼티에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 읽기로 변경 또는 쓰기로 변경을 선택합니다.
7. 프로퍼티 노드가 객체의 참조나 참조 배열을 반환하는 경우, [참조 닫기] 함수를 사용하여 참조를 닫습니다.



노트 While 루프 또는 For 루프에 프로퍼티를 추가하고 프로퍼티가 참조를 객체에 반환하는 경우, 루프가 반복될 때마다 해당 참조를 닫아야 합니다. 루프 밖으로 프로퍼티를 옮겨서, 루프 실행의 모든 반복이 완료된 후에 참조를 닫을 수 있습니다.

프로퍼티와 메소드를 생성하는 다른 방법

- 클래스 탐색기 윈도우를 사용하여 **프로퍼티 및 메소드를 생성**하고 탐색할 수 있습니다.
- 프런트패널 객체 또는 프런트패널 객체의 블록다이어그램 터미널에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성»프로퍼티 노드 또는 생성»인보크 노드를 선택하고 리스트에서 프로퍼티나 메소드를 선택하여 객체의 프로퍼티나 메소드를 생성할 수 있습니다. 노드가 객체에 내부적으로 연결되어 있기 때문에, 참조를 결과의 프로퍼티 또는 인보크 노드에 연결할 필요가 없습니다. 객체가 라벨을 가지고 있다면, 프로퍼티는 똑같은 라벨을 가집니다. 노드 생성 후에 라벨을 변경할 수 있습니다. 메소드를 선택했다면, 연관된 파라미터가 노드에 나타납니다.
- (Windows) .NET 프로퍼티나 메소드를 생성하려면 **.NET 컨테이너** 또는 **.NET 참조**를 노드의 참조 터미널에 와이어로 연결합니다.
- ActiveX 프로퍼티나 메소드를 생성하려면, **ActiveX 컨테이너** 또는 **오토메이션 참조 번호**를 노드의 참조 터미널에 와이어로 연결합니다.

클래스 탐색기 윈도우로 프로퍼티와 메소드 생성하기

클래스 탐색기 윈도우를 사용하여, 객체 라이브러리를 선택하고 새 프로퍼티나 메소드를 생성할 수 있습니다.

다음 단계에 따라 클래스 탐색기 윈도우를 사용하여 새 프로퍼티 또는 메소드를 생성합니다.

1. 보기>>클래스 탐색기를 선택하여 클래스 탐색기 윈도우를 디스플레이합니다.
2. 객체 라이브러리 풀다운 메뉴에서 라이브러리를 선택합니다.



노트

(Windows) .NET 또는 ActiveX 라이브러리를 선택하려면 .NET>>어셈블리 선택 또는 ActiveX>>타입 라이브러리 선택을 선택하여 각각 어셈블리 선택 또는 타입 라이브러리 선택 대화 상자를 디스플레이해야 합니다. 원하는 어셈블리나 라이브러리의 확인란에 확인 표시를 하고 확인 버튼을 클릭하여 클래스 탐색기 윈도우로 돌아옵니다.

1. 클래스 풀다운 메뉴에서 클래스를 선택합니다. 다음 버튼을 사용하여 클래스를 탐색합니다.
 - 보기 선택 버튼을 클릭하여 클래스 풀다운 메뉴와 프로퍼티와 메소드 리스트에서 아이템의 가나다 순서로 보기와 계층구조로 보기 사이를 토글합니다.
 - 검색 버튼을 클릭하여 클래스 탐색기 검색 대화 상자를 시작합니다.
2. 클래스 탐색기 윈도우의 프로퍼티와 메소드 리스트에서 프로퍼티 또는 메소드를 선택합니다. 점으로 표시할 수 있는 프로퍼티는 이 리스트의 프로퍼티 이름 옆에 파란색 문양을 가지고 있습니다. **점으로 표시된 프로퍼티를 생성**하려면, 점으로 표시할 수 있는 프로퍼티를 더블 클릭합니다. 선택한 프로퍼티 또는 메소드는 선택된 프로퍼티 또는 메소드 상자에 나타납니다.



노트 **LabVIEW 클래스**는 인보크 노드를 지원하지 않습니다. LabVIEW 클래스 메소드는 멤버 VI입니다. 생성 버튼을 사용하거나 메소드를 직접 블록 다이어그램으로 끌면, SubVI가 나타납니다. LabVIEW 클래스 메소드는 인보크 노드가 아닌 subVI로 나타납니다. 이 LabVIEW 버전으로는 LabVIEW 클래스 프로퍼티를 정의할 수 없습니다.

1. 생성 버튼이나 쓰기 생성 버튼을 클릭하여, 선택한 프로퍼티나 메소드를 포함하는 노드를 마우스 커서에 붙이고 블록 다이어그램에 추가합니다. 생성 버튼은 읽기용 프로퍼티나 메소드를 생성합니다. 쓰기 전용 프로퍼티를 선택하면 이 버튼은 비활성화됩니다. 쓰기를 위한 프로퍼티를 생성하려면, 쓰기 생성 버튼을 클릭합니다. 쓰기 생성 버튼은 메소드나 읽기 전용 프로퍼티가 선택되었을 때 비활성화됩니다. 또한, 프로퍼티와 메소드 리스트에서 직접 프로퍼티나 메소드를 끌어와서 블록 다이어그램에 놓아도 됩니다.

2. 추가로 다른 프로퍼티나 메소드를 생성하고 블록다이어그램에 놓으려면, 단계 2부터 단계 5까지를 반복합니다.

.NET와 ActiveX 이벤트 등록 및 핸들링하기

.NET 또는 ActiveX 이벤트를 다룰 때 이벤트를 등록(1~5단계 확인)하고 해당 이벤트를 핸들링할 콜백 VI를 생성(6~7단계 확인)해야 합니다.



노트

National Instruments는 이벤트를 처리할 필요가 없어지면 [이벤트 등록 해제] 함수를 사용하여 이벤트를 등록 해제할 것을 권장합니다. 이벤트를 등록 해제하지 않으면 해당 이벤트를 처리하는 이벤트 구조가 없어도 VI가 실행되는 한 계속해서 LabVIEW는 이벤트를 생성하고 큐에 넣습니다. 따라서 메모리가 소모되고, 이 이벤트로 인해 프론트패널 잠금이 활성화되고, VI가 작동하지 않을 수 있습니다.

다음 단계를 따라 LabVIEW에서 .NET과 ActiveX 이벤트를 등록하고 처리합니다.

1. **.NET 컨테이너** 또는 **ActiveX 컨테이너**에 새 컨트롤을 생성합니다. 또한 [컨스트럭터 노드]를 사용하여 **.NET 객체를 생성**하거나 오토메이션 참조 번호를 사용하여 LabVIEW로부터 **ActiveX 객체를 호출**할 수 있습니다.
2. 블록다이어그램에 [이벤트 콜백 등록] 함수를 추가합니다.
3. 참조 번호 또는 객체를 [이벤트 콜백 등록] 함수의 이벤트 입력에 연결합니다.
4. [이벤트 콜백 등록] 함수의 이벤트 입력 옆에 있는 아래 화살표를 클릭하고 바로 가기 메뉴에서 이벤트를 선택합니다.
5. (옵션) 콜백 VI에 전달하려는 데이터를 사용자 파라미터 입력에 연결합니다. 클러스터를 포함하여 모든 LabVIEW 데이터 타입을 이 파라미터에 연결할 수 있습니다.
6. VI 참조 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 콜백 VI 생성을 선택합니다. LabVIEW는 선택한 이벤트와 사용자 파라미터에 연결한 데이터에 기초하여 VI를 생성합니다. LabVIEW는 콜백 VI를 VI 참조 입력에 연결합니다.



노트 사용하려는 VI의 커넥터 팬이 이벤트 데이터의 커넥터 팬과 일치하는 한, 기존의 VI를 콜백 VI로 사용할 수 있습니다. 콜백 VI는 **재호출**이어야 하고 콜백 VI에 대한 참조는 엄격한 타입이어야 합니다.

1. 콜백 VI를 열고 블록다이어그램을 편집하여 이벤트를 처리합니다. 콜백 VI는 다음의 요소를 포함합니다.
 - 이벤트 공통 데이터는 다음의 요소를 포함합니다.
 - 이벤트 소스는 LabVIEW, .NET, 또는 ActiveX와 같은 이벤트의 소스를 지정하는 숫자 컨트롤입니다. 1의 값은 ActiveX 이벤트를 나타냅니다. 2의 값은 .NET 이벤트를 나타냅니다.
 - 이벤트 타입은 어떤 이벤트가 발생했는지 지정합니다. 이는 사용자 인터페이스 이벤트에서는 열거형 타입이며 .NET, ActiveX와 다른 이벤트 소스에서는 32비트 부호없는 정수 타입입니다. .NET와 ActiveX 이벤트일 경우, 이벤트 타입은 등록된 이벤트의 메소드 코드 또는 ID를 나타냅니다.

- 타임스탬프는 이벤트가 생성된 시간을 밀리초 단위로 지정하는 타임스탬프입니다.
- 컨트롤 참조는 이벤트가 발생한 .NET, ActiveX 객체, 또는 오토메이션 참조 번호의 참조입니다.
- 이벤트 데이터는 콜백 VI가 처리하는 이벤트에 특정한 파라미터의 클러스터입니다. LabVIEW는 [이벤트 콜백 등록] 함수로부터 이벤트를 선택할 때 적절한 이벤트 데이터를 결정합니다. 이벤트에 관련된 데이터가 없는 경우, LabVIEW는 콜백 VI에서 이 컨트롤을 생성하지 않습니다.
- 이벤트 데이터 출력은 콜백 VI가 처리한 이벤트에 대한 변경 가능한 파라미터의 클러스터입니다. 이 원소는 이벤트가 출력 파라미터를 가지고 있을 때에만 사용 가능합니다.
- (옵션) 사용자 파라미터는 .NET 또는 ActiveX 객체가 이벤트를 생성할 때 콜백 VI에 전달하려는 데이터입니다.

노트 LabVIEW는 이벤트 데이터 입력을 이벤트 데이터 출력 출력에 자동으로 연결하지 않습니다. 그러므로, 이벤트 데이터 입력의 일부를 콜백 VI를 통해 전달하려면, 이벤트 데이터 입력을 이벤트 데이터 출력 출력에 직접 연결해야 합니다.

.NET 컨테이너에서 .NET 컨트롤 생성하기

다음 단계를 따라 .NET 컨테이너에 .NET 컨트롤을 생성합니다.



노트

LabVIEW에서 .NET 객체를 생성하고 이 객체와 통신하려면 LabVIEW와 함께 설치되는 .NET CLR 4.0이 필요합니다. .NET 2.0 혼합 모드 어셈블리를 로드하려면 .NET 2.0 설정 파일을 사용해야 합니다. LabVIEW에서 .NET 제한에 대한 더 자세한 정보는 LabVIEW에서 .NET를 사용할 때의 요구조건을 참조하십시오. National Instruments는 LabVIEW 프로젝트 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

1. 프런트패널 윈도우에 .NET 컨테이너를 추가합니다.
2. 프런트패널 윈도우의 .NET 컨테이너에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 .NET 컨트롤 삽입을 선택하여 .NET 컨트롤 선택 대화 상자를 디스플레이합니다.
3. 어셈블리 풀다운 메뉴에서 어셈블리를 선택합니다. 또한 탐색 버튼을 클릭하여 프라이빗 어셈블리에 접근할 수 있습니다. .NET 어셈블리는 .dll 파일 타입입니다. 프라이빗 어셈블리를 선택한 후 다시 이 대화 상자를 띄우면 어셈블리 풀다운 메뉴에 선택한 어셈블리가 나타납니다.



노트 프라이빗 어셈블리를 사용하는 VI를 다른 폴더나 컴퓨터로 이동하는 경우, 관련된 프라이빗 .NET 어셈블리 파일을 LabVIEW가 검색할 수 있는 서브디렉토리로 이동하거나 파일을 Global Assembly Cache(GAC)에 저장해야 합니다. 프라이빗 어셈블리를 사용하는 VI를 공유 라이브러리 또는 독립 어플리케이션에 만들려는 경우, LabVIEW는 관련된 프라이빗 .NET 어셈블리 파일들을 라이브러리 또는 어플리케이션과 같은 디렉토리의 data 서브디렉토리에 복사합니다.

1. 컨트롤 리스트에서 .NET 컨트롤을 선택한 후 확인 버튼을 클릭합니다. 프런트패널 윈도우의 .NET 컨테이너에 .NET 컨트롤이 나타납니다.
2. 블록다이어그램에서 .NET 컨트롤을 프로퍼티 노드, 인보크 노드, 또는 이벤트 콜백 등록 노드에 연결하여 프로퍼티나 메소드를 설정하거나 컨트롤의 이벤트를 처리합니다.

.NET 객체의 참조를 닫기 위해 [참조 닫기] 함수를 사용하지 마십시오. 참조를 열어두어 VI를 닫을 때까지 컨트롤이 올바르게 디스플레이하도록 합니다. LabVIEW는 사용자가 VI를 닫으면 자동으로 참조를 닫습니다. 이와 같은 권장사항은 블록다이어그램 .NET 객체의 참조 관리 방법과 다릅니다.



팁 특정 어셈블리를 로드하거나 어셈블리의 객체에 접근하는데 문제가 발생하는 경우, 어셈블리 호출을 디버그하거나 메모리의 .NET 어셈블리 대화 상자를 사용하여 메모리의 어셈블리가 올바른 버전인지 확인합니다.



노트 .NET 컨트롤은 자동으로 사용자 인터페이스 실행 시스템을 사용합니다.

또한 추후 사용을 위해 .NET 컨트롤을 컨트롤 팔레트에 추가할 수 있습니다.



관련 정보

[LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건](#) (745 페이지 참조)

[LabVIEW에서 .NET 2.0, 3.0, 3.5 어셈블리 로드하기](#) (1435 페이지 참조)

[LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건](#) (745 페이지 참조)

[LabVIEW에서 프로젝트 관리하기](#) (290 페이지 참조)

[LabVIEW에서 .NET 사용하기](#) (743 페이지 참조)

[블록다이어그램에서 .NET 객체 생성하기](#) (1432 페이지 참조)

[.NET 어셈블리 호출 디버그하기](#) (1437 페이지 참조)

[LabVIEW에서 멀티태스킹](#) (707 페이지 참조)

[.NET 서브팔레트 생성하기](#) (797 페이지 참조)

블록다이어그램에서 .NET 객체 생성하기

LabVIEW를 사용하여 타사 .NET 서버와 연관된 객체에 접근합니다. 블록다이어그램에 보이거나 보이지 않는 .NET 객체를 생성하여 해당 객체에 대한 프로퍼티나 메소드를 설정하거나 이벤트를 핸들링할 수 있습니다. 예를 들어, 블록다이어그램에 .NET 컨테이너를 놓은 후 타사 브라우저에 접근하는 컨테이너의 내부에 버튼을 놓습니다. 다음 단계를 따라 블록다이어그램에 .NET 객체를 생성하고 설정합니다.

단계 1: 컨스트럭터 노드 설정하기

블록다이어그램에 컨스트럭터 노드를 추가하여 .NET 컨스트럭터 선택 대화 상자를 시작합니다. 이후 .NET 객체에 대한 어셈블리, 객체 및 컨스트럭터를 선택합니다.

.NET 컨스트럭터 선택 대화 상자의 각 구성요소 설정에 대한 더 중요한 상세 정보는 다음 리스트를 참조하십시오.

- 어셈블리—이 구성요소는 Global Assembly Cache (GAC)의 모든 퍼블릭 어셈블리를 나열합니다. 프라이빗 어셈블리에 접근하려면, 탐색 버튼을 클릭합니다. 프라이빗 어셈블리를 탐색할 때에는 .NET 어셈블리는 .dll 파일 타입임에 유의하십시오. 프라이빗 어셈블리를 선택한 후 다시 이 대화 상자를 띄우면 어셈블리 풀다운 메뉴에 선택한 어셈블리가 나타납니다.



노트

프라이빗 어셈블리를 사용하는 VI를 다른 폴더나 컴퓨터로 이동하는 경우, 관련된 프라이빗 .NET 어셈블리 파일을 LabVIEW가 검색할 수 있는 서브디렉토리로 이동하거나 파일을 Global Assembly Cache (GAC)에 저장해야 합니다. 프라이빗 어셈블리를 사용하는 VI를 공유 라이브러리 또는 독립 어플리케이션에 만들려는 경우, LabVIEW는 관련된 프라이빗 .NET 어셈블리 파일들을 라이브러리 또는 어플리케이션과 같은 디렉토리의 data 서브디렉토리에 복사합니다.

- 객체—이전 단계에서 선택한 어셈블리에 의해 사용가능한 객체 클래스가 결정됩니다. 객체를 선택한 후, 해당 클래스의 컨스트럭터가 .NET 컨스트럭터 선택 대화 상자의 컨스트럭터 섹션에 나타납니다.
- 컨스트럭터—컨스트럭터를 선택한 후 확인 버튼을 클릭합니다. 이후 LabVIEW는 컨스트럭터 노드에서 선택한 클래스의 이름을 디스플레이합니다.

단계 2: .NET 객체에 대한 프로퍼티, 메소드 또는 이벤트 설정하기

- .NET 컨스트럭터 노드는 .NET 객체의 인스턴스에 대한 참조를 생성합니다. 참조를 갖게 된 이후에는 .NET 객체에 대한 작업을 할 수 있게 됩니다. 객체에 작업을 수행하려면, 컨스트럭터 노드의 새 참조 출력을 프로퍼티 노드, 인보크 노드 또는 이벤트 콜백 등록에 연결한 후 해당 객체에 대한 프로퍼티나 메소드를 설정하거나, 이벤트를 핸들링합니다.
- [참조 닫기] 함수를 사용하여 .NET 객체에 대한 참조를 닫고 LabVIEW가 해당 객체에 할당한 메모리를 해제합니다.

팁 및 문제 해결 정보

- .NET CLR 4.0은 LabVIEW와 함께 설치됩니다. .NET 2.0 혼합 모드 어셈블리를 로드하려면 반드시 .NET 2.0 설정 파일을 사용해야 합니다.
- 특정 어셈블리를 로드하거나 어셈블리에서 객체에 접근할 때 문제가 발생하는 경우, 어셈블리 호출을 디버그하거나 메모리의 .NET 어셈블리 대화 상자를 사용하여 메모리의 어셈블리가 올바른 버전인지 확인합니다.

관련 정보

[LabVIEW에서 .NET 2.0, 3.0, 3.5 어셈블리 로드하기](#)

[LabVIEW에서 .NET 어셈블리 사용에 대한 요구조건](#)

[LabVIEW에서 .NET 사용하기](#)

[프로퍼티와 메소드 생성하기](#)

[.NET와 ActiveX 이벤트 등록 및 핸들링하기](#)

정적 .NET 프로퍼티와 메소드 사용하기

LabVIEW는 정적 .NET 프로퍼티와 메소드를 지원합니다. .NET 서버 참조 번호를 프로퍼티 노드 또는 인보크 노드에 연결하지 않고도 클래스에서 정적 프로퍼티와 메소드를 실행할 수 있습니다.



노트

LabVIEW에서 .NET 객체를 생성하고 이 객체와 통신하려면 LabVIEW와 함께 설치되는 **.NET CLR 4.0**이 필요합니다. .NET 2.0 혼합 모드 어셈블리를 로드하려면 **.NET 2.0 설정 파일**을 사용해야 합니다. LabVIEW에서 .NET 제한에 대한 더 자세한 정보는 [LabVIEW에서 .NET을 사용할 때의 요구조건](#)을 참조하십시오. National Instruments는 [LabVIEW 프로젝트](#) 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

다음 단계를 따라 정적 프로퍼티와 메소드를 .NET 객체와 함께 사용합니다.

1. 다음 노드 중에서 하나를 블록 다이어그램에 추가합니다:
 - 프로퍼티 노드
 - 인보크 노드
2. 노드에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 클래스 선택 >> .NET >> 탐색을 선택하여 어셈블리로부터 객체 선택 대화 상자를 디스플레이합니다.
3. 어셈블리 풀다운 메뉴에서 클래스를 선택합니다.
4. 객체 리스트박스에서 객체를 선택합니다.
5. 확인 버튼을 클릭합니다.
6. 프로퍼티 노드 또는 인보크 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 정적 프로퍼티 또는 메소드를 선택합니다. LabVIEW는 프로퍼티 또는 메소드 이름의 처음에 [S]를 붙여 정적 프로퍼티와 메소드를 나타냅니다.



노트 프로퍼티 노드를 크기 조정하여 더 많은 프로퍼티를 포함시키고 그 프로퍼티 중 일부가 정적 프로퍼티가 아닌 경우, 컨스트럭터 노드를 사용하여 **.NET 객체**를 생성함으로써 .NET 서버 참조를 프로퍼티 노드에 연결해야 합니다.

LabVIEW에서 .NET 2.0, 3.0, 3.5 어셈블리 로드하기

.NET Framework 4.0을 사용하면 모든 버전의 .NET Framework에서 만든 순수 관리 어셈블리와 .NET 4.0에서 만든 혼합 모드 어셈블리를 로드할 수 있습니다. LabVIEW는 기본적으로 .NET Common Language Runtime (CLR) 4.0을 로드하지만 LabVIEW가 CLR 4.0을 사용하여 CLR 2.0을 타겟으로 하는 .NET 혼합 모드 어셈블리를 로드하도록 강제 지정할 수 있습니다.



노트

.NET Framework 2.0, 3.0, 3.5는 CLR 2.0을 사용합니다. 그러나 LabVIEW가 CLR 2.0을 사용하도록 설정할 수는 없습니다. 이전 버전의 CLR을 타겟으로 하는 어셈블리는 CLR 4.0에서 실행은 되지만, 예기치 않은 동작을 할 수 있습니다.

다음의 객체가 CLR 2.0을 타겟으로 하는 .NET 혼합 모드 어셈블리를 로드할 수 있도록 하려면 아래에 설명되는 설정 파일을 사용하십시오.

- LabVIEW 2013
- LabVIEW 2013으로 생성한 실행 파일(EXE)
- LabVIEW 밖에서 생성되었지만, interop 어셈블리나 LabVIEW로 생성한 DLL을 호출하는 실행 파일(EXE)

다음 단계에 따라 LabVIEW가 CLR 2.0을 타겟으로 하는 .NET 혼합 모드 어셈블리를 로드하도록 설정합니다.

1. 텍스트 편집기를 엽니다.
2. 다음 텍스트를 텍스트 편집기에 복사합니다.


```
<?xml version="1.0" encoding="utf-8" ?> <configuration> <startup useLegacyV2RuntimeActivationPolicy="true"> <supportedRuntime version="v4.0.30319"> </startup> </configuration>
```
3. LabVIEW.exe와 같은 디렉토리에 새 설정 파일을 저장합니다. 파일의 이름을 LabVIEW.exe.config로 합니다.
4. LabVIEW를 다시 시작합니다.

이제 LabVIEW는 CLR 2.0을 타겟으로 하는 .NET 혼합 모드 어셈블리를 로드할 수 있습니다.



노트 .NET 2.0 혼합 모드 어셈블리를 사용하는 LabVIEW VI 또는 프로젝트에서 독립 어플리케이션을 만드는 경우, 어플리케이션의 사용자는 시스템에 .NET Framework 2.0을 설치해야 하고 이름이 application name.exe.config인 설정 파일을 어플리케이션과 같은 디렉토리에 두어야 합니다. 이 설정 파일의 텍스트는 이전 과정의 단계 2에서 언급된 설정 파일의 텍스트와 같아야 합니다. 사용자가 자신의 설정 파일을 생성하지 않아도 되게 하려면, 설정 파일을 포함하는 **설치 프로그램을 만들도록** 고려해 보십시오.

메모리의 .NET 어셈블리 변경 사항 로드하기

메모리에 있는 어셈블리를 변경하면, LabVIEW에서 강제로 어셈블리를 다시 로드해야만 변경 사항에 접근할 수 있습니다. 다음 단계를 따라 LabVIEW에서 강제로 어셈블리의 변경 사항을 로드합니다.



노트

어셈블리를 참조하는 모든 노드를 VI에서 삭제한다 하더라도, 어셈블리는 메모리에 남아 있습니다.

1. 변경한 어셈블리의 이름, 버전 번호, 디스크상 경로를 확인합니다.
2. 메모리의 .NET 어셈블리 대화 상자를 사용하여 수정된 어셈블리 버전이 메모리에 있는 모든 LabVIEW 프로젝트를 확인합니다. 해당 어셈블리를 사용하는 각 프로젝트에 대해, 메모리에 있는 어셈블리의 버전 번호 및 경로가 변경한 어셈블리의 버전 번호 및 경로와 일치하는지 확인합니다. 이 정보가 일치하지 않는 경우, LabVIEW는 사용자가 업데이트한 것과 다른 어셈블리 복사본을 사용하고 있는 것입니다. [어셈블리 호출을 디버그](#)하여 LabVIEW에서 올바른 어셈블리를 로드할 수 있습니다.
3. 수정된 어셈블리를 참조하는 모든 프로젝트를 닫고 다시 엽니다.

노트 주요 어플리케이션 인스턴스에 있는 VI가 수정된 어셈블리를 참조하는 경우, 변경 사항에 접근하려면 LabVIEW를 다시 시작해야 합니다. National Instruments는 [LabVIEW 프로젝트](#) 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

어떤 .NET 어셈블리 버전을 사용할지 지정하기

컴퓨터에 같은 어셈블리의 버전이 여러 개 있는 경우, 다음 상황에서 어떤 어셈블리 버전을 사용할지 지정해야 합니다:

- 처음에 .NET 컨스트럭터 노드가 로드하도록 설정한 어셈블리 버전을 나중에 다른 어셈블리 버전으로 변경하고 싶습니다. LabVIEW는 처음 사용한 어셈블리 버전을 어플리케이션과 연관시킵니다. 그러므로, 설정 파일을 생성하기 전에는 다른 버전의 프로퍼티, 메소드, 동작을 사용할 수 없습니다.
- 어플리케이션을 여러 어셈블리 버전이 있는 다른 컴퓨터에 배포했습니다. 어떤 어셈블리 버전을 사용할지 지정하면 배포된 어플리케이션이 올바른 버전을 감지하고 사용하게 됩니다. 설정 파일을 어플리케이션과 함께 배포해야 합니다.

LabVIEW 프로젝트 또는 생성한 어플리케이션이 .NET 어셈블리의 특정 버전을 참조하도록 하려면 프로젝트를 닫고, 다음 파일에서 이탤릭체 코드를 이 토픽의 후반에 나오는 테이블을 따라 대체하여 아래 파일과 비슷한 [.NET 설정 파일](#)을 생성합니다.

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      < probing privatePath="Assembly\ver2"/ >
    </assemblyBinding>
  </runtime>
</configuration>
```



```

<dependentAssembly>
  <assemblyIdentity name=" AssemblyName "
    publicKeyToken=" xee7d93a67000dr7 "
    culture="Neutral" />
  <bindingRedirect oldVersion=" oldVersionNumber "
    newVersion=" newVersionNumber " />
</dependentAssembly>
</assemblyBinding>
</runtime>
</configuration>

```

대체할 코드	가이드라인
probing privatePath="Assembly\ver2"	LabVIEW가 어셈블리를 로드하는 디렉토리를 프로젝트 안에서 지정합니다. 어셈블리가 Global Assembly Cache (GAC)가 확실한 경우, 설정 파일에서 이 라인을 생략합니다. 그렇지 않으면, LabVIEW는 다른 위치 를 검색하기 전에 이 디렉토리를 가장 먼저 검색합니다.
어셈블리 이름	여러 버전이 있는 어셈블리의 이름을 나타냅니다.
xee7d93a67000dr7	어셈블리의 고유 식별 키. Microsoft Developer Network (MSDN)에서 강력한 이름 도구를 사용하여 이 키를 생성하십시오.
oldVersionNumber	LabVIEW가 현재 사용하고 있는 어셈블리 버전을 n.n.n.n 형태로 나타냅니다. LabVIEW가 어셈블리의 여러 다른 버전을 사용하는 것을 방지하려면 여기서 n.n.n.n - n.n.n.n의 형태를 사용하여 버전의 범위를 지정합니다.
newVersionNumber	oldVersionNumber 대신 LabVIEW가 사용할 어셈블리의 버전을 나타냅니다.

.NET 설정 파일을 LabVIEW와 함께 사용하는 예는 ni.com에서 기술지원 데이터베이스를 참조하십시오.

.NET 어셈블리 호출 디버깅하기

LabVIEW의 메모리에 있는 어셈블리 버전이 사용자가 사용하려는 버전과 다를 때, .NET 객체는 예상치 못한 동작을 보일 수 있습니다. 이러한 .NET 어셈블리 충돌을 다수 진단하려면, 메모리의 .NET 어셈블리 대화 상자를 사용합니다. 이 대화 상자는 메모리의 어셈블리에 대한 식별 정보를 디스플레이하기 때문에, 이를 통해 LabVIEW가 올바른 버전의 어셈블리를 로드했는지 확인할 수 있습니다.

다음 단계를 따라 .NET 어셈블리 호출을 디버깅합니다.

1. LabVIEW에서 로드하려는 어셈블리의 이름, 버전 번호, 디스크에서의 위치를 결정합니다. 어셈블리가 **LabVIEW가 어셈블리를 찾을 때 사용하는 검색 경로** 안에 있는지 확인합니다.

2. 보기>>메모리의 .NET 어셈블리를 선택하여 메모리의 .NET 어셈블리 대화 상자를 디스플레이합니다.
3. 대화 상자에서, 로드하려는 어셈블리의 이름을 찾습니다.
4. 대화 상자의 어셈블리 정보와 단계 1에서 원하는 어셈블리의 식별 정보 사이에 불일치가 있는지 확인합니다. 불일치가 없으면서 LabVIEW에 최근 어셈블리 변경이 반영되지 않은 경우, **강제로 LabVIEW가 업데이트된 어셈블리를 로드하도록 합니다.**
5. 메모리의 어셈블리를 원하는 어셈블리로 대체하는데 다음 과정 중 하나를 선택합니다.
 - 로드된 어셈블리를 **어셈블리 검색 경로** 밖으로 이동합니다. 이 동작으로 LabVIEW는 다른 버전의 동일한 어셈블리를 강제로 검색하게 됩니다.
 - 프로젝트 설정 파일을 생성하여 **원하는 어셈블리 버전을 LabVIEW가 강제로 로드하도록 합니다.**
6. 어셈블리를 호출하는 프로젝트를 닫고 다시 엽니다. 이 동작으로 LabVIEW는 강제로 올바르지 않은 버전의 어셈블리를 메모리에서 로드 해제한 후 올바른 버전을 로드하게 됩니다.



노트

올바르지 않은 어셈블리를 로드한 VI가 프로젝트의 일부가 아닌 경우, 올바르지 않은 어셈블리를 메모리에서 지우려면 LabVIEW를 다시 시작해야 합니다. 이는 **LabVIEW 프로젝트** 내에서만 .NET 객체를 사용할 것을 적극 권장합니다.

ActiveX

ActiveX를 지원하는 어플리케이션에 접근하기

다음 단계에 따라 LabVIEW에서 ActiveX 활성화 어플리케이션에 접근합니다:

1. VI의 프런트패널에 오토메이션 참조 번호 컨트롤을 추가합니다.
2. 오토메이션 참조 번호 컨트롤의 ActiveX 클래스를 선택하여 접근하려는 ActiveX 객체를 지정합니다.
3. [오토메이션 열기] 함수를 사용하여 ActiveX 객체에 대한 참조를 엽니다. 참조를 열면, LabVIEW가 참조된 ActiveX 객체와 통신하게 됩니다.
4. 참조를 프로퍼티 노드 또는 인보트 노드로 전달하여 객체의 프로퍼티에 접근하거나 메소드를 인보크합니다. 이 중 하나의 노드를 클릭하면, 해당 객체에서 사용할 수 있는 모든 프로퍼티 또는 메소드를 LabVIEW가 디스플레이합니다. 객체는 COM (Component Object Model) 인터페이스를 구현하며, 사용가능한 프로퍼티 및 메소드를 결정합니다.

특정 ActiveX 객체에 대해 기대하는 프로퍼티나 메소드를 LabVIEW가 디스플레이하지 않는 경우, 객체가 여러 COM 인터페이스를 구현할 수 있습니다. LabVIEW가 한 번에 한 객체가 구현하는 오직 하나의 COM 인터페이스로부터 프로퍼티나 메소드를 디스플레이하는 것일 수 있습니다. [배리언트를 데이터로] 함수를 사용하여 객체가 구현하는 다른 COM 인터페이스의 프로퍼티와 메소드를 LabVIEW로 하여금 강제로 디스플레이하게 하여 객체를 다른 COM 인터페이스로 캐스트할 수 있습니다.

5. 객체를 메모리에서 삭제하려면 [참조 닫기] 함수를 사용합니다.

이전 단계를 사용하여 Microsoft Office 어플리케이션과 같은 일반적인 ActiveX 활성화 어플리케이션에 접근할 수 있습니다. 예를 들어, Microsoft Excel이 ActiveX 활성 어플리케이션입니다. 엑셀에 대한 참조를 열 수 있을 뿐만 아니라 엑셀을 열어 사용자의 스크린에 나타나게 하고, 워크북을 생성하고, 스프레드시트를 생성하고, 테이블을 LabVIEW에서 엑셀 스프레드시트로 작성하는 프로퍼티와 메소드를 호출할 수 있습니다.

LabVIEW를 엑셀 클라이언트로 사용하는 예제는 labview\examples\Connectivity\Excel 디렉토리의 Excel - Write Table VI를 참조하십시오.

 예제 열기  관련 예제 찾기



노트

ActiveX 사용자 인터페이스를 포함하는 어플리케이션은  아이콘과 함께 나타납니다. 아이콘을 클릭하여 사용자 인터페이스의 객체를 선택합니다.

ActiveX 프로퍼티 접근하기

다음 방법으로 ActiveX 객체에 대한 프로퍼티에 접근할 수 있습니다:

- ActiveX 컨트롤 프로퍼티 브라우저—ActiveX 객체가 컨테이너에 있을 때에만 사용 가능합니다.
- ActiveX 프로퍼티 페이지—오직 ActiveX 객체가 컨테이너에 있을 때에만 사용 가능하며, 모든 ActiveX 객체에서 사용 가능하지는 않습니다.
- 프로퍼티 노드—프로그램적으로 ActiveX 프로퍼티를 설정할 때 사용합니다.



관련 정보

[ActiveX 컨트롤 프로퍼티 브라우저 사용하기](#) (1445 페이지 참조)

[ActiveX 프로퍼티 페이지 사용하기](#) (1444 페이지 참조)

[프로퍼티와 메소드 생성하기](#) (1427 페이지 참조)

ActiveX 서버로 LabVIEW 활성화하기

ActiveX 서버로 LabVIEW를 활성화할 때, LabVIEW는 어플리케이션 클래스와 버추얼 인스트루먼트 클래스의 두 개의 클래스를 반환합니다. 어플리케이션 클래스는 버추얼 인스트루먼트 클래스에 대한 참조를 엽니다.

다음 단계를 따라 ActiveX 서버로 LabVIEW를 활성화합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 VI 서버를 선택합니다.
3. VI 서버 페이지의 프로토콜 섹션에서 ActiveX 확인란에 확인 표시를 합니다.
4. 필요한 보안 수준에 따라 페이지의 접근 가능한 서버 리소스 섹션에서 옵션을 설정하십시오.

5. VI 반출 리스트 섹션에서 VI를 설정하여, 필요한 보안 수준에 따라 외부 클라이언트의 VI 접근 허용 또는 차단합니다.
6. 확인 버튼을 클릭합니다.

Visual Basic과 같이 ActiveX 오토메이션 클라이언트로부터 독립 어플리케이션을 호출하려는 경우, 어플리케이션을 빌드할 때 어플리케이션에 대한 ActiveX 서버를 반드시 활성화해야 합니다. 그렇게 하려면 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서 ActiveX 서버 활성화 확인란에 확인 표시를 하십시오.

ActiveX 컨테이너 개요

ActiveX 컨테이너를 사용하여 VI 프런트패널에 ActiveX 객체를 임베드합니다. 이를 통해 Windows 기반의 어플리케이션이 나타나서 LabVIEW 컨트롤과 상호작용하게 됩니다. 두 타입의 ActiveX 객체를 ActiveX 컨테이너에 놓을 수 있습니다. 새 ActiveX 컨트롤 또는 문서를 생성할 수 있습니다. 또한 기존 ActiveX 컨트롤 또는 문서를 삽입할 수도 있습니다.



관련 정보

[ActiveX 컨테이너에 새 컨트롤 생성하기](#) (1442 페이지 참조)

[ActiveX 컨테이너에 새 문서 생성하기](#) (1442 페이지 참조)

[ActiveX 컨테이너에 기존의 컨트롤이나 문서 삽입하기](#) (1443 페이지 참조)

LabVIEW에서 ActiveX 객체 호출하기

LabVIEW 프런트패널 윈도우에 ActiveX 객체 또는 컨트롤을 임베드하지 않으려는 경우 오토메이션 참조 번호를 사용합니다.

다음 단계를 따라 LabVIEW에서 ActiveX 객체를 호출합니다.

1. 프런트패널 윈도우에 [오토메이션 참조 번호](#) 컨트롤을 추가합니다.
2. ActiveX 객체를 오토메이션 참조 번호 컨트롤과 연계합니다.
 1. 컨트롤에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 ActiveX 클래스 선택 >> 탐색을 선택하여 타입라이브러리로부터 객체 선택 대화 상자를 디스플레이합니다.
 2. 타입 라이브러리로부터 객체 선택 대화 상자의 풀다운 메뉴에서 타입 라이브러리를 선택합니다.
 3. 객체 상자에서 객체를 선택한 후 확인 버튼을 클릭합니다.
3. 오토메이션 참조 번호에 대한 참조를 엽니다.
 1. 블록다이어그램에 [오토메이션 열기] 함수를 추가합니다.
 2. [오토메이션 열기] 함수의 오토메이션 참조 번호에 오토메이션 참조 번호를 연결합니다.
4. 프로퍼티에 접근하거나 오토메이션 참조 번호에서 메소드를 실행합니다.
 1. 다음 노드 중에서 하나를 블록다이어그램에 추가합니다.

- 프로퍼티 노드
 - 인보크 노드
2. [오토메이션 열기] 함수의 오토메이션 참조 번호 출력을 노드의 참조 입력에 연결합니다.
 3. 노드를 클릭하여 참조된 객체에 사용 가능한 프로퍼티나 메소드의 리스트를 디스플레이합니다.



노트

참조된 ActiveX 객체에 대해 기대하는 프로퍼티나 메소드를 LabVIEW가 디스플레이하지 않는 경우, 객체는 여러 구성요소 객체 모델 (COM) 인터페이스를 구현할 수도 있습니다. 객체가 구현하는 COM 인터페이스에서, LabVIEW는 한 번에 오직 하나의 COM 인터페이스에 해당하는 프로퍼티나 메소드를 디스플레이합니다. [배리언트를 데이터로] 함수를 사용하여 오토메이션 참조 번호를 다른 COM 인터페이스로 캐스트하여, 객체가 구현하는 다른 COM 인터페이스의 프로퍼티나 메소드를 강제로 LabVIEW가 디스플레이하도록 할 수 있습니다.

4. 프로퍼티 또는 메소드를 선택합니다.



노트 모든 ActiveX 객체가 메소드를 노출하지는 않습니다.

1. ActiveX 객체에 대한 참조를 닫습니다.
 1. 블록 다이어그램에 [참조 닫기] 함수를 추가합니다.
 2. [인보크 노드]의 참조 출력 출력을 [참조 닫기] 함수의 참조 입력에 연결합니다.
2. VI를 실행합니다. [오토메이션 열기] 함수는 프런트패널 윈도우의 오토메이션 참조 번호 컨트롤에 요청한 ActiveX 객체에 대한 참조를 열고 인보크 노드에 요청한 메소드를 수행합니다.

COM/오토메이션과 LabVIEW 데이터 타입

다음 테이블은 COM/오토메이션 데이터 타입과 대응하는 LabVIEW 데이터 타입을 나열합니다.

COM/오토메이션 데이터 타입	LabVIEW 데이터 타입
정수	정수
부동	부동
BSTR	문자열
날짜	문자열
Currency	문자열

COM/오토메이션 데이터 타입	LabVIEW 데이터 타입
Scode (에러)	정수
IDispatch	오토메이션 참조 번호
IUnknown	오토메이션 참조 번호
ActiveX 배리언트	LabVIEW 배리언트
하나의 데이터 타입의 SafeArray	대응하는 데이터 타입의 LabVIEW 배열
배리언트의 SafeArray	LabVIEW 클러스터

ActiveX 컨테이너에 새 컨트롤 생성하기

다음 단계를 따라 ActiveX 컨테이너에 새 컨트롤을 생성합니다.

1. 프런트패널 윈도우에 **ActiveX 컨테이너**를 추가합니다.
2. 프런트패널 윈도우의 ActiveX 컨테이너에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 ActiveX 객체 삽입을 선택하여 ActiveX 객체 선택 대화 상자를 디스플레이합니다.
3. ActiveX 객체 선택 대화 상자에서 풀다운 메뉴의 컨트롤 생성을 선택합니다.
4. 컴퓨터에 등록되어 있는 리스트에서 ActiveX 컨트롤을 선택하고 확인 버튼을 클릭합니다. ActiveX 프런트패널 컨테이너에 ActiveX 컨트롤이 나타납니다.



노트

서버 확인 확인란에 확인 표시를 하면, LabVIEW는 서버를 검증하고 컴퓨터에 실제 존재하는 레지스트리에서 서버들의 리스트만 열거합니다. 서버 확인 확인란에 확인 표시를 하지 않으면, LabVIEW는 서버를 검증하지 않고 레지스트리의 모든 서버를 열거합니다.

1. **컨트롤 또는 문서에 관련된 ActiveX 프로퍼티에 접근**하여 컨트롤 또는 문서의 프로퍼티를 설정하거나 **객체에서 메소드를 인보크**합니다.

ActiveX 컨테이너에 새 문서 생성하기

다음 단계를 따라 ActiveX 컨테이너에 새로운 ActiveX 문서를 생성합니다.

1. 프런트패널 윈도우에 **ActiveX 컨테이너**를 추가합니다.
2. ActiveX 컨테이너에서 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 ActiveX 객체 삽입을 선택하여 ActiveX 객체 선택 대화 상자를 디스플레이합니다.
3. ActiveX 객체 선택 대화 상자에서 풀다운 메뉴의 문서 생성을 선택합니다.
4. 컴퓨터에 등록되어 있는 리스트에서 문서 타입을 선택하고 확인 버튼을 클릭합니다. ActiveX 문서가 ActiveX 컨테이너에 나타납니다.

ActiveX 문서 편집하기

다음 단계를 따라 ActiveX 컨테이너의 ActiveX 문서를 편집합니다.

1. ActiveX 컨테이너에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 객체 편집을 선택합니다. 문서의 어플리케이션은 문서를 새 윈도우에서 엽니다.
2. 새 윈도우에서 문서를 편집합니다.

ActiveX 문서가 ActiveX 오토메이션 인터페이스를 지원하는 경우, 블록 다이어그램에서 문서를 ActiveX 프로퍼티 노드 및 인보크 노드와 함께 사용할 수 있습니다.

프런트패널에 ActiveX 객체 삽입하기

다음 단계에 따라 프런트패널에 ActiveX 객체를 삽입합니다:

1. ActiveX 컨테이너에서 마우스 오른쪽 버튼을 클릭한 후 ActiveX 객체 삽입 바로 가기 메뉴에서 삽입하려는 ActiveX 컨트롤을 선택합니다.
2. (옵션) ActiveX 컨트롤 프로퍼티 브라우저, 프로퍼티 페이지, 또는 프로퍼티 노드를 사용하여 객체의 **프로퍼티를 설정**합니다.



노트

프로그램적으로 프로퍼티를 설정하려면, 프로퍼티 노드를 사용하십시오.

3. (옵션) 인보크 노드를 사용하여 객체와 연관된 메소드를 인보크할 수 있습니다.

ActiveX 객체를 사용하여 Microsoft와 같은 일반 ActiveX 활성화 어플리케이션에 접근할 수 있습니다. 예를 들어, ActiveX 컨테이너로 Microsoft 웹 브라우저 컨트롤에 접근한 후, 메소드의 탐색 클래스를 선택하고, URL 메소드를 선택하여, URL을 지정하면 프런트패널에서 웹 페이지를 디스플레이할 수 있습니다.

ActiveX 컨테이너를 사용하면, 컨테이너에 대한 참조를 열거나 닫을 필요가 없습니다. ActiveX 컨테이너는 LabVIEW의 호출 어플리케이션을 내장하므로, 참조를 직접 [인보크 노드] 또는 [프로퍼티 노드]에 연결할 수 있습니다. 그러나, ActiveX 컨테이너가 다른 오토메이션 참조 번호를 반환하는 프로퍼티나 메소드를 포함할 경우, 이러한 추가적인 참조를 닫아야 합니다.

ActiveX 컨테이너에 기존의 컨트롤이나 문서 삽입하기

다음 단계를 따라 ActiveX 컨테이너에 기존의 ActiveX 컨트롤이나 문서를 삽입합니다.

1. 프런트패널에 **ActiveX 컨테이너**를 추가합니다.
2. ActiveX 컨테이너에서 마우스 오른쪽 버튼을 클릭합니다.
3. 바로 가기 메뉴에서 ActiveX 객체 삽입을 선택하여 ActiveX 객체 선택 대화 상자를 디스플레이합니다.

4. ActiveX 객체 선택 대화 상자에서 최상위 풀다운 메뉴의 파일로부터 객체 생성을 선택합니다.
5. 탐색 버튼을 클릭하고 삽입하려는 컨트롤이나 문서를 탐색합니다. 컴퓨터의 모든 곳의 객체를 선택할 수 있습니다. ActiveX 객체를 업데이트할 때 컨트롤이나 문서를 업데이트하려면 파일에 링크걸기 확인란에 확인 표시를 합니다. 파일에 링크걸기 확인란에 확인 표시를 하지 않으면, LabVIEW는 컨트롤과 문서의 고정된 버전을 삽입합니다.
6. 확인 버튼을 클릭합니다. ActiveX 컨트롤이나 문서가 ActiveX 컨테이너에 나타납니다.

ActiveX 클래스 선택하기

다음 단계를 따라 ActiveX 서버 클래스를 선택합니다.

1. 프런트패널 윈도우에 오토메이션 참조 번호 컨트롤을 추가합니다.
2. 오토메이션 참조 번호 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 ActiveX 클래스 선택 >> 탐색을 선택합니다.
3. 타입 라이브러리로부터 객체 선택 대화 상자로부터 ActiveX 서버 클래스를 선택합니다.

ActiveX 서버 클래스를 선택한 후에, 그 객체에서 메소드를 실행하거나 프로퍼티를 설정할 수 있습니다.



관련 정보

[참조 번호 컨트롤의 타입](#) (1021 페이지 참조)

[프로퍼티와 메소드 생성하기](#) (1427 페이지 참조)

[프로퍼티와 메소드 생성하기](#) (1427 페이지 참조)

ActiveX 프로퍼티 페이지 사용하기

많은 ActiveX 객체에는 각 개별 탭에 객체와 관련된 프로퍼티를 정리해 놓은 프로퍼티 페이지가 있습니다.

ActiveX 컨트롤 프로퍼티 브라우저처럼, ActiveX 프로퍼티 페이지는 대화식으로 ActiveX 객체의 프로퍼티를 설정하는 쉬운 방법이지만, 프로그램적으로 프로퍼티를 설정하는데 사용할 수는 없습니다. 프로퍼티 페이지는 컨테이너 내부의 ActiveX 객체에만 사용할 수 있습니다. 프로퍼티 페이지는 모든 ActiveX 객체에 사용 가능한 것은 아닙니다. 또한, 프로퍼티 페이지는 실행 모드 또는 VI가 실행 중일 경우에는 사용할 수 없습니다.

다음 단계를 따라 ActiveX 프로퍼티 페이지를 사용하여 ActiveX 프로퍼티를 보고 설정합니다.

1. ActiveX 컨테이너에서 **새 컨트롤 생성**, **새 문서 생성**, 또는 **기존 컨트롤 또는 문서 삽입**을 합니다.

2. 컨테이너의 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 객체의 이름을 탐색합니다. 프로퍼티를 선택하여 해당 객체의 컨트롤 프로퍼티 대화 상자를 시작합니다.
3. 탭을 선택하고 설정하고자 하는 프로퍼티를 편집합니다.
4. 확인 버튼을 클릭하여 프로퍼티 페이지 대화 상자를 닫습니다. 변경은 즉시 적용됩니다.

ActiveX 컨트롤 프로퍼티 브라우저 사용하기

ActiveX 컨트롤 프로퍼티 브라우저를 사용하여 ActiveX 컨테이너 안의 ActiveX 컨트롤 또는 문서와 관련된 모든 프로퍼티를 보고 설정합니다.

ActiveX 컨트롤 프로퍼티 브라우저는 대화식으로 ActiveX 객체의 프로퍼티를 설정하는 쉬운 방법이나, 프로그램적으로 프로퍼티를 설정하는데 브라우저를 사용할 수는 없습니다. ActiveX 컨트롤 프로퍼티 브라우저는 컨테이너 내부의 ActiveX 객체에만 사용할 수 있습니다. ActiveX 컨트롤 프로퍼티 브라우저는 실행 모드 또는 VI가 실행중일 경우에는 사용할 수 없습니다.

다음 단계를 따라 ActiveX 컨트롤 프로퍼티 브라우저를 사용하여 ActiveX 프로퍼티를 보고 설정합니다.

1. ActiveX 컨테이너에서 **새 컨트롤 생성**, **새 문서 생성**, 또는 **기존 컨트롤 또는 문서 삽입**을 합니다.
2. 컨테이너의 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 프로퍼티 브라우저를 선택합니다.
3. 프로퍼티를 설정합니다. 프로퍼티 브라우저의 왼쪽 열에서 설정하고자 하는 프로퍼티를 찾아 프로퍼티 브라우저의 오른쪽 열에서 해당 값을 수정합니다. 프로퍼티 브라우저 아래의 도움말은 업데이트되어 현재 선택한 프로퍼티를 설명합니다. 변경은 즉시 적용됩니다.
4. 프로퍼티 브라우저를 닫습니다.

기존의 프로퍼티와 인보크 노드 편집하기

프로퍼티 선택 또는 메소드 선택 대화 상자를 사용하여 기존 프로퍼티 노드 또는 인보크 노드를 편집할 수 있습니다.

다음 단계를 따라 프로퍼티 노드 또는 인보크 노드를 편집합니다.

1. 프로퍼티 노드의 프로퍼티 또는 인보크 노드의 메소드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **프로퍼티 선택** > **탐색** 또는 **메소드 선택** > **탐색**을 선택합니다. 프로퍼티 선택 또는 메소드 선택 대화 상자가 나타납니다.
2. 대화 상자의 프로퍼티 또는 메소드와 점으로 표시할 수 있는 프로퍼티 리스트에서 프로퍼티나 메소드를 선택하여 현재 프로퍼티 노드 또는 인보크 노드에 보이는 것을 대체합니다. 선택한 프로퍼티 또는 메소드는 선택된 프로퍼티 또는 선택된 메소드 상자에 나타납니다.
3. 확인 버튼을 클릭하여 블록 다이어그램의 노드를 업데이트합니다.

프로퍼티 노드와 인보크 노드를 터미널에 링크하기

프로퍼티 노드 또는 인보크 노드를 **복사하거나 복제**할 경우, LabVIEW는 프런트패널 객체와 노드의 연결을 끊을 수 있습니다.

다음 단계를 따라 프로퍼티 노드 또는 인보크 노드를 프런트패널 객체에 연결합니다.

1. 노드에서 마우스 오른쪽 버튼을 클릭합니다.
2. 링크를 선택하고 바로 가기 메뉴에서 객체의 이름을 선택합니다. LabVIEW는 노드를 **함축적으로 연결된 노드**로 대체합니다.
3. 다시 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 컨트롤로부터 연결 끊기를 선택하여 함축적으로 연결된 노드를 제거합니다.

또한, 링크 바로 가기 메뉴를 사용하여 함축적으로 연결된 노드를 다른 프런트패널 객체와 연계할 수 있습니다. 링크 바로 가기 메뉴에는 VI의 각 프런트패널 객체에 대한 옵션이 있습니다.

LabVIEW에서의 프로그램 연결

웹 브라우저에서 원격 디바이스 모니터 및 설정하기

RT 타겟과 같은 원격 디바이스에 NI 웹 기반 설정 및 모니터링을 설치한 후, 웹 브라우저를 사용하여 디바이스를 모니터하고 설정할 수 있습니다.

(Real-Time Module) NI Measurement & Automation Explorer (MAX)에서 LabVIEW Real-Time 소프트웨어 마법사를 사용하여 RT 타겟에 NI 웹 기반 설정 및 모니터링이 설치되었는지 알아보고 필요한 경우 이 소프트웨어를 설치합니다.



노트

NI 웹 기반 설정 및 모니터링을 사용하려면 Microsoft Silverlight을 설치해야 합니다. Microsoft Silverlight을 설치하지 않은 경우, NI 웹 기반 설정 및 모니터링이 설치를 요청합니다.

설정 페이지 접근하기

다음 방법을 사용하여 원격 디바이스에 대한 NI 웹 기반 설정 및 모니터링에 접근합니다:

- 프로젝트 탐색기 윈도우에서, 디바이스에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 유틸리티»브라우저에서 보기를 선택합니다.
- 웹 브라우저에서 URL `http://[원격 디바이스 IP 주소]`를 입력하며, 이때 [원격 디바이스 IP 주소]는 원격 디바이스의 실제 IP 주소입니다.
- **로컬 호스트 시스템**에서, 옵션 대화 상자의 웹 서버 페이지에 있는 웹 어플리케이션 서버 설정 버튼을 클릭합니다.

- ni.com에서 정보 코드 NetworkBrowserHelp를 입력하여 NI 네트워크 브라우저를 다운로드한 후 실행합니다. 원격 디바이스 리스트에서 원격 디바이스의 이름을 클릭합니다. 방화벽이 활성화된 경우, 방화벽이 NI 네트워크 브라우저를 허용할지 물어보는 대화 상자를 디스플레이할 수 있습니다. 방화벽이 NI 네트워크 브라우저를 허용하도록 해야 제대로 기능할 수 있습니다.
- (NI Linux Real-Time) 컨트롤러에서 SSH를 활성화한 경우, SSH 또는 시리얼 연결을 사용하여 타겟에 로그인할 수 있습니다. 보다 상세한 내용은 [NI Linux Real-Time 타겟에서의 Real-Time Module](#) 토픽을 참조하십시오.

설정 페이지에 로그인하기

NI 웹 기반 설정 및 모니터링은 사용 권한을 묻는 보안 모델을 사용합니다. 기본으로 각 원격 디바이스는 다음의 이름과 암호를 사용하는 한 명의 사용자를 포함합니다.

- 사용자 이름 — Admin
- 암호 — <공백>

원격 디바이스에 대한 NI 웹 기반 설정 및 모니터링 어플리케이션에 접근한 후, 다음 단계를 따라 처음으로 로그인합니다:

1. 페이지의 우측 상단에 있는 로그인 버튼을 클릭합니다.
2. 사용자 이름 필드에 Admin을 입력합니다.
3. 암호는 공백으로 둡니다.
4. 확인 버튼을 클릭합니다.

다음 단계에 따라 사용자 암호를 변경할 수 있습니다.



노트 암호 필드는 대소문자를 구별합니다. (NI Linux Real-Time) SSH 또는 시리얼 연결을 사용하여 로그인하려면 암호를 공백이 아닌 암호로 변경해야 합니다.

1. 보안 설정 페이지를 탐색합니다.
2. 이 페이지의 왼쪽에 있는 사용자 탭을 클릭하십시오.
3. 새 암호를 입력하고 확인한 후 확인을 클릭합니다.
4. 변경사항을 저장하려면 저장을 클릭하십시오.

디바이스 모니터하고 설정하기

로그인한 후, 디바이스를 모니터하고 설정하는데 대한 정보를 얻으려면 NI 웹 기반 설정 및 모니터링의 각 페이지에 있는 도움말을 클릭합니다.

어플리케이션 개발하기 및 배포하기

요구사항: 어플리케이션 빌더 또는 Professional Development System

LabVIEW 프로젝트를 다른 컴퓨터 및 다른 프로그래밍 언어에서 사용할 수 있도록 배포 가능한 어플리케이션으로 변환시킬 수 있습니다. 이 문서의 단계에 따라 다음의 배포 가능한 어플리케이션의 타입들을 생성합니다.

- 독립 어플리케이션
- 설치 프로그램
- .NET Interop 어셈블리
- 패키지
- 묶음 프로젝트 라이브러리
- 공유 라이브러리
- 소스 배포
- 압축(Zip) 파일

어플리케이션을 만들 준비하기

1. 어플리케이션을 만들 때 사용할 **LabVIEW 프로젝트**를 엽니다. 반드시 개별 VI가 아닌 프로젝트에서 어플리케이션을 만들어야 합니다.
2. 전체 프로젝트를 저장하여 모든 VI가 LabVIEW의 현재 버전으로 저장되도록 합니다.
3. 각 VI의 VI 프로퍼티 대화 상자에서 설정을 확인합니다. 어플리케이션을 배포할 계획이 있는 경우에는, VI 프로퍼티 대화 상자의 셋팅이 VI의 빌드 버전과 맞는지 확인합니다. 예를 들어, 빌드된 어플리케이션의 모양을 손질하려면, VI 프로퍼티 대화 상자의 다음 페이지에 있는 셋팅을 확인합니다.
 - 윈도우 모양
 - 윈도우 크기
 - 윈도우 런타임 위치



노트

어플리케이션에 **컴파일된 개별 코드**를 포함하는 VI가 있는 경우, 어플리케이션의 프로퍼티 대화 상자의 소스 파일 셋팅 페이지에서 이러한 VI의 셋팅을 설정할 수 있습니다.

4. 개발 환경에서 사용한 경로가 대상 컴퓨터에서도 올바르게 동작하는지 확인합니다. 프로젝트가 **VI를 다이나믹하게 로드**하는 경우, 절대 경로 대신 상대 경로를 사용하여 VI의 위치를 지정합니다. 컴퓨터가 다르면 파일 계층구조도 달라질 수 있습니다. 컴퓨터에 따라 파일 계층 구조가 다를 수 있기 때문에, **상대 경로**를 사용하면 개발 환경 및 어플리케이션이 실행될 대상 컴퓨터에서 경로가 모두 올바르게 작동하게 됩니다.

노트 어플리케이션이 **8.x 파일 레이아웃을 사용**하는 경우, 대상 컴퓨터에서는 개발 환경에서 사용한 상대 경로와 다른 상대 경로를 사용해야 합니다.

또한 만드는 과정에서 에러가 발생하는 것을 방지하려면, 빌드에서 생성한 대상 디렉토리의 파일 경로가 파일 이름을 포함하여 255자 이하가 되도록 합니다. 생성 중인 빌드 스펙에 대한 프로퍼티 대화 상자의 대상 페이지에서 생성한 파일에 대한 대상을 지정할 수 있습니다.

5. [현재 VI의 경로] 함수에서 예상된 경로가 반환되는지 확인합니다. 독립 어플리케이션 또는 공유 라이브러리에서, [현재 VI의 경로] 함수는 어플리케이션 파일의 VI에 대한 경로를 반환하며 어플리케이션 파일을 LLB로 취급합니다. 예를 들어, foo.vi를 어플리케이션으로 빌드한 경우, 이 함수는
C:\...\Application.exe\foo.vi라는 경로를 반환하며, 이때
C:\...\Application.exe는 어플리케이션의 경로와 파일 이름을 나타냅니다.
6. LabVIEW 런타임 엔진에서 의도한대로 VI 서버 프로퍼티와 메소드가 작동하는지 확인합니다. LabVIEW 런타임 엔진은 일부 VI 서버 프로퍼티와 메소드를 지원하지 않습니다. 그러므로, 어플리케이션 또는 공유 라이브러리에 들어갈 VI에는 이러한 프로퍼티와 메소드를 사용하는 것을 피하십시오. VI Analyzer Toolkit의 빌드된 어플리케이션 호환성 테스트를 실행하여, 사용자의 VI 서버 프로퍼티가 LabVIEW 런타임 엔진과 호환되는지 확인합니다.

빌드된 어플리케이션의 스펙 설정하기



노트 프론트패널 컨트롤의 사용자 런타임 메뉴를 별도의 런타임 메뉴 파일 또는 .rtm 파일로 저장한 후 이 컨트롤을 어플리케이션 빌드에 포함시키려는 경우, 해당 .rtm 파일을 어플리케이션의 프로퍼티 대화 상자의 소스 파일 페이지에 있는 항상 포함 리스트박스에 반드시 추가해야 합니다.

1. 빌드 스펙을 생성합니다. 프로젝트 탐색기 윈도우의 내 컴퓨터를 확장합니다. 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 어플리케이션 타입을 선택하여, 어플리케이션의 프로퍼티 대화 상자를 디스플레이합니다. 프로젝트 탐색기 윈도우에서 빌드 스펙을 숨긴 경우, 이에 접근하려면 반드시 해당 아이템을 디스플레이해야 합니다.
2. 어플리케이션의 프로퍼티 대화 상자에서 빌드 스펙의 필수 페이지를 설정합니다.



노트 설치 프로그램에 어떤 타입의 어플리케이션을 포함시키려면, 어플리케이션의 모든 파일이 어플리케이션의 주요 대상과 상대적이 되도록 지정해야 합니다. 그렇지 않으면, 설치 프로그램에 어플리케이션의 빌드 출력을 포함시킬 때 설치 프로그램이 주요 대상과 상대적이지 않은 모든 파일을 이동하여 어플리케이션 원본 파일의 구조를 재구성합니다. 모든 타입의 어플리케이션에 대한 주요 대상은 해당 어플리케이션의 프로퍼티 대화 상자의 대상 페이지에서 지정할 수 있습니다.

만들려는 어플리케이션의 타입을 다음 리스트에서 선택하십시오:

독립 어플리케이션

- 필수 설정
 - 소스 파일
- 권장 설정
 - 정보

- 대상
- 소스 파일 셋팅
- 추가 정보
 - LabVIEW 런타임 엔진이 모든 VI를 로드하게 하려면, 컴파일된 코드 분리를 위해 프로젝트 아이템 표시 대화 상자를 사용하여 모든 VI의 컴파일된 코드를 포함시킵니다.

설치 프로그램

- 필수 설정
 - 소스 파일
- 권장 설정
 - 제품 정보
 - 대상
 - 소스 파일 셋팅
 - 버전 정보
- 추가 정보
 - LabVIEW 런타임 엔진이 모든 VI를 로드하게 하려면, 컴파일된 코드 분리를 위해 프로젝트 아이템 표시 대화 상자를 사용하여 모든 VI의 컴파일된 코드를 포함시킵니다. 더 자세한 정보는 [설치 프로그램 만들기에 대한 유의점과 권장사항](#)을 참조하십시오.

.NET Interop 어셈블리

사용자 개발 컴퓨터에 .NET Framework 4.0이 설치되어 있는 경우, 어플리케이션 빌더를 사용하여 .NET interop 어셈블리를 만들 수 있습니다.

- 필수 설정
 - 소스 파일
- 권장 설정
 - 정보
 - 대상
 - 소스 파일 셋팅
- 추가 정보
 - .NET interop 어셈블리를 통해 LabVIEW 클래스들에 접근할 수도 있습니다. LabVIEW는 사용자가 지정한 LabVIEW 클래스에 대해 .NET 클래스를 생성할 수 있으며, .NET 어셈블리를 통해 이러한 .NET 클래스들에 접근합니다.
 - LabVIEW 런타임 엔진이 모든 VI를 로드하게 하려면, 컴파일된 코드 분리를 위해 프로젝트 아이템 표시 대화 상자를 사용하여 모든 VI의 컴파일된 코드를 포함시킵니다. 더 자세한 정보는 [.NET interop 어셈블리 만들기의 특징](#)을 참조하십시오.

패키지

- **권장 설정**
 - 소스 파일
 - 정보
 - 대상
 - 패키지
 - 패키지 설치 프로그램
 - 피드
- **추가 정보**
 - 독립형 VI는 배포할 수 없습니다.
 - 패키지 생성에 대한 자세한 정보는 [배포용 패키지 생성하기](#) 토픽을 참조하십시오.

묶음 프로젝트 라이브러리

- **필수 설정**
 - 소스 파일
- **권장 설정**
 - 정보
 - 대상
 - 소스 파일 셋팅
- **추가 정보** 더 자세한 정보는 [묶음 프로젝트 라이브러리의 유의점과 권장사항](#)을 참조하십시오.

공유 라이브러리

- **필수 설정**
 - 소스 파일
- **권장 설정**
 - 정보
 - 대상
 - 소스 파일 셋팅
- **추가 정보** 더 자세한 정보는 [LabVIEW로 만든 공유 라이브러리의 특징](#)을 참조하십시오.

소스 배포

- **필수 설정**
 - 소스 파일

- 권장 설정
 - 정보
 - 대상
 - 소스 파일 셋팅

압축(Zip) 파일

- 필수 설정
 - 소스 파일
- 권장 설정
 - 압축(Zip) 정보
 - 압축(Zip) 파일 구조
- 추가 정보
 - 압축(Zip) 파일 프로퍼티 대화 상자의 소스 파일 페이지에서 전체 프로젝트 압축(Zip) 옵션을 선택한 경우, LabVIEW는 포함된 아이템 트리에 있는 다른 빌드 스펙의 출력을 자동적으로 포함하지 않습니다. 소스 배포, 공유 라이브러리 또는 어플리케이션을 압축(Zip) 파일에 포함시키려면 화살표 버튼을 사용해야 합니다.
 - 압축(Zip) 또는 설치 프로그램 빌드 스펙의 출력은 소스 파일 페이지의 프로젝트 파일 트리에 나타나지 않습니다. 다른 압축(Zip) 파일 또는 설치 프로그램을 압축 파일에 포함시키려면, 압축 파일을 빌드하고 있는 타겟 아래에 LabVIEW 프로젝트에 포함시키려는 압축 파일이나 설치 프로그램을 추가하십시오.
 - 또한 압축[Zip] VI를 사용하거나 [빌드] VI를 사용하여 프로그램적으로 압축(Zip) 파일을 만들 수 있습니다. 압축[Zip] VI를 사용하려면 Zip 파일에 포함시키려는 모든 파일에 대한 상대 경로를 알아야 합니다.

3. 다이나믹하게 로드된 VI를 빌드 스펙에 명확히 포함시킵니다.

어떤 VI가 **VI 서버**를 사용하여 다른 VI를 **다이나믹하게** 로드하거나 [참조에 의한 호출] 또는 [비동기화 호출 시작] 노드를 통해 다이나믹하게 로드된 VI를 호출하는 경우, 어플리케이션의 프로퍼티 대화 상자의 소스 파일 페이지에 있는 항상 포함 리스트박스에 그러한 VI들을 반드시 추가해야 합니다.

또한 어플리케이션의 다이나믹하게 로드된 VI를 소스 배포에 포함시켜서 배포할 수도 있습니다.

4. 업데이트된 빌드 스펙 셋팅을 저장합니다. 확인 버튼을 클릭하여 프로젝트의 빌드 스펙을 업데이트하고 대화 상자를 닫습니다. 빌드 스펙 아래에 있는 프로젝트에 빌드 스펙 이름이 나타납니다. 빌드 스펙에 변경 사항을 저장하려면, 해당 빌드 스펙을 포함하는 프로젝트를 저장해야 합니다.

어플리케이션 빌드하기

빌드하려는 어플리케이션의 빌드 스펙 이름에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 빌드를 선택합니다. 또한 [빌드] VI 또는 **ExecuteBuildSpec** 명령 라인 작업을 사용하여 어플리케이션을 프로그램적으로 빌드할 수 있습니다.



팁 어플리케이션을 미리 보기하여 올바르게 나타나는지 확인할 수 있습니다. 어플리케이션의 프로퍼티 대화 상자의 미리 보기 페이지에서 미리 보기 생성 버튼을 클릭하여 어플리케이션에 대해 생성된 파일을 살펴 볼 수 있습니다.

빌드된 어플리케이션 배포하기

1. 어플리케이션을 실행하려는 컴퓨터가 LabVIEW 런타임 엔진을 사용할 수 있도록 합니다. 어플리케이션 또는 공유 라이브러리를 실행하려는 모든 컴퓨터에 **LabVIEW 런타임 엔진**이 설치되어 있어야 합니다. LabVIEW 런타임 엔진을 어플리케이션 또는 공유 라이브러리와 함께 배포할 수 있습니다. (Windows) 또한 설치 프로그램에 LabVIEW 런타임 엔진을 포함시킬 수 있습니다.
2. 최종 사용자를 위한 법적 정보를 배포합니다. 설치 프로그램을 사용하여 어플리케이션을 배포하는 경우, 설치 프로그램 프로퍼티 대화 상자의 대화 상자 정보 페이지에 사용자 라이선스 협약 정보를 입력합니다. NI 제품으로 빌드된 설치 프로그램에 법적 정보를 포함시키는 방법은 <National Instruments>\Legal Information.txt를 참조하십시오.
3. LabVIEW로 만든 어플리케이션의 배포에 도움이 되는 팁은 다음 테이블을 참조하십시오.

작업	장점
빌드한 어플리케이션이 올바르게 실행되는지 확인하려면, 어플리케이션의 프로퍼티 대화 상자의 고급 페이지에 있는 디버깅 활성화 확인란에 확인 표시를 합니다. 또한 수행 >> 어플리케이션 또는 공유 라이브러리 디버깅을 선택하여 빌드한 어플리케이션에 연결할 수 있습니다.	빌드한 어플리케이션을 테스트할 때에는, 개발 환경과 배포 환경 모두에서 동일한 동작을 하는지 확인해야 합니다.
어플리케이션의 테스트를 마친 후에는 VI에 대한 디버깅을 비활성화 합니다.	VI에 대한 디버깅을 비활성화하면 파일 크기를 줄이고 런타임 성능을 향상시킬 수 있습니다.
독립 어플리케이션용 정보 대화 상자를 생성합니다.	대부분의 전문적인 어플리케이션에는 버전, 저작권 및 지원 정보와 같이 어플리케이션에 대한 일반적인 정보를 제공하는 정보 대화 상자가 들어 있습니다.
최종 사용자의 사용 언어 요구 사항이 원본 어플리케이션의 언어와 다를 경우, 어플리케이션의 프로퍼티 대화 상자의 런타임 언어 페이지에서 어플리케이션의 기본 언어 설정을 조절할 수 있습니다.	어플리케이션을 배포하면, 대화 상자 및 메뉴는 어플리케이션이 실행되는 OS의 언어를 채택하지만 실행 텍스트 및 컨트롤은 실제 어플리케이션을 빌드한 OS 시스템의 언어로 남아 있게 됩니다. 기본으로, 독립 어플리케이션과 공유 라이브러리 빌드 스펙은 중국어(간체), 영어, 불어, 독일어, 일본어, 한국어를 지원합니다.

작업	장점
어플리케이션이 다른 어플리케이션과 동일한 VI 서버를 사용하는 경우, 사용자 설정 파일을 생성 하십시오.	동일한 VI 서버를 사용하는 어플리케이션을 상충하는 어플리케이션과 동시에 실행하려는 경우, 해당 VI 서버는 실행되지 않으며 LabVIEW는 경고를 주지 않습니다.
사용자 설정 파일을 빌드한 어플리케이션과 함께 배포하십시오.	어플리케이션이 일부 LabVIEW 환경 설정에서 항상 실행되도록 하려는 경우, 어플리케이션과 함께 사용자 설정 파일 또는 환경 파일을 배포하여 해당 설정을 보존 및 재배포할 수 있습니다.

macOS에서 Universal Framework 만들기

Universal Framework 또는 공유 라이브러리에는 64비트 및 32비트 어플리케이션에서 모두 프레임워크를 호출할 수 있도록 64비트 및 32비트 코드가 모두 포함됩니다. Universal Framework를 만드려면 LabVIEW의 64비트 버전과 32비트 버전이 모두 필요합니다. 다음 단계를 따라 LabVIEW에서 Universal Framework 어플리케이션을 만듭니다:

1. 64비트 LabVIEW를 사용하여 범용으로 사용하려는 프레임워크를 만듭니다. LabVIEW는 64비트 VI 코드를 사용하여 프레임워크를 만듭니다.
2. 32비트 LabVIEW를 사용하여, 이전 단계에서 만든 것과 동일한 프레임워크를 만듭니다. LabVIEW는 32비트 VI 코드를 사용하여 프레임워크를 만듭니다.
3. 디스크상의 32비트 프레임워크를 탐색한 후, 32비트 프레임워크의 .11b 파일을 단계 1에서 생성한 64비트 프레임워크에 복사합니다. .11b 파일을 64비트 프레임워크로 복사한 후, 단계 2에서 생성한 32비트 프레임워크는 더 이상 필요하지 않습니다.



노트

반드시 32비트 프레임워크의 .11b 파일을 64비트 프레임워크로 복사해야 합니다. 64비트 프레임워크에서 32비트 프레임워크로 .11b 파일을 복사하는 경우 프레임워크를 범용으로 사용할 수 없게 됩니다.

이제 단계 1에서 만든 프레임워크를 32비트와 64비트 어플리케이션에서 모두 호출할 수 있습니다.

관련 정보

[어플리케이션 만들기 및 배포하기](#)

[프레임워크\(공유 라이브러리\) 만드는 방법](#)

[어플리케이션과 공유 라이브러리 디버그하기](#)

배포용 패키지 생성하기

패키지를 생성하고 NI 패키지 관리자 또는 SystemLink를 통해 클라이언트에 패키지를 배포할 수 있습니다.

패키지를 생성하기 전에 패키지에 포함시킬 소스 배포, 묶음 프로젝트 라이브러리, 공유 라이브러리, .NET 어셈블리, 실행 파일을 생성합니다.



노트

독립형 VI는 배포할 수 없습니다.

1. 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 새로 만들기»패키지를 선택합니다.
2. 패키지 프로퍼티 대화 상자의 정보 아래에서 패키지에 사용할 이름을 패키지 이름에 입력합니다. 패키지 이름은 모두 소문자여야 하고 공백이 있으면 안 됩니다. NI 패키지 관리자 및 다른 패키지는 이 이름을 사용하여 생성한 패키지를 참조합니다.
3. 대상 아래에서 패키지를 설치하려는 클라이언트의 대상 폴더를 설정합니다.
4. 소스 파일 아래에서 설치할 파일 또는 빌드 스펙 출력을 지정합니다.
5. (Windows) 바로 가기 아래에서 시스템 시작 후 바로 어플리케이션이 실행되기를 원한다면 시작 메뉴에 바로가기를 추가할 수 있습니다. 또는 Windows의 바탕 화면이나 프로그램 메뉴에 바로 가기를 추가할 수도 있습니다. 기본적으로, LabVIEW는 패키지에 포함된 각 어플리케이션의 바로 가기를 Windows 프로그램 메뉴에 추가합니다.
6. 패키지 아래에서 패키지의 디스플레이 이름 및 개요를 지정합니다. 디스플레이 이름은 사용자가 패키지를 설치할 때 패키지 관리자 또는 NI SystemLink에 나타나는 이름입니다. 개요는 사용자가 패키지를 클릭하여 전체 설명을 보기 전에 나타나는 설명의 첫 번째 줄입니다.
7. 의존성 아래에서 현재 설치된 모든 패키지의 의존성을 지정합니다.
 1. 의존성 관계를 설정합니다. 의존성을 필수, 권장 또는 제안으로 정의하고 버전 범위를 지정할 수 있습니다.
 2. 사용자의 패키지를 관련 패키지 설치 시 함께 설치하도록 요청하려면 강화된 관계 포함 확인란에 확인 표시를 합니다.
8. 버전 정보 아래에서 패키지의 버전과 사용자에게 디스플레이할 버전을 지정합니다. 디스플레이 버전을 제거하면 사용자는 패키지 관리자에서 전체 버전을 보게 됩니다.
9. 고급 아래에서 설치 또는 제거 중에 실행 파일이나 스크립트를 실행하도록 패키지를 설정합니다. 예를 들어, 패키지 설치 또는 제거 전에 특정 작업이 완료되기를 기다리도록 지정할 수 있습니다.
10. (옵션) 패키지 설치 프로그램 아래에서 패키지 설치 프로그램 생성 확인란에 확인 표시를 하면 배포를 패키지 설치 프로그램 파일(.exe)로 빌드할 수 있습니다.
11. (옵션) 피드 아래에서, 패키지를 로컬 피드에 추가할지 아니면 SystemLink 서버의 피드에 공개할지 선택합니다.
12. 빌드를 클릭합니다. 지정한 대상에 (Windows).nipkg 파일이나 (NI Linux Real-Time).ipk 파일이 나타납니다.

패키지 관리자 또는 SystemLink를 통해 클라이언트에 패키지를 배포할 수 있습니다. (Windows) 클라이언트는 패키지 관리자 또는 SystemLink를 통해 피드를 구독하여, 패키지를 찾고 설치할 수 있습니다. (NI Linux Real-Time) 클라이언트는 SystemLink 또는 NI Linux Real-Time 타겟의 명령 라인을 사용하여 패키지를 설치할 수 있습니다. 패키지 관리자는 .ipk 파일을 지원하지 않습니다.

어플리케이션 및 공유 라이브러리에서 런타임 언어 설정하기

독립 어플리케이션 또는 공유 라이브러리에서 사용할 언어를 하나 이상 지정할 수 있습니다. 기본적으로, LabVIEW는 Windows와 동일한 런타임 언어를 사용합니다. LabVIEW가 Windows 또는 Windows가 아닌 플랫폼의 런타임 언어를 사용할 수 없는 경우, LabVIEW는 기본적으로 영어를 사용합니다. 런타임 언어 옵션을 설정하여 이러한 기본 동작을 변경할 수 있습니다.



노트

LabVIEW는 기본 런타임 메뉴, 사용자 런타임 메뉴의 기본 메뉴 아이템, 에러 코드 설명, LabVIEW 대화 상자와 같은 아이템의 텍스트를 현지화합니다. 하지만, 사용자가 직접 캡션 및 컨트롤 설명과 같은 아이템에 대한 **텍스트를 현지화**해야 합니다.

다음 메소드를 통해 독립 어플리케이션을 위한 런타임 언어 옵션을 설정할 수 있습니다. 메소드는 우선 순위에 따라 나타납니다.

- -lang 언어 명령 라인 옵션을 가진 어플리케이션을 실행합니다. 이때, 언어는 사용하려는 언어의 영어 이름입니다.
- 어플리케이션의 **설정 파일**에 다음 라인을 추가합니다. AppLanguage= 언어. 이때, 언어는 사용하려는 언어의 영어 이름입니다. 어플리케이션이 프로젝트 셋팅을 수정하지 않으므로, 설정 파일을 수정합니다.
- (Windows) 명령 라인이나 또는 어플리케이션의 설정 파일에 사용하려는 언어를 지정하지 않으면, 어플리케이션은 OS의 언어를 사용합니다. 제어판 >> 국가 및 언어 옵션을 탐색하여 OS의 언어 셋팅을 확인합니다. 어플리케이션이 OS 언어를 지원하지 않으면, 어플리케이션은 어플리케이션의 기본 언어를 사용합니다.
- 기본 언어 옵션을 사용하여 어플리케이션의 기본 언어를 설정합니다. 프로젝트 탐색기 윈도우의 빌드 스펙 하단에 있는 어플리케이션에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택하고 어플리케이션 프로퍼티 대화 상자의 런타임 언어 페이지를 탐색합니다. 프로젝트 탐색기 윈도우에서 빌드 스펙을 숨긴 경우, 이에 접근하려면 반드시 **해당 아이템을 디스플레이**해야 합니다.
- 지원되는 언어 옵션을 사용하여 빌드 스펙에서 어플리케이션에 대한 지원 언어를 설정합니다. 프로젝트 탐색기 윈도우의 빌드 스펙 하단에 있는 어플리케이션에서 마우스 오른쪽 버튼을 클릭한 후, 바로 가기 메뉴에서 프로퍼티를 선택하고 어플리케이션 프로퍼티 대화 상자의 런타임 언어 페이지를 탐색하여 어플리케이션의 언어 옵션을 설정합니다. 런타임 언어 페이지의 지원되는 언어 옵션은 지원되는 모든 언어를 나열합니다.

앞의 옵션을 사용하여 언어를 지정하지 않으면, 어플리케이션은 알파벳 순서에 따라 첫 번째 사용 가능한 지원되는 언어를 사용합니다. 어플리케이션 프로퍼티 대화 상자의 런타임 언어 페이지의 지원되는 언어 옵션은 지원되는 모든 언어를 나열합니다. 필요한 언

어 파일이 삭제되거나 손상되거나 읽을 수 없어 어플리케이션이 지원되는 언어를 사용할 수 없으면 LabVIEW는 해당 어플리케이션을 로드할 수 없습니다.



노트 **LabVIEW 런타임 엔진**은 여러 언어를 지원하므로 현지화된 텍스트를 사용하기 위해 어플리케이션, 공유 라이브러리 또는 설치 프로그램과 함께 여러 언어 버전을 포함시키지 않아도 됩니다.

어플리케이션:언어 프로퍼티를 사용하여 프로그램적으로 어플리케이션의 기본 언어를 결정할 수 있습니다.

공유 라이브러리 프로퍼티 대화 상자의 런타임 언어 페이지를 사용하여 공유 라이브러리의 언어 환경을 설정합니다.

LabVIEW 작업 환경 옵션 복사하기

LabVIEW 작업 환경에 **설정된 옵션**을 사용자가 만든 독립 어플리케이션에 복사할 수 있습니다.

독립 어플리케이션에서 **LabVIEW 설정 파일**을 사용하는 경우, LabVIEW 작업 환경 옵션을 수동으로 복사할 필요가 없습니다. 어플리케이션 빌더가 LabVIEW 작업 환경에 설정된 옵션을 자동으로 복사합니다. 독립 어플리케이션에서 **사용자 설정 파일**을 사용하고 이 파일에 원하는 LabVIEW 작업 환경 옵션이 포함되어 있지 않은 경우, LabVIEW 설정 파일에서 LabVIEW 작업 환경 옵션을 지정하는 엔트리를 수동으로 복사할 수 있습니다.

다음 단계에 따라 LabVIEW 작업 환경 옵션을 수동으로 복사합니다.

1. LabVIEW 설정 파일을 엽니다. LabVIEW 설정 파일의 이름 및 위치는 다음 테이블을 참조하십시오. 다음 테이블에서, x는 현재 사용 중인 LabVIEW 버전입니다.

플랫폼	LabVIEW 설정 파일
LabVIEW x \labview.ini	
~/Library/Preferences/LabVIEW.app x 64-bit Preferences	
/home/<username>/natinst/.config/LabVIEW-x/labview.conf	

2. LabVIEW 설정 파일에서, 원하는 LabVIEW 작업 환경 옵션이 지정된 엔트리를 복사합니다. LabVIEW 작업 환경 옵션을 지정하는 엔트리는 다음과 같은 포맷으로 되어 있습니다: preference_name = value.
3. 이 엔트리를 독립 어플리케이션 설정 파일에 붙여넣습니다. 독립 어플리케이션 설정 파일의 이름은 다음 테이블을 참조하십시오. 다음 테이블에서, myapp는 독립 어플리케이션의 이름이고 x는 현재 사용 중인 LabVIEW 버전을 가리킵니다.

플랫폼

독립 어플리케이션 설정 파일

myapp.ini

myapp.app x Preferences

myapp.conf

4. 독립 어플리케이션 설정 파일을 저장합니다.

관련 정보

LabVIEW가 옵션을 저장하는 방법

독립 어플리케이션의 정보 대화 상자 생성하기

대부분의 어플리케이션에는 이 어플리케이션에 대한 일반적인 정보와 이것을 만든 사용자와 회사를 디스플레이하는 정보 대화 상자가 있습니다. 사용자가 생성한 독립 어플리케이션에 대한 정보를 디스플레이하기 위해서 도움말>>정보를 선택할 때 LabVIEW가 실행하게 될 VI를 생성할 수 있습니다. VI가 실행된 이후, LabVIEW는 대화 상자를 닫습니다.

어플리케이션은 단 하나의 정보 VI를 가질 수 있습니다. 정보 VI를 공급하지 않는 경우, LabVIEW는 도움말>>정보를 선택할 때 LabVIEW 개발 시스템에서 나타내는 대화 상자와 비슷한 기본 대화 상자를 디스플레이합니다.

다음 단계를 따라 정보 VI를 생성하고 독립 어플리케이션에 포함시킵니다.

1. 버전 번호, 회사 이름, 저작권과 같이 정보 대화 상자에 디스플레이하려는 정보를 포함하는 프런트패널 윈도우가 있는 VI를 만듭니다. 생성한 정보 VI는 어플리케이션에 포함시킨 다른 VI와 subVI를 공유할 수 있습니다. 하지만, 다른 VI가 어플리케이션에서 실행될 때 정보 VI는 실행될 수 없기 때문에 정보 VI를 subVI로 사용할 수 없습니다.



노트

프런트패널 윈도우는 반드시 National Instruments 저작권 공지를 포함해야만 합니다. LabVIEW 어플리케이션에 대해서 생성한 정보 대화 상자의 요구사항에 대한 추가적인 정보는 LabVIEW 플랫폼 미디어에 위치한 National Instruments Software License Agreement를 참조하십시오.

1. 정보 VI의 블록 다이어그램을 만듭니다. 예를 들어, 사용자가 버튼을 클릭할 때까지 정보 대화 상자를 디스플레이 하려는 경우, 블록 다이어그램에 이벤트 구조가 있는 While 루프를 사용합니다.
2. VI를 저장합니다. 이 VI의 이름은 반드시 About으로 시작해야 합니다.
3. VI를 어플리케이션을 만드는 프로젝트에 추가합니다.
4. 어플리케이션 빌드 설정을 설정할 때, 정보 VI를 어플리케이션 프로퍼티 대화 상자의 소스 파일 페이지의 항상 포함 리스트에 추가합니다.

5. 소스 파일 셋팅 페이지의 프로젝트 파일 트리에서 정보 VI를 선택합니다. 기본 저장 셋팅 사용 섹션에서, 프론트패널 제거 확인란에 확인 표시가 없는지 그리고 대상이 어플리케이션인지 확인합니다.
6. 남아있는 어플리케이션 빌드 셋팅을 설정하고 어플리케이션을 만듭니다.

독립 어플리케이션에서 웹 서버 활성화하기

HTML 문서에 독립 어플리케이션의 **프론트패널을 넣으려면** VI에서와 마찬가지로 어플리케이션에서 웹 서버를 **활성화하고 설정**하면 됩니다. VI의 프론트패널과 HTML 파일을 설치 프로그램에 포함하십시오.



노트

어플리케이션을 빌드할 때, 어플리케이션 프로퍼티 대화 상자의 소스 파일 셋팅 페이지에서, 공개하려는 VI의 프론트패널 제거 확인란의 확인 표시를 제거하여 HTML 문서에 프론트패널을 넣습니다.

어플리케이션을 생성한 후, 출력 폴더는 .ini 설정 파일과 niwebserver.conf 설정 파일을 둘 다 포함합니다.

.ini 파일에 WebServer.Enabled=True라는 문장이 이미 존재하지 않을 경우, 이 문장을 추가하십시오. 또한 LabVIEW 개발 시스템의 LabVIEW INI 파일에서 웹서버* 환경을 어플리케이션의 .ini 파일에 복사할 수 있습니다. 어플리케이션에 맞도록 이 환경을 수정해야 할 수도 있습니다.



노트 어플리케이션을 실행할 때 LabVIEW 개발 환경을 포함하여 동일한 포트를 사용하는 다른 어플리케이션이 없어야 합니다.

niwebserver.conf 파일을 사용하여, **웹 서버의 루트 디렉토리**와 같은 어플리케이션의 다른 웹 서버 기능을 설정합니다.

프로그램적으로 어플리케이션 및 공유 라이브러리 버전 검색하기(Windows)

labview\vi.lib\Platform 디렉토리의 fileVersionInfo.llb를 사용하여 독립 어플리케이션이나 공유 라이브러리의 버전 정보를 프로그램적으로 검색할 수 있습니다. 이 VI는 함수 팔레트에 없습니다.



LLB 열기



노트

LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. LabVIEW Base Development System 또는 Full Development System을 사용하는 경우, National Instruments 웹사이트를 방문하여 어플리케이션 빌더를 별도로 구매할 수 있습니다.

다음 단계를 따라 어플리케이션이나 공유 라이브러리의 버전 정보를 프로그램적으로 검색합니다.

1. LLB의 최상위 VI인 FileVersionInfo VI에서, 어플리케이션 경로 입력 경로 컨트롤에서 버전 정보를 검색하려는 어플리케이션이나 공유 라이브러리의 디스크 위치를 지정합니다.
2. VI를 실행합니다.

LabVIEW는 File Version Info와 FileVersionInfo Out 인디케이터에 버전 정보를 반환합니다.

독립 어플리케이션에서 설정 파일 사용하기

LabVIEW 설정 파일 또는 다른 사용자 설정 파일을 LabVIEW 독립 어플리케이션과 함께 사용할 수 있습니다.

LabVIEW 설정 파일

LabVIEW 설정 파일을 독립 어플리케이션 파일과 함께 사용하기로 결정한 경우, LabVIEW는 labview 디렉토리에서 LabVIEW 설정 파일에 나타나는 설정 토큰의 서브셋을 포함하는 설정 파일을 생성합니다. 어플리케이션 프로퍼티 대화 상자를 사용하여 추가적으로 빌드 스펙 셋팅을 설정할 수 있습니다. 예를 들어, 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서 디버깅 활성화를 선택하면, LabVIEW는 독립 어플리케이션의 설정 파일에 디버깅 토큰을 추가합니다.

다음 단계를 완료하여 독립 어플리케이션에서 LabVIEW 설정 파일을 사용합니다.

1. **프로젝트 탐색기** 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기 >> 어플리케이션을 선택합니다. 이전에 어플리케이션에 대한 빌드 스펙을 생성한 경우, 프로젝트 탐색기 윈도우에서 빌드를 더블 클릭하여 빌드를 편집합니다.
2. 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서 사용자 정의 설정 파일 사용 확인란이 활성화되지 않도록 합니다.
3. (옵션) 고급 페이지에서 다른 설정을 선택할 수 있습니다. 이에 따라 독립 어플리케이션의 설정 파일에 설정 토큰이 추가될 수 있습니다.
4. 확인 버튼을 클릭하여 변경을 적용하고 대화 상자를 닫습니다.

사용자 설정 파일

독립 어플리케이션과 함께 사용자 설정 파일을 사용하는 경우, LabVIEW는 사용자가 템플릿으로 지정한 설정 파일을 사용합니다. LabVIEW는 어플리케이션 프로퍼티 대화 상자에서 선택한 옵션에 따라 추가 토큰을 사용자가 지정하는 설정 파일에 추가합니다. 예를 들어, 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서 디버깅 활성화를 선택하면, LabVIEW는 사용자가 포함시킨 사용자 설정 파일에 디버깅 토큰을 추가합니다.

다음 단계를 완료하여 독립 어플리케이션에서 사용자 설정 파일을 사용합니다.

1. 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기>>어플리케이션을 선택합니다. 이전에 어플리케이션에 대한 빌드 스펙을 생성한 경우, 프로젝트 탐색기 윈도우에서 빌드를 더블 클릭하여 빌드를 편집합니다.
2. 어플리케이션 프로퍼티 대화 상자의 고급 페이지에서, 사용자 정의 설정 파일 사용 확인란에 확인 표시를 합니다. 프로젝트 파일 선택 대화 상자가 나타납니다.
3. 프로젝트의 파일 리스트에서 포함시키고 싶은 설정 파일을 선택합니다. 포함시키려는 파일이 프로젝트 안에 없는 경우, 다음 단계를 완료합니다.
 1. 추가 버튼을 클릭합니다. 파일 대화 상자가 나타납니다.
 2. 포함시키려는 파일을 탐색하고 추가 버튼을 클릭합니다. 대화 상자가 닫히고, 사용자가 선택한 파일이 어플리케이션 프로퍼티 대화 상자의 고급 페이지에 있는 프로젝트 설정 파일 텍스트 박스에 나타납니다.
4. (옵션) 고급 페이지에서 다른 설정을 선택할 수 있습니다. 이에 따라 사용자 설정 파일에 설정 토큰이 추가될 수 있습니다.
5. 확인 버튼을 클릭하여 변경을 적용하고 대화 상자를 닫습니다.

LabVIEW로 만든 .NET Interop 어셈블리의 특징

없음

다른 프로그램에서 .NET Interop 어셈블리 호출하기

.NET Interop 어셈블리를 만든 후, .NET을 지원하는 다른 프로그램에서 해당 어셈블리를 호출할 수 있습니다. 그러나 다른 프로그램이 실행되는 컴퓨터는 다음의 필수조건과 권장사항을 충족해야 합니다.

- 사용자가 .NET Interop 어셈블리를 실행하는 컴퓨터에는 모두 LabVIEW 런타임 엔진이 설치되어 있어야 합니다. LabVIEW 런타임 엔진을 .NET Interop 어셈블리와 함께 배포할 수 있습니다. (Windows) 또한 설치 프로그램에 LabVIEW 런타임 엔진을 포함시킬 수 있습니다.
- National Instruments는 .NET Interop 어셈블리의 타겟 컴퓨터에 설치된 .NET Framework 버전이 LabVIEW가 어플리케이션을 만드는데 사용한 버전과 같거나 이후 버전인 것을 권장합니다.
- LabVIEW 개발 환경 밖에서 .NET interop 어셈블리를 호출하려면 Microsoft Visual Studio 프로젝트의 NationalInstruments.LabVIEW.Interop.dll을 참조해야 합니다. LabVIEW 런타임 엔진은 자동으로 이 DLL을 National Instruments\Shared\LabVIEW Run-Time 디렉토리로 설치합니다. 선택적으로, NationalInstruments.LabVIEW.Interop.dll를 Global Assembly Cache (GAC)에 직접 설치할 수 있습니다. 이 DLL을 GAC에 설치하려면 .NET 어셈블리를 강력한 이름 키 파일에 인증해야 합니다.
- 디버깅 가능한 .NET interop 어셈블리를 생성할 때 LabVIEW는 설정 파일(.ini)을 어셈블리와 동일한 디렉토리에 놓습니다. 다른 프로그램에서 어셈블리를 디버깅하려면, 해당 .ini 파일을 어셈블리와 함께 배포해야 합니다.

.NET Interop 어셈블리의 LabVIEW 데이터 타입

.NET interop 어셈블리를 만들 때 LabVIEW는 숫자형, 불리언, 문자열과 같은 단순 데이터 타입 그리고 단순 데이터 타입의 배열을 대응하는 .NET 데이터 타입으로 직접 바꿉니다. 그러나 클러스터, 웨이브폼, 복소수, 참조 번호, LabVIEW 클래스와 같은 LabVIEW 특정 데이터 타입의 경우, 생성된 어셈블리에서 LabVIEW가 새로운 .NET 데이터 타입을 정의해야 합니다.

LabVIEW가 자체의 데이터 타입을 .NET 데이터 타입으로 변환하는 한 가지 방법은 .NET interop 어셈블리를 만든 다음 Microsoft Visual Studio와 같은 .NET 프로그래밍 환경에서 어셈블리의 매니페스트를 보는 것입니다. 또한 Microsoft Developer Network (MSDN)에서 얻을 수 있는 MSIL 디스어셈블러 도구를 사용하여 생성된 어셈블리를 볼 수 있습니다. 그러나 다음 섹션에서는 LabVIEW가 클러스터, 열거형 타입 컨트롤, LabVIEW 클래스를 .NET 데이터 타입으로 변환하는 방법에 대해 기본적인 몇 가지 사항을 상세하게 설명합니다.

클러스터 및 열거형

LabVIEW는 클러스터와 열거형을 .NET 구조로 변환하며 이 구조의 원소는 클러스터나 열거형의 원소에 대응하게 됩니다. LabVIEW는 다음 규칙을 사용하여 클러스터나 열거형에 대응하는 .NET 구조의 이름을 정합니다.

- 타입 정의나 엄격한 타입 정의로 정의된 클러스터와 열거형 — LabVIEW는 타입 정의 또는 엄격한 타입 정의의 라벨을 .NET 구조의 이름으로 사용
- 다른 모든 클러스터와 열거형 — LabVIEW는 .NET 구조의 이름을 LVCluster_# 또는 LVEnum_#로 정함

LabVIEW 클래스

LabVIEW 클래스의 멤버 VI는 반출이 가능합니다. LabVIEW에서, 이러한 VI에는 소유 LabVIEW 클래스가 입력으로 필요합니다. 따라서 이러한 VI를 .NET interop 어셈블리로 반출하는 경우, LabVIEW는 어셈블리에서 상응하는 .NET 클래스 정의를 생성해야 합니다. .NET 클래스 정의는 각 멤버 VI의 정적 메소드를 포함합니다. 이와 같은 각각의 정적 메소드에는 파라미터로 새롭게 정의된 .NET 클래스의 인스턴스가 필요합니다. LabVIEW는 소유 LabVIEW 클래스의 프라이빗 데이터를 반출하지 않습니다.

LabVIEW는 또한 사용자가 반출하는 메소드 VI가 속한 LabVIEW 클래스의 상속 구조를 반출합니다. 생성된 .NET 클래스는 동일한 상속 구조를 갖습니다.

LabVIEW 에러

LabVIEW가 VI의 .NET 메소드를 생성하는 경우, LabVIEW는 에러 입력 및 에러 출력 클러스터를 새 메소드의 파라미터로 반출하지 않습니다. 그 대신, 메소드 실행 중 에러가 발생하면 새 메소드에 .NET 예외가 생깁니다. 이 예외는 에러 클러스터와 동일한 정보를 포함합니다.

LabVIEW로 만든 공유 라이브러리의 특징

없음

LVDLLStatus 함수

반출하는 함수 외에 LabVIEW에서 만든 모든 DLL은 다음의 원형을 가진 LVDLLStatus 라는 반출된 함수를 포함합니다.

```
MgErr LVDLLStatus(CStr errStr, int32 errStrLen, void *module)
```

호출 프로그램은 이 함수를 사용하여 LabVIEW DLL이 올바르게 로드되었는지를 확인합니다. DLL을 로드하는 동안 에러가 발생하는 경우, 함수는 에러를 반환합니다. 문자열 버퍼를 errStr 파라미터에 전달하여 에러에 대한 추가적인 정보를 받습니다.

errStrLen 파라미터를 errStr로 전달된 문자열 버퍼의 바이트의 개수로 설정합니다. module 파라미터를 사용하여 DLL에서 사용되는 LabVIEW 런타임 엔진에 대한 핸들을 가져올 수 있습니다. 일반적으로, 이 파라미터는 NULL으로 둘 수 있습니다.

[LVDLLStatus] 함수는 LabVIEW 런타임 엔진이 로드되는 동안 LabVIEW가 이 함수를 호출하는 경우, 실행을 차단합니다.



노트

LabVIEW에서 LabVIEW로 만든 DLL을 호출하는 경우, LabVIEW는 자동으로 LVDLLStatus 함수를 호출합니다.

SetExecuteVIsInPrivateExecutionSystem 함수

반출하는 함수 외에 LabVIEW에서 만든 모든 DLL은 다음의 원형을 가진 SetExecuteVIsInPrivateExecutionSystem라는 반출된 함수를 포함합니다.

```
Void SetExecuteVIsInPrivateExecutionSystem(Bool32 value)
```

이 함수를 사용하여 LabVIEW로 빌드한 공유 라이브러리를 다른 언어의 어플리케이션으로 호출할 때 LabVIEW가 멀티스레드 실행 시스템에서 VI를 실행할지 여부를 지정합니다. 값 파라미터에 1을 전달하면, LabVIEW로 빌드한 공유 라이브러리를 다른 언어의 어플리케이션으로 호출할 때 LabVIEW가 멀티스레드 실행 시스템에서 VI를 실행합니다. 이 함수를 호출하면, 이 함수의 값 파라미터가 공유 라이브러리 프로퍼티 대화 상자의 고급 페이지에서 프라이빗 실행 시스템에서 VI 실행 확인란의 값을 덮어씁니다.

(macOS) LabVIEW에서 공유 라이브러리의 일부를 만드려면, 어플리케이션 빌더는 Unix 및 macOS의 표준 명령 라인 컴파일러인 gcc를 사용해야 합니다. gcc를 설치하려면, Apple 컴퓨터의 표준 개발 환경인 Xcode를 사용하십시오. Xcode는 Mac App Store에서 무료로 다운로드할 수 있습니다.

메모리 할당

LabVIEW로 만든 공유 라이브러리는 핸들로 나타나는 각 LabVIEW 배열 타입에 대해 할당, 할당 해제 및 크기 조정 함수를 제공하여 이러한 배열 타입의 메모리 할당을 관리합니다. 생성된 헤더 파일을 열어 생성된 공유 라이브러리에 특정한 함수 원형을 봅니다.

설치 프로그램 배포 구성요소 캐시하기(Windows)

배포는 설치 프로그램을 이루는 파일의 집합입니다. 어플리케이션 빌더를 사용하여 추가적인 설치 프로그램이나 구성요소가 있는 **설치 프로그램을 만드는** 경우, 어플리케이션 빌더는 구성요소의 원래의 배포의 위치를 찾도록 입력요청합니다. 설치 프로그램 배포 구성요소를 캐시하면 해당 구성요소를 로컬 시스템의 영구적인 위치로 복사하여 설치 프로그램을 빌드할 때마다 추가 구성요소에 대해 위치를 지정할 필요가 없게 됩니다. 다음에 구성요소를 포함하는 설치 프로그램을 만들 때, 어플리케이션 빌더는 배포 미디어를 요구하지 않고 자동으로 로컬 드라이브에서 구성요소를 복사합니다.

다음 단계를 완료하여 설치 프로그램 배포 구성요소를 캐시합니다.

1. 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새로 만들기» 설치 프로그램을 선택합니다.
2. 항목 리스트에서 추가적인 설치 프로그램을 선택합니다.
3. 선택한 설치 프로그램 및 이후에 설치하려는 모든 프로그램을 이 컴퓨터에 복사하여, 설치 프로그램 생성 시 미디어 삽입 요청을 최소화하기 확인란에 확인 표시를 합니다.
4. 확인을 클릭합니다.



노트

LabVIEW Professional Development System은 어플리케이션 빌더를 포함합니다. LabVIEW Base Development System 또는 Full Development System을 사용하는 경우, ni.com에서 어플리케이션 빌더를 별도로 구매할 수 있습니다.

설치 프로그램 만들기에 대한 유의점과 권장사항

다음 리스트는 설치 프로그램을 만들 때 고려해야 할 몇가지 유의점 및 권장사항에 대해 설명합니다.

- 설치 프로그램 프로퍼티 대화 상자의 소스 파일 페이지에서 설치 프로그램 빌드에 포함시킬 파일을 선택할 때 빌드 스펙의 일부만 선택할 수 없습니다. 빌드 스펙 전체를 포함해야 합니다. 전체 스펙은 대상 보기 트리에 나타납니다.
- 설치 프로그램 빌드에 프로젝트 라이브러리를 포함하려는 경우, 프로젝트 라이브러리에 네트워크의 파일이나 이동하면 링크가 깨지는 다른 파일이 포함되지 않도록 하십시오. 파일이 다른 드라이브에 있거나 파일이 LabVIEW 프로젝트 파일(.lvproj)과 일반적인 경로를 공유하지 않는 경우, 프로젝트 파일 보기 디렉토리에서 대상 보기 디렉토리로 프로젝트 라이브러리를 복사할 수 없습니다.

- 네트워크 또는 다른 드라이브의 파일에 대한 링크가 깨지게 될 가능성을 낮추려면, 설치 프로그램 빌드에 들어갈 파일을 포함하는 소스 배포를 만드는 것을 고려하십시오.
- 설치 프로그램 빌드에 **National Instruments 제품 설치 프로그램을 추가**하면, 설치 프로그램에 포함된 NI 제품 드라이버와 구성 요소는 이 설치 프로그램을 빌드한 컴퓨터에 설치된 기능만을 포함합니다. 예를 들어, 빌드 컴퓨터에 설치된 NI 제품에서 가능한 기능이 20개이나 이 중 10개의 기능만이 있는 상태에서 해당 제품의 풀버전을 설치 프로그램에 포함하도록 선택하면, 설치 프로그램은 해당 제품에서 사용 가능한 20개의 기능이 아닌 빌드 컴퓨터에 설치된 10개의 기능만을 포함하게 됩니다. 생성한 설치 프로그램이 이 같은 구성 요소의 현재 버전을 다시 설치하지 않고도 설치된 컴퓨터의 구성 요소를 설치 해제할 수 있습니다. 이 같은 동작은 빌드 컴퓨터에 설치된 NI 제품과 설치 프로그램을 실행하는 컴퓨터에 설치된 제품에 따라 다릅니다. 설치 프로그램을 실행하는 컴퓨터에 빌드 컴퓨터에 설치되지 않은 NI 제품이 설치되어 있는 경우, 설치되지 않은 제품은 설치과정에서 제거될 수 있습니다. 설치 프로그램은 설치 중 구성 요소를 설치 해제하기 전 어떤 기능이 제거될 지를 보여주며 사용자에게 설치를 중단할 수 있는 옵션을 제공합니다.
- 설치 프로그램에 **사용자 에러 코드**를 포함시킬 수 있습니다. 대상 보기 디렉토리의 [LabVIEW x Run-Time] 폴더는 Shared\LabVIEW Run-Time\x.x 디렉토리에 대응됩니다. 여기서 xx는 현재 사용하는 LabVIEW 버전입니다. 고급 페이지의 사용자 에러 코드 파일 설치 확인란에 확인 표시를 한 경우, 설치 프로그램 빌드는 labview\project\errors와 labview\user.lib\errors의 모든 에러 코드 파일을 포함하며 이를 Shared\LabVIEW Run-Time\x.x\errors 디렉토리에 설치합니다.



노트

사용자가 직접 LabVIEW\user.lib 디렉토리 안에 errors 폴더를 생성해야 사용자 에러 코드 파일들을 정리할 수 있습니다.

- 설치 프로그램 빌드의 어플리케이션을 만들 때 메모리 부족 에러가 발생하는 경우, 다음 방법 중 하나를 사용하여 문제를 해결합니다:
 - 어플리케이션에 크기가 크거나 복잡한 VI가 있는 경우, LabVIEW가 편집기 응답을 실행 속도보다 우선하기 시작하는 복잡도 임계점을 낮춥니다. 이렇게 하면 LabVIEW는 빌드 과정에 더 많은 메모리를 할당할 수 있습니다.
 - 어플리케이션의 일부를 여러 개의 소스 배포로 나누는 방법으로 어플리케이션의 VI 개수를 줄이십시오. 어플리케이션의 VI는 참조에 의한 호출 노드를 사용하여 개별 영역을 호출합니다.
- 설치 프로그램 빌드의 독립 어플리케이션, 공유 라이브러리, .NET 어셈블리, 혹은 **웹 서비스**를 테스트할 때 로드 시간이 예상보다 오래 걸리면 VI 중 다른 VI가 한번 혹은 때때로 다이내믹하게 호출하는 부분을 제거하십시오. 예를 들어, 어플리케이션의 설정 대화 상자가 다수의 VI를 가지고 있고, 사용자가 설정 대화 상자를 로드해야 할 경우, VI 호출 설정 대화 상자를 사용하여 개별 호출마다 다시 로드 옵션을 선택해서 **SubVI가 언제 로드할지 설정**합니다. 어플리케이션, 공유 라이브러리 그리고 .NET 어셈블리를 생성할 때 추가적인 대상을 가질 수 있도록 빌드를 수정합니다. 로드 시간을 줄이려면, 다이내믹하게 호출되는 VI들이 주요 어플리케이션이나 공유 라이브러리로 가는 대신 추가적인 대상으로 가도록 설정합니다.
- 대상 보기 디렉토리의 [LabVIEW x] 폴더를 사용하여 사용자가 설치한 최신 버전 LabVIEW의 디렉토리에 설치할 파일을 추가합니다.

- 설치 프로그램의 모든 타입의 어플리케이션에 빌드 스펙을 포함시키려면, 빌드 스펙의 빌드 출력은 빌드 스펙의 주요 대상에 상대적이어야 합니다. 주요 대상은 어플리케이션의 프로퍼티 대화 상자의 대상 페이지에서 지정되어 있습니다. 빌드 출력에 주요 대상과 관련되지 않은 파일이 포함된 경우, 설치 프로그램은 어플리케이션의 원본 파일 계층구조를 재구성하게 됩니다. 특히 설치 프로그램은 주요 대상과 관련되지 않은 모든 파일을 이동시킵니다.
- LabVIEW로 만든 어플리케이션과 함께 LabVIEW 배포 구성요소를 사용할 때, 추가적인 라이선스 파일이 필요하지 않습니다.
- 설치 프로그램 프로퍼티 대화 상자의 소스 파일 페이지에 있는 대상 보기 트리의 폴더에 빌드 스펙을 추가할 때, 해당 빌드의 이름을 가진 폴더가 존재하지 않는 경우에도 빌드의 이름이 대상 보기 트리에 나타납니다.

설치 프로그램 빌드에 추가적인 National Instrument 설치 프로그램 포함하기

설치 프로그램을 빌드할 때, 설치 프로그램 빌드에 디바이스 또는 다른 National Instruments 제품의 설치 프로그램을 포함시켜야 하는 경우도 있습니다. 예를 들어, 스토리지/DataPlugin VI를 사용하는 독립 어플리케이션 또는 공유 라이브러리에 대한 설치 프로그램을 만들 때, NI-USB 설치 프로그램을 반드시 포함해야 합니다. 기본으로, LabVIEW는 National Instruments 제품을 설치한 컴퓨터에 설치 프로그램을 자동으로 추가합니다.



주의

설치 프로그램 빌드에 National Instruments 제품 설치 프로그램을 추가하는 경우, 사용하는 컴퓨터에 설치된 설치 프로그램의 구성요소만 추가하게 됩니다. 사용자가 빌드된 설치 프로그램을 실행할 때, 설치 프로그램은 사용자 컴퓨터에서 설치 프로그램의 일부가 아닌 제품 구성요소를 삭제할 수 있습니다. 사용자 컴퓨터에서 제품 구성요소 제거의 위험을 최소화하려면, 빌드에 제품 설치 프로그램을 추가하기 전에 사용하는 컴퓨터에 모든 옵션 구성요소를 포함한 제품의 최신 판이 완전히 설치되었는지 확인하십시오.

추가적인 설치 프로그램을 직접 선택하기

LabVIEW가 설치 프로그램을 자동으로 선택하는 것을 방지하려면 설치 프로그램의 설치 프로그램 프로퍼티 대화 상자의 추가적인 설치 프로그램 페이지에 있는 권장 설치 프로그램 자동 선택 확인란에서 확인 표시를 제거하십시오. 포함할 National Instruments 설치 프로그램 리스트에서 설치 프로그램에 포함시키려는 설치 프로그램을 선택할 수 있습니다. 이 리스트에는 다음 및 그 외의 설치 프로그램이 포함되어 있습니다:

- **NI LabVIEW 런타임 엔진**—어플리케이션 및 공유 라이브러리와 함께 이 설치 프로그램을 포함시킵니다. LabVIEW 런타임 엔진은 어플리케이션과 공유 라이브러리에서 **LabVIEW 리포트 생성 지원**을 제공하는 3D 그래프 지원과 NI 리포트를 포함합니다. 다음과 같은 LabVIEW 런타임 엔진의 구성 요소를 포함할지 지정할 수 있습니다.
 - Math Kernel 라이브러리—분석 VI가 특정 프로세서에서 사용하는 알고리즘을 위해 이 구성요소 포함.
 - NI ActiveX 컨테이너—설치 프로그램으로 NI ActiveX 객체, 프로퍼티, 메소드 및 이벤트를 배포하기 위해 이 구성요소 포함.

- NI 에러 리포트—어플리케이션 충돌 및 내부 경고를 감지하기 위해 이 구성요소 포함. 또한 이 구성요소를 활성화하면 디스크에서 메모리 덤프 및 로그 파일을 포함하는 리포트를 생성할 수 있어 문제를 해결할 수 있습니다.
- NI LabVIEW Real-Time NBFifo—데스크탑에서 LabVIEW Real-Time FIFO를 위해 포함.
- NI LabVIEW 런타임 엔진 영어외 다른 언어 지원—LabVIEW 런타임 엔진이 어플리케이션 타입을 실행하는 OS의 언어로 에러 리포트를 생성하기 위해 이 구성요소를 포함.
- NI LabVIEW 런타임 엔진용 웹 서버—독립적인 웹 서버와 공유 라이브러리를 위해 포함.
- NI System Web Server—LabVIEW 및 Measurement & Automation Explorer와 같이 National Instruments 소프트웨어 프로그램 사이의 통신을 위해 웹 서비스를 사용할 때 이 구성요소 포함.
- NI Logos 5.4—네트워크 공개 공유 변수를 사용하는 네트워크 프로토콜을 위해 이 구성요소 포함.
- NI TDM Streaming—파일 I/O 작업을 수행하기 위해 TDMS VI를 사용할 때 이 구성요소 포함.
- NI VC2008MSMs—LabVIEW 런타임 엔진이 이전에 설치되어 있지 않은 컴퓨터에서 LabVIEW 런타임 엔진을 로드하고 실행하기 위해 Microsoft Visual C++ Runtime을 설치할 때 이 구성요소 포함.
- NI VC2010MSMs—이 구성요소는 .NET 지원이 포함된 LabVIEW 런타임 엔진의 로드 및 실행을 위해 Microsoft Visual C++ Runtime을 설치합니다. LabVIEW 런타임 엔진이 이미 설치되어 있지 않은 컴퓨터의 경우 이 구성요소 포함.
- NI LabWindows/CVI; 런타임 엔진—이 설치 프로그램을 LabWindows™/CVI™에 내장된 공유 라이브러리(DLLs)와 함께 포함시킵니다.
- NI USI—어플리케이션 또는 공유 라이브러리가 스토리지/DataPlugin VI를 사용하는 경우 이 설치 프로그램을 포함시킵니다.
- NI LabVIEW DSC Module 런타임—어플리케이션 또는 공유 라이브러리가 보안, 알람, 로깅 등의 LabVIEW DSC Module 기능을 사용하는 경우 이 설치 프로그램을 포함시킵니다.
- NI DataSocket—어플리케이션 또는 공유 라이브러리가 **DataSocket**을 사용하는 경우 이 설치 프로그램을 포함시킵니다. DataSocket 서버, DataSocket 서버 관리자, OPC Demo 서버를 포함시킵니다.
- NI Measurement & Automation Explorer (MAX)—National Instruments 하드웨어를 설정 및 테스트하고 이와 통신하기 위해 이 설치 프로그램을 포함.
- **NI TDM Excel Add-In**—Microsoft Excel로부터 .tdm와 .tdms 파일을 로드할 수 있도록 이 설치 프로그램을 포함시킵니다.
- NI 배포된 시스템 관리자—변수, 프로세스, I/O 서버, 웹 서비스를 생성 및 모니터링하고, 공유 변수 엔진과 상호 작용하며, 보안 및 가명을 관리하는 어플리케이션.
- NI 변수 엔진—공유 변수를 호스트하거나, 설정 또는 생성하려는 어플리케이션 및 공유 변수를 사용하여 실시간 데이터를 읽고 쓰려는 어플리케이션에 이 설치 프로그램을 포함시킵니다.
- NI 웹 기반 설정 및 모니터링—NI 웹 기반 설정 및 모니터링을 사용하려면 이 설치 프로그램을 포함시킵니다. NI 웹 기반 설정 및 모니터링을 사용하면 RT 타겟과 같은 원격 디바이스뿐만 아니라 어플리케이션 웹 서버도 설정 및 제어할 수 있습니다.

LabVIEW로 만든 설치 프로그램 내부적으로 설치하기

명령 라인 윈도우를 사용하여 LabVIEW로 만든 설치 프로그램을 내부적으로 설치할 수 있습니다. 명령 라인 윈도우를 사용하여 LabVIEW로 만든 설치 프로그램을 내부적으로 설치하려는 경우, 설치 프로그램의 크기를 고려해야 합니다. 설치 프로그램에 설치 과정에서 진행 막대를 디스플레이하지 않기 때문에, 설치 프로그램의 진행 상황을 파악하기 힘들 수도 있습니다. 따라서, 설치 프로그램이 클수록 시간이 많이 소요되고 사용자는 앞으로 얼마나 오래동안 설치 프로그램이 실행될지를 알 수 없습니다.

다음 단계를 완료하여 LabVIEW로 만든 설치 프로그램을 내부적으로 설치합니다.

1. LabVIEW로 만든 어플리케이션을 위해 **LabVIEW로 만든 설치 프로그램을 생성**합니다.
2. 명령 라인 윈도우를 엽니다.
3. 다음 명령을 입력하여 설치 프로그램을 내부적으로 실행합니다.
`path \install.exe /q /acceptlicenses yes`(이때, path는 `install.exe`의 경로임)

LabVIEW로 만든 설치 프로그램 설치 제거하기

명령 라인 윈도우를 사용하여 LabVIEW가 만든 설치 프로그램과 설치한 파일을 설치 제거할 수 있습니다. 다음 단계를 완료하여 LabVIEW로 만든 설치 프로그램을 설치 제거합니다.

1. 명령 라인 윈도우를 열고 NIUninstaller 또는 `uninst.exe` 파일을 포함하는 디렉토리로 변경합니다. `uninst.exe` 파일은 `C:\Program Files\National Instruments\Shared\NIUninstaller` 디렉토리 또는 (Windows 64-bit)의 경우 `C:\Program Files (x86)\National Instruments\Shared\NIUninstaller`에 있습니다.
2. 다음의 명령을 입력하여 설치 프로그램을 내부적으로 설치 제거합니다.
`uninst.exe /qb /x " product name"`(여기서 `product name`은 제거하려는 제품의 이름입니다.)



노트

`product name`은 대소문자를 구분합니다. 정확한 이름을 사용하려면 제어판의 프로그램 및 기능에서 나타나는 제품 이름을 그대로 사용합니다. 또한, 공백이 있는 제품 이름에는 따옴표를 사용합니다.

제어판의 프로그램 및 기능을 사용하여 LabVIEW를 사용하여 만든 설치 프로그램을 설치 제거할 수도 있습니다. 프로그램 및 기능을 사용하여 설치 프로그램을 설치 제거해도 이 설치 프로그램과 함께 설치된 NI 구성요소는 제거되지 않습니다. 이러한 NI 구성요소는 따로 설치 제거해야 합니다.

TDM Excel Add-In 사용하기 (Windows)

TDM Excel Add-In을 사용하여, Microsoft Excel에 .tdm와 .tdms 파일을 로드할 수 있습니다. Excel에 있는 도구 모음에서 파일, 그룹, 채널 수준에서 Excel에 로드할 프로퍼티(사용자 정의 프로퍼티도 포함)를 선택합니다.

TDM Excel Add-In에 대한 더 자세한 정보는 ni.com에서 Editing TDM(S) files with the TDM Excel Add-In을 참조하십시오.



관련 정보

[TDM Excel Add-In으로 TDM\(S\) 파일 편집하기](#)

어플리케이션 매니페스트 작업하기

Microsoft는 어플리케이션의 **필요한 실행 레벨**을 asInvoker로 지정할 것을 적극 권장합니다. 따라서, asInvoker는 LabVIEW에서 생성한 모든 어플리케이션에서 기본적인 필요한 실행 레벨입니다. 그러나 asInvoker 외의 값을 사용해야 하는 경우도 있습니다. 다음 단계에 따라 독립 어플리케이션, .NET interop 어셈블리 또는 공유 라이브러리의 매니페스트를 편집합니다.

어플리케이션 매니페스트 편집하기

1. 매니페스트를 편집하기 전 먼저 컴퓨터에 mt.exe가 설치되었는지 확인하십시오. mt.exe를 사용하여 명령 라인 옵션을 사용하는 매니페스트와 상호 작용하십시오. 컴퓨터에 mt.exe가 없는 경우, Microsoft 웹 사이트에서 Software Development Kit을 다운로드하십시오.
2. 매니페스트를 편집하려면 먼저 매니페스트를 검색해야 합니다. 매니페스트를 검색하려면 다음을 수행하십시오.
 1. 명령 라인 윈도우를 엽니다.



노트

대부분의 Windows OS에서는 시작 >> 실행을 선택하고, 열기 텍스트 박스에 cmd를 입력한 후 확인을 클릭하면 명령 라인 윈도우를 열 수 있습니다.

2. 명령 라인 윈도우에서 다음 명령을 입력하여 매니페스트를 검색합니다.
`mt.exe -inputresource: directory path \ Application Name .exe -out: Application Name .manifest`(여기에서 directory path는 내장 어플리케이션에 대한 경로이며 Application Name은 어플리케이션의 이름임)

노트 mt.exe가 검색하려는 매니페스트가 있는 어플리케이션과 같은 디렉토리에 없는 경우, 이전 명령에서 mt.exe에 대한 경로를 지정해야 합니다.

3. <Enter> 키를 눌러서 명령을 실행합니다. 파일 확장명 .manifest가 있는 어플리케이션 매니페스트가 어플리케이션과 같은 디렉토리에 나타납니다.
3. 다음을 수행하여 매니페스트를 편집하십시오.

1. 대상 폴더의 매니페스트에서 마우스 오른쪽 버튼을 클릭하고 텍스트 편집기나 XML 편집기에서 파일을 엽니다.
2. 다음 코드를 위치합니다: `<requestedExecutionLevel level="asInvoker" uiAccess="false">`. `asInvoker` 값을 내장 어플리케이션의 실행 레벨에 따라 `requireAdministrator` 또는 `highestAvailable`으로 변경하십시오.
3. 매니페스트 파일을 저장합니다.

어플리케이션 매니페스트 확인하기

어플리케이션 매니페스트(Manifest) 내의 태그는 대소문자를 구분합니다. 편집 후에 매니페스트에 대한 변경사항을 확인하는 것이 좋습니다. 그러나 매니페스트를 확인해도 맞춤법 에러를 찾지는 못합니다. 매니페스트를 확인하려면 다음을 수행하십시오.

1. 명령 라인 윈도우에서 다음의 명령을 입력하여 매니페스트를 확인하십시오:
`mt.exe -manifest Application Name.manifest -validate_manifest` 여기서 Application Name은 어플리케이션의 이름입니다.
2. <Enter> 키를 눌러서 명령을 실행합니다.

어플리케이션 매니페스트 임베드하기

어플리케이션 매니페스트를 편집하고 검증한 후, 독립 어플리케이션, .NET interop 어셈블리 또는 공유 라이브러리의 설정을 설정할 때 변경사항을 임베드할 수 있습니다.

1. 프로퍼티 대화 상자의 Windows 보안 페이지에서, 매니페스트 파일 임베드 확인란에 확인 표시를 합니다.
2. 프로젝트 파일 선택 대화 상자를 사용하여 매니페스트 파일을 지정합니다.

Windows 보안 페이지를 사용하지 않고 매니페스트를 임베드할 수도 있습니다. 다음 단계에 따라 명령 라인 윈도우를 사용하여 매니페스트를 임베드합니다.

1. 명령 라인 윈도우에 명령 `mt.exe -manifest Application Name.manifest -outputresource: directory path \Application Name.exe;#1`을 입력합니다. 여기에서 directory path는 빌드된 어플리케이션에 대한 경로이며 Application Name은 어플리케이션의 이름입니다.
2. <Enter> 키를 눌러서 명령을 실행합니다.

LabVIEW의 명령 라인 인터페이스를 사용하여 작업 실행하기

LabVIEW의 명령 라인 인터페이스 (CLI)를 사용하여 LabVIEW에서 **미리 정의된 일련의 작업**을 실행할 수 있습니다. 예를 들어, LabVIEW의 CLI를 사용하여 LabVIEW 어플리케이션의 빌드 프로세스를 자동화합니다. 또한 생성한 **사용자 정의 작업**을 LabVIEW에서 실행

할 수 있습니다. LabVIEW의 CLI를 사용하여 작업을 실행할 때 발생할 수 있는 알려진 유의 사항을 확인하려면 labview\readme에서 LabVIEW Readme를 참조하십시오.



노트

LabVIEW의 CLI는 별도의 패키지로 제공됩니다. LabVIEW의 CLI 및 호환되는 LabVIEW 버전 다운로드에 대한 더 자세한 정보는 NI 웹 사이트를 참조하십시오.

LabVIEW의 CLI를 사용하여 작업을 실행하기 전에 도구 >> 옵션 >> VI 서버를 선택하고 TCP/IP 확인란에 확인 표시를 하여 LabVIEW의 CLI와 통신하도록 LabVIEW를 설정합니다.

LabVIEW의 CLI를 사용하여 작업을 실행하려면, 다음 인수 중 하나 이상을 사용하여 명령을 실행합니다. 필수 인수는 굵은 체로 표시됩니다.



노트 모든 인수는 대소문자를 구분합니다.



팁 LabVIEW의 CLI 사용에 대해 더 자세히 알아보려면 LabVIEWCLI -Help 명령을 실행합니다. 특정 작업에 대해 더 자세히 알아보려면 LabVIEWCLI -OperationName < operation name > -Help 명령을 실행합니다. 예를 들어, **RunVI** 작업에 대해 더 알아보려면 LabVIEWCLI -OperationName RunVI -Help 명령을 실행합니다.

인수	설명
LabVIEWCLI -OperationName	특정한 작업을 실행합니다. 작업에 추가적인 인수가 있는 작업 이름 뒤에 인수를 추가합니다.
-AdditionalOperationDirectory	작업에 필요한 파일을 저장할 디렉토리를 지정합니다. 이 사용자 정의 작업을 실행하고, 이 작업에 필요한 파일이 디렉토리에 저장되어 있지 않은 경우에만 필요합니다. (Windows) C:\Program Files (x86)\National Instruments\Shared\LabVIEW CLI\Operations (macOS) /Library/Application Support/National Instruments/LabVIEW CLI/Operations (Linux) /usr/local/natinst/nilvcli/Operations
-LabVIEWPath	작업을 실행하는데 사용할 LabVIEW 버전을 지정합니다. macOS 및 Linux에서는 필수이지만 Windows에서는 선택 사항입니다. Windows의 경우, 기본값은 이 머신에서 가장 사용한 LabVIEW 버전입니다.
-PortNumber	VI의 포트를 지정합니다. 이 인수는 도구 >> 옵션 >> VI 서버 >> 아래의 포트 번호가 3363이 아닌 경우 필요합니다.
-LogFilePath	LabVIEW의 CLI 로그 파일의 경로를 지정합니다. 기본값은 위치의 경로입니다.

인수	설명
-LogToConsole	출력을 콘솔과 로그 파일 모두에 저장할지 지정합니다. 이 값은 다음 중 하나여야 합니다. - 참 - 거짓 기본값은 참입니다.
-Verbosity	로그 파일에 어느 정도의 세부사항을 포함할지 지정합니다. 인수의 값은 다음 중 하나여야 합니다. - Minimal - 기본 - Detailed - Diagnostic 기본값은 Default입니다.

사용자 정의 명령 라인 작업 생성하기

LabVIEW 명령 라인 인터페이스 (CLI)에서 **실행** 하기 위해 사용자 정의 명령 라인 작업을 생성하려면, 다음 태스크를 완료하십시오.

- 1단계: 작업 이름을 정의합니다.
- 2단계: 작업 동작을 정의합니다.
- 3단계(옵션): 작업에 대한 도움말을 정의합니다.

1단계: 작업 이름 정의하기

LabVIEW의 CLI에서 명령 라인 작업은 상위 클래스를 가지며, 각 작업은 상위 클래스의 하위 클래스로부터 만들어 집니다. 작업의 이름을 정의하려면, 상위 클래스에 하위 클래스를 생성하고 이 하위 클래스의 이름을 지정해야 합니다.

다음 단계를 따라 작업의 이름을 정의합니다.

- 다음 디렉토리 중 하나로 이동합니다.
 - (Windows) C:\Program Files (x86)\National Instruments\Shared\LabVIEW CLI\Operations
 - (macOS) /Library/Application Support/National Instruments/LabVIEW CLI/Operations
 - (Linux) /usr/local/natinst/nilvcli/Operations

2. 작업이 LabVIEW 개발 시스템만 필요한지 또는 추가적인 LabVIEW 모듈 및 툴킷이 필요한지에 따라 `Operations.lvproj` 또는 `Toolkit-Operations.lvproj`를 엽니다.



노트

`Toolkit-Operations.lvproj`를 열려면 다음 툴킷과 라이브러리를 설치해야 합니다.

- LabVIEW VI Analyzer Toolkit
- LabVIEW Unit Test Framework Toolkit
- UTF JUnit Report 라이브러리—JKI VI Package Manager (VIPM) 소프트웨어를 사용하여 이 라이브러리를 설치합니다.

3. 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후 새로 만들기>>클래스를 선택합니다.
4. 새 클래스 윈도우에서 `AddTwoNumbers`와 같은 클래스 이름을 지정합니다. 클래스 이름은 생성 중인 작업의 이름입니다.
5. 프로젝트 트리에서 생성한 클래스를 마우스 오른쪽 버튼으로 클릭하고 프로퍼티를 선택합니다.
6. 상속 페이지에서 직상위 클래스 변경을 클릭합니다.
7. `CoreOperation.lvclass`를 선택하고 선택된 것에서 상속을 클릭한 후 확인 클릭합니다.

2단계: 작업 동작 정의하기

작업의 동작을 정의하려면 이전 단계에서 만든 클래스의 메소드를 덮어쓰고, 덮어쓴 메소드 중 하나에 코드를 추가해야 합니다.

다음 단계를 따라 작업의 동작을 정의합니다.

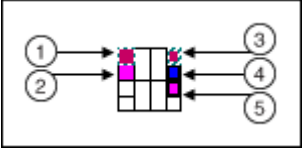
1. 프로젝트 트리에서, 이전 단계에서 생성한 클래스를 마우스 오른쪽 버튼으로 클릭하고 새로 만들기>>덮어쓰기 위한 VI를 선택합니다.
2. 새로 덮어쓰기 윈도우에서 `GetHelp.vi` 및 `RunOperation.vi`를 선택하고 확인을 클릭합니다.
3. 프로젝트를 저장합니다. 입력 요청을 받으면, 클래스 및 덮어쓰인 메소드를 작업과 같은 이름의 폴더에 저장합니다.

노트 NI는 다음 디렉토리 중 하나에 폴더를 두도록 권장합니다. 이렇게 하면 작업을 실행할 때 `-AdditionalOperationDirectory` 인수를 사용하여 폴더의 경로를 지정할 필요가 없습니다.

- (Windows) `C:\Program Files (x86)\National Instruments\Shared\LabVIEW CLI\Operations`
- (macOS) `/Library/Application Support/National Instruments/LabVIEW CLI/Operations`
- (Linux) `/usr/local/natinst/nilvcli/Operations`

4. `RunOperation.vi` 메소드의 블록다이어그램을 엽니다.
5. SubVI를 제거하고 다시 연결합니다.

6. 코드를 추가하여 작업의 동작을 정의하고 변경 사항을 저장합니다. 다음 그림과



같이, 커넥터 팬을 수정하지 마십시오.

터미널	데이터 타입	설명
		이전 섹션에서 생성한 클래스의 LabVIEW 객체입니다.
		이 작업을 실행할 때 사용자가 LabVIEW의 CLI에 입력하는 인수입니다. 배열의 각 문자열은 인수를 나타냅니다.
		이전 섹션에서 생성한 클래스의 LabVIEW 객체입니다.
		LabVIEW의 CLI가 Jenkins 애플리케이션과 같은 호출 애플리케이션에 반환하는 코드. 예를 들어, LabVIEW가 성공적으로 실행되면 0을 반환하고 에러가 발생하면 -1을 반환하도록 LabVIEW의 CLI를 설정할 수 있습니다.
		LabVIEW의 CLI 콘솔에 표시하거나 로그 파일에 저장할 작업 출력 또는 에러 메시지.

두 개의 입력 숫자를 더하고 합계를 출력에 반환하는 AddTwoNumbers 작업의 동작을 정의하는 예는 다음 디렉토리의 RunOperation.vi에서 참조할 수 있습니다.

- (Windows) C:\Users\Public\Documents\National Instruments\LabVIEW CLI\Examples\AddTwoNumbers
- (macOS) /Library/Application Support/National Instruments\LabVIEW CLI\Examples\AddTwoNumbers
- (Linux) /usr/local/natinst/nilvcli/Examples/AddTwoNumbers

3단계: 작업에 대한 도움말 정의하기

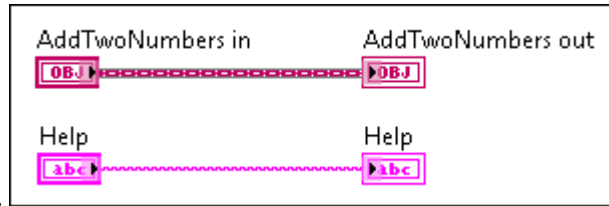
사용자가 작업을 용이하게 시작하도록, 이전 섹션에서 생성한 GetHelp.vi 메소드를 수정하여 이 작업을 문서화할 수 있습니다. 사용자는 LabVIEWCLI - OperationName <작업 이름> -Help 명령어를 실행하여 문서에 접근할 수 있습니다.

다음 단계에 따라 작업을 문서화합니다.

1. 프로젝트 트리에서 GetHelp.vi 메소드를 엽니다.
2. 프런트패널에서 문자열 컨트롤을 생성하고 Help로 이름을 지정한 후, 이 컨트롤에 문서를 입력합니다.
3. Help 컨트롤에서 마우스 오른쪽 버튼을 클릭하고 데이터 처리>>현재값을 기본값으로 선택하여 이 텍스트를 컨트롤의 기본값으로 설정합니다. 텍스트를 편집할 때마다 이 단계를 반복합니다.

4. 블록다이어그램에서 SubVI를 **제거하고 다시 연결**합니다.
5. 다음과 같은 방법으로 객체를 연결합니다. 이 때 AddTwoNumbers는 작업 이름입

니다. 변경 사항을 저장하십시오.



TCP 기반 NI-PSP의 기본 포트 변경하기 (NI Linux Real-Time, Windows)

다음 단계에 따라 LogosXT.ini 파일을 생성하고, 이를 통해 **NI Publish-Subscribe Protocol (NI-PSP)**이 사용 가능성을 점검하는 TCP 포트를 수정합니다.

1. 사용자의 OS에 적절한 디렉토리인 Windows 디렉토리에 새 텍스트 문서를 생성합니다.
 1. Windows를 설정하여 숨겨진 모든 폴더를 나타냅니다.
 2. 일반 어플리케이션 폴더의 \National Instruments\Logos XT 서브폴더로 갑니다. 이 디렉토리의 위치는 LabVIEW 설치를 설정하는 방식에 따라 다릅니다. 기본 위치는 C:\ProgramData\National Instruments\Logos XT입니다. 이 서브폴더가 아직 없는 경우, 이 폴더를 생성하십시오.
 3. 새 텍스트 문서를 생성합니다.
 4. 새 텍스트 문서 LogosXT.ini에 이름을 지정하고 문서를 Logos XT 서브폴더에 놓습니다.

NI Linux Real-Time 디렉토리

1. RT 타겟에 관리자 권한으로 로그인합니다.
2. /etc/natinst/ 디렉토리로 갑니다.
3. 새 텍스트 문서를 생성합니다.
4. 새 텍스트 문서를 LogosXT.ini로 이름붙입니다.



노트

파일 시스템은 대소문자를 구별합니다.

2. LogosXT.ini를 열고 다음의 텍스트를 입력합니다. [LogosXT] LogosXT_PortBase = 59110 LogosXT_NumPortsToCheck = 100

노트 LogosXT_PortBase는 NI-PSP 프로토콜이 사용가능성 점검을 시작하는 포트를 지정합니다. LogosXT_NumPortsToCheck는 프로토콜이 점검하는 최대 포트 개수를 지정합니다. 기본적으로, NI-PSP는 포트 59110에서 포트 사용가능성을 시작하고 최대 100개의 포트를 점검합니다.

3. NI-PSP 프로토콜이 사용가능성을 점검하는 첫번째 포트를 수정하려면, 59110를 삭제하고 LogosXT_PortBase에 대한 다른 포트 번호를 입력하십시오.. NI-PSP 프로토콜이 사용가능성을 점검하는 최대 포트의 개수를 수정하려면, 100을 삭제하고 LogosXT_NumPortsToCheck에 대한 다른 개수를 입력하십시오..

4. LogosXT.ini를 저장하고 닫습니다. LabVIEW를 다시 실행하면 변경사항이 적용됩니다.

JSONP 지원

대부분의 브라우저에서 동일 출처 정책을 방지하기 위해 JSONP를 사용하려면, 정상적으로 모든 요청을 하고 \$callback 파라미터를 한 개 더 추가합니다. 예: GET `http://my_server:3580/nivariable/VariableObjects('ni.var.psp://localhost')/type?$format=json&$callback=my_callback`

JSON 데이터는 \$callback 파라미터에 제공한 my_callback 함수에 파라미터로 전달됩니다. JSONP 지원은 JSON 포맷이 요청된 경우에만 유효합니다. XML 포맷이 요청된 경우 JSONP 지원은 무시됩니다.

웨이브폼 형

LabVIEW의 웨이브폼은 다음 네 개의 구성요소를 이루어진 클러스터입니다: t0, dt, Y, 속성. JSON에서 웨이브폼은 JSON 객체 형의 숫자값으로 나타납니다. XML에서 LabVIEW는 이 값을 XML 원형 VI로 패턴화합니다. "<" 및 ">"은 "<" 및 ">"로 이스케이프되고 값 프로퍼티의 문자열 값으로 저장됩니다.



노트

JSON형은 항상 기본 에러 상수를 포함합니다.

DataSocket 함수로 버퍼링 사용하기

다음 단계를 따라 DataSocket 함수를 사용하여 클라이언트 측 버퍼링을 활성화합니다. 구축하는 각 클라이언트 연결에 대해 이 단계를 마쳐야 합니다.

1. 데이터 연결을 엽니다.
2. 데이터 클라이언트의 버퍼 크기를 지정합니다.
3. 서버가 쓴 버퍼된 데이터를 읽습니다.
4. 버퍼 크기를 모니터합니다.
5. 데이터 연결을 닫습니다.
6. 블록 다이어그램의 모든 에러 입력과 에러 출력 터미널을 연결합니다.

또한, 공유 데이터가 DataSocket 함수를 사용할 때 문자열?

`maxbytes=1000&maxpackets=10`을 URL의 끝에 추가하여 버퍼링을 활성화할 수 있

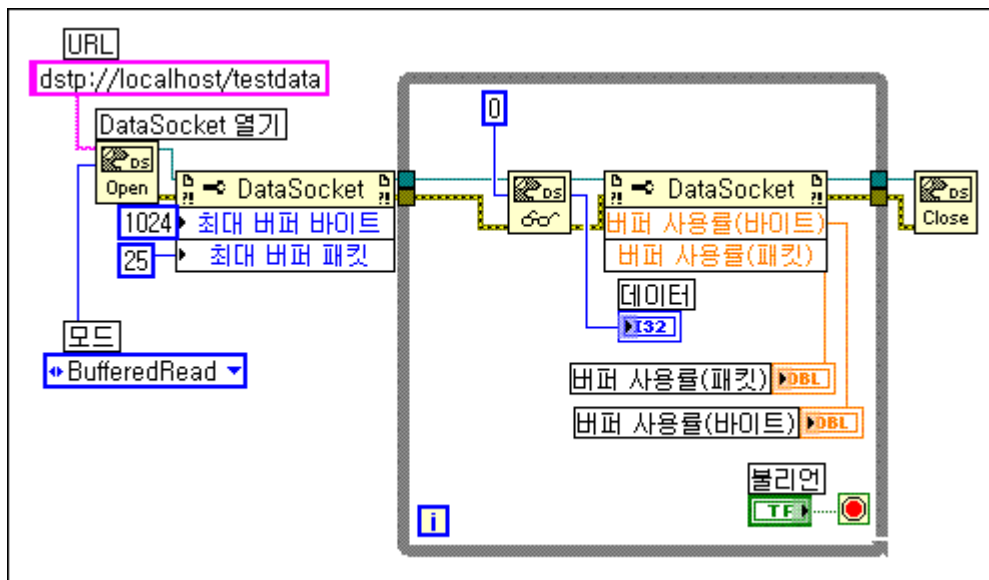
습니다. 이 때 1000은 버퍼가 포함하는 바이트의 최대 개수이고 10은 버퍼가 포함하는 패킷의 최대 개수입니다.



노트

버퍼링은 서버가 쓰는 데이터를 [DataSocket 읽기] 함수를 사용하여 읽을 때에만 적용됩니다. 프런트패널 DataSocket 데이터 연결을 사용하여 서버가 쓰는 데이터를 읽을 때에는 버퍼링을 사용할 수 없습니다. 공유 변수 프로퍼티 대화 상자의 네트워크 페이지에 있는 버퍼링을 활성화하고 컨트롤을 공유 변수에 연결한 경우, 공유 변수 엔진을 통해 프런트패널 데이터 연결에 버퍼링을 사용할 수 있습니다.

다음 예제는 DataSocket 함수로 버퍼링을 사용합니다.



관련 정보

[데이터 버퍼링하기](#) (598 페이지 참조)

[프로그램적으로 데이터 연결 열기와 닫기](#) (1478 페이지 참조)

[DataSocket 함수를 사용하여 버퍼 크기 지정하기](#) (1479 페이지 참조)

[서버가 쓴 버퍼된 데이터를 읽기](#) (1479 페이지 참조)

[데이터 연결의 버퍼 크기 모니터하기](#) (1480 페이지 참조)

[프로그램적으로 데이터 연결 열기와 닫기](#) (1478 페이지 참조)

[VI에서 여러 I/O 설정하기](#) (1388 페이지 참조)

[실시간 데이터를 프로그램적으로 공유하기](#) (593 페이지 참조)

[프로그램적으로 실시간 데이터 읽기](#) (1481 페이지 참조)

[dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 프런트패널 데이터 읽기](#) (1484 페이지 참조)

[컨트롤 및 인디케이터로 공유 변수 읽고 쓰기](#) (585 페이지 참조)

[공유 변수 기술 이해하기](#) (564 페이지 참조)

프로그램적으로 데이터 연결 열기와 닫기

[DataSocket 열기]와 [DataSocket 닫기] 함수를 사용하여 프로그램적으로 데이터 연결을 열고 닫습니다.

다음 단계를 따라 데이터 연결을 프로그램적으로 열고 닫습니다.

1. 블록다이어그램에 [DataSocket 열기] 함수를 추가합니다.
2. pspURL을 URL 입력에 연결하거나 공유 변수 컨트롤을 URL 입력에 연결합니다.
3. 모드 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>> 상수를 선택합니다.
4. 모드 상수를 클릭하고 Read(0), Write(1), ReadWrite(2), BufferedRead(3) 또는 BufferedReadWrite(4)를 선택합니다. 또한, DataSocket:모드 프로퍼티를 사용하여 프로그램적으로 데이터 연결을 설정할 수 있습니다.
5. 다음 함수 중 하나를 블록다이어그램에 추가하고, 적절한 입력에 대해 컨트롤 또는 상수를 생성합니다:
 - [DataSocket 읽기] 함수
 - [DataSocket 쓰기] 함수
6. [DataSocket 열기] 함수의 연결 id 출력을 [DataSocket 읽기] 또는 [DataSocket 쓰기] 함수의 연결 입력 입력에 연결합니다.
7. 블록다이어그램에 [DataSocket 닫기] 함수를 추가합니다.
8. [DataSocket 읽기] 또는 [DataSocket 쓰기] 함수의 연결 출력 출력을 [DataSocket 닫기] 함수의 연결 id 입력에 연결하여 데이터 연결을 닫습니다. 또한, VI가 정지될 때에도 연결은 닫힙니다.
9. 블록다이어그램의 모든 에러 입력과 에러 출력 터미널을 연결합니다.

실시간 데이터를 프로그램적으로 쓰기

프로그램적으로 실시간 데이터를 써서 프로젝트에 있지 않은 VI로부터의 실시간 데이터 업데이트를 보내거나, 데이터 연결의 프로퍼티를 컨트롤하고 보거나, 데이터를 VI의 프런트패널에 디스플레이하지 않고 다른 VI에 실시간 데이터 업데이트를 보내거나, 프런트패널의 데이터를 쓸 때 발생할 수 있는 데이터 손실을 피합니다.

(Real-Time, Windows) National Instruments는 공유 변수 노드를 사용하여 블록다이어그램에서 실시간으로 데이터를 쓰도록 권장합니다.

다음 단계를 따라 프로그램적으로 실시간 데이터를 씁니다.

1. 블록다이어그램에 [DataSocket 쓰기] 함수를 추가합니다.
2. [DataSocket 쓰기] 함수의 연결 입력 입력에 pspURL을 지정하거나 공유 변수 컨트롤을 연결합니다.
3. 쓰려는 데이터를 [DataSocket 쓰기] 함수의 데이터 입력에 연결합니다. 다른 어플리케이션이 실시간 데이터를 읽는 경우, 배리언트 데이터로 데이터를 변환합니다.

4. VI를 실행하거나 연속적으로 VI를 실행하여 실시간 데이터 업데이트를 데이터 연결에 씁니다.

DataSocket 함수를 사용하여 버퍼 크기 지정하기

다음 단계를 따라 DataSocket 함수를 사용하여 클라이언트 측 버퍼 크기를 지정합니다.

1. 블록다이어그램에 [DataSocket 열기] 함수를 추가합니다.
2. pspURL을 URL 입력에 연결하거나 공유 변수 컨트롤을 URL 입력에 연결합니다.
3. 모드 입력에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 생성>> 상수를 선택합니다.
4. 모드 상수를 클릭하고 BufferedRead를 선택합니다.
5. 블록다이어그램에 프로퍼티 노드를 추가합니다.
6. 프로퍼티 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 클래스 선택>>DataSocket>>DataSocket을 선택합니다.
7. 프로퍼티 노드의 크기를 재조정하고 터미널을 추가합니다.
8. 프로퍼티 노드의 첫번째 하얀 영역을 클릭하고 최대 버퍼 바이트 프로퍼티를 선택합니다.
9. 프로퍼티 노드의 두번째 하얀 영역을 클릭하고 최대 버퍼 패킷 프로퍼티를 선택합니다.
10. 프로퍼티 노드에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 모든 것을 쓰기로 변경을 선택합니다.
11. 각 프로퍼티를 위한 상수를 생성하고 최대 버퍼 바이트에 1024 그리고 최대 버퍼 패킷에 25와 같이 각 프로퍼티에 값을 입력합니다. LabVIEW는 첫번째 버퍼 크기 리미트가 최대 버퍼 바이트 프로퍼티 또는 최대 버퍼 패킷 프로퍼티에 도달하면 클라이언트 측 버퍼 채우기를 멈춥니다.
12. [DataSocket 열기] 함수의 연결 ID 출력을 프로퍼티 노드의 참조 입력에 연결합니다.

클라이언트의 버퍼 크기를 지정한 후에, 버퍼 사용 데이터를 읽을 수 있습니다.

서버가 쓴 버퍼된 데이터를 읽기

DataSocket 함수를 사용하여 버퍼 크기를 지정한 후, 버퍼된 데이터를 읽을 수 있습니다.

다음 단계를 따라 서버가 쓴 버퍼된 데이터를 읽습니다.

1. 블록다이어그램에 While 루프를 추가합니다.
2. While 루프에 [DataSocket 읽기] 함수를 추가합니다.
3. 컨트롤이나 상수를 DataSocket 읽기 함수의 타입 입력에 연결합니다. 컨트롤이나 상수의 데이터 타입은 읽으려는 데이터의 데이터 타입과 일치해야 합니다. 기본적인 데이터 타입은 배리언트 데이터 타입입니다. 읽으려고 하는 데이터의 데이

터 타입이 불확실하거나 다른 어플리케이션이 데이터를 생성하는 경우, 배리언트 데이터 타입을 사용합니다. LabVIEW가 다룰 수 있는 데이터로 배리언트 데이터를 변경할 수 있습니다.

4. 클라이언트에 버퍼 크기를 지정할 때 생성한 프로퍼티 노드의 연결 ID 출력을 [DataSocket 읽기] 함수의 연결 입력에 연결합니다.

버퍼된 데이터를 읽은 후에, 버퍼의 크기를 모니터할 수 있습니다.



관련 정보

[DataSocket 함수를 사용하여 버퍼 크기 지정하기](#) (1479 페이지 참조)

[실시간 데이터를 프로그램적으로 공유하기](#) (593 페이지 참조)

[배리언트 데이터를 LabVIEW 데이터로 변환하기](#) (1122 페이지 참조)

[DataSocket 함수를 사용하여 버퍼 크기 지정하기](#) (1479 페이지 참조)

[데이터 연결의 버퍼 크기 모니터하기](#) (1480 페이지 참조)

데이터 연결의 버퍼 크기 모니터하기

버퍼 사용 데이터를 읽은 후에, 버퍼 크기를 모니터할 수 있고 지정한 버퍼의 진단 정보를 요청할 수 있습니다.

다음 단계를 따라 버퍼 크기를 모니터합니다.

1. 블록 다이어그램에 While 루프를 추가합니다.
2. [DataSocket 열기] 함수를 While 루프의 왼쪽에 추가하여 프로그램적으로 데이터 연결을 엽니다.
3. 클라이언트 연결을 위해 버퍼 크기를 지정합니다.
4. While 루프 안에 [DataSocket 읽기] 함수를 추가합니다.
5. [DataSocket 열기] 함수의 연결 id 출력을 [DataSocket 읽기] 함수의 연결 입력 입력에 연결합니다.
6. While 루프 안에서 [DataSocket 읽기] 함수의 오른쪽에 프로퍼티 노드를 추가합니다.
7. [DataSocket 읽기] 함수의 연결 출력 출력을 프로퍼티 노드의 참조 입력에 연결합니다.
8. 프로퍼티 노드의 크기를 재조정하고 터미널을 추가합니다.
9. 프로퍼티 노드의 첫번째 하얀 영역을 클릭하고 버퍼 사용률(바이트) 프로퍼티를 선택합니다.
10. 프로퍼티 노드의 두번째 하얀 영역을 클릭하고 버퍼 사용률(패킷) 프로퍼티를 선택합니다.
11. 각 프로퍼티에 대한 인디케이터를 생성하여 지정된 버퍼 사용의 제약을 모니터합니다.

LabVIEW는 첫번째 버퍼 크기 리미트가 최대 버퍼 바이트 프로퍼티 또는 최대 버퍼 패킷 프로퍼티에 도달하면 클라이언트 측 버퍼 채우기를 멈춥니다.



관련 정보

[서버가 쓴 버퍼된 데이터를 읽기](#) (1479 페이지 참조)

[프로그램적으로 데이터 연결 열기와 닫기](#) (1478 페이지 참조)

[DataSocket 함수를 사용하여 버퍼 크기 지정하기](#) (1479 페이지 참조)

[노드 크기 조정하기](#) (849 페이지 참조)

프로그램적으로 실시간 데이터 읽기

데이터 연결에서 실시간 **데이터**를 읽고 프런트패널에 데이터를 추가하기 전 또는 데이터를 블록다이어그램의 VI 또는 함수에 전달하기 전에 데이터를 수정합니다.

(Real-Time, Windows) National Instruments는 **공유 변수 노드 사용**을 통해 블록다이어그램에서 실시간으로 데이터를 읽도록 권장합니다.

다음 단계를 따라 프로그램적으로 실시간 데이터를 읽습니다.

1. 블록다이어그램에 [DataSocket 읽기] 함수를 추가합니다.
2. [DataSocket 읽기] 함수의 연결 입력 입력에서 pspURL을 지정하거나 **공유 변수 컨트롤**을 연결 입력 입력에 연결합니다.
3. 컨트롤이나 상수를 DataSocket 읽기 함수의 타입 입력에 연결합니다. 컨트롤이나 상수의 데이터 타입은 읽으려는 데이터의 데이터 타입과 일치해야 합니다. 기본적인 데이터 타입은 배리언트 데이터 타입입니다. 읽으려고 하는 데이터의 데이터 타입이 불확실하거나 다른 어플리케이션이 데이터를 생성하고 있다면, 배리언트 데이터 타입을 사용합니다. LabVIEW가 다룰 수 있는 데이터로 **배리언트 데이터를 변환**할 수 있습니다.
4. VI를 실행하거나 연속적으로 VI를 실행하여 실시간 데이터를 업데이트합니다.

dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 원격으로 프런트패널 컨트롤 컨트롤하기

프런트패널 데이터 연결을 사용하여 **공유 변수 엔진**을 통해 원격으로 프런트패널 컨트롤을 제어하면 dstp에 비해 더 신속하며 신뢰성 있는 데이터 전송이 지원될 뿐만 아니라, 더 원활하게 공유 변수 및 **NI Publish-Subscribe Protocol (NI-PSP) 데이터** 아이템과 연결할 수 있습니다.

다음 단계에 따라 Windows에서는 opc, ftp, 또는 http 프로토콜을 사용하고, LabVIEW가 지원되는 모든 플랫폼에서는 dstp 및 file 프로토콜을 사용하여, 다른 VI에서 프런트패널 컨트롤을 제어할 수 있습니다.



노트

dstp 프로토콜을 사용하여 프런트패널 컨트롤을 원격으로 제어하려면 반드시 DataSocket 서버를 설치해야 합니다.

1. dstp 연결을 사용하는 경우, DataSocket 서버를 시작하고 설정합니다. opc, ftp, http, 또는 file 연결을 사용하는 경우, 2 단계로 건너뛩니다.
 1. 시작>> 프로그램>> National Instruments>> DataSocket>> DataSocket Server를 선택하여 DataSocket 서버를 엽니다.
 2. 시작>> 프로그램>> National Instruments>> DataSocket>> DataSocket Server Manager를 선택하여 DataSocket 서버 관리자를 시작합니다.
 3. New Item 버튼을 클릭하여 미리 정의된 데이터 아이템을 생성하고 Allow Multiple Writers 확인란에 확인 표시를 합니다. 여러 쓰기의 연결을 설정하는 것에 대한 추가적인 정보는 DataSocket Server Help를 참조하십시오.
2. 프런트패널의 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티 대화 상자를 디스플레이합니다.
3. 오른쪽 화살표 버튼을 클릭하여 프로퍼티 대화 상자의 데이터 연결 페이지를 탐색합니다.
4. 데이터 연결 선택 풀다운 리스트에서 DataSocket을 선택합니다.
5. Windows에서는 opc, ftp, 또는 http URL을, LabVIEW가 지원되는 모든 플랫폼에서는 file 또는 dstpURL을 경로 텍스트 박스에 입력합니다. 또한 LabVIEW가 지원되는 모든 플랫폼에서는 탐색 풀다운 메뉴에서 DSTP 서버 또는 파일 시스템을 선택하여 1c 단계에서 생성한 사전 지정 데이터 아이템의 URL을 찾을 수 있습니다.
6. 데이터 연결 페이지의 접근 타입 풀다운 메뉴에서 읽기/쓰기를 선택합니다.
7. 확인 버튼을 클릭합니다.
8. VI를 실행합니다.
9. 두 번째 VI(로컬 컴퓨터 또는 원격 컴퓨터)에서, 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티 대화 상자를 디스플레이합니다.
10. 오른쪽 화살표 버튼을 클릭하여 프로퍼티 대화 상자의 데이터 연결 페이지를 탐색합니다.
11. 데이터 연결 선택 리스트에서 DataSocket을 선택합니다.
12. 탐색 풀다운 메뉴에서 DSTP 서버 또는 파일 시스템을 선택하여 단계 5에서 입력한 URL을 탐색합니다.
13. 데이터 연결 페이지의 접근 타입 풀다운 메뉴에서 읽기/쓰기를 선택합니다.
14. 두 VI를 실행하거나 연속 실행합니다. VI가 실행중일 때, DataSocket 서버에 대한 연결이 성공적인 경우, 작은 녹색 인디케이터가 VI의 프런트패널 객체 옆에 나타납니다. 연결이 성공적이지 못한 경우, 작은 붉은색 인디케이터가 나타납니다. LabVIEW가 서버에 연결을 시도하기 전에, 작은 투명한 인디케이터가 나타납니다. 또한, 커서를 인디케이터 위로 이동하는 경우 상태 메시지가 나타납니다. 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보이는 아이템>> DataSocket LED를 선택하여 이 인디케이터를 숨길 수 있습니다. 또한,

DataSocket:LED 보이기 프로퍼티를 사용하여 프로그램적으로 컨트롤과 인디케이터를 숨길 수 있습니다.

첫번째 VI의 프런트패널 컨트롤은 두번째 VI가 전달하는 값을 DataSocket 연결을 통해서 가져옵니다. 사용자가 두번째 VI의 컨트롤 값을 변경할 때, DataSocket 연결은 첫번째 VI의 프런트패널 컨트롤에 새 값을 씁니다. 프런트패널 컨트롤에 쓰기 전용 권한만 있으면, 컨트롤은 VI가 실행될 때 또는 VI가 연속적으로 실행되는 경우 VI가 리셋될 때마다 초기값을 씁니다. 프런트패널 컨트롤에 읽기 전용 또는 읽기/쓰기 권한이 있으면, 컨트롤은 VI가 실행될 때 초기값을 읽습니다.

컨트롤의 dstp, opc, ftp, http, 또는 file 연결 활성화하기

프런트패널 데이터 연결을 사용하여 **공유 변수 엔진**을 통해 원격으로 프런트패널 컨트롤을 제어하면 dstp에 비해 더 신속하며 신뢰성 있는 데이터 전송이 지원될 뿐만 아니라, 더 원활하게 **공유 변수** 및 **NI Publish-Subscribe Protocol (NI-PSP)** 데이터 아이템과 연결할 수 있습니다.

Windows 상에서의 컨트롤에 대한 opc, ftp, http 연결을 활성화하거나, LabVIEW가 지원되는 모든 플랫폼 상에서의 컨트롤에 대한 dstp 또는 file 연결을 활성화하면, DataSocket 서버나 Windows에서는 OPC 서버, FTP 서버, Web 서버를 통해 컨트롤이 실시간 **데이터**를 쓰고 읽을 수 있습니다. 객체의 URL을 쓰거나 저장하지만 데이터를 쓰거나 읽지 않으려는 경우 컨트롤의 연결을 활성화하지 마십시오.

다음 단계에 따라 Windows에서는 opc, ftp, 또는 http 연결에 대한 컨트롤을, LabVIEW가 지원되는 모든 플랫폼에서는 dstp 및 file 연결에 대한 컨트롤을 활성화합니다.

1. dstp 연결을 사용하는 경우, DataSocket 서버를 시작하고 설정합니다. opc, ftp, http, 또는 file 연결을 사용하는 경우, 2 단계로 건너뛵니다. 시작 >> 프로그램 >> National Instruments >> DataSocket >> DataSocket Server를 선택하여 DataSocket 서버를 엽니다.
2. 프런트패널의 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티 대화 상자를 디스플레이합니다.



노트

타입 정의를 위해 프런트패널 데이터 연결을 생성할 수 없습니다.

1. 오른쪽 화살표 버튼을 클릭하여 프로퍼티 대화 상자의 데이터 연결 페이지를 탐색합니다.
2. 데이터 연결 선택 풀다운 리스트에서 DataSocket을 선택합니다.
3. Windows에서는 opc, ftp, 또는 http **URL**을, LabVIEW가 지원되는 모든 플랫폼에서는 file 또는 dstp **URL**을 경로 텍스트 박스에 입력합니다. 또한, LabVIEW가 지원되는 모든 플랫폼에서는 탐색 풀다운 메뉴에서 DSTP 서버 또는 파일 시스템을 선택하여 URL을 탐색할 수 있습니다.
4. 접근 타입 풀다운 메뉴에서 읽기/쓰기를 선택합니다.
5. 확인 버튼을 클릭합니다.

dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 프런트패널 데이터 읽기

프런트패널 데이터 연결을 사용하여 **공유 변수 엔진**을 통해 원격으로 프런트패널 컨트롤을 제어하면 dstp에 비해 더 신속하며 신뢰성 있는 데이터 전송이 지원될 뿐만 아니라, 더 원활하게 **공유 변수** 및 **NI Publish-Subscribe Protocol (NI-PSP)** 데이터 아이템과 연결할 수 있습니다.

다음 단계에 따라 Windows에서는 opc, ftp, 또는 http 프로토콜을 사용하고, LabVIEW가 지원되는 모든 플랫폼에서는 dstp 및 file 프로토콜을 사용하여 프런트패널 **데이터**를 읽습니다.



노트

dstp 프런트패널 데이터를 읽으려면 반드시 DataSocket 서버를 설치해야 합니다.

1. dstp 연결을 사용하는 경우, DataSocket 서버를 시작하고 설정합니다. opc, ftp, http, 또는 file 연결을 사용하는 경우, 2 단계로 건너뛩니다. 시작 >> 프로그램 >> National Instruments >> DataSocket >> DataSocket Server를 선택하여 DataSocket 서버를 엽니다.
2. 프런트패널의 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 프로퍼티 대화 상자를 디스플레이합니다.



노트 **타입 정의**를 위해 프런트패널 데이터 연결을 생성할 수 없습니다.

1. 프로퍼티 대화 상자의 데이터 연결 탭을 클릭합니다.
2. 데이터 연결 선택 풀다운 리스트에서 DataSocket을 선택합니다.
3. Windows에서는 opc, ftp, 또는 http **URL**을, LabVIEW가 지원되는 모든 플랫폼에서는 file 또는 dstpURL을 경로 텍스트 박스에 입력합니다. 또한, LabVIEW가 지원되는 모든 플랫폼에서는 탐색 풀다운 메뉴에서 DSTP 서버 또는 파일 시스템을 선택하여 URL을 탐색할 수 있습니다.
4. 접근 타입 풀다운 메뉴에서 읽기 전용을 선택합니다.
5. 확인 버튼을 클릭합니다.
6. 블록 다이어그램에서 **While 루프**를 생성하여 VI가 연속적으로 실행하도록 설정합니다.
7. VI를 실행합니다. LabVIEW는 DataSocket 서버로부터 데이터를 가져오고 그것을 VI의 프런트패널 객체에 디스플레이합니다. VI가 실행중일 때, DataSocket 서버에 대한 연결이 성공적인 경우, 작은 녹색 인디케이터가 VI의 프런트패널 객체 옆에 나타납니다. 연결이 성공적이지 못한 경우, 작은 붉은색 인디케이터가 나타납니다. LabVIEW가 서버에 연결을 시도하기 전에, 작은 투명한 인디케이터가 나타납니다. 또한, 커서를 인디케이터 위로 이동하는 경우 상태 메시지가 나타납니다. 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 VI가 실행되고 있지 않을 때 바로 가기 메뉴에서 보이는 아이템 >> DataSocket LED를 선택하여 이 인디케이터를 숨길 수 있습니다. 또한, 보이기 프로퍼티를 사용하여 프로그램적으로 컨트롤과 인디케이터를 숨길 수 있습니다.

dstp, opc, ftp, http, 또는 file 프로토콜을 사용하여 프런트패널 데이터 쓰기

프런트패널 데이터 연결을 사용하여 **공유 변수 엔진**을 통해 원격으로 프런트패널 컨트롤을 제어하면 dstp에 비해 더 신속하며 신뢰성 있는 데이터 전송이 지원될 뿐만 아니라, 더 원활하게 **공유 변수** 및 **NI Publish-Subscribe Protocol (NI-PSP)** 데이터 아이템과 연결할 수 있습니다.

다음 단계에 따라 Windows에서는 opc, ftp, 또는 http 프로토콜을 사용하고, LabVIEW가 지원되는 모든 플랫폼에서는 dstp 및 file 프로토콜을 사용하여 프런트패널 **데이터**를 씁니다.



노트

dstp 프로토콜을 사용하여 프런트패널 데이터를 쓰려면 반드시 DataSocket 서버를 설치해야 합니다. 프런트패널에서 데이터를 쓰면 데이터를 잃을 수 있습니다. 가장 최근 데이터뿐만 아니라 모든 프런트패널 데이터를 쓰려면, [DataSocket 쓰기] 함수를 사용하여 **실시간 데이터를 프로그램적으로 씁니다**.

1. dstp 연결을 사용하는 경우, DataSocket 서버를 시작하고 설정합니다. opc, ftp, http, 또는 file 연결을 사용하는 경우, 2 단계로 건너뛩니다. national instruments 디렉토리에서 DataSocket 서버를 시작합니다.
2. 프런트패널의 객체에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 객체의 **프로퍼티** 대화 상자를 디스플레이합니다.
3. 프로퍼티 대화 상자의 데이터 연결 페이지에 있는 데이터 연결 선택 풀다운 메뉴에서 DataSocket을 선택합니다.
4. Windows에서는 opc, ftp, 또는 http **URL**을, LabVIEW가 지원되는 모든 플랫폼에서는 file 또는 dstp URL을 경로 텍스트 박스에 입력합니다. 또한, LabVIEW가 지원되는 모든 플랫폼에서는 탐색 풀다운 메뉴에서 DSTP 서버 또는 파일 시스템을 선택하여 URL을 탐색할 수 있습니다.
5. 접근 타입 풀다운 메뉴에서 읽기/쓰기를 선택합니다.
6. 확인 버튼을 클릭합니다.
7. 블록 다이어그램에서 **While 루프를 생성**하여 VI가 연속적으로 실행하도록 설정합니다.
8. VI를 실행합니다. LabVIEW는 프런트패널 객체의 데이터를 URL에 입력한 서버 대상에 씁니다. VI가 실행중일 때, DataSocket 서버에 대한 연결이 성공적인 경우, 작은 녹색 인디케이터가 VI의 프런트패널 객체 옆에 나타납니다. 연결이 성공적이지 못한 경우, 작은 붉은색 인디케이터가 나타납니다. LabVIEW가 서버에 연결을 시도하기 전에, 작은 투명한 인디케이터가 나타납니다. 또한, 커서를 인디케이터 위로 이동하는 경우 상태 메시지가 나타납니다. 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭한 후 VI가 실행되고 있지 않을 때 바로 가기 메뉴에서 보이는 **아이템 >> DataSocket LED**를 선택하여 이 인디케이터를 숨길 수 있습니다. 또한, DataSocket:LED 보이기 프로퍼티를 사용하여 프로그램적으로 컨트롤과 인디케이터를 숨길 수 있습니다.

웹 서비스

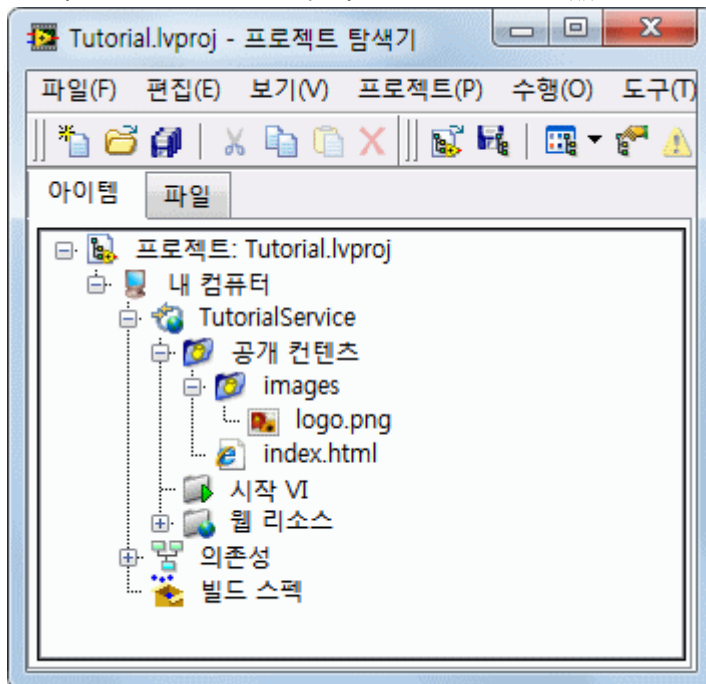
웹 서비스에 정적 콘텐츠 통합하기(Real-Time, Windows)

상위 토픽: [웹 서비스 개발하기](#)

웹 서비스를 빌드할 때, HTML 파일, JavaScript 파일, CSS 파일, 이미지 및 비디오 같은 정적 콘텐츠를 LabVIEW 프로젝트에서 통합시킬 수 있습니다. 예를 들어 웹 서비스를 사용하여 독립 웹 페이지를 공개하거나, 웹 서비스에서 HTTP 메소드 VI와 상호작용하는 사용자 인터페이스를 제공할 수 있습니다.

다음 단계에 따라 웹 서비스에 정적 콘텐츠를 통합합니다.

1. 디스크 상의 정적 콘텐츠를 공개 콘텐츠를 포함한 폴더와 사용자에게 공개하고 싶지 않은 비공개 콘텐츠를 포함하는 폴더라는 두 개의 기본 폴더로 구성합니다.
2. 웹 서비스 파일이 포함된 LabVIEW 프로젝트를 엽니다.
3. 프로젝트 트리에 있는 웹 서비스 프로젝트 아이템(🌐)에서 마우스 오른쪽 버튼을 클릭한 후, 공개 콘텐츠 폴더 추가 또는 비공개 콘텐츠 폴더 추가를 선택합니다.
4. 나타나는 탐색 대화 상자에서, 웹 서비스에 포함시키려는 파일이 있는 디스크 상의 폴더를 선택합니다.
5. LabVIEW는 웹 서비스 프로젝트 아이템 아래에 공개 콘텐츠 또는 비공개 콘텐츠라는 이름의 자동 적용 폴더를 생성합니다. 디스크 상의 해당 폴더에 있는 모든 파일이 프로젝트 트리에 추가한 폴더 아래에 나타납니다. 다음 프로젝트는 공개 콘텐츠 폴더를 포함하는 프로젝트를 보여줍니다. 디스크상의 해당 폴더에는 프로젝트 트리과 동일한 파일 및 서브 폴더가 들어 있습니다.



웹 서비스를 공개한 이후에는, 접근하려는 파일에 대한 URL 매핑을 탐색하여 웹 브라우저와 같은 클라이언트에서 정적 콘텐츠에 접근할 수 있습니다.

공개 파일에 대한 URL 매핑 찾기

LabVIEW는 자동으로 각 공개 정적 파일에 URL 매핑을 할당합니다. 이 매핑은 사용자가 변경할 수 없습니다. 다음 단계를 완료하여 공개 정적 파일에 대한 URL 매핑을 확인합니다:

- 특정 정적 파일 또는 폴더에 대해, 프로젝트 트리에 있는 해당 파일에서 마우스 오른쪽 버튼을 클릭한 후 공개 URL 보이기를 선택합니다.
- 모든 공개 정적 파일 및 폴더에 대한 URL 매핑을 확인하려면, 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택하여 웹 서비스 프로퍼티 대화 상자를 디스플레이합니다. 사이트 맵 페이지에서, 테이블은 각각의 공개 정적 파일 및 폴더에 대한 URL 매핑을 나타내는 행으로 구성됩니다.

기본 HTML 파일 통합하기

프로젝트의 각 공개 콘텐츠 폴더 및 서브폴더에 index.html 파일을 포함시킬 수 있습니다. 클라이언트가 `http://localhost/TutorialService/`와 같은 URL을 사용하여 HTTP 요청을 보내면, 웹 서비스는 index.html 파일을 반환합니다.

다음 단계를 완료하여 특정 공개 콘텐츠 폴더에 기본 index.html 파일을 통합합니다:

1. 디스크에서, 프로젝트의 공개 콘텐츠 폴더 아래에 나타나는 폴더에 index.html이라는 이름의 파일을 추가합니다. 위의 예에서, 최상위 레벨 공개 콘텐츠 폴더와 연관된 디스크 상의 폴더에 해당 파일을 놓습니다. 디스크에 있는 해당 폴더의 변화에 대한 자동 적용을 중지시킨 경우가 아니라면, 파일은 자동으로 프로젝트 트리 아래에 나타납니다.

노트 파일 확장자는 반드시 .html이어야 합니다.

2. 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택합니다.
3. 서비스 셋팅 페이지에서, 퍼블릭 폴더에 기본 index.html 사용 확인란에 확인 표시를 한 후 확인을 클릭합니다.
4. 웹 브라우저에서 index.html 파일이 들어 있는 폴더에 해당하는 URL을 탐색합니다. 해당 HTML 파일이 브라우저에 로드됩니다.

웹 서비스에 비공개 콘텐츠 통합하기

비공개 정적 파일은 웹 클라이언트가 비공개 콘텐츠에 접근할 수 없기 때문에 URL 매핑을 갖지 않습니다. 웹 서비스 소스 파일 안에서 비공개 정적 콘텐츠에 프로그램적으로 접근하려면, [서비스 속성 읽기] VI를 사용하여 실행시 비공개 콘텐츠가 들어있는 폴더의 경로를 반환합니다. 비공개 콘텐츠 폴더에 대한 경로를 사용하여, 특정 파일에 대한 경로를 만들 수 있습니다. 더 자세한 정보는 서비스 속성 읽기 VI 도움말을 참조하십시오.

관련 정보

웹 서비스의 구성요소: 정적 파일

NI 웹 서버에서 웹 서비스 테스트 및 디버깅하기 (Windows)

상위 토픽: 웹 서비스 개발하기

웹 서비스를 타겟에 공개하기 전, 클라이언트가 HTTP 메소드 VI와 정적 콘텐츠를 예상대로 실행하고 통신하는지 테스트합니다. 예상치 않은 동작이나 에러가 발견된 경우, 수정 후 웹 서비스를 다시 테스트합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

웹 서비스 디버깅 작업흐름

웹 서비스의 테스트 및 디버깅은 일반적으로 웹 서비스 파일이 포함된 LabVIEW 프로젝트를 완성하기 위해 필요한 반복 절차입니다. 다음 단계에 따라 NI 웹 서버에서 웹 서비스를 테스트하고 디버깅합니다.



노트 이 토픽에서는 NI 웹 서버에서 웹 서비스를 테스트하고 디버깅합니다. 또한 [어플리케이션 웹 서버에서 웹 서비스 테스트 및 디버깅하기](#)를 참조하여 어플리케이션 웹 서버에서 웹 서비스를 테스트할 수도 있습니다.

1. 웹 서비스가 독립 어플리케이션의 일부인 경우, 프로젝트에서 해당 어플리케이션에 대한 시작 VI를 열고 실행합니다.
2. 프로젝트 트리에 있는 웹 서비스 프로젝트 아이템 에서 마우스 오른쪽 버튼을 클릭한 후, 시작을 선택합니다. LabVIEW는 NI 웹 서버에 웹 서비스를 놓으며, 여기서 웹 서비스는 타겟에 공개된 것과 같이 동작합니다.
3. 사용자의 컴퓨터에서 NI 웹 서버를 처음 사용하는 경우, 다음 단계에 따라 NI 웹 서버를 활성화하고 설정합니다.
 1. 웹 서비스 시작 대화 상자에서 NI 웹 서버 설정을 클릭하여 NI Web Server Configuration을 시작합니다.
 2. 테스트 및 디버깅을 위해 단순 로컬 접근의 미리 설정된 값을 선택한 후, 다음을 클릭합니다.
 3. 인증 페이지에서 서버 관리를 위한 관리자 사용자로 로그인을 선택한 후 암호를 생성합니다. 다음을 클릭합니다.
 4. 마침을 클릭하여 이 설정을 적용합니다.
 5. LabVIEW 프로젝트로 돌아가서 2 단계를 반복합니다.
4. 프로젝트에서 HTTP 메소드 VI를 엽니다. 프로브나 브레이크포인트, 실행 하이라이트와 같은 표준 LabVIEW 디버깅 기법을 사용하여 클라이언트가 호출할 때의 동작을 관찰합니다.
5. VI에 맵핑된 URL을 사용하여 웹 브라우저와 같은 클라이언트에서 HTTP 메소드 VI를 실행합니다. 프로젝트에서 직접 적절한 URL을 찾아서 복사할 수 있습니다. 다음 단계에 따라 HTTP 메소드 VI에 접근합니다.

1. HTTP 메소드 VI에서 마우스 오른쪽 버튼을 클릭한 후, 메소드 URL 보이기 를 선택하여 HTTP 메소드 URL 대화 상자를 디스플레이합니다.
2. 사용 가능한 서버 폴다운 다운 메뉴에서 NI 웹 서버가 포함된 항목을 선택한 후, URL 복사를 클릭합니다.
3. 대화 상자를 닫습니다.

노트 HTTP 메소드 URL 대화 상자가 열려 있는 경우, 웹 서비스는 클라이언트의 요청에 응답하지 않습니다.

4. URL을 웹 브라우저에 붙여 넣고, {value} 변수를 숫자형 값(예: http://127.0.0.1:80/TutorialService/Add?b=2&a=3)으로 대체해야 합니다.

노트 NI 웹 서버는 HTTP/HTTPS 프로토콜에 따라 80/443을 기본 포트로 사용합니다. 그러나 다른 어플리케이션이 해당 포트를 점유하면 포트 번호가 변경될 수 있습니다. NI Web Server Configuration에서 포트 번호를 사용자 정의할 수 있습니다. 인증, 포트 및 원격 연결과 같은 서버 설정에 더 자세한 내용은 NI 웹 서버 매뉴얼의 NI 웹 서버 설정하기를 참조하십시오.

5. URL을 탐색합니다.
6. 에러가 발견되거나 테스트가 완료되면 프로젝트 트리에 있는 웹 서비스 프로젝트 항목에서 마우스 오른쪽 버튼을 클릭한 후 정지를 선택합니다. LabVIEW 독립 어플리케이션에 대한 시작 VI가 실행되고 있는 경우, 정지할 필요가 없습니다.

노트 디버깅하는 동안 실행되도록 예약되었기 때문에 디버깅 세션을 정지할 때까지 HTTP 메소드 VI를 편집할 수 없습니다.

7. VI를 편집하여 발견한 문제를 모두 수정합니다.
8. (옵션) 단계 2-7을 반복하여 웹 서비스 파일에 작성한 변경사항을 검증합니다.

관련 정보

NI 웹 서버 이해하기

웹 서비스를 위해 NI 웹 서버 설정하기 (Windows)

상위 토픽: 웹 서비스 호스팅하기

NI 웹 서버에서 웹 서비스를 호스팅하려면, 웹 서비스를 실행하거나 공개하기 전에 먼저 웹 서버를 설정해야 합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 따라 웹 서비스를 호스팅할 수 있도록 [NI 웹 서버](#)를 활성화하고 설정합니다. 어플리케이션 웹 서버를 사용하여 웹 서비스를 호스팅하는 경우, 자세한 설명은 [웹 서비스를 위해 어플리케이션 웹 서버 설정하기](#)를 참조하십시오.

1. NI Web Server Configuration을 시작합니다.

1. 프로젝트 탐색기 윈도우에서 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 웹 서버 페이지에서 NI 웹 서버 설정을 클릭합니다.

노트 웹 서비스를 호스팅하기 위해 NI 웹 서버를 처음 사용하는 경우 LabVIEW는 NI Web Server Configuration을 시작합니다.

2. NI Web Server Configuration을 처음 시작하는 경우 NI Web Server Guided Setup이 나타납니다.
3. 필요에 따라 미리 설정된 값을 선택하거나 Help me choose a preset을 선택합니다. Next를 클릭합니다.
4. 입력요청에 따라 설정 옵션 중에서 선택합니다. 마침을 클릭합니다.
5. 설정이 끝나면 NI 웹 서버가 실행되기 시작합니다.
6. 컨트롤 페이지에서 서버 상태를 확인합니다. 이 페이지에서 NI 웹 서버를 다시 시작, 중지, 활성화 또는 비활성화 할 수 있습니다.

인증, 포트 및 원격 연결과 같은 서버 설정에 대한 더 자세한 정보는 NI 웹 서버 설정하기를 참조하십시오.

NI 웹 서버 사용 시 웹 서비스 보안 설정하기 (Windows)

HTTPS 및 권한 기반 인증을 사용하여 웹 클라이언트와 LabVIEW 웹 서비스 간의 안전한 통신을 설정할 수 있습니다.



노트

이 토픽에서는 NI 웹 서버의 보안 셋팅을 설정합니다. 어플리케이션 웹 서버를 사용하여 웹 서비스를 호스팅하는 경우, 자세한 설명은 [어플리케이션 웹 서버 사용 시 웹 서비스 보안 설정하기](#)를 참조하십시오.

다음 절차에 따라 여러 다양한 웹 서비스 보안 방법을 설정할 수 있습니다:

- [NI 웹 서버에서 HTTPS 활성화하기](#)
- [사용자 역할 및 권한 할당하기](#)

NI 웹 서버에서 HTTPS 활성화하기

웹 서비스용 NI 웹 서버에 SSL (Secure Sockets Layer) 암호화를 활성화할 수 있습니다. SSL 암호화를 사용하여 클라이언트와 웹 서버 간의 데이터 전송 시 안전하고 암호화된 연결을 생성할 수 있습니다.

다음 단계에 따라 HTTPS 연결을 활성화합니다.

1. 다음 단계에 따라 NI 웹 서버 설정을 시작합니다.

1. 프로젝트 탐색기 윈도우에서 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 웹 서버 페이지에서 NI 웹 서버 설정을 클릭합니다.
2. NI Web Server Configuration의 HTTPS 탭에서, HTTPS settings 중 하나를 선택합니다. 각 HTTPS 셋팅에 대한 자세한 정보는 NI 웹 서버 매뉴얼의 HTTPS 셋팅 선택하기를 참조하십시오.
3. 설명에 따라 자체 서명한 인증서를 생성하거나 인증 기관의 인증서를 설치합니다.
4. 서버 HTTPS 포트 번호를 설정합니다.
5. 적용 및 다시 시작을 클릭합니다. NI 웹 서버는 재부팅 후 HTTPS를 사용하여 클라이언트와 통신합니다.



노트 LabVIEW는 NI 웹 서버의 HTTP 연결 및 포트 셋팅 변경에 따라 HTTP 메소드 URL을 자동으로 업데이트합니다.

사용자 역할 및 권한 할당하기

사용자 역할을 정의하고 각 역할에 다른 권한을 할당하여 각 HTTP 메소드 VI 및 NI 웹 서버 어플리케이션에 대한 각 클라이언트의 접근을 제어할 수 있습니다.

NI 웹 서버 어플리케이션에 대한 사용자 역할 및 권한 정의하기

NI Web Server Configuration에서 NI 웹 서버 어플리케이션에 대해 사용자 역할을 정의하고 권한을 할당하는 방법에 대한 정보는 NI 웹 서버 매뉴얼의 사용자 역할 및 권한 정의하기를 참조하십시오.

LabVIEW 웹 서비스에 사용자 역할 및 권한 할당하기

다음 단계를 완료하여 LabVIEW에서 HTTP 메소드 VI가 유효한 권한이 있는 웹 클라이언트가 보낸 요청만 수락하도록 설정합니다.

1. 프로젝트 트리에 있는 웹 서비스 프로젝트 아이템(🌐)에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택하여, 웹 서비스 프로퍼티 대화 상자를 디스플레이합니다.
2. NI 웹 서버 권한 페이지에서 권한 열 아래에 사용자 정의된 새 권한의 이름을 입력하여 권한을 생성합니다.
3. 역할 열 아래에서 사용자 역할을 선택하거나 NI Web Server Configuration에서 생성한 사용자 정의 역할을 입력하여 지정한 사용자 역할에 권한을 할당합니다. 예를 들어, 데이터 쓰기 권한을 관리자 사용자 역할에만 할당할 수 있습니다.
4. HTTP 메소드 VI 셋팅 페이지의 웹 서비스 VI 테이블에서 VI를 선택한 후 웹 서비스 VI 프로퍼티에 있는 NI 웹 서버 탭을 클릭합니다.
5. 필요한 권한 풀다운 메뉴에서 권한을 선택합니다.
6. 확인을 클릭하여 변경 사항을 저장합니다.

권한을 사용하여 HTTP 메소드 VI에 접근하기

웹 클라이언트는 클라이언트에서 권한으로 보호된 VI에 HTTP 요청을 보낼 때 반드시 올바른 사용자 이름과 암호를 제공해야 합니다. 사용자를 역할별로 그룹화하려면 NI 웹 서버 매뉴얼의 인증 셋팅 선택하기를 참조하여 NI Web Server Configuration의 Authentication 탭에서 셋팅을 설정합니다.

패키지를 통해 웹 서비스 공개하기 (Windows)

상위 토픽: [웹 서비스 공개하기](#)

NI 패키지.nipkg를 통해 웹 서비스를 NI 웹 서버에 공개할 수 있습니다. 웹 서비스는 패키지 설치/제거가 완료된 후에 실행을 시작/중지합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 완료하여 NI 패키지를 통해 웹 서비스를 공개합니다.

1. 패키지에 웹 서비스를 포함시킵니다.
 1. 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기»패키지를 선택하여 설치 패키지 프로퍼티 대화 상자를 시작합니다.
 2. 웹 서비스 페이지에서 포함시키려는 웹 서비스의 확인란에 확인 표시를 합니다.
 3. (옵션) 패키지 설치 프로그램 페이지에서 패키지 설치 프로그램 생성 확인란에 확인 표시를 하면 배포를 패키지 설치 프로그램 파일 .exe로 빌드할 수 있습니다.
 4. (옵션) 피드 페이지에서, 패키지를 로컬 피드에 추가할지 아니면 SystemLink 서버의 피드에 공개할지 선택합니다.
2. 웹 서비스를 포함한 패키지를 빌드합니다.
 1. 빌드를 클릭하면 빌드 상태 대화 상자가 나타납니다.
 2. LabVIEW가 패키지 빌드를 완료하면, 탐색을 클릭하여 LabVIEW가 패키지 파일을 저장할 대상 디렉토리를 엽니다.
3. 타겟 컴퓨터에 패키지를 설치하여 타겟 컴퓨터에서 웹 서비스가 실행을 시작하도록 합니다.
 1. 대상 디렉토리에서 패키지 .nipkg를 더블 클릭하여 NI 패키지 관리자를 통해 패키지를 설치합니다. 설치가 완료되면 웹 서비스가 로컬 컴퓨터의 NI 웹 서버에 공개됩니다.

노트 웹 서비스를 NI 웹 서버에 게시하려면 관리자 권한이 필요합니다.

2. LabVIEW 프로젝트를 닫습니다. 공개된 웹 서비스는 LabVIEW 런타임에서 실행되며 LabVIEW 개발 시스템에 의존하지 않습니다.

어떤 경우에 패키지 설치 프로그램을 생성합니까?

패키지 설치 프로그램은 타겟이 네트워크 접근이나 NI 소프트웨어 없이도 패키지를 설치할 수 있도록 모든 패키지 의존성을 포함합니다. 인터넷 연결이 없거나 패키지를 설치하여 웹 서비스를 실행하는데 필요한 소프트웨어(NI 패키지 관리자와 LabVIEW 런타임 등)가 없는 컴퓨터에 웹 서비스를 공개하려면 패키지 설치 프로그램을 생성해야 합니다.

어떤 경우에 피드에 웹 서비스를 추가합니까?

배포 타겟에서 피드를 구독하면 패키지를 검색 및 설치할 수 있으며 네트워크 접근을 통해 NI 패키지 관리자 또는 NI SystemLink에서 업데이트 알림을 받을 수 있습니다. NI는 서로 의존성이 있는 여러 패키지를 배포할 때 피드를 사용하도록 권장합니다.

관련 정보

피드에서 패키지 설치하기

서버의 클라이언트에 어플리케이션 배포하기

독립 어플리케이션을 통해 웹 서비스 공개하기 (Real-Time, Windows)

상위 토픽: [웹 서비스 공개하기](#)

독립 어플리케이션을 통해 웹 서비스를 NI 웹 서비스나 어플리케이션 웹 서버에 공개할 수 있습니다. 어플리케이션이 실행을 시작/중지하면 웹 서비스가 실행을 시작/중지합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 완료하여 독립 어플리케이션을 통해 웹 서비스를 공개합니다.

1. 어플리케이션에 웹 서비스를 포함시킵니다.
 1. (Windows) 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기»어플리케이션을 선택하여 어플리케이션 프로퍼티 대화 상자를 시작합니다. (Real-Time) 프로젝트 탐색기의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기»Real-Time 어플리케이션을 선택하여 Real-Time 어플리케이션 프로퍼티 대화 상자를 시작합니다.
 2. 웹 서비스 페이지에서 포함시키려는 웹 서비스의 확인란에 확인 표시를 합니다.
 3. 웹 서비스를 호스팅할 웹 서버를 선택합니다. 각 웹 서버에 대한 자세한 정보는 [웹 서비스 호스팅하기](#)를 참조하십시오.

노트 (Real-Time) NI 웹 서버는 RT 타겟을 지원하지 않기 때문에 RT 타겟에서 웹 서비스를 호스팅하려면 어플리케이션 웹 서버만 사용할 수 있습니다.

4. (옵션) c 단계에서 어플리케이션 웹 서버를 선택한 경우, 웹 서비스에 대한 요청을 승인하는 데 사용할 포트를 설정합니다. 고유한 포트를 사용하려면 동시에 실행되는 어플리케이션을 각각 설정해야 합니다. 클라이언트가 HTTPS 프로토콜을 사용하여 웹 서비스를 호출하는 것을 허용하려면, SSL 활성화 확인란에 확인 표시를 하고 다른 아이템을 설정합니다.
2. 웹 서비스를 포함한 어플리케이션을 만듭니다.
 1. 빌드를 클릭하면 빌드 상태 대화 상자가 나타납니다.
 2. LabVIEW가 어플리케이션 빌드를 완료하면, 탐색을 클릭하여 LabVIEW가 어플리케이션 파일을 저장할 대상 디렉토리를 엽니다.
3. 타겟 디바이스에서 어플리케이션을 실행하여 디바이스에서 웹 서비스가 실행되도록 합니다.
 1. (Windows) 대상 디렉토리에 있는 실행 파일을 더블 클릭하여 실행합니다. (Real-Time) 프로젝트 탐색기의 빌드 스펙 아래에 빌드한 어플리케이션에서 마우스 오른쪽 버튼을 클릭한 후, Run as startup을 클릭합니다. 어플리케이션은 타겟 디바이스의 웹 서버에 웹 서비스를 게시합니다.

노트 웹 서비스를 NI 웹 서버에 게시하려면 관리자 권한이 필요합니다.

2. LabVIEW 프로젝트를 닫습니다. 공개된 웹 서비스는 LabVIEW 런타임에서 실행되며 LabVIEW 개발 시스템에 의존하지 않습니다.

설치 프로그램을 통해 웹 서비스 공개하기 (Windows)

상위 토픽: 웹 서비스 공개하기

Windows 설치 프로그램을 통해 웹 서비스를 어플리케이션 웹 서버에 공개할 수 있습니다. 웹 서비스는 설치/제거 프로세스가 완료된 후에 실행을 시작/중지합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 완료하여 Windows 설치 프로그램을 통해 웹 서비스를 공개합니다.

1. 설치 프로그램에 웹 서비스를 포함시킵니다.
 1. 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기 » 설치 프로그램을 선택하여 설치 프로그램 프로퍼티 대화 상자를 시작합니다.
 2. 웹 서비스 페이지에서 포함시키려는 웹 서비스의 확인란에 확인 표시를 합니다.
2. 웹 서비스를 포함한 설치 프로그램을 만듭니다.
 1. 빌드를 클릭하면 빌드 상태 대화 상자가 나타납니다.

2. LabVIEW가 설치 프로그램 빌드를 완료하면, 탐색을 클릭하여 LabVIEW가 설치 프로그램 파일을 저장할 대상 디렉토리를 엽니다.
3. 타겟 컴퓨터에서 설치 프로그램을 실행하여 타겟 컴퓨터에서 웹 서비스가 실행되도록 합니다.
 1. 대상 디렉토리에 있는 설치 프로그램 .exe를 더블 클릭하여 어플리케이션을 설치합니다. 설치된 어플리케이션은 로컬 컴퓨터의 웹 서버에 웹 서비스를 공개합니다.
 2. LabVIEW 프로젝트를 닫습니다. 공개된 웹 서비스는 LabVIEW 런타임에서 실행되며 LabVIEW 개발 시스템에 의존하지 않습니다.

독립 웹 서비스 공개 및 비공개하기(Real-Time, Windows)

상위 토픽: 웹 서비스 공개하기

LabVIEW 프로젝트에서 직접 독립 웹 서비스를 어플리케이션 웹 서버에 공개하거나 비공개할 수 있습니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

웹 서비스 개발 컴퓨터 또는 접속된 RT 타겟에 독립 웹 서비스를 공개하려면 프로젝트 탐색기 윈도우의 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 어플리케이션 웹 서버>>공개를 선택합니다. 공개된 웹 서비스는 LabVIEW 런타임에서 실행되며 LabVIEW 개발 시스템에 의존하지 않습니다.

독립 웹 서비스를 비공개하려면 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 어플리케이션 웹 서버>>비공개를 선택합니다.

어플리케이션 웹 서버에서 웹 서비스 테스트 및 디버깅하기 (Real-Time, Windows)

웹 서비스를 타겟에 공개하기 전, 클라이언트가 HTTP 메소드 VI와 정적 콘텐츠를 예상대로 실행하고 통신하는지 테스트합니다. 예상치 않은 동작이나 에러가 발견된 경우, 수정 후 웹 서비스를 다시 테스트합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

웹 서비스 디버깅 작업흐름

웹 서비스의 테스트 및 디버깅은 일반적으로 웹 서비스 파일이 포함된 LabVIEW 프로젝트를 완성하기 위해 필요한 반복 절차입니다. 다음 단계에 따라 어플리케이션 웹 서버에서 웹 서비스를 테스트하고 디버깅합니다.



노트 이 토픽에서는 어플리케이션 웹 서버에서 웹 서비스를 테스트하고 디버깅합니다. 또한 [NI 웹 서버에서 웹 서비스 테스트 및 디버깅하기](#)를 참조하여 NI 웹 서버에서 웹 서비스를 테스트할 수도 있습니다.

1. 웹 서비스가 독립 어플리케이션의 일부인 경우, 프로젝트에서 해당 어플리케이션에 대한 시작 VI를 열고 실행합니다.
2. (Windows) 프로젝트 트리에 있는 내 컴퓨터 아래의 웹 서비스 프로젝트 항목에서 마우스 오른쪽 버튼을 클릭한 후, 어플리케이션 웹 서버>>시작(디버그 서버)를 선택합니다. (Real-Time) 프로젝트 트리에 있는 RT 타겟 아래의 웹 서비스 프로젝트 항목에서 마우스 오른쪽 버튼을 클릭한 후, 시작(디버그 서버)를 선택합니다. LabVIEW는 웹 서비스를 디버깅 서버에 놓고, 디버깅 서버에서 웹 서비스는 타겟에 공개된 것과 같이 동작합니다.
3. 프로젝트에서 HTTP 메소드 VI를 엽니다. 프로브나 브레이크포인트, 실행 하이라이트와 같은 표준 LabVIEW 디버깅 기법을 사용하여 클라이언트가 호출할 때의 동작을 관찰합니다.
4. VI에 맵핑된 URL을 사용하여 웹 브라우저와 같은 클라이언트에서 HTTP 메소드 VI를 실행합니다. 프로젝트에서 직접 적절한 URL을 찾아서 복사할 수 있습니다.
5. 에러가 발견되거나 웹 서비스의 테스트가 완료되면, (Windows) 프로젝트 트리에 있는 웹 서비스 프로젝트 항목에서 마우스 오른쪽 버튼을 클릭한 후 어플리케이션 웹 서버>>정지(디버그 서버)를 선택합니다. (Real-Time) 웹 서비스 프로젝트 항목에서 마우스 오른쪽 버튼을 클릭한 후, 정지(디버그 서버)를 선택합니다. LabVIEW 독립 어플리케이션에 대한 시작 VI가 실행되고 있는 경우, 정지할 필요가 없습니다.

노트 디버깅하는 동안 HTTP 메소드가 실행되도록 예약되었기 때문에 디버깅 세션을 정지할 때까지 HTTP 메소드 VI를 편집할 수 없습니다.

6. VI를 편집하여 발견한 문제를 모두 수정합니다.
7. (옵션) 단계 2-6을 반복하여 웹 서비스 파일에 작성한 변경사항을 검증합니다.

디버깅 모드에서 HTTP 메소드 VI 접근하기

다음 단계를 완료하여 디버깅 세션 동안 HTTP 메소드 VI의 URL을 확인합니다:

1. HTTP 메소드 VI에서 마우스 오른쪽 버튼을 클릭한 후, 메소드 URL 보이기를 선택하여 HTTP 메소드 URL 대화 상자를 디스플레이합니다.
2. 사용 가능한 서버 폴다운 다운 메뉴에서 로컬 디버깅이 포함된 항목을 선택한 후, URL 복사를 클릭합니다. 웹 서비스가 디버깅 환경에서 실행되므로, 이미 타겟에 공개한 경우 다른 서버와 포트로 접근하여야 합니다.

노트 HTTP 메소드 URL 대화 상자가 열려 있는 경우, 웹 서비스는 클라이언트의 요청에 응답하지 않습니다.

- URL을 표준 웹 브라우저에 붙여 넣고, 다음과 같이 {value} 변수를 숫자형 값으로 대체해야 합니다: `http://127.0.0.1:8001/TutorialService/Add?b=2&a=3`.

관련 정보

웹 서비스 개발하기

서버 디버깅하기

웹 서비스용 어플리케이션 웹 서버 설정하기

어플리케이션 웹 서버에서 웹 서비스를 호스팅하려면, 웹 서비스를 실행하거나 공개하기 전에 먼저 웹 서버를 설정해야 합니다.



노트

이 토픽의 개념에 대한 더 자세한 정보는 [웹 서비스 개요](#) 및 [길라잡이](#)를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

다음 단계를 완료하여 LabVIEW 독립 어플리케이션의 일부가 아닌 웹 서비스를 호스트할 [어플리케이션 웹 서버](#)를 활성화하고 설정합니다. NI 웹 서버를 사용하여 웹 서비스를 호스팅하는 경우, 자세한 설명은 [웹 서비스를 위해 NI 웹 서버 설정하기](#)를 참조하십시오.



노트 독립 어플리케이션에 포함되어 웹 서비스를 호스트하는 웹 서버는 자동으로 활성화됩니다.

- 웹 브라우저를 엽니다.
- 브라우저에서 어플리케이션 웹 서버를 활성화하려는 타겟의 URL을 입력합니다. URL은 다음의 구문을 따라야 합니다.
 - 로컬 호스트 시스템에서 어플리케이션 웹 서버를 설정하려면 다음 URL을 입력하십시오. `http://localhost:3582`.
 - 원격 타겟 시스템에서 어플리케이션 웹 서버를 설정하려면, LabVIEW 프로젝트에 있는 타겟에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택하여 웹 서버 페이지를 탐색합니다. 웹 어플리케이션 서버 설정 버튼을 클릭합니다. 버튼이 비활성화되어 있는 경우, 타겟에 어플리케이션 웹 서버가 설치되어 있는지 확인합니다.

팁 또한 웹 브라우저에 다음 URL을 입력할 수 있습니다. `http://[원격 디바이스 IP 주소]`. 여기서 [원격 디바이스 IP 주소]는 원격 디바이스의 IP 주소입니다.

노트 NI 웹 기반 설정 및 모니터링에는 Microsoft Silverlight가 필요하므로 Microsoft Internet Explorer를 사용하여 URL을 찾아야 할 수 있습니다. 이 경우 Microsoft Silverlight와의 호환성이 향상됩니다.

- 로그인을 클릭한 후 사용자 이름과 암호를 입력하여 해당 시스템의 NI 웹 기반 설정 및 모니터링에 접근합니다. 시스템에서 NI 웹 기반 설정 및 모니터링에 대한 보안 설정을 아직 설정하지 않은 경우, 사용자 이름은 Admin을 입력하고 암호는 공백으로 두어 기본 인증을 사용하여 접근할 수 있습니다.
- 웹 서버 설정 버튼을 클릭하여 웹 서버 설정 페이지를 디스플레이합니다.
- 어플리케이션 웹 서버의 활성화와 설정에 대한 정보를 보려면 도움말을 클릭합니다.

어플리케이션 웹 서버는 독립적인 시스템 서비스로 실행됩니다. 호스트 시스템에서 LabVIEW를 실행하지 않고, 어플리케이션 웹 서버를 활성화하고 실행할 수 있습니다.

관련 정보

웹 서비스 호스팅하기

NI 웹 기반의 설정 및 모니터링

어플리케이션 웹 서버 사용 시 웹 서비스 보안 설정하기 (Real-Time, Windows)

이전에는 SSL (Secure Sockets Layer)이라고 알려진 **TLS (Transport Layer Security)** 암호화, 사용자 및 그룹 권한, 그리고 API 키를 통해 웹 클라이언트와 LabVIEW 웹 서비스 사이에서 안전하게 통신할 수 있습니다.

다음 절차에 따라 여러 다양한 웹 서비스 보안 방법을 설정할 수 있습니다:

- 웹 서비스 파일에 권한 할당하기
- API 키 보안 설정하기



노트

- 이 토픽에서는 어플리케이션 웹 서버의 보안 설정을 설정합니다. NI 웹 서버를 사용하여 웹 서비스를 호스팅하는 경우, 자세한 설명은 **NI 웹 서버 사용 시 웹 서비스 보안 설정하기**를 참조하십시오.
- 추가적인 웹 서비스 보안 방법 설정에 대한 정보는 **NI 웹 기반 설정 및 모니터링**의 보안 설정 페이지에 있는 도움말을 클릭하십시오.

웹 서비스 파일에 권한 할당하기

다음 타입의 파일에 대해 특정 권한을 요청할 수 있습니다:

- HTTP 메소드 VI—개별 VI에 적용되는 권한을 설정합니다.
- 공개 정적 파일—웹 서비스에 있는 모든 공개 정적 파일에 적용되는 권한을 설정합니다.

다음 단계를 완료하여 웹 서비스가 유효한 권한이 있는 웹 클라이언트가 보낸 HTTP 요청만 수락하도록 설정합니다.

1. 프로젝트 트리에 있는 웹 서비스 프로젝트 아이템(🌐)에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택하여, 웹 서비스 프로퍼티 대화 상자를 디스플레이합니다.
2. 설정하려는 파일의 타입에 적용되는 페이지를 탐색합니다:
 - HTTP 메소드 VI의 경우, HTTP 메소드 VI 셋팅 페이지를 탐색하여 웹 서비스 VI 테이블에서 VI를 선택한 후, 웹 서비스 VI 프로퍼티에 있는 어플리케이션 웹 서버 탭을 클릭합니다.
 - 공개 정적 파일의 경우, 서비스 셋팅 페이지와 정적 문서 섹션을 탐색합니다.
3. 필요한 권한 테이블에서, NI 웹 기반 설정 및 모니터링의 보안 설정 페이지에서 이전에 생성한 권한의 이름을 추가합니다.

권한이 있는 웹 서비스 파일에 접근하기

클라이언트에서 권한으로 보호된 파일에 HTTP 요청을 제출할 때, 웹 클라이언트는 반드시 올바른 사용자 이름과 암호를 제공해야 합니다. 사용자 이름과 암호는 반드시 HTTP 메소드 VI 또는 공개 정적 파일에 추가된 것과 동일한 권한이 부여된 NI 웹 기반 설정 및 모니터링 유틸리티의 사용자 이름과 암호에 상응되는 것이어야 합니다.

[핸들 열기] VI를 사용하여 LabVIEW 에서 만들어진 웹 클라이언트의 HTTP 요청에 대한 사용자 이름과 암호를 제공합니다.

API 키 보안 설정하기

또한 API 키를 사용하여 HTTP 메소드 VI에 **HTTP 요청**을 보낼 수 있는 클라이언트를 제한할 수 있습니다. API 키는 임의의 문자로 구성된 문자열로, 접근 ID와 비밀 ID의 두 부분으로 구성되어 있습니다. 접근 ID는 사용자 이름과 유사하며, 비밀 ID는 허용된 클라이언트만을 위해 설정된 암호와 유사합니다.

어플리케이션 웹 서버, 임베디드 어플리케이션용 웹 서버 및 웹 서비스 디버깅 서버를 포함하는 모든 웹 서버에서 실행되는 모든 웹 서비스에 적용되는 하나의 API 키를 설정할 수 있습니다. API 키를 생성하려면 먼저 접근 ID와 비밀 ID를 웹 서버에서 설정해야 합니다. 그 후 보안을 적용하려는 모든 URL 맵핑에 각각 API 보안 키를 활성화해야 합니다.

웹 서버에서 API 키 보안 설정하기

어플리케이션 웹 서버에서 접근 ID 및 비밀 ID를 만드려면, NI 웹 기반 설정 및 모니터링 유틸리티를 시작하고 웹 서버 설정 페이지를 탐색합니다. 웹 서비스 API 키 탭을 사용하여 API 키를 생성, 적용 및 리셋합니다.

API 키 보안을 사용하여 URL 맵핑 보안하기

다음 단계에 따라 웹 서비스에서 HTTP 메소드 VI에 대한 API 키 보안을 활성화합니다:

1. 프로젝트 트리에 있는 웹 서비스 프로젝트 아이템(🌐)에서 마우스 오른쪽 버튼을 클릭한 후 프로퍼티를 선택하여, 웹 서비스 프로퍼티 대화 상자를 디스플레이합니다.
2. HTTP 메소드 VI 셋팅 페이지를 탐색하여 웹 서비스 VI 테이블에서 VI를 선택한 후, 웹 서비스 VI 프로퍼티에 있는 어플리케이션 웹 서버 탭을 클릭합니다.
3. API 키 필요 확인란에 확인 표시를 합니다.

API 키 보안을 사용하는 URL 맵핑에 접근하기

HTTP 메소드 VI에 대한 API 키를 만들고 활성화하면, VI와 상응하는 웹 클라이언트에서 오는 HTTP 요청에는 올바른 API 키가 포함되어야 합니다. HTTP 요청을 생성할 때, [API 키 설정] VI를 사용해서 LabVIEW 에서 만들어진 웹 클라이언트 API 키를 제공하도록 설정할 수 있습니다.

관련 정보

웹 서비스 호스트하기

웹 서버의 루트 디렉토리 설정하기

내장된 LabVIEW **웹 서버**를 사용하는 경우, 웹 서버가 문서 요청에 따라 파일을 로드하는 위치에 루트 디렉토리가 있습니다. 웹 서버의 기본 루트 디렉토리는 labview\www입니다.

다음 단계에 따라 웹 서버의 루트 디렉토리를 설정합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 옵션 대화 상자의 항목 리스트에서 웹 서버 페이지를 선택합니다.
3. 리모트 패널 서버 섹션의 리모트 패널 서버 활성화 확인란에 확인 표시를 합니다.
4. 웹 서버가 사용하도록 만드려는 루트 디렉토리의 경로를 루트 디렉토리 텍스트 박스에 입력합니다. 또한, 탐색 버튼을 클릭하여 디렉토리를 탐색하고 선택할 수 있습니다.
5. 확인 버튼을 클릭하여 현재의 설정을 저장하고 대화 상자를 닫습니다.



노트

사용자가 파일 이름을 지정하지 않은 채로 웹 서버의 URL을 입력하면, 에러 페이지나 디렉토리 리스트가 나타날 수 있습니다. 이 대신에 자동으로 로드되는 index.htm 파일을 웹 서버 루트 디렉토리에 생성할 수 있습니다.

길라잡이

길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 (Windows)

다음 단계에 따라 LabVIEW 웹 서비스를 생성하고 공개하여, LabVIEW 웹 서비스와 웹 클라이언트 (이 경우는 웹 브라우저) 사이에서 데이터를 교환합니다.



노트

LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

이 길라잡이는 다음 섹션으로 구성되어 있습니다.

1. 웹 서비스 생성하기
2. 웹 서비스 테스트 및 디버깅하기
3. 클라이언트가 웹 서비스에 접근하는 방법 설정하기
4. NI 웹 서버에 웹 서비스 공개하기
5. 클라이언트로 웹 서비스 접근하기

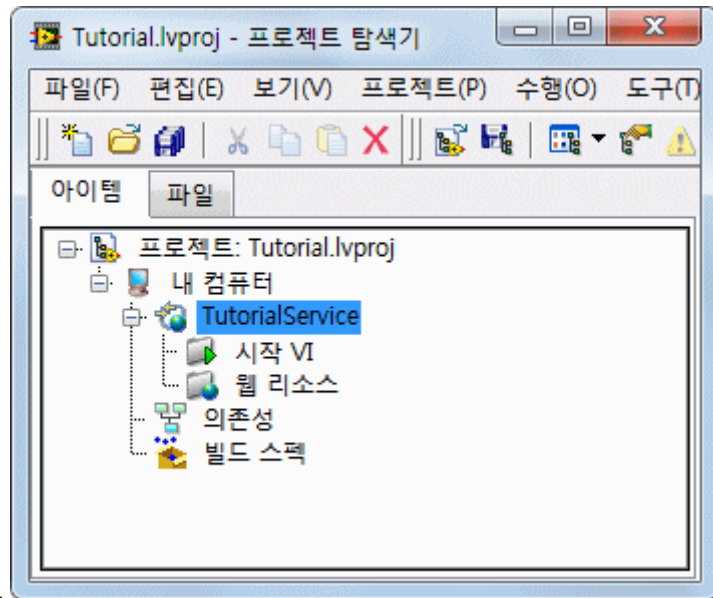
웹 서비스 생성하기

LabVIEW 프로젝트에서 웹 서비스를 개발하여 웹 서비스를 구성하는 VI와 그 외 파일들을 함께 묶고 설정합니다. 다음 단계에 따라 웹 서비스 프로젝트와 2개의 HTTP 메소드 VI를 생성합니다.

1. LabVIEW 프로젝트를 생성하여 웹 서비스 파일을 구성합니다. 프로젝트를 Tutorial1.lvproj로 저장합니다.
2. 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기» 웹 서비스를 선택합니다. LabVIEW는 타겟 아래에 웹 서비스 프로젝트 아이템과 폴더를 추가합니다.

노트 이 길라잡이에서는 웹 서비스를 생성하여 호스트 컴퓨터에 공개합니다. 프로젝트 탐색기의 Real-Time (RT) 타겟 아래에 웹 서비스 프로젝트를 생성하여 웹 서비스를 RT 타겟에 공개할 수 있습니다. 웹 서비스를 RT 타겟에 공개하려면, 웹 서비스 호스팅에 어플리케이션 웹 서버만 사용할 수 있습니다. 자세한 설명은 [길라잡이: 어플리케이션 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기](#)를 참조하십시오.

3. 웹 서비스 프로젝트 아이콘 에서 마우스 오른쪽 버튼을 클릭한 후, 이름 바꾸기를 선택하고 웹 서비스의 이름을 TutorialService로 변경합니다. 프로젝트는

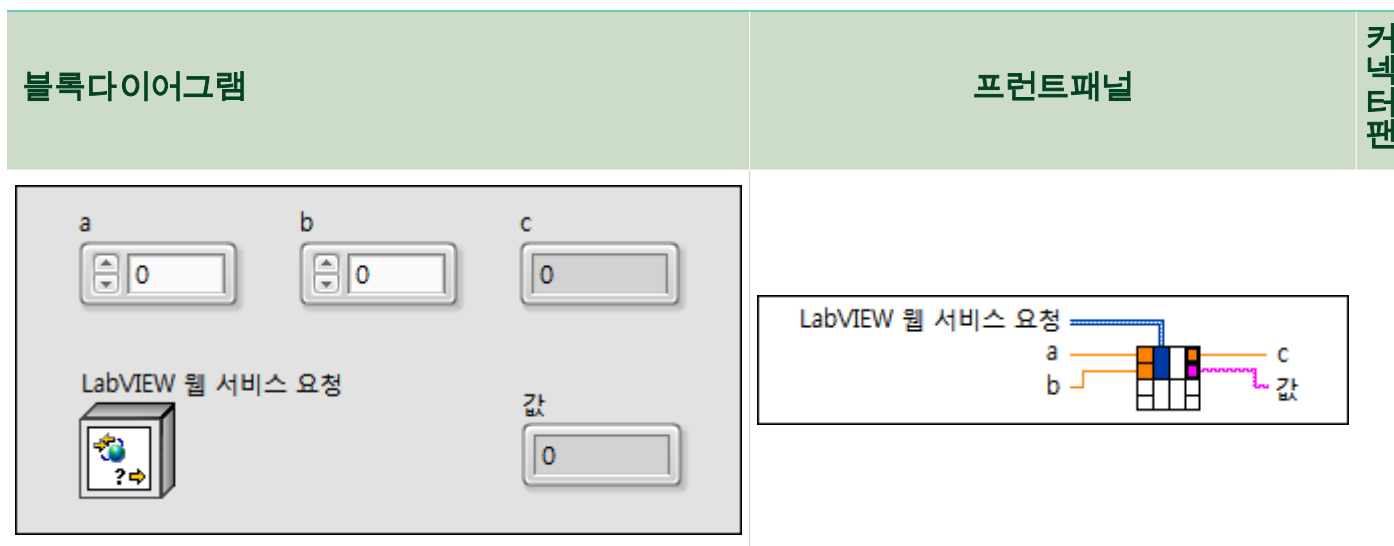


다음과 같이 나타납니다.

4. 웹 리소스에서 마우스 오른쪽 버튼을 클릭한 후, 새 VI를 선택하여 새 HTTP 메소드 VI를 생성합니다. HTTP 메소드 VI는 클라이언트로부터 HTTP 요청을 받아서 클라이언트에게 데이터를 반환하는 VI입니다. LabVIEW가 템플릿에서 새 VI를 엽니다.

노트 기본적으로 HTTP 메소드 VI는 커넥터 팬을 통해 클라이언트로 데이터를 반환합니다. 이 길라잡이에서 HTTP 메소드 VI는 기본 JSON 포맷의 텍스트 응답을 반환합니다. 이 길라잡이와 관련 없이 HTTP 메소드 VI를 생성하는 경우에는, 사용자 정의된 HTML 페이지 또는 이미지 데이터와 같은 사용자 포맷으로 클라이언트에 데이터를 스트리밍할 수도 있습니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 출력 포맷 타입을 설정합니다.

5. 새 VI의 블록 다이어그램, 프런트패널, 커넥터 팬을 다음 그림과 같이 만듭니다.



이 VI를 웹 서비스의 일부로 공개할 때, 웹 클라이언트는 VI를 인보크하도록 HTTP 요청을 보냅니다. 실행 시, 이 VI는 다음과 같은 작업을 수행합니다:

- 클라이언트가 만든 HTTP 요청(LabVIEW 웹 서비스 요청 제외)의 커넥터 팬에 할당된 모든 컨트롤의 값을 받습니다. LabVIEW 웹 서비스 요청은 현재 HTTP 요청을 식별하는 참조 번호와 같은 역할을 합니다.
- a와 b의 합계를 계산합니다.
- 커넥터 팬에 할당된 c 인디케이터를 통해 클라이언트에게 그 값의 합을 반환합니다.
- [요청 변수 읽기] VI를 실행하여, 클라이언트가 현재 HTTP 요청에 포함시킨 쿼리 문자열을 반환합니다. LabVIEW 웹 서비스 요청이 현재 HTTP 요청을 식별합니다.

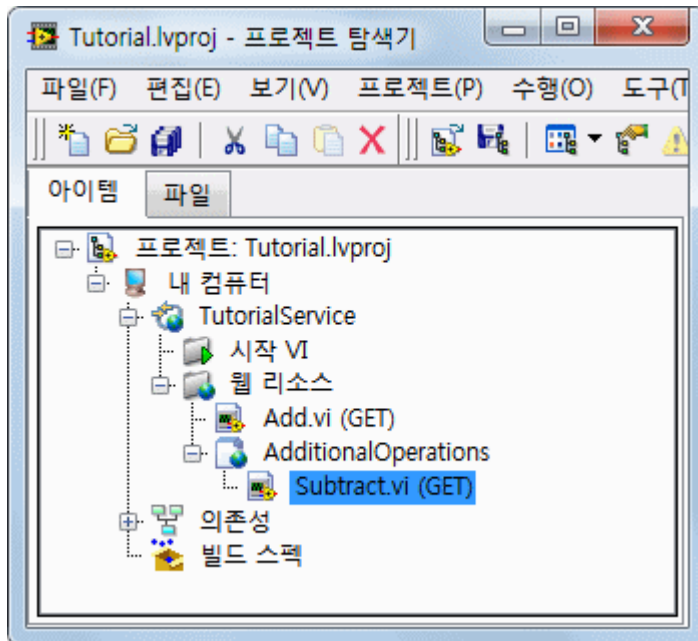
팁 보다 복잡한 웹 서비스에서는 웹 서비스 팔레트의 다른 VI들을 사용하여 HTTP 헤더 및 인증 설정과 같은 기능들을 활성화할 수 있습니다.

6. 각 컨트롤과 인디케이터가 커넥터 팬에 나타납니다. 기본 터미널 출력 타입을 사용하는 경우, 웹 서비스는 HTTP 메소드 VI의 커넥터 팬에 할당된 입력과 출력을 통해서만 클라이언트로부터 데이터를 받거나 클라이언트에 데이터를 보낼 수 있습니다.

팁 각 객체에 알아보기 쉬운 이름을 붙입니다. 이 이름은 웹 클라이언트가 VI와 데이터를 교환할 때 사용하는 URL의 일부가 됩니다.

7. VI를 Add.vi라는 이름으로 저장합니다.
8. 파일>>다른 이름으로 저장을 선택한 후, 입력 요청된 대화 상자에서 복사>>모두 열기를 선택합니다. Tutorial.lvproj에 복사본 추가 확인란에 확인 표시가 되어 있는지 확인한 후, 계속을 클릭합니다.
9. 새 복사본을 Subtract.vi로 저장합니다.
10. Subtract.vi의 블록 다이어그램을 열고, [더하기] 함수를 [빼기] 함수로 대체합니다. 컨트롤과 인디케이터를 다시 연결합니다.
11. 프로젝트로 다시 돌아와 웹 리소스에서 마우스 오른쪽 버튼을 클릭한 후, 새 웹 리소스를 선택합니다. 새 프로젝트 아이템의 이름을 AdditionalOperations라고 변경합니다.

12. Subtract.vi를 AdditionalOperations 프로젝트 아이템 아래로 끌어옵니다. 프로젝트를 저장합니다. 프로젝트는 다음 그림과 같이 됩니다.



프로젝트 트리에서 다음 구성요소를 확인합니다:

- LabVIEW는 HTTP 요청 타입을 각 HTTP 메소드 VI의 이름에 추가합니다. 기본으로 HTTP 요청 타입은 GET입니다. 이 길라잡이의 후반에 클라이언트가 특정 HTTP 메소드 VI에 보낸 HTTP 요청의 타입을 변경할 수 있습니다.
- 웹 서비스 프로젝트 아이템 은 또한 시작 VI 폴더를 포함합니다. 이 길라잡이에서는 시작 VI를 사용하지 않지만, 길라잡이의 후반에 그와 관련된 더 자세한 정보가 들어 있습니다.

관련 정보

웹 서비스의 구성요소

컨트롤과 인디케이터에 터미널 지정하기

HTTP 메소드 VI 개발하기: 출력 데이터 포맷 선택하기

HTTP 헤더, 스트리밍, 버퍼 설정하기

LabVIEW에서 프로젝트 관리하기

웹 서비스 테스트 및 디버깅하기

웹 서비스를 타겟에 공개하기 전, HTTP 메소드 VI가 클라이언트와 예상대로 통신하는지 테스트합니다. LabVIEW 프로젝트에서 NI 웹 서버로 바로 웹 서비스를 실행하여 클라이언트가 HTTP 메소드 VI에 요청을 보낼 수 있도록 합니다.

1. 웹 서비스 프로젝트 아이템 에서 마우스 오른쪽 버튼을 클릭한 후 시작을 선택합니다.
2. 사용자의 컴퓨터에서 NI 웹 서버를 처음 사용하는 경우, 다음 단계에 따라 NI 웹 서버를 활성화하고 설정합니다.
 1. 웹 서비스 시작 대화 상자에서 NI 웹 서버 설정을 클릭하여 NI Web Server Configuration을 시작합니다.
 2. 이 길라잡이에서는 단순 로컬 접근의 미리 설정된 값을 선택한 후, 다음을 클릭합니다.
 3. 인증 탭에서 서버 관리를 위한 관리자 사용자로 로그인을 선택한 후 암호를 생성합니다. 다음을 클릭합니다.
 4. 마침을 클릭하여 이 설정을 적용합니다.
3. LabVIEW 프로젝트로 돌아가서 1 단계를 반복합니다. 웹 서비스가 이제 NI 웹 서버에서 실행되고 있습니다.
4. Add.vi를 엽니다. 이때 실행 버튼은 VI가 실행을 위해 예약된 상태()임을 보여줍니다. 이는 LabVIEW가 웹 서비스를 로드하여 NI 웹 서버에서 클라이언트와 통신이 가능하도록 했기 때문입니다.
5. Add.vi를 닫습니다.
6. 다음 단계에 따라 클라이언트가 Add.vi를 실행하기 위해 사용할 URL을 얻습니다.
 1. Add.vi에서 마우스 오른쪽 버튼을 클릭한 후, 메소드 URL 보이기를 선택하여 HTTP 메소드 URL 대화 상자를 디스플레이합니다.
 2. 사용 가능한 서버 폴다운 다운 메뉴에서 NI 웹 서버가 포함된 아이템을 선택한 후, URL 복사를 클릭합니다.
 3. 대화 상자를 닫습니다.

노트 HTTP 메소드 URL 대화 상자가 열려 있는 경우, 웹 서비스는 클라이언트의 요청에 응답하지 않습니다.

7. 6 단계에서의 URL을 표준 웹 브라우저에 붙여 넣고, 다음과 같이 {value} 변수를 숫자형 값으로 대체해야 합니다: `http://127.0.0.1:80/TutorialService/Add?b=2&a=3`.

노트 NI 웹 서버는 HTTP/HTTPS 프로토콜에 따라 80/443을 기본 포트로 사용합니다. 그러나 다른 어플리케이션이 해당 포트를 점유하면 포트 번호가 변경될 수 있습니다. NI Web Server Configuration에서 포트 번호를 사용자 정의할 수 있습니다. 인증, 포트 및 원격 연결과 같은 서버 설정에 대한 더 자세한 정보는 NI 웹 서버 설정하기를 참조하십시오.

8. URL을 탐색합니다. Add.vi는 브라우저에서 HTTP 요청을 받고 a와 b의 값의 합계를 계산한 후, Add.vi의 커넥터 팬에 할당된 각 인디케이터의 이름과 출력값으로 응답합니다. 이 길라잡이에서, 응답은 기본 터미널 출력 포맷의 JSON 문자열입니다. Add.vi의 커넥터 팬에는 c와 값이라는 두 개의 인디케이터가 할당되고, HTTP 메소드 VI는 그 값을 응답으로 반환한다는 점에 유의하십시오.
- ```
{ "c": 5, "value": "b=2&a=3" }
```

노트 예상치 못한 동작을 발견하는 경우, 웹 서비스 파일을 편집하기 전에 디버깅 세션을 중지해야 합니다. 이는 LabVIEW가 파일들을 실행을 위해 예약하는 동안은 파일을 편집할 수 없기 때문입니다. 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 정지를 선택합니다.

9. Subtract.vi에 대해 6단계와 7단계를 반복하거나 7단계의 URL을 업데이트하여 Subtract.vi에 대해 다음과 같이 고유한 URL 매핑을 반영합니다. `http://127.0.0.1:80/TutorialService/AdditionalOperations/Subtract?b=2&a=3`.
10. Subtract.vi에 대해 단계 8을 반복합니다.
11. 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 정지를 선택합니다.

## 관련 정보

### 웹 서비스 테스트 및 디버깅하기

### 웹 서비스용 NI 웹 서버 설정하기

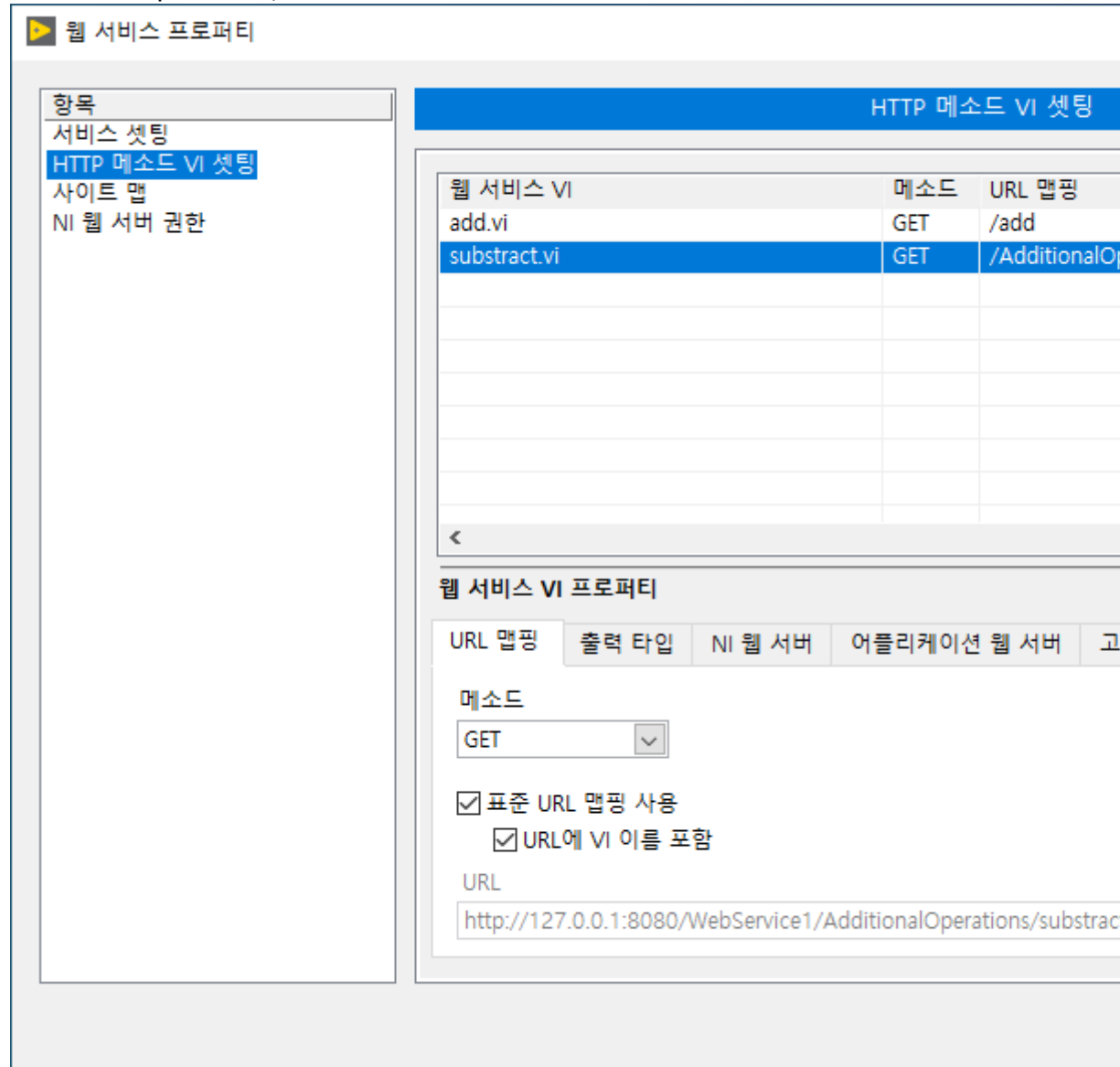
#### 클라이언트가 웹 서비스에 접근하는 방법 설정하기

웹 클라이언트는 URL을 사용하여 HTTP 메소드 VI와 데이터를 교환합니다. 이 길라잡이의 후반부에서, 웹 브라우저에 특정 URL을 입력하여 데이터를 HTTP 메소드 VI의 커넥터 팬에 있는 컨트롤로 직접 보낼 것입니다.

다음 단계를 완료하여, 클라이언트가 HTTP 메소드 VI에 접근할 때 사용할 URL을 정의합니다:

1. 웹 서비스 프로젝트 아이템 에서 마우스 오른쪽 버튼을 클릭한 후, 프로퍼티를 선택하여 웹 서비스 프로퍼티 대화 상자를 디스플레이합니다.
2. HTTP 메소드 VI 셋팅 페이지에서, LabVIEW 프로젝트의 웹 리소스 폴더에 있는 HTTP 메소드 VI는 웹 서비스 VI의 테이블에 나타납니다. 또한 AdditionalOperations 프로젝트 아래에 Subtract.vi를 놓으면 Add.vi의 경우와 비교해서 URL 매핑에 어떤 영향을 미치는지 확인합니다.
3. 테이블에서 Subtract.vi를 선택한 후, 페이지 아래쪽의 URL 매핑 탭에서 다음의 구성요소를 확인합니다.
  - 메소드 풀다운 메뉴에서 HTTP 요청 타입을 변경할 수 있습니다. Subtract.vi에 대해 기본 GET 값을 사용합니다.
  - LabVIEW가 VI에 대한 URL을 매핑할 때 사용하는 표준 포맷을 비활성화하고, 대신 서비스 이름 다음에 나오는 URL의 일부를 사용자 정의하도록 선택할 수 있습니다. 이 길라잡이에서는 표준 URL 매핑을 사용하고 VI 이름을 포함시킵니다.
  - URL 필드는 LabVIEW가 HTTP 메소드 VI에 대해 생성하는 베이스 URL을 디스플레이합니다. 예를 들어 Subtract.vi의 경우, LabVIEW는 /

AdditionalOperations/Subtract라는 기본 URL 매핑을 생성합니다.



노트 보다 복잡한 웹 서비스에서는 이 페이지의 다른 탭을 사용하여 권한과 같은 보안 설정과 VI가 웹 클라이언트에 출력 데이터를 반환하는 방식을 설정할 수도 있습니다.

4. 확인 버튼을 클릭하여 웹 서비스 프로퍼티 대화 상자를 종료합니다.
5. Tutorial.lvproj를 저장합니다.

## 관련 정보

### 웹 서비스에서 URL 매핑 및 쿼리 문자열 이해하기

## NI 웹 서버에 웹 서비스 공개하기

생성한 웹 서비스를 NI 웹 서버에 공개할 수 있습니다. NI 웹 서버는 NI가 만든 LabVIEW 웹 서비스 및 SystemLink™ 서비스와 같이 사용자가 생성한 서비스를 호스팅할 수 있는 산업용 급 웹 서버입니다. NI 웹 서버는 일반적인 웹 보안 위협으로부터 웹 어플리케이션을 보호하고 많은 엔터프라이즈 등급 데이터 서비스에 높은 확장성을 제공하며 디바이스 관리를 가능하게 합니다. 이 길라잡이에서는 웹 서비스를 NI 패키지에 포함시키고 패키지를 설치하여 공개합니다.



노트 또한 웹 서비스를 어플리케이션 웹 서버에 공개할 수 있습니다. 자세한 내용은 [길라잡이: 어플리케이션 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기](#)를 참조하십시오.

다음 단계를 완료하여, 패키지를 통해 로컬 컴퓨터에서 실행되는 NI 웹 서버에 웹 서비스를 공개합니다.

1. 프로젝트 탐색기 윈도우의 빌드 스펙에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기»패키지를 선택하여 패키지 프로퍼티 대화 상자를 시작합니다.
2. 정보 페이지에서, 패키지 출력 디렉토리 텍스트 박스에 패키지를 빌드할 위치를 지정합니다.
3. 웹 서비스 페이지에서 패키지의 웹 서비스에 포함할 TutorialService 확인란에 확인 표시를 합니다.
4. 빌드를 클릭하면 빌드 상태 대화 상자가 나타납니다.
5. LabVIEW가 패키지 빌드를 완료하면 탐색을 클릭하여 LabVIEW가 패키지 파일 .nipkg를 저장할 대상 디렉토리를 엽니다.
6. 패키지를 더블 클릭하여 NI 패키지 관리자를 통해 패키지를 설치합니다. 설치가 완료되면 웹 서비스가 호스트 컴퓨터의 NI 웹 서버에 공개됩니다.
7. LabVIEW 프로젝트를 닫습니다. 공개된 웹 서비스는 LabVIEW 런타임에서 실행되며 LabVIEW 개발 시스템에 의존하지 않습니다.

## 관련 정보

### 웹 서비스 공개하기

## 클라이언트로 웹 서비스 접근하기

다음 단계를 완료하여 URL을 통해 웹 브라우저에 공개된 웹 서비스로 데이터를 전송합니다:

1. [웹 서비스 테스트 및 디버깅하기](#) 섹션의 7 단계에서의 URL을 표준 웹 브라우저에 붙여 넣고, 다음과 같이 {value} 변수를 숫자형 값으로 대체해야 합니다:  
http://127.0.0.1:80/TutorialService/Add?b=2&a=3.

노트 클라이언트가 웹 서비스가 공개된 시스템과 다른 시스템에 위치하는 경우, 127.0.0.1을 호스트 시스템 또는 NI Web Server Configuration에서 사용자 정의한 호스트 이름의 실제 IP 주소로 대체해야 합니다.



2. 브라우저는 커넥터 팬에 할당된 각 인디케이터의 출력 값을 JSON 응답으로 반환합니다.

## 관련 정보

### URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기

## 기타 관련 리소스

- **개요: LabVIEW 어플리케이션과의 웹 기반 통신** — LabVIEW 웹 서비스에 대한 전반적인 개요
- **패키지를 통해 웹 서비스 공개하기** — 패키지를 설치할 때 웹 서비스가 실행되도록 NI 패키지에 웹 서비스를 포함시킵니다.
- **독립 어플리케이션을 통해 웹 서비스 공개하기** — 어플리케이션이 실행될 때 웹 서비스가 실행되도록 LabVIEW 독립 어플리케이션에 웹 서비스를 포함시킵니다.
- **HTTP 메소드 VI 개발하기** 또는 **HTTP 메소드 VI를 초기화하고 통신하는 시작 VI 개발하기** — 웹 서비스 어플리케이션의 일부로 실행되지만 웹 클라이언트와 데이터를 교환하지는 않는 시작 VI와, 보다 복잡한 HTTP 메소드 VI를 개발합니다.
- **웹 서비스에 정적 콘텐츠 통합하기** — 웹 서비스를 사용하여 HTML 페이지 또는 이미지와 같은 정적 콘텐츠를 공개합니다.

## 예제 코드

웹 서비스를 사용하는 예제는 다음 LabVIEW 프로젝트를 참조하십시오:

- `examples\Connectivity\Web Services\Weather Monitor\Web Services` — `Weather Monitor.lvproj`  예제 열기  관련 예제 찾기

길라잡이: 어플리케이션 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기 (Real-Time, Windows)

다음 단계에 따라 LabVIEW 웹 서비스를 생성하고 공개하여, LabVIEW 웹 서비스와 웹 클라이언트 (이 경우는 웹 브라우저) 사이에서 데이터를 교환합니다.



### 노트

이 토픽의 개념에 대한 더 자세한 정보는 **웹 서비스 개요**를 참조하십시오. LabVIEW 웹 서비스는 LabVIEW Full Development System과 LabVIEW Professional Development System에서만 사용 가능합니다.

이 길라잡이는 다음 섹션으로 구성되어 있습니다.

1. **웹 서비스 생성하기**
2. **웹 서비스 테스트 및 디버깅하기**
3. **클라이언트가 웹 서비스에 접근하는 방법 설정하기**

4. 어플리케이션 웹 서버에 웹 서비스 공개하기
5. 클라이언트로 웹 서비스 접근하기
6. 웹 서비스 모니터하기

## 웹 서비스 생성하기

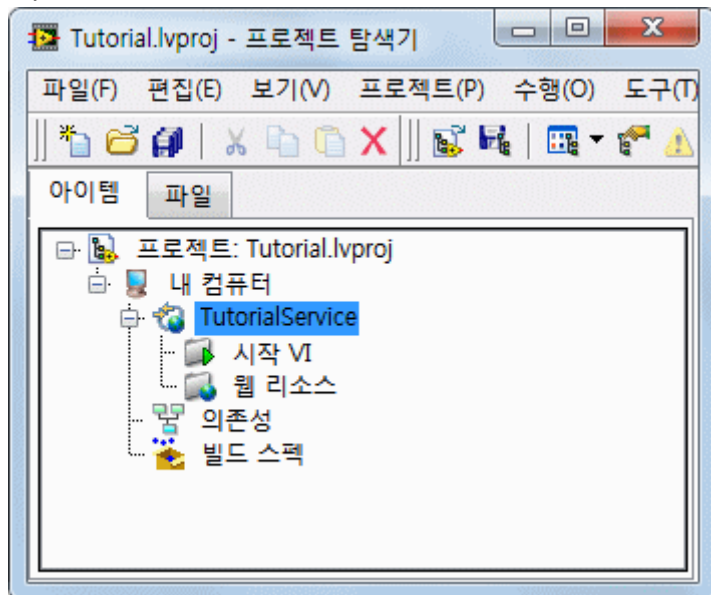
LabVIEW 프로젝트에서 웹 서비스를 개발하여 웹 서비스를 구성하는 VI와 그 외 파일들을 함께 묶고 설정합니다. 다음 단계에 따라 웹 서비스 프로젝트와 2개의 HTTP 메소드 VI를 생성합니다.

1. LabVIEW 프로젝트를 생성하여 웹 서비스 파일을 구성합니다. 프로젝트를 Tutorial.lvproj로 저장합니다.
2. 내 컴퓨터에서 마우스 오른쪽 버튼을 클릭한 후, 새로 만들기»웹 서비스를 선택합니다. LabVIEW는 타겟 아래에 웹 서비스 프로젝트 아이템과 폴더를 추가합니다.

### 노트

- 이 길라잡이에서는 웹 서비스를 생성하여 호스트 컴퓨터에 공개합니다. LabVIEW 프로젝트의 RT 타겟 아래에 웹 서비스 프로젝트를 생성하여 웹 서비스를 Real Time 타겟에 공개할 수 있습니다.
- (Windows) NI는 NI 웹 서버를 사용하여 Windows 컴퓨터에 웹 서비스를 호스팅할 것을 권장합니다. 자세한 설명은 [길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기](#)를 참조하십시오.

3. 웹 서비스 프로젝트 아이콘 에서 마우스 오른쪽 버튼을 클릭한 후, 이름 바꾸기를 선택하고 웹 서비스의 이름을 TutorialService로 변경합니다. 프로젝트는

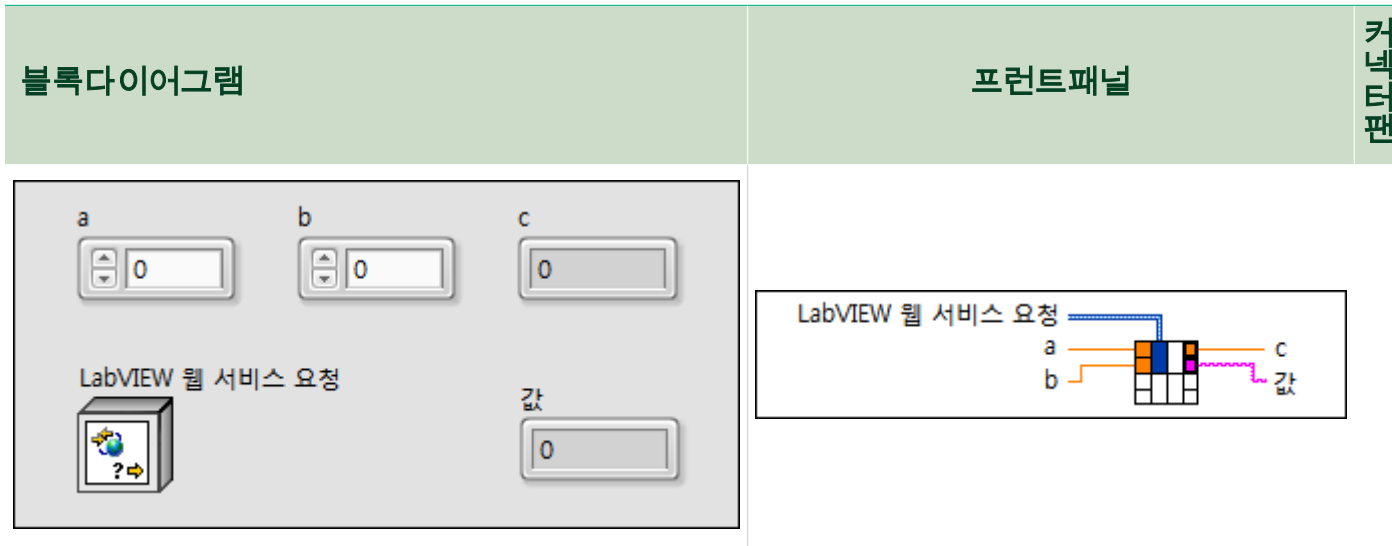


다음과 같이 나타납니다.

4. 웹 리소스에서 마우스 오른쪽 버튼을 클릭한 후, 새 VI를 선택하여 새 HTTP 메소드 VI를 생성합니다. HTTP 메소드 VI는 클라이언트로부터 HTTP 요청을 받아서 클라이언트에게 데이터를 반환하는 VI입니다. LabVIEW가 템플릿에서 새 VI를 엽니다.

노트 기본적으로 HTTP 메소드 VI는 커넥터 팬을 통해 클라이언트로 데이터를 반환합니다. 이 길라잡이에서 HTTP 메소드 VI는 기본 JSON 포맷의 텍스트 응답을 반환합니다. 이 길라잡이와 관련 없이 HTTP 메소드 VI를 생성하는 경우에는, 사용자 정의된 HTML 페이지 또는 이미지 데이터와 같은 사용자 포맷으로 클라이언트에 데이터를 스트리밍할 수도 있습니다. 웹 서비스 프로퍼티 대화 상자의 HTTP 메소드 VI 셋팅 페이지에서 출력 포맷 타입을 설정합니다.

5. 새 VI의 블록다이어그램, 프런트패널, 커넥터 팬을 다음 그림과 같이 만듭니다.



이 VI를 웹 서비스의 일부로 공개할 때, 웹 클라이언트는 VI를 인보크하도록 HTTP 요청을 보냅니다. 실행 시, 이 VI는 다음과 같은 작업을 수행합니다:

- 클라이언트가 만든 HTTP 요청(LabVIEW 웹 서비스 요청 제외)의 커넥터 팬에 할당된 모든 컨트롤의 값을 받습니다. LabVIEW 웹 서비스 요청은 현재 HTTP 요청을 식별하는 참조 번호와 같은 역할을 합니다.
- a와 b의 합계를 계산합니다.
- 커넥터 팬에 할당된 c 인디케이터를 통해 클라이언트에게 그 값의 합을 반환합니다.
- [요청 변수 읽기] VI를 실행하여, 클라이언트가 현재 HTTP 요청에 포함시킨 쿼리 문자열을 반환합니다. LabVIEW 웹 서비스 요청이 현재 HTTP 요청을 식별합니다.

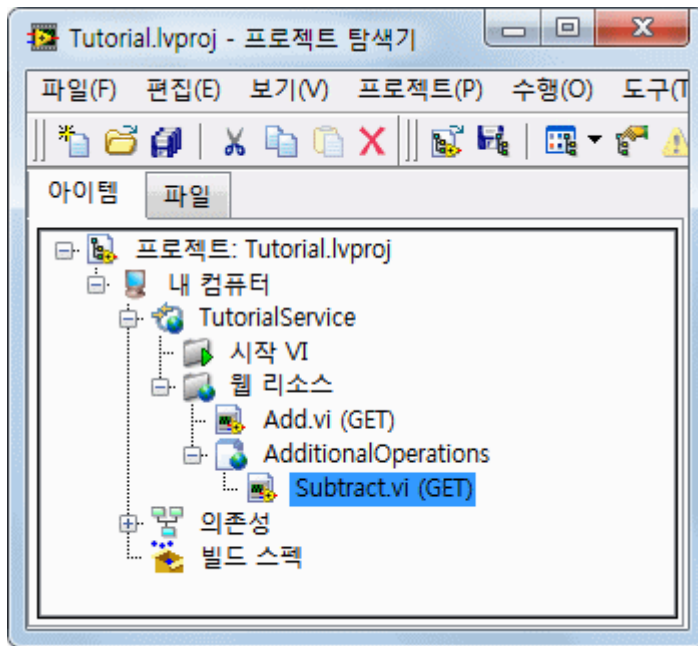
팁 보다 복잡한 웹 서비스에서는 웹 서비스 팔레트의 다른 VI들을 사용하여 HTTP 세션, 인증, HTTP 헤더 및 쿠키와 같은 기능들을 활성화하십시오.

6. 각 컨트롤과 인디케이터가 커넥터 팬에 나타납니다. 기본 터미널 출력 타입을 사용하는 경우, 웹 서비스는 HTTP 메소드 VI의 커넥터 팬에 할당된 입력과 출력을 통해서만 클라이언트로부터 데이터를 받거나 클라이언트에 데이터를 보낼 수 있습니다.

팁 각 객체에 알아보기 쉬운 이름을 붙입니다. 이 이름은 웹 클라이언트가 VI와 데이터를 교환할 때 사용하는 URL의 일부가 됩니다.

7. VI를 Add.vi라는 이름으로 저장합니다.
8. 파일>>다른 이름으로 저장을 선택한 후, 입력 요청된 대화 상자에서 복사>>모두 열기를 선택합니다. Tutorial.lvproj에 복사본 추가 확인란에 확인 표시가 되어 있는지 확인한 후, 계속을 클릭합니다.

9. 새 복사본을 Subtract.vi로 저장합니다.
10. Subtract.vi의 블록다이어그램을 열고, [더하기] 함수를 [빼기] 함수로 대체합니다. 컨트롤과 인디케이터를 다시 연결합니다.
11. 프로젝트로 다시 돌아와 웹 리소스에서 마우스 오른쪽 버튼을 클릭한 후, 새 웹 리소스를 선택합니다. 새 프로젝트 아이템의 이름을 AdditionalOperations라고 변경합니다.
12. Subtract.vi를 AdditionalOperations 프로젝트 아이템 아래로 끌어옵니다. 프로젝트를 저장합니다. 프로젝트는 다음 그림과 같이 됩니다.



프로젝트 트리에서 다음 구성요소를 확인합니다:

- LabVIEW는 HTTP 요청 타입을 각 HTTP 메소드 VI의 이름에 추가합니다. 기본으로 HTTP 요청 타입은 GET입니다. 이 길라잡이의 후반에 클라이언트가 특정 HTTP 메소드 VI에 보낸 HTTP 요청의 타입을 변경할 수 있습니다.
- 웹 서비스 프로젝트 아이템 은 또한 시작 VI 폴더를 포함합니다. 이 길라잡이에서는 시작 VI를 사용하지 않지만, 길라잡이의 후반에 그와 관련된 더 자세한 정보가 들어 있습니다.

관련 정보

[웹 서비스의 구성요소](#)

[컨트롤과 인디케이터에 터미널 지정하기](#)

[HTTP 메소드 VI 개발하기: 출력 데이터 포맷 선택하기](#)

[HTTP 헤더, 스트리밍, 버퍼 설정하기](#)

[LabVIEW에서 프로젝트 관리하기](#)

## 웹 서비스 테스트 및 디버깅하기

웹 서비스를 타겟에 공개하기 전, HTTP 메소드 VI가 클라이언트와 예상대로 통신하는지 테스트합니다. 웹 서비스를 LabVIEW 프로젝트에서 직접 디버깅 서버에 놓아 클라이언트가 HTTP 메소드 VI에 요청을 보낼 수 있도록 합니다. 디버깅 서버는 Sandbox 환경을 제공합니다.

1. (Windows) 내 컴퓨터 아래의 웹 서비스 프로젝트 아이콘 에서 마우스 오른쪽 버튼을 클릭한 후, 어플리케이션 웹 서버>>시작 (디버그 서버)를 선택합니다.  
(Real-Time) RT 타겟 아래의 웹 서비스 프로젝트 아이콘 에서 마우스 오른쪽 버튼을 클릭한 후, 시작 (디버그 서버)를 선택합니다.
2. Add.vi를 엽니다. 이때 실행 버튼은 VI가 실행을 위해 예약된 상태()임을 보여줍니다. 이는 LabVIEW가 웹 서비스를 로드하여 디버깅 서버에서 클라이언트와 통신이 가능하도록 했기 때문입니다.
3. Add.vi를 닫습니다.
4. 다음 단계에 따라 클라이언트가 Add.vi를 실행하기 위해 사용할 URL을 얻습니다.
  1. Add.vi에서 마우스 오른쪽 버튼을 클릭한 후, 메소드 URL 보이기를 선택하여 HTTP 메소드 URL 대화 상자를 디스플레이합니다.
  2. 사용 가능한 서버 풀다운 다운 메뉴에서 로컬 디버깅이 포함된 아이콘을 선택한 후, URL 복사 버튼을 클릭합니다. 웹 서비스를 웹 서버에 공개한 경우, 클라이언트는 다른 포트를 사용하여 디버깅 서버에서 웹 서비스와 통신해야 합니다.
  3. 대화 상자를 닫습니다.

---

노트 HTTP 메소드 URL 대화 상자가 열려 있는 경우, 웹 서비스는 클라이언트의 요청에 응답하지 않습니다.

---

5. 4 단계에서의 URL을 표준 웹 브라우저에 붙여 넣고, 다음과 같이 {value} 변수를 숫자형 값으로 대체해야 합니다: `http://127.0.0.1:8001/TutorialService/Add?b=2&a=3`.
6. URL을 탐색합니다. Add.vi는 브라우저에서 HTTP 요청을 받고 a와 b의 값의 합계를 계산한 후, Add.vi의 커넥터 팬에 할당된 각 인디케이터의 이름과 출력값으로 응답합니다. 이 길라잡이에서, 응답은 기본 터미널 출력 포맷의 JSON 문자열입니다. Add.vi의 커넥터 팬에는 c와 값이라는 두 개의 인디케이터가 할당되며 HTTP 메소드 VI는 그 값을 응답으로 반환한다는 점에 유의하십시오.  

```
{ "c": 5, "value": "b=2&a=3" }
```

---

노트 예상치 못한 동작을 발견하는 경우, 웹 서비스 파일을 편집하기 전에 디버깅 세션을 중지해야 합니다. 이는 LabVIEW가 파일들을 실행을 위해 예약하는 동안은 파일을 편집할 수 없기 때문입니다.

---

7. Subtract.vi에 대해 4단계와 5단계를 반복하거나 5단계의 URL을 업데이트하여 Subtract.vi에 대해 다음과 같이 고유한 URL 매핑을 반영합니다: `http://127.0.0.1:8001/TutorialService/AdditionalOperations/Subtract?b=2&a=3`.
8. Subtract.vi에 대해 단계 6을 반복합니다.

9. (Windows) 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 어플리케이션 웹 서버 >> 정지(디버그 서버)를 선택합니다. (Real-Time) 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 정지(디버그 서버)를 선택합니다.

## 관련 정보

### 어플리케이션 웹 서버에서 웹 서비스 테스트 및 디버깅하기

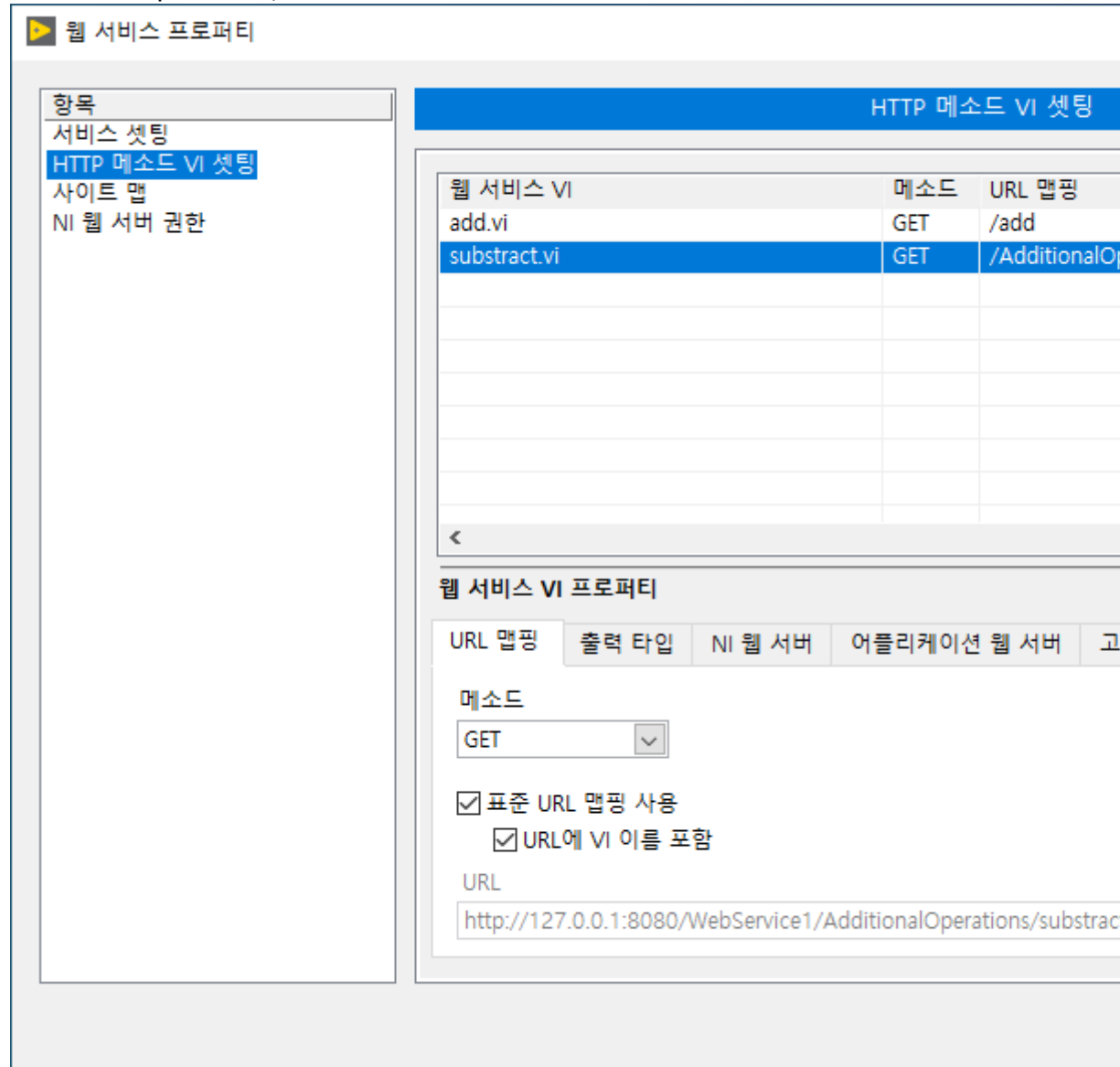
#### 클라이언트가 웹 서비스에 접근하는 방법 설정하기

웹 클라이언트는 URL을 사용하여 HTTP 메소드 VI와 데이터를 교환합니다. 이 길라잡이의 후반부에서, 웹 브라우저에 특정 URL을 입력하여 데이터를 HTTP 메소드 VI의 커넥터 팬에 있는 컨트롤로 직접 보낼 것입니다.

다음 단계를 완료하여, 클라이언트가 HTTP 메소드 VI에 접근할 때 사용할 URL을 정의합니다:

1. 웹 서비스 프로젝트 아이템 에서 마우스 오른쪽 버튼을 클릭한 후, 프로퍼티를 선택하여 웹 서비스 프로퍼티 대화 상자를 디스플레이합니다.
2. HTTP 메소드 VI 셋팅 페이지에서, LabVIEW 프로젝트의 웹 리소스 폴더에 있는 HTTP 메소드 VI는 웹 서비스 VI의 테이블에 나타납니다. 또한 AdditionalOperations 프로젝트 아래에 Subtract.vi를 놓으면 Add.vi의 경우와 비교해서 URL 매핑에 어떤 영향을 미치는지 확인합니다.
3. 테이블에서 Subtract.vi를 선택한 후, 페이지 아래쪽의 URL 매핑 탭에서 다음의 구성요소를 확인합니다.
  - 메소드 폴다운 메뉴에서 HTTP 요청 타입을 변경할 수 있습니다. Subtract.vi에 대해 기본 GET 값을 사용합니다.
  - LabVIEW가 VI에 대한 URL을 매핑할 때 사용하는 표준 포맷을 비활성화하고, 대신 서비스 이름 다음에 나오는 URL의 일부를 사용자 정의하도록 선택할 수 있습니다. 이 길라잡이에서는 표준 URL 매핑을 사용하고 VI 이름을 포함시킵니다.
  - URL 필드는 LabVIEW가 HTTP 메소드 VI에 대해 생성하는 베이스 URL을 디스플레이합니다. 예를 들어 Subtract.vi의 경우, LabVIEW는 /

AdditionalOperations/Subtract라는 기본 URL 매핑을 생성합니다.



노트 보다 복잡한 웹 서비스에서는 이 페이지의 다른 탭을 사용하여 권한과 같은 보안 설정과 VI가 웹 클라이언트에 출력 데이터를 반환하는 방식을 설정할 수도 있습니다.

4. 확인 버튼을 클릭하여 웹 서비스 프로퍼티 대화 상자를 종료합니다.
5. Tutorial.lvproj를 저장합니다.

## 관련 정보

### 웹 서비스에서 URL 매핑 및 쿼리 문자열 이해하기

## 어플리케이션 웹 서버에 웹 서비스 공개하기

생성한 웹 서비스를 어플리케이션 웹 서버에 공개할 수 있습니다. 이 길라잡이에서는 독립 웹 서비스를 공개합니다. 하지만 보다 복잡한 어플리케이션에서는 LabVIEW 독립 어플리케이션 또는 설치 프로그램을 통해 웹 서비스를 공개할 수 있습니다. 어플리케이션 웹 서버는 네트워크에 웹 서비스를 호스트하며, SSL (Secure Sockets Layer) 암호화를 비롯하여 네트워크 데이터 교환을 보호하는 여러 보안 관련 도구를 제공합니다.



노트 (Windows) 또한 웹 서비스를 NI 웹 서버에 공개할 수 있습니다. 자세한 설명은 [길라잡이: NI 웹 서버에 LabVIEW 웹 서비스 생성 및 공개하기](#)를 참조하십시오.

다음 단계를 완료하여, 로컬 컴퓨터에서 실행되는 어플리케이션 웹 서버에 웹 서비스를 공개합니다:

1. 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후 어플리케이션 웹 서버» 공개를 선택합니다. 배포 진행 대화 상자가 나타납니다.
2. LabVIEW가 에러 없이 웹 서비스를 공개하면, 닫기 버튼을 클릭합니다. 이제 웹 클라이언트는 HTTP 메소드 VI와 데이터를 교환할 수 있습니다.

## 관련 정보

### 웹 서비스 공개하기

### 웹 서비스용 어플리케이션 웹 서버 설정하기

## 클라이언트로 웹 서비스 접근하기

다음 단계를 완료하여 URL을 통해 웹 브라우저에 공개된 웹 서비스로 데이터를 전송합니다:

1. Add.vi에서 마우스 오른쪽 버튼을 클릭한 후, 메소드 URL 보이기를 선택하여 HTTP 메소드 URL 대화 상자를 디스플레이합니다.
2. 사용 가능한 서버 풀다운 다운 메뉴에서 어플리케이션이 포함된 아이템을 선택한 후, URL 복사 버튼을 클릭합니다. 이 URL에는 디버깅 서버의 경우의 URL과는 다른 포트 번호가 포함됩니다. LabVIEW는 이 포트 번호를 어플리케이션 웹 서버에서 실행되는 웹 서비스와의 통신에 사용합니다.
3. 대화 상자를 닫습니다.
4. URL을 표준 웹 브라우저에 붙여 넣고, 다음과 같이 {value} 변수를 숫자형 값으로 대체해야 합니다: `http://127.0.0.1:8080/TutorialService/Add?b=2&a=3`.

노트 클라이언트가 웹 서비스가 공개된 시스템과 다른 시스템에 위치한 경우, 127.0.0.1을 호스트 시스템 또는 [NI 웹 기반 설정 및 모니터링](#)에서 사용자 정의한 호스트 이름의 실제 IP 주소로 대체해야 합니다.



5. 브라우저는 커넥터 팬에 할당된 각 인디케이터의 출력 값을 JSON 응답으로 반환합니다.

## 관련 정보

### URL을 사용하여 배포된 어플리케이션으로 데이터 전송하기

#### 웹 서비스 모니터링하기

다음 단계에 따라 어플리케이션 웹 서버에 공개된 웹 서비스의 상태를 확인합니다:

1. 웹 서비스 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭한 후, 어플리케이션 웹 서버 >> 웹 서버 관리를 선택합니다. 웹 브라우저에 NI 웹 기반 설정 및 모니터링이 열립니다.



2. 왼쪽 사이드바의 웹 서비스 관리 버튼을 클릭합니다.
3. 공개된 웹 서비스 리스트에서 TutorialService를 선택합니다. 이 리스트가 비어 있는 경우, 새로 고침 버튼을 클릭합니다.
4. 웹 서비스의 상태가 실행 중이 됩니다. 페이지 아래쪽의 버튼을 사용하여 웹 서비스를 일시 중지하거나, 다시 시작하거나 비공개할 수 있습니다.
5. HTTP 메소드 VI, Add 및 AdditionalOperations/Subtract에 대한 URL 맵핑이 실행 중인 HTTP 메소드와 함께 페이지의 오른쪽에 나타납니다.

#### 기타 관련 리소스

- **웹 클라이언트에서 LabVIEW 어플리케이션과 통신하기**
- **개요: LabVIEW 어플리케이션과의 웹 기반 통신** — LabVIEW 웹 서비스에 대한 전반적인 개요
- **독립 어플리케이션을 통해 웹 서비스 공개하기** — 어플리케이션이 실행될 때 웹 서비스가 실행되도록 LabVIEW 독립 어플리케이션에 웹 서비스를 포함시킵니다.
- **설치 프로그램을 통해 웹 서비스 공개하기** — 설치 프로세스를 완료할 때 웹 서비스가 실행되도록 Windows 설치 프로그램에 웹 서비스를 포함시킵니다.
- **HTTP 메소드 VI 개발하기** 또는 **HTTP 메소드 VI를 초기화하고 통신하는 시작 VI 개발하기** — 웹 서비스 어플리케이션의 일부로 실행되지만 웹 클라이언트와 데이터를 교환하지는 않는 시작 VI와, 보다 복잡한 HTTP 메소드 VI를 개발합니다.
- **웹 서비스에 정적 콘텐츠 통합하기** — 웹 서비스를 사용하여 HTML 페이지 또는 이미지와 같은 정적 콘텐츠를 공개합니다.

#### 예제 코드

웹 서비스를 사용하는 예제는 다음 LabVIEW 프로젝트를 참조하십시오:

- examples\Connectivity\Web Services\Weather Monitor\Web Services — Weather Monitor.lvproj  예제 열기  관련 예제 찾기

## 새 브라우저 접근 리스트 엔트리 추가하기

다음 단계를 따라 브라우저 접근 리스트에 새 엔트리를 추가합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 웹 서버를 선택합니다.
3. 브라우저 주소 텍스트 박스에 유효한 브라우저 주소를 입력합니다. mydomain.com과 같은 도메인 이름이나 130.164.123.123과 같은 IP 주소를 입력합니다. 또한, 브라우저 주소에 **와일드 카드를 사용**할 수 있습니다.



### 노트

DNS 서버에 접근할 수 없는 컴퓨터에서 웹 서버가 실행될 경우, 브라우저 접근 리스트에 도메인 이름 엔트리를 사용하지 마십시오. 도메인 이름 또는 IP 주소 분석 요청이 실패하면 컴퓨터가 느려집니다.

1. 보기와 컨트롤링 허용, 보기 허용 또는 접근 거부 버튼을 클릭하여 선택된 아이템에 대한 접근을 각각 허용하거나 거부합니다. 프런트패널의 보기와 컨트롤링을 허용하면 두 개의 녹색 확인 표시가 나타나고, 프런트패널의 보기만을 허용하면 한 개의 녹색 확인 표시가 나타나며, 접근을 거부하면 붉은색 X가 나타납니다.
2. 엔트리의 이름 옆에 하나 또는 두 개의 녹색 확인 표시나 붉은 X가 없는 경우, 엔트리에 대한 구문은 올바르지 않은 것입니다. 엔트리를 편집하거나 제거 버튼을 클릭하여 엔트리를 삭제합니다.
3. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

## 웹 브라우저 접근 허용 및 거부하기

옵션 대화 상자의 웹 서버 페이지에 있는 브라우저 접근 섹션을 사용하여 사용자 VI 프런트패널을 볼 수 있는 브라우저 주소를 설정할 수 있습니다. 개별 브라우저 주소에 접근을 허용하고 거부하는 브라우저 접근 리스트를 생성합니다. 또한, 웹 서버: TCP/IP 접근 리스트 프로퍼티를 사용하여 프로그램적으로 브라우저 주소에 접근을 허용하고 거부할 수 있습니다.

웹 브라우저가 VI 프런트패널 이미지를 얻으려고 시도할 때, 웹 서버는 브라우저 주소를 브라우저 접근 리스트에 있는 엔트리와 비교해서 접근 허용 여부를 결정합니다. 브라우저 접근 리스트의 엔트리가 브라우저 주소와 일치할 경우, 웹 서버는 엔트리 설정을 어떻게 했는지에 따라서 접근을 허용하거나 거부합니다. 기본적으로 모든 브라우저는 LabVIEW 웹 서버에 접근할 수 있습니다. 새로운 머신 접근 리스트 엔트리를 추가하거나,

기존 엔트리를 편집하거나, 엔트리를 제거하는 경우, 엔트리에 올바른 구문, 순서, 와일드카드를 사용하십시오.



#### 관련 정보

[브라우저 주소 리스트를 위한 구문](#) ( 1527 페이지 참조)

[새 브라우저 접근 리스트 엔트리 추가하기](#) ( 1518 페이지 참조)

[기존의 브라우저 접근 리스트 엔트리 편집하기](#) ( 1519 페이지 참조)

[브라우저 접근 리스트 엔트리 제거하기](#) ( 1412 페이지 참조)

[브라우저 주소 리스트를 위한 구문](#) ( 1527 페이지 참조)

## 기존의 브라우저 접근 리스트 엔트리 편집하기

다음 단계를 따라 브라우저 접근 리스트에서 브라우저 주소를 편집합니다.

1. 도구 » 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 웹 서버를 선택합니다.
3. 브라우저 접근 섹션의 브라우저 접근 리스트에서 수정하려는 주소를 선택합니다. 주소가 텍스트 박스에 나타납니다.
4. 텍스트 박스의 주소를 편집하거나 보기와 컨트롤링 허용, 보기 허용 또는 접근 거부 버튼을 클릭합니다. 또한, 브라우저 주소에 [와일드카드를 사용](#)할 수 있습니다.
5. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다.

## 프런트패널

### 브라우저 또는 LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널 컨트롤하기

다음 단계에 따라 웹 브라우저 또는 LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널을 컨트롤합니다.



#### 노트

보려는 어플리케이션 또는 VI가 서버 컴퓨터의 메모리에 있어야 합니다.

1. [LabVIEW](#) 또는 [브라우저에서 원격으로 어플리케이션 또는 프런트패널](#)을 봅니다.
2. 리모트 프런트패널 윈도우의 아래를 클릭하거나 프런트패널의 어느 곳에서나 마우스 오른쪽 버튼을 클릭하고, 바로 가기 메뉴에서 VI의 컨트롤 요청을 선택합니다. 현재 다른 클라이언트가 컨트롤 권한을 행사하고 있지 않을 경우, 현재 사용자에게 프런트패널의 컨트롤 권한이 부여됩니다. 현재 다른 클라이언트가 VI를 컨트롤하고 있는 경우, 서버는 다른 클라이언트가 컨트롤을 해제하거나 컨트롤 시간 제한이 타임아웃될 때까지 요청을 큐합니다. 서버 컴퓨터의 사용자만 [큐 리스트를 모니터](#)할 수 있습니다.

서버 컴퓨터에서 실행되는 웹 서버와의 연결을 완전히 끊으려면, 브라우저 윈도우나 LabVIEW 윈도우 경계에서 닫기 문양을 클릭하십시오. 그러나, 프런트패널의 컨트롤은 해제하나 연결은 유지하여 프런트패널을 보고자 하는 경우, 프런트패널의 빈 스페이스에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 VI의 컨트롤 해제를 선택하거나 그래프와 같은 프런트패널 객체에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 리모트 패널 클라이언트 >> VI의 컨트롤 해제를 선택합니다.



노트 원격 컴퓨터에서 실행되는 VI로 생성된 데이터를 저장하려는 경우, 리모트 프런트패널 대신 **DataSocket** 또는 **TCP**를 사용하십시오.

또한, 어플리케이션 메소드를 사용하여 프로그램적으로 리모트 프런트패널의 컨트롤을 해제하고 요청할 수 있습니다.

## 리모트 프런트패널을 위한 웹 서버를 활성화하고 설정하기

다음 단계를 따라 **프런트패널을 원격으로 보고, 설정**할 수 있도록 LabVIEW 웹 서버를 활성화하고 설정합니다. 이 웹 서버는 LabVIEW가 호스트 머신에서 열려있을 때만 실행될 수 있습니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 옵션 대화 상자의 항목 리스트에서 웹 서버 페이지를 선택합니다.
3. 리모트 패널 서버 섹션의 리모트 패널 서버 활성화 확인란에 확인 표시를 합니다. 기본 웹 서버 설정은 대부분의 어플리케이션에 적합합니다.



### 노트

(Linux) 기본적으로, LabVIEW는 8000 포트를 사용하여 루트에 접근할 수 없는 계층에서 웹 서버가 실행될 수 있도록 합니다. 루트 접근을 가지는 경우, 포트를 80으로 변경할 수 있습니다. 80 이외의 다른 포트를 사용하는 경우, 다음 예제와 같이 서버를 참조하는 URL에 반드시 포트 번호를 지정해야 합니다.

`http://hostname:8000/index.htm`

1. 프런트패널에 대한 **브라우저 접근을 제어**하거나 웹에서 **볼 수 있는 VI를 제어**하려는 경우, 웹 서버 페이지의 브라우저 접근과 VI 보이기 섹션을 설정합니다.
2. 확인 버튼을 클릭하여 옵션 대화 상자를 닫고 변경사항을 저장합니다. 웹 서버가 실행 중입니다.

또한, 웹 서버 프로퍼티를 사용하여 웹 서버를 프로그램적으로 활성화하고 설정할 수 있습니다. 또한 **빌드된 어플리케이션에서 웹 서버를 활성화하고 설정**할 수 있습니다.

## 리모트 프론트 패널에 대한 TLS/SSL 보안 활성화하기 (Windows)

**리모트 프론트패널** 관련 네트워크 통신을 위해 이전에는 SSL (Secure Sockets Layer)이라고 알려진 **TLS (Transport Layer Security)** 암호화를 활성화할 수 있습니다.



### 노트

TLS/SSL 암호화를 **LabVIEW 웹 서비스**와 함께 사용하려면 **어플리케이션 웹 서버에서 TLS/SSL을 활성화**해야 합니다.

다음 단계를 따라 LabVIEW 웹 서버에서 리모트 프론트패널에 대한 TLS/SSL을 활성화합니다.

1. 옵션 대화 상자의 웹 서버 페이지에 있는 리모트 패널 서버 섹션을 찾습니다. 필요하다면 **LabVIEW 웹 서버를 리모트 프론트패널과 함께 사용할 수 있도록 활성화**하십시오.
2. SSL 확인란에 확인 표시를 합니다.
3. SSL 포트 필드에 TLS/SSL로 암호화할 네트워크 포트를 입력하거나 또는 기본 TLS/SSL 포트인 443을 사용합니다. 기본 포트인 80을 사용하는 네트워크 통신에서는 TLS/SSL을 활성화할 수 없습니다.
4. SSL 인증서 파일 텍스트 박스를 공란으로 남겨두면, **기본 LabVIEW 자체 서명 인증서**를 사용할 수 있습니다.
5. (옵션) **NI 웹 기반 설정 및 모니터링**를 사용하여 **사용자 자체 서명 인증서를 생성**합니다. 또한, **인증 기관의 서명을 받은 인증서**를 사용할 수 있습니다.
6. (옵션) 인증서를 포함하는 시스템 또는 타겟의 주소를 서버 주소 텍스트 박스에 지정합니다.

노트 또한 SSL 인증서 파일 텍스트 박스를 사용하여 사용자가 생성한 인증서 파일을 직접 찾을 수 있습니다.

7. (옵션) 쿼리 버튼을 클릭하여, 서버에 있는 사용 가능한 인증서를 발견된 인증서 리스트 박스에 디스플레이합니다.
8. (옵션) 발견된 인증서 리스트박스에서 인증서를 선택합니다.

## 리모트 프론트패널에 연결할 때 에러

다음 테이블은 **웹 브라우저** 또는 **LabVIEW에서 리모트 프론트패널에 연결**할 때 발생할 수 있는 에러를 나열합니다.

| 에러                             | 해결 방법                                                                                        |
|--------------------------------|----------------------------------------------------------------------------------------------|
| 유효하지 않은 서버 IP 주소               | 연결하려는 컴퓨터의 정확한 IP 주소 또는 이름을 얻기 위해서 서버 관리자에게 연락합니다.                                           |
| 리모트 패널 연결이 지정된 서버에 의하여 거부되었습니다 | 서버 관리자는 반드시 <b>LabVIEW 웹 서버를 활성화</b> 해야 합니다. 리모트 패널에 연결 대화 상자에 입력한 서버 IP 주소와 포트가 올바른지 확인합니다. |

| 에러                                         | 해결 방법                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 클라이언트는 리모트패널 접근 권한이 없습니다.                  | 서버 관리자는 반드시 리모트 프런트패널을 보거나 제어하려는 모든 잠재적 클라이언트에 대해서 <b>IP 주소</b> 를 지정해야 합니다. 서버 관리자가 어떤 IP 주소도 지정하지 않는 경우, LabVIEW는 모든 클라이언트가 리모트 프런트패널에 접근하도록 허용합니다.                                                                                                                                                                                               |
| 리모트 패널 프로토콜 버전이 호환되지 않습니다                  | 클라이언트와 서버 컴퓨터는 반드시 동일한 버전의 LabVIEW를 실행해야 합니다. 리모트 프런트패널을 보고 컨트롤하기 위해서 브라우저를 사용하는 경우, 반드시 서버 컴퓨터의 LabVIEW 버전과 호환되는 <b>LabVIEW 런타임 엔진 버전을 사용</b> 해야 합니다. 또한, 서버 관리자와 연락하여 HTML 문서가 LabVIEW 런타임 엔진의 <b>정확한 버전을 지정</b> 하는지 확인합니다.                                                                                                                      |
| LabVIEW 클라이언트 버전이 LabVIEW 서버 버전과 호환되지 않습니다 | 반드시 서버 컴퓨터의 LabVIEW의 버전과 호환되는 LabVIEW의 버전을 사용해야 합니다. 리모트 프런트패널을 보고 컨트롤하기 위해서 브라우저를 사용하는 경우, 반드시 서버 컴퓨터의 LabVIEW 버전과 호환되는 <b>LabVIEW 런타임 엔진 버전을 사용</b> 해야 합니다. 또한, 서버 관리자와 연락하여 HTML 문서가 LabVIEW 런타임 엔진의 <b>정확한 버전을 지정</b> 하는지 확인합니다.                                                                                                               |
| 서버는 리모트 패널을 지원하지 않습니다                      | 서버의 IP 주소 또는 컴퓨터 이름을 정확하게 입력했는지를 확인합니다.                                                                                                                                                                                                                                                                                                            |
| 리모트 패널 서버가 요청한 VI를 보내는 데 실패하였습니다           | 메모리 또는 네트워크 장애가 발생했습니다. 이 에러는 매우 큰 프런트패널에 연결을 시도할 경우에 발생할 수도 있습니다. 프런트패널에 연결을 다시 시도하거나 보다 적은 메모리를 사용할 수 있도록 프런트패널 수정을 서버 관리자에게 요구합니다. 또한 이 에러는 사용자가 웹 서버를 활성화하지 않은 상태에서 VI를 리얼타임 타겟에 배포한 이후에 웹 서버를 활성화할 때 발생할 수 있습니다. LabVIEW가 웹 서버가 활성화되어 있을 때 리얼타임 타겟에 배포된 VI의 프런트패널만을 보존하는데, 이 경우에는 배포된 VI가 프런트패널을 갖고 있지 않기 때문에 에러가 발생합니다. 이는 성능 최적화를 위한 동작입니다. |
| 클라이언트는 지정된 VI에 접근 권한이 없습니다.                | 서버 관리자는 반드시 클라이언트가 보거나 제어할 수 있는 <b>VI를 지정</b> 해야 합니다.                                                                                                                                                                                                                                                                                              |
| 요청된 VI는 서버 컴퓨터에서 메모리로 로드되지 않았습니다.          | VI는 VI가 서버 컴퓨터에 열려 있을 때마다 메모리에 있는 것으로 간주됩니다. 이 VI가 실행 중일 필요는 없습니다. 또한 입력한 VI 이름이 .vi 확장자를 포함하여 호스트 컴퓨터와 정확히 일치하도록 합니다.                                                                                                                                                                                                                             |

## 리모트 프런트패널 보기와 컨트롤에서 지원되지 않는 기능

다음 리스트에서는 리모트 프런트패널을 확인하고 컨트롤할 때 지원되지 않는 기능과 권장사항에 대해 설명합니다.

- 웹 브라우저의 제약 때문에, 프런트패널이 웹 페이지의 일부로 디스플레이될 때, 프런트패널의 넓이나 위치를 조작하는 사용자 인터페이스 어플리케이션은 잘 작동하지 않습니다. 웹 서버와 LabVIEW 브라우저 플러그인은 복잡한 사용자 인터페이스 어플리케이션도 LabVIEW에서 구현하는 것과 동일하게 실행하도록 만들지만, 특히 대화상자나 SubVI 윈도우가 있는 경우, 일부 어플리케이션은 웹 브라우저의 환경에서 잘 작동하지 않을 수 있습니다. National Instruments는 웹 브라우저에서 그와 같은 타입의 어플리케이션을 실행하지 않을 것을 권유합니다.
- [기다림] 함수를 가지지 않고 While 루프를 가진 VI의 반출은 피합니다. 이러한 VI는 우선 순위가 낮은 태스크가 일정한 시간동안 수행하는 것을 방해하여 원격에서 프런트패널을 보고 컨트롤할 때 프런트패널이 반응을 하지 않을 수 있습니다.
- 특정 VI는 원격 컴퓨터에서 실행할 때와 로컬 컴퓨터에서 실행할 때 동일하게 수행되지 않을 수도 있습니다. 임베디드 .NET과 ActiveX 컨트롤은 LabVIEW에 상관없이 거의 독립적으로 그림을 그리거나 수행되므로 리모트 클라이언트에서는 나타나지 않습니다. VI가 표준 파일 대화 상자를 사용할 경우, 원격에서 파일 시스템을 탐색할 수 없으므로 컨트롤러는 에러를 받게 됩니다. 또한, 경로 컨트롤의 탐색 버튼은 리모트 패널에서 비활성화됩니다.
- 클라이언트가 원격에서 프런트패널을 볼 때, 이 프런트패널이 생성된 어플리케이션과 연결이 되어 있는지 아닌지에 따라서 다른 작동을 합니다. 특히, 생성된 어플리케이션의 프런트패널인 경우, 클라이언트가 이 프런트패널에 접속하기 전에 프로그램적으로 변경된 부분은 클라이언트의 컴퓨터에 적용되지 않습니다. 예를 들어, 클라이언트가 프런트패널에 접속하기 전에 프로퍼티 노드로 컨트롤의 캡션을 변경할 경우, 클라이언트는 변경된 캡션이 아닌 원래의 캡션을 보게 됩니다.
- VI 서버를 이용하여 다이나믹하게 열고 실행된 VI의 프런트패널 또는 호출될 때 프런트패널을 디스플레이하도록 설정된 subVI의 프런트패널을 컨트롤러만이 원격으로 볼 수 있습니다. VI를 컨트롤할 권한이 없는 클라이언트는 그 프런트패널을 볼 수 없습니다.
- 프런트패널 컨트롤의 프로퍼티를 폴링하여 사용자 인터페이스의 효과를 얻는 블록 다이어그램은 원격 컴퓨터에서 VI를 컨트롤할 때 성능이 저하될 수 있습니다. 이러한 VI의 성능은 [프런트패널 작업 기다림] 함수를 사용하여 향상시킬 수 있습니다.
- LabVIEW는 원격에서 보기 또는 컨트롤하려는 VI의 패널 닫기 이벤트를 생성할 수 없습니다. VI를 원격에서 보거나 컨트롤하는 경우, LabVIEW는 어플리케이션 또는 VI 클래스가 아닌 컨트롤 클래스에서만 이벤트를 생성할 수 있습니다.
- SubVI 호출 시 SubVI의 프런트패널을 디스플레이하는 VI를 원격으로 보거나 컨트롤하는 경우, 수동으로 SubVI의 프런트패널을 닫지 마십시오. 그렇지 않으면, 클라이언트 컴퓨터와 서버 컴퓨터에 있는 VI의 컨트롤을 잃을 수도 있습니다. 호출된 SubVI의 프런트패널을 확인해야 하는 경우에는, LabVIEW가 **SubVI의 실행 후 이를 닫도록** 설정하십시오.

## 리모트 프런트패널 잠그기와 잠금해제

클라이언트가 원격으로 볼 수 있는 프런트패널을 잠그거나 잠금해제하여 클라이언트가 프런트패널 컨트롤의 값을 변경하거나 변경하지 못하도록 할 수 있습니다.

실행되는 VI의 프런트패널에서 마우스 오른쪽 버튼을 클릭하고 컨트롤 잠금을 선택하여 클라이언트가 원격으로 보는 프런트패널을 잠급니다. 클라이언트가 원격으로 보는 프런트패널을 잠그면, LabVIEW는 클라이언트의 요청을 큐하여 프런트패널을 컨트롤합니다.



또한, 리모트 패널:컨트롤 잠금 메소드를 사용하여 프로그램적으로 리모트 프론트패널의 컨트롤을 잠글 수 있습니다.

실행되는 VI의 프론트패널에서 마우스 오른쪽 버튼을 클릭하고 컨트롤 잠금해제를 선택하여 클라이언트가 원격으로 프론트패널을 보고 프론트패널 컨트롤의 값을 변경하는 것을 허용합니다. 클라이언트가 원격으로 보는 프론트패널을 잠금해제할 때, LabVIEW는 큐의 다음 클라이언트에게 컨트롤을 줍니다. 큐에 클라이언트가 없는 경우, 메소드는 프론트패널을 잠금해제합니다.

또한, 리모트 패널:컨트롤 잠금해제 메소드를 사용하여 프로그램적으로 리모트 프론트패널의 컨트롤을 잠금해제할 수 있습니다.

## 프론트패널의 클라이언트 트래픽 관리하기

클라이언트가 원격 컴퓨터로 보고 컨트롤하도록 제공하는 서버를 관리한다면, 서버에 대한 모든 클라이언트 트래픽을 모니터할 수 있습니다. 도구>>리모트 패널 연결 관리자 를 선택하여 리모트 패널 연결 관리자 윈도우를 디스플레이합니다.

이 윈도우를 사용하여 클라이언트 큐 리스트를 관리하고, 클라이언트에 대한 연결을 끊고, 네트워크 클라이언트 트래픽 데이터를 얼마나 오래 유지하는지 지정합니다. 모든 클라이언트의 연결을 끊으려면, VI 이름이나 전체 네트워크 트래픽을 선택하고 클라이언트 연결 끊기 버튼을 클릭합니다.

또한, 리모트 패널:서버에 연결 및 리모트 패널:클라이언트에 연결 프로퍼티를 사용하여 리모트 프론트패널 연결에 대한 정보를 받을 수 있습니다.

## 웹에 VI와 독립 어플리케이션 공개하기

LabVIEW 웹 서버를 이용하여, VI나 독립 어플리케이션의 프론트패널 이미지를 웹에 공개합니다. 웹 서버가 활성화되면, 기본적으로 모든 VI와 어플리케이션은 모든 웹 브라우저에 보이도록 되어 있습니다. 그러나 공개된 프론트패널에 브라우저 접근을 컨트롤하고, 웹에 어느 VI를 나타낼 것인지를 설정할 수 있습니다. 웹에 프론트패널을 디스플레이하기 위해, VI는 컴퓨터의 메모리에 있어야 합니다.

웹 서버는 프론트패널의 이미지를 JPEG 또는 PNG 이미지 포맷으로 생성할 수 있습니다. JPEG 포맷은 그래픽 압축을 잘 하지만 정밀한 그래픽의 일부가 손상될 수 있습니다. 이 포맷은 사진에 가장 적합합니다. 라인 아트, 프론트패널, 블록다이어그램에서, JPEG 압축은 희미한 그래픽과 균일하지 않은 색을 가져올 수 있습니다. PNG 포맷 또한 그래픽을 잘 압축하지만, JPEG 포맷만큼 항상 잘 하지는 못합니다. 하지만, PNG 압축시 그래픽의 정밀한 부분이 손상되지 않습니다.

웹 출판 도구를 사용하여 HTML 문서를 생성하고 프론트패널의 정적 또는 애니메이션 이미지를 내장하여 클라이언트 컴퓨터가 원격으로 프론트패널을 보고 제어할 수 있도록 합니다.. 또한, 프론트패널의 정적 이미지를 기존의 HTML 문서에 넣을 수 있습니다. 이 어플리케이션을 이용하여 HTML 문서를 배포하려면, 독립 어플리케이션에서 웹 서버를 활성화하고 설정합니다.

웹 출판 도구의 스냅샷 옵션을 사용하여 현재 서버 컴퓨터의 메모리 내에 있는 VI의 프론트패널의 정적 이미지를 반환합니다. 웹 출판 도구의 모니터 옵션을 사용하여 현재 서버 컴퓨터의 메모리 내에 있는 VI의 프론트패널의 애니메이션 이미지를 반환합니다. 웹 출



판 도구의 임베디드 옵션을 사용하여 프런트패널을 HTML 문서에 임베디드하여 클라이언트 컴퓨터가 프런트패널을 원격으로 보고 제어할 수 있도록 합니다.



#### 노트

공개하려는 VI나 어플리케이션의 프런트패널이 클 수 있습니다. 웹 서버가 큰 이미지를 생성하는데 더 오랜 시간이 걸릴 수 있고 이것은 VI의 실행 속도를 늦출 수 있습니다. 또한, 큰 이미지를 브라우저를 통해 전달하는데, 특히 느린 모뎀 연결로 상당한 시간이 걸립니다. 그러므로, 공개하려는 VI와 어플리케이션의 프런트패널이 적합한 크기인지 확인해야 합니다.

VI 혹은 어플리케이션을 공개한 후, 어플리케이션 혹은 프런트패널을 원격 제어하거나, 브라우저를 사용하여 또는 LabVIEW를 사용하여 확인할 수 있습니다.

또한 ni.com에서 등록된 사용자와 VI를 공유할 수 있습니다.



#### 관련 정보

[리모트 프런트패널을 위한 웹 서버를 활성화하고 설정하기](#) ( 1520 페이지 참조)

[웹 브라우저 접근 허용 및 거부하기](#) ( 1518 페이지 참조)

[어떤 VI가 웹에서 보이는지 지정하기](#) ( 1527 페이지 참조)

[HTML 문서 생성하기 및 프런트패널 이미지 내장하기](#) ( 1400 페이지 참조)

[브라우저를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기](#) ( 1530 페이지 참조)

[기존의 HTML 문서에 프런트패널 이미지 내장하기](#) ( 1402 페이지 참조)

[독립 어플리케이션에서 웹 서버 활성화하기](#) ( 1459 페이지 참조)

[브라우저 또는 LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널 컨트롤하기](#) ( 1519 페이지 참조)

[브라우저를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기](#) ( 1530 페이지 참조)

[LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기](#) ( 1531 페이지 참조)

## VI에서 E-메일 보내기

[E-메일 보내기] 익스프레스 VI를 사용하면 LabVIEW에서 지정한 수신자 리스트로 신속하게 E-메일을 보낼 수 있습니다.

다음 단계에 따라 [E-메일 보내기] 익스프레스를 설정하여 이메일을 전송합니다:



#### 노트

헤더를 설정하거나, TLS (Transport Layer Security) 셋팅을 설정하거나, 첨부 파일을 포함하려는 경우, 일반 SMTP E-메일 VI를 사용하여 보다 고급 셋팅으로 E-메일을 설정하고 보낼 수 있습니다. 일반 SMTP E-메일 VI를 사용하여 이메일을 발송하는 예제는 labview\examples\Data Communication\Protocols\SMTP\Send Email using SMTP Client.vi를 참조하십시오.

1. [E-메일 보내기] 익스프레스 VI를 블록다이어그램에 추가합니다. E-메일 보내기 설정 대화 상자가 자동으로 열립니다.
2. 사용자 정보 섹션에 있는 발신자 E-메일 주소 텍스트 박스에 사용자의 E-메일 주소 또는 E-메일을 발송할 E-메일 주소를 지정합니다.
3. E-메일 발송에 사용할 호스트 이름 또는 SMTP 서버의 IP 주소를 `my.mailserver.com`와 같이 발신 메일 서버 (SMTP) 텍스트 박스에 지정합니다. 어느 메일 서버를 사용해야 할지 알고 있지 않다면, 유효한 서버의 이름을 위해 네트워크 관리자에게 연락하기 바랍니다.
4. 포트 텍스트 박스에 E-메일 전송에 사용할 SMTP 서버의 포트 번호를 지정합니다. 기본값은 25이며, 이는 일반적으로 공개 연결에 사용됩니다. 그러나 보안 연결을 사용하는 경우, 포트 번호를 변경해야 할 수 있습니다.
5. (옵션) 로그인 필요 확인란에 확인 표시를 하여 SMTP 서버로 로그인 자격 증명을 전송합니다. 대부분의 공개 SMTP 서버는 로그인 자격 증명을 요구합니다.
  1. 사용자 이름 텍스트 박스에서 지정한 SMTP 서버의 로그인 사용자 이름을 지정합니다. 대부분의 공개 서버에서 이는 E-메일 주소이거나 또는 E-메일을 보내는 발신 E-메일의 주소입니다.
  2. 암호 텍스트 박스에서 지정한 SMTP 서버의 로그인 암호를 지정합니다.
6. (옵션) 보안 연결 사용(TLS\SSL) 확인란에 확인 표시를 하여, LabVIEW가 SMTP 서버와 통신할 때 TLS\SSL 보안 연결을 요구하도록 합니다.
7. E-메일 섹션의 수신자 텍스트 박스에 수신자(들)의 이메일 주소(복수)를 지정합니다. 여러 수신자에게 E-메일을 보내려면, 각 E-메일 주소를 쉼표로 구분합니다. 적어도 한 개의 E-메일 주소를 입력해야 합니다. 또한 수신자 리스트를 블록다이어그램의 수신자 입력에 연결할 수도 있습니다.
8. (옵션) 제목 텍스트 박스에 메시지의 제목을 지정합니다. 또한 제목 라인을 블록다이어그램의 제목 입력에 연결할 수도 있습니다.
9. (옵션) 메시지 텍스트 박스에서 수신자에게 보내려는 메시지를 일반 텍스트로 지정합니다. 또한 메시지를 블록다이어그램의 메시지 입력에 연결할 수도 있습니다.
10. (옵션) 테스트 E-메일 보내기 버튼을 클릭하고, 발신자 E-메일 주소에 지정한 E-메일 주소로 테스트 이메일을 전송하여 VI의 설정을 테스트합니다.
11. 확인 버튼을 클릭하여 설정을 저장하고 E-메일 보내기 설정 대화 상자를 닫습니다.
12. 블록다이어그램의 모든 에러 입력과 에러 출력 터미널을 연결합니다.
13. VI를 실행합니다. LabVIEW는 설정한 셋팅에 따라 E-메일을 전송합니다.

## 클라이언트 컨트롤에서 시간 제한 설정하기

웹 서버와 함께, 여러 클라이언트가 VI 제어를 기다릴 때 원격 클라이언트가 VI를 제어할 수 있는 시간 제한을 설정할 수 있습니다. 시간 제한을 설정하여 모든 클라이언트가 VI의 컨트롤을 얻는 기회를 가지도록 보장합니다. 특정한 클라이언트가 시간 제한에 도달할

때, 큐에 있는 다음 클라이언트는 컨트롤러입니다. 그러나, 모든 클라이언트는 여전히 프론트패널을 볼 수 있습니다.



#### 노트

LabVIEW는 특정 VI에서 설정된 시간 제한을 두 번째 클라이언트가 그 VI의 컨트롤을 요구할 때까지 모니터링하지 않습니다. 두 번째 클라이언트가 VI의 컨트롤을 요구하지 않으면, 첫 번째 클라이언트는 VI의 컨트롤을 계속 유지합니다.

다음 단계를 따라 각 클라이언트가 특정한 VI를 컨트롤할 수 있는 시간 제한을 설정합니다.

1. 도구 >> 옵션을 선택하여 옵션 대화 상자를 디스플레이합니다.
2. 항목 리스트에서 웹 서버를 선택합니다.
3. VI 보이기 섹션의 VI 보이기 리스트에서 VI를 클릭합니다. VI가 리스트에 나타나지 않으면, 리스트에 **VI 보이기 엔트리를 추가**할 수 있고 그것을 선택할 수 있습니다.
4. 접근 허용 확인란에 확인 표시를 합니다.
5. 기본적인 시간 제한에 변경하려면, 기본 사용 확인란에서 확인 표시를 제거합니다. 기본은 300초입니다.
6. 컨트롤 시간 제한(초) 필드에서, 각 클라이언트가 VI 보이기 리스트에서 선택된 VI의 프론트패널을 컨트롤할 수 있는 시간을 초 단위로 입력합니다.
7. 확인 버튼을 클릭합니다.

## 어떤 VI가 웹에서 보이는지 지정하기

웹에서 보이는 VI를 설정하려면, 웹 서버 페이지의 VI 보이기 섹션을 사용합니다. 개별적인 VI나 그룹 VI, 또는 디렉토리 경로에 대한 접근을 허용하고 거부하는 VI 보이기 리스트를 생성합니다. 또한, VI 접근 리스트 프로퍼티를 사용하여 VI에 대한 접근을 프로그램적으로 허용하거나 거부할 수 있습니다.

웹 브라우저가 VI 프론트패널 이미지를 얻으려고 시도할 때, 웹 서버는 VI 이름을 VI 보이기 리스트에 있는 엔트리와 비교해서 접근을 허용할 것인지 결정합니다. VI 보이기 리스트의 엔트리가 VI 이름과 일치하면, 웹 서버는 사용자가 어떻게 **엔트리를 설정**했는지에 따라 해당 VI의 이미지에 대한 접근을 허용하거나 거부합니다. 기본적으로 모든 VI의 프론트패널 이미지는 볼 수 있습니다. 새로운 VI 보이기 리스트 엔트리를 **추가**하거나, 기존의 엔트리를 **편집**하거나, 엔트리를 **제거**할 때, 엔트리에 올바른 **구문, 순서, 와일드카드**를 사용합니다.

## 브라우저 주소 리스트를 위한 구문

옵션 대화 상자의 웹 서버 페이지에 있는 브라우저 접근 섹션의 브라우저 접근 리스트 또는 옵션 대화 상자의 VI 서버 페이지에 있는 머신 접근 리스트에 130.164.140.12 등의 IP 주소나 www.ni.com 등의 도메인 이름을 입력할 수 있습니다.

\* 와일드카드를 사용하여 브라우저 주소의 그룹을 지정합니다. 예를 들면,  
 \*.domain.com의 엔트리를 가진 domain.com 도메인 안의 모든 브라우저를 지정할 수 있습니다. 서브넷의 모든 브라우저를 지정할 수 있는데 이 서브넷의 처음 두 개의 8진

수는 130.164.\*의 엔트리를 가진 130.164입니다. 도메인 이름의 시작이나 IP 주소의 끝에 \* 와일드카드를 사용할 수 있습니다.

브라우저 접근 리스트 또는 머신 접근 리스트의 엔트리에 대한 권한은 이전의 권한을 덮어씁니다. 순서를 변경하기 위해 리스트에서 엔트리를 클릭하고 끌어서 놓을 수 있습니다.

예를 들면, \*.test.site.com으로 끝나는 모든 브라우저 주소에 대한 접근을 거부하지만, 그 엔트리 후에 a.test.site.com과 b.test.site.com에 접근하면, 이 두 개의 브라우저는 여전히 접근할 수 있는데, 그 이유는 리스트에서 나중의 엔트리가 처음의 엔트리를 덮어쓰기 때문입니다.

일반적으로, \* 와일드카드를 사용하여 일반적인 허가나 거부를 설정해야 하고, 이전 허가의 일부를 뒤집는 보다 특정한 엔트리를 갖는 엔트리가 뒤따르도록 합니다. 최고의 성능을 위해 가장 자주 일치되는 엔트리를 리스트의 아래에 놓습니다.

다음 예제는 \* 와일드카드를 올바르게 사용하는 방법을 설명합니다.

| 접근<br>리스트<br>엔트리 | 권한 상태           |                                                     |
|------------------|-----------------|-----------------------------------------------------|
| ✓                | *               | 모든 브라우저 주소에 접근할 수 있습니다.                             |
| ✓                | *.site.com      | .site.com로 끝나는 모든 브라우저 주소에 접근할 수 있습니다.              |
| X                | public.site.com | 이전 엔트리가 접근을 허용해도 이 브라우저 주소에 대한 접근을 거부합니다.           |
| X                | *.test.site.com | .test.site.com로 끝나는 모든 브라우저 주소에 접근할 수 없습니다.         |
| ✓                | a.test.site.com | 이전 엔트리가 접근을 거부해도 이 브라우저 주소에 접근할 수 있습니다.             |
| X                | 130.164.123.*   | 130.164.123로 시작하는 IP 주소를 가진 모든 브라우저 주소에 접근할 수 없습니다. |
| ✓                | 130164123123    | 이전 엔트리가 접근을 거부해도 이 브라우저 주소에 접근할 수 있습니다.             |

## VI의 구문 열거하기

보이기와 VI 반출 리스트의 각 엔트리는 VI 이름이나 VI 경로를 설명하고 와일드카드 문자를 가질 수 있습니다. 웹과 VI 서버는 VI 경로에 대한 경로 분리자를 가진 엔트리를 비교하고 VI 이름을 상대로 경로 분리자를 갖지 않은 엔트리를 비교합니다.

보이기와 VI 반출 리스트의 엔트리를 지정할 때 다음 와일드카드 문자를 사용합니다.

| 와일드카드 | 작업                                      |
|-------|-----------------------------------------|
| ?     | 경로 분리자를 제외하고 정확히 하나의 임의의 문자와 일치합니다.     |
| *     | 경로 분리자를 제외하고 제로 또는 그 이상의 임의의 문자와 일치합니다. |
| **    | 경로 분리자를 포함하여 제로 또는 그 이상의 임의의 문자와 일치합니다. |
| \     | 특수 문자의 해석을 취소합니다.                       |

리스트에서 나중의 엔트리에 대한 권한이 이전 권한을 덮어씁니다. 순서를 변경하기 위해 리스트에서 엔트리를 클릭하고 끌어서 놓을 수 있습니다.

예를 들면, `c:\labview\server`에서 모든 VI 프런트패널에 대한 접근을 거부하지만, `c:\labview\server\local.vi`에 대한 접근을 허용한 그 엔트리 후에, 리스트에서 나중의 엔트리가 처음 엔트리를 덮어쓰기 때문에 그 VI는 볼 수 있습니다.

일반적으로, \* 와일드카드를 사용하여 일반적인 허가나 거부를 설정해야 하고, 이전 허가의 일부를 뒤집는 보다 특정한 엔트리를 갖는 엔트리가 뒤따르도록 합니다. 최고의 성능을 위해 가장 자주 일치되는 엔트리를 리스트의 아래에 놓습니다.

다음 예제는 올바르게 와일드카드를 사용하는 방법을 설명합니다.

| VI가 엔트리를 열거합니다. | 권한 상태                                   |                                                                               |
|-----------------|-----------------------------------------|-------------------------------------------------------------------------------|
| ✓               | *                                       | 모든 VI에 접근할 수 있습니다.                                                            |
| ✓               | <code>c:\labview\server\*</code>        | <code>c:\labview\server</code> 의 모든 VI에 접근할 수 있습니다.                           |
| ✓               | <code>c:\labview\test\**</code>         | <code>c:\labview\test</code> 와 그 서브디렉토리의 모든 VI에 접근할 수 있습니다.                   |
| X               | <code>c:\labview\test\private.vi</code> | 이전 엔트리가 접근을 허용해도 이 VI에 대한 접근을 거부합니다.                                          |
| ✓               | <code>srvr_*.vi</code>                  | <code>srvr_</code> 문자열을 가지고 시작하고 <code>.vi</code> 문자열로 끝나는 모든 VI에 접근할 수 있습니다. |
| X               | <code>Are You Finished\?.vi</code>      | <code>완료하였습니까?.vi</code> 에 대한 접근을 거부합니다.                                      |

| VI가 엔트리틀 열거합니다. | 권한 상태    |                                                                         |
|-----------------|----------|-------------------------------------------------------------------------|
| X               | Open?.vi | 이름이 Open?으로 지정된 모든 VI에 대한 접근을 거부하는데, 이때 ?는 경로 분리자를 제외한 임의의 한 문자를 나타냅니다. |

## 브라우저를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기

웹 브라우저를 사용하여 원격으로 어플리케이션 또는 프런트패널을 보려면, 우선 서버를 설정해야 합니다. 다음 단계를 따라 서버를 설정하여 클라이언트 컴퓨터가 브라우저를 사용하여 원격으로 어플리케이션 프런트패널을 볼 수 있도록 합니다.



### 팁

원격으로 보거나 컨트롤되는 VI에서 프런트패널은 되도록 작게 생성하십시오. 웹 서버는 작은 프런트패널 이미지보다 큰 프런트패널 이미지를 로드하고 디스플레이하는데 더 많은 시스템 자원을 사용하므로 VI 자체의 실행이 지연될 수 있습니다.

1. 확인하려는 **어플리케이션** 또는 **VI**가 위치해 있는 서버 컴퓨터의 웹 서버를 활성화 시키십시오.
2. **브라우저 접근 리스트**를 설정하여 클라이언트 컴퓨터가 서버에 연결될 수 있도록 합니다.
3. 어플리케이션이 실행 중이거나 클라이언트가 보는 프런트패널을 가진 VI가 서버 컴퓨터의 메모리에 있음을 확인합니다.
4. 웹 출판 도구를 사용하여 **HTML 파일을 생성**합니다. 또한, **기존의 HTML 파일**을 사용할 수 있습니다.
5. 웹 출판 도구를 사용하지 않는 경우, HTML파일을 **labview\www** 디렉토리에 놓습니다. <OBJECT>와 <EMBED> 태그의 LVFPVNAME 파라미터를 메모리에 로드한 어플리케이션 또는 최상위 VI의 이름으로 변경합니다. 웹 출판 도구를 사용하는 경우, 이 단계는 자동으로 완성됩니다.

또한, <OBJECT>와 <EMBED> 태그의 WIDTH와 HEIGHT 파라미터를 변경하여 클라이언트 브라우저에서 어플리케이션 또는 프런트패널 모양을 사용자 정의합니다.

클라이언트 서버에 있고 LabVIEW가 설치되어 있지 않으나 브라우저를 사용하여 원격으로 프런트패널을 보려면 다음 단계를 따르십시오.



### 노트

브라우저에서 프런트패널을 원격으로 볼 때, 보려는 어플리케이션이나 VI가 서버 컴퓨터에서 실행 중이어야 합니다.

1. **LabVIEW 런타임 엔진**을 설치합니다.
2. 기본 브라우저를 시작합니다.
3. 브라우저의 URL에서, 서버의 IP 주소 또는 컴퓨터 이름을 입력하고 그 뒤에 HTML 파일의 이름을 타입합니다. 예를 들어 `http://ipaddress/filename.html` 입니다. 접근이 거부되는 경우, 브라우저에 **에러 메시지**가 나타납니다. 어플리케이션 또는 리모트 프런트패널의 **컨트롤을 요청**할 수 있습니다.

서버 컴퓨터에서 실행되는 웹 서버와의 연결을 완전히 끊으려면, 브라우저 윈도우 경계에서 닫기 문양을 클릭하십시오.

또한, 어플리케이션 메소드를 사용하여 연결을 열고 프로그램적으로 리모트 프런트패널의 컨트롤을 요청할 수 있습니다.

## LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널 보기

LabVIEW를 사용하여 원격으로 어플리케이션 또는 프런트패널을 보려면, 우선 서버를 설정해야 합니다. 다음 단계를 따라 서버를 설정하여 클라이언트 컴퓨터가 LabVIEW를 사용하여 원격으로 프런트패널을 볼 수 있도록 합니다.



### 팁

원격으로 보거나 컨트롤되는 VI에서 프런트패널은 되도록 작게 생성하십시오. 웹 서버는 작은 프런트패널 이미지보다 큰 프런트패널 이미지를 로드하고 디스플레이하는데 더 많은 시스템 자원을 사용하므로 VI 자체의 실행이 지연될 수 있습니다.

1. 확인하려는 **VI** 또는 **어플리케이션**이 위치해 있는 서버 컴퓨터의 웹 서버를 활성화 시키십시오.
2. **브라우저 접근 리스트**를 설정하여 클라이언트 컴퓨터가 서버에 연결될 수 있도록 합니다.
3. **보이는 VI 리스트**를 설정하여 클라이언트 컴퓨터가 원격으로 VI 또는 어플리케이션 프런트패널에 접근할 수 있게 합니다. 보이는 VI 리스트를 설정하지 않는 경우, 클라이언트는 서버 컴퓨터의 메모리에 로드된 모든 VI에 접근할 수 있습니다.
4. 어플리케이션이 실행 중이거나 클라이언트가 보는 프런트패널을 가진 VI가 서버 컴퓨터의 메모리에 있음을 확인합니다.

클라이언트 서버에 있고 LabVIEW를 사용하여 원격으로 프런트패널을 보려면 다음 단계를 따르십시오.

1. 수행>>리모트 패널에 연결을 선택하여 리모트 패널에 연결 대화 상자를 디스플레이합니다.

2. 서버 IP 주소 필드에, 연결하려는 서버의 IP 주소 또는 컴퓨터 이름을 입력합니다.
3. VI 이름 필드에, 프런트패널을 보려는 VI의 이름을 입력합니다. 또는, 보려는 어플리케이션의 최상위 VI의 이름을 입력합니다.
4. HTTP 포트를 입력합니다. 기본은 80입니다.
5. 즉시 프런트패널의 컨트롤을 요청하려는 경우, 컨트롤 요청 확인란에 확인 표시를 합니다. 또한, 리모트 프런트패널이 컴퓨터에 나타났을 때 리모트 프런트패널 윈도우의 아래에서 **컨트롤을 요청**할 수 있습니다.
6. 연결 버튼을 클릭합니다. 프런트패널이 컴퓨터에 나타나 보도록 해줍니다. 접근이 거부되는 경우, 리모트 패널에 연결 대화 상자의 연결 상태 섹션에 **에러 메시지**가 나타납니다.



#### 노트

원격 컴퓨터에서 실행되는 VI로 생성된 데이터를 저장하려는 경우, 리모트 프런트패널 대신 **DataSocket** 또는 **TCP**를 사용하십시오.

서버 컴퓨터에서 실행되는 웹 서버와의 연결을 완전히 끊으려면, LabVIEW 윈도우 경계에서 닫기 문양을 클릭하십시오.



노트 리모트 프런트패널을 관리하는 서버 컴퓨터는 LabVIEW Full 또는 Professional Development System을 가지고 있거나 프런트패널을 독립 어플리케이션으로 구축해야 합니다. 클라이언트는 어떤 LabVIEW 개발 시스템을 사용해서도 원격으로 프런트패널을 보고 컨트롤할 수 있습니다.

또한, 어플리케이션 메소드를 사용하여 연결을 열고 프로그램적으로 리모트 프런트패널의 컨트롤을 요청할 수 있습니다.

## 웹 서버 구문 예제

웹 출판 도구를 사용하여 **프런트패널의 정적 또는 애니메이션 이미지를 HTML 문서에 포함**시키거나 또는 프런트패널을 HTML 문서에 포함시켜 클라이언트 컴퓨터가 **원격으로 프런트패널을 보고 제어**할 수 있도록 합니다. 프런트패널의 정적 이미지를 기존 HTML 문서에 포함하려면 다음의 시나리오에 어플리케이션의 최상위 VI 이름을 입력합니다.

HTML 문서에서 동일한 컴퓨터에서 실행하는 Example.vi라고 이름 지어진 VI의 정적 이미지를 넣기 위해, HTML 문서에서 다음의 코드를 사용합니다:

```

```

HTML 문서에 foo 컴퓨터에서 실행하는 Example.vi라고 이름 지어진 VI의 정적 이미지를 넣기 위해, HTML 문서에 다음의 코드를 사용합니다:

```

```

type=png의 이미지와 depth=24의 이미지를 사용하여 foo 컴퓨터에서 실행 중인 Test 1.vi라는 VI의 정적 이미지를 넣으려면 HTML 문서에 다음 코드를 사용합니다.



```

```

## TCP 클라이언트 생성하기

다음 단계를 따라 TCP 함수를 사용하여 TCP 클라이언트를 생성합니다.

1. (옵션) TCP 서버를 생성합니다. TCP 서버를 생성할 때, 원격 포트나 서비스 이름을 지정하여 서버가 통신에 사용할 포트를 나타냅니다.
2. [TCP 연결 열기] 함수를 이용하여 서버와의 연결을 엽니다. 서버에 대한 원격 포트 또는 서비스 이름을 지정해야 합니다. 원격 포트 또는 서비스 이름은 서버가 통신 요청에 대해 응답할 때 사용하는 컴퓨터의 통신 채널을 식별합니다. 서비스 이름을 지정하면, LabVIEW는 서버가 등록한 포트 번호에 대한 NI Service Locator를 쿼리합니다. 원격 컴퓨터와 연결을 설정하고자 하는 경우, 서버의 인터넷 주소를 지정해야 합니다. 주소는 네트워크 상의 컴퓨터를 식별합니다. 주소를 지정하지 않는 경우, LabVIEW는 로컬 컴퓨터에 연결을 만듭니다.
3. [TCP 읽기] 함수를 사용하여 서버에서 메시지를 받습니다. 반드시 읽으려는 바이트의 크기를 지정해야 합니다.
4. [TCP 쓰기] 함수를 사용하여 서버에 메시지를 보냅니다.
5. [TCP 연결 닫기] 함수를 이용하여 서버와의 연결을 닫습니다.

TCP 클라이언트의 예제는 다음 VI를 참조하십시오:

- labview\examples\Data Communication\Protocols\TCP\TCP Named Service\TCP Named Service.lvproj  예제 열기  관련 예제 찾기



### 관련 정보

[TCP 서버 생성하기](#) ( 1533 페이지 참조)

[NI Service Locator](#) ( 705 페이지 참조)

## TCP 서버 생성하기

다음 단계를 따라 TCP 함수를 사용하여 TCP 서버를 생성합니다.

1. [TCP 리슨] VI나 [TCP 리스너 생성] 및 [TCP 리스너 기다림] 함수를 사용하여 연결을 기다립니다. 포트와 서비스 이름(옵션)을 지정해야 합니다. 이 포트나 서비스 이름은 반드시 클라이언트가 접근을 시도하는 포트나 서비스 이름과 같아야 합니다.
  - [TCP 리슨] VI
  - [TCP 리스너 생성] 함수
  - [TCP 리스너 기다림] 함수

2. 연결이 설정되면, [TCP 쓰기] 함수를 사용하여 클라이언트에 메시지를 보냅니다. 데이터는 반드시 클라이언트가 받아들일 수 있는 형식을 따라야 합니다.
3. [TCP 읽기] 함수를 사용하여 클라이언트로부터의 메시지를 읽습니다.
4. [TCP 연결 닫기] 함수를 이용하여 서버와의 연결을 닫습니다.

TCP 서버의 예제는 다음 VI를 참조하십시오:

- labview\examples\Data Communication\Protocols\TCP\TCP Named Service\TCP Named Service.lvproj  예제 열기  관련 예제 찾기



관련 정보

[TCP 클라이언트 생성하기](#) ( 1533 페이지 참조)

## 도메인

### 도메인 계정 관리자 사용하기(Windows)

도구 >> 보안 >> 도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다. 도메인 계정 관리자를 사용하여 로컬 도메인을 [생성](#) 및 [삭제](#)하고, [로컬 도메인을 반출 및 반입](#)하고, [네트워크 도메인을 추가 및 제거](#)하고, [Lookout 계정을 반입](#)하고, 도메인 [사용자](#)와 [그룹](#)을 관리합니다.

도메인 계정 관리자는 다음의 요소를 포함합니다:

- 내 컴퓨터—로컬 도메인을 디스플레이합니다.
- 네트워크 도메인—관리자가 설정한 네트워크 도메인을 나열합니다.
- 사용자 리스트—선택된 도메인의 사용자를 나열합니다.
- 그룹 리스트—선택된 도메인의 사용자를 나열합니다.



(Windows) 관련된 토픽을 보려면 이 윈도우의 맨 위에 있는 도구 모음에서 왼쪽 그림과 같은 위치 보기 버튼을 클릭합니다. LabVIEW 도움말의 내용 탭에서 이 토픽이 하이라이트되어 사용자가 관련 토픽을 탐색할 수 있게 합니다.

### 네트워크 도메인 추가하기(Windows)

[도메인 계정 관리자](#)를 사용하여 로컬 도메인을 [생성](#) 및 [삭제](#)하고, [로컬 도메인을 반입 및 반출](#)하고, 네트워크 도메인을 [추가 및 제거](#)하고, [Lookout 계정을 반입](#)하고, 도메인 [사용자](#)와 [그룹](#)을 관리합니다.

다음 단계를 따라 도메인 계정 관리자를 사용하여 네트워크 도메인 리스트에 네트워크 도메인을 추가합니다.

1. 도구»보안»도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 도메인»도메인 탐색을 선택하거나 또는 네트워크 도메인 리스트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 도메인 탐색을 선택하여 도메인 탐색 대화 상자를 디스플레이합니다.
3. 관리하려는 네트워크 도메인을 선택합니다.
4. 확인 버튼을 클릭하여 대화 상자를 닫습니다. LabVIEW가 네트워크 도메인을 네트워크 도메인 리스트에 추가합니다.

또한 도메인 찾기 대화 상자에서 호스트 컴퓨터의 이름이나 IP 주소를 입력하여 네트워크 도메인을 찾을 수도 있습니다. 도메인»도메인 찾기를 선택하거나 또는 네트워크 도메인 리스트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 도메인 찾기를 선택하여 도메인 찾기 대화 상자를 디스플레이합니다.

특정 네트워크 도메인을 보이지 않게 하려면, 네트워크 도메인을 클릭한 후 편집»보기에서 제거를 선택하거나 또는 도메인에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 보기에서 제거를 선택합니다. 보기에서 도메인을 제거해도 **도메인이 삭제**되지 않습니다.

## 로컬 도메인 생성하기(Windows)

**도메인 계정 관리자**를 사용하여 로컬 도메인을 생성 및 **삭제**하고, **로컬 도메인을 반출 및 반입**하고, **네트워크 도메인을 추가 및 제거**하고, **Lookout 계정을 반입**하고, 도메인 **사용자**와 **그룹**을 관리합니다.

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 컴퓨터에서 로컬 도메인을 생성합니다.

1. 도구»보안»도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.



### 노트

로컬 도메인이 이미 컴퓨터에 있으면, 도메인 계정 관리자는 자동으로 로컬 도메인을 로드합니다. 새 로컬 도메인을 생성하려면, 먼저 기존의 로컬 도메인을 삭제합니다.

2. 파일»새로 만들기»로컬 도메인을 선택하거나 도메인 리스트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 새 로컬 도메인을 선택하여 도메인 프로퍼티 대화 상자를 디스플레이합니다.
3. 일반 탭을 클릭한 후 도메인 텍스트 박스에서 도메인 이름을 입력합니다.
4. 암호 탭을 클릭한 후 최소 및 최대 암호 길이와 암호 만료 날짜를 설정합니다.



노트 지정한 암호 길이와 만료 날짜는 해당 도메인에 있는 모든 사용자 계정에 대한 기본값이 됩니다.

1. 컨트롤 접근 탭을 클릭하여 머신 또는 IP 주소를 허용 리스트 또는 거부 리스트에 추가합니다. 활성화할 리스트 선택은 도메인이 현재 사용 중인 리스트를 지정합니다.
2. 확인 버튼을 클릭하여 암호 변경 대화 상자를 디스플레이합니다.
3. 관리자 계정 암호를 입력하고 OK 버튼을 클릭합니다. 기본으로, 생성하려는 계정의 사용자 이름은 Administrator입니다.



노트 도메인 계정 관리자는 로컬 도메인을 생성할 때 포함된 사용자 및 그룹에 대한 리스트를 생성합니다. 사용자로는 관리자와 게스트가 포함되어 있으며, 그룹으로는 관리자, 게스트 및 운영자가 포함됩니다.

## 도메인 그룹 생성 및 설정하기(Windows)

도메인 계정 관리자를 사용하여 로컬 도메인을 생성 및 삭제하고, 로컬 도메인을 반출 및 반입하고, 네트워크 도메인을 추가 및 제거하고, Lookout 계정을 반입하고, 도메인 사용자와 그룹을 관리합니다.

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 도메인 그룹을 생성, 설정 및 삭제합니다. 그룹을 생성, 설정 및 삭제하려면 당 로컬 도메인에 로그인해야 합니다.

1. 도구>>보안>>도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 그룹 리스트를 클릭하고 파일>>새 파일>>그룹을 선택하거나 그룹 리스트에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새 파일을 선택하여 그룹 프로퍼티 대화 상자를 디스플레이합니다.
3. 그룹 이름 텍스트 박스에 그룹 이름을 입력합니다.
4. 설명 텍스트 박스에 설명을 입력합니다.
5. 그룹에 속하지 않는 사용자 리스트에서 사용자를 선택하고 오른쪽 화살표 버튼을 클릭하여, 그룹에 사용자를 추가합니다. 그룹에서 사용자를 삭제하려면, 그룹에 속하는 사용자 리스트에서 사용자를 선택하고 왼쪽 화살표 버튼을 클릭합니다.
6. 확인 버튼을 클릭합니다.

그룹 프로퍼티 대화 상자에서 도메인 그룹 프로퍼티를 설정할 수도 있습니다. 그룹을 클릭하고 편집>>프로퍼티를 선택하거나 그룹에서 오른쪽 마우스를 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여 그룹 프로퍼티 대화 상자를 디스플레이합니다.

도메인 그룹을 삭제하려면 그룹을 클릭하고 편집>>삭제를 선택하거나 그룹에서 오른쪽 마우스를 클릭한 후 바로 가기 메뉴에서 삭제를 선택합니다. 내장된 그룹은 삭제할 수 없습니다. 그룹을 삭제하면 해당 이름의 모든 이전 그룹이 무효화되며, 컨트롤 접근 목록의 엔트리도 무효화될 수 있습니다. 같은 이름을 가진 그룹을 다시 생성해도 사용자 계정을 직접 새로운 그룹에 추가해야 합니다.

## 도메인 사용자 생성 및 설정하기(Windows)

도메인 계정 관리자를 사용하여 로컬 도메인을 생성 및 삭제하고, 로컬 도메인을 반출 및 반입하고, 네트워크 도메인을 추가 및 제거하고, Lookout 계정을 반입하고, 도메인 사용자와 그룹을 관리합니다.

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 도메인 사용자를 생성, 설정 및 삭제합니다. 사용자를 생성, 설정 및 삭제하려면 해당 로컬 도메인에 로그인해야 합니다.

1. 도구 >> 보안 >> 도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 사용자 리스트를 클릭하고 파일 >> 새로 만들기 >> 사용자를 선택하거나 그룹 리스트에서 오른쪽 마우스를 클릭한 후 바로 가기 메뉴에서 새로 만들기를 선택하여 사용자 프로퍼티 대화 상자를 디스플레이합니다.
3. 일반 탭을 클릭한 후 사용자 이름 텍스트 박스에서 사용자 이름을 입력합니다.
4. 전체 이름 텍스트 박스에 사용자의 이름을 입력합니다.
5. 설명 텍스트 박스에 설명을 입력합니다.
6. 계정 잠김의 확인란에 확인 표시를 하면 도메인에서 사용자를 제거하지 않은 상태로 사용자 계정을 잠급니다.
7. 암호 변경 버튼을 클릭하여 암호 변경 대화 상자를 디스플레이합니다.
8. 사용자 암호를 입력하고 재입력한 후 확인 버튼을 클릭합니다.
9. 멤버십 탭을 클릭하고, 사용자가 참여할 수 있는 그룹 리스트에서 그룹을 선택하고 오른쪽 화살표 버튼을 클릭하여, 그룹에 새로운 사용자를 추가합니다. 그룹에서 사용자를 제거하려면, 사용자가 속해있는 그룹 리스트에서 그룹을 선택한 후 왼쪽 화살표 버튼을 클릭합니다.
10. 확인 버튼을 클릭합니다.

도메인 사용자 프로퍼티를 설정할 수 도 있습니다. 사용자를 클릭하고 편집 >> 프로퍼티를 선택하거나 그룹에서 오른쪽 마우스를 클릭한 후 바로 가기 메뉴에서 프로퍼티를 선택하여, 사용자 프로퍼티 대화 상자를 디스플레이하고 사용자 셋팅을 설정합니다.

도메인 사용자를 삭제하려면, 사용자를 클릭하고 편집 >> 삭제를 선택하거나 그룹에서 오른쪽 마우스를 클릭한 후 바로 가기 메뉴에서 삭제를 선택합니다. 포함된 사용자는 삭제할 수 없습니다. 사용자를 삭제하면 해당 이름의 모든 이전 사용자가 무효화되며, 접근 컨트롤 리스트의 엔트리도 무효화될 수 있습니다. 같은 이름으로 사용자를 다시 생성해도 내부 사용자 식별 번호는 다릅니다.

## 로컬 도메인 삭제하기(Windows)

도메인 계정 관리자를 사용하여 로컬 도메인을 생성 및 삭제하고, 로컬 도메인을 반출 및 반입하고, 네트워크 도메인을 추가 및 제거하고, Lookout 계정을 반입하고, 도메인 사용자와 그룹을 관리합니다.

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 로컬 도메인을 삭제합니다. 로컬 도메인을 삭제하려면 먼저 로컬 도메인에 로그인해야 합니다.

1. 도구 » 보안 » 도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 로컬 도메인을 클릭하고 도메인 » 로컬 도메인 삭제를 선택하거나, 로컬 도메인에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 로컬 도메인 삭제를 선택합니다.
3. 도메인 삭제를 원하는지 확인 요청을 하는 대화 상자가 나타나면, 예 버튼을 클릭합니다.
4. 관리자 암호 입력 대화 상자에 암호를 입력하고 확인 버튼을 클릭합니다.

도메인에 대한 암호가 없는 경우처럼, 도메인 계정 관리자를 통해 도메인을 제거할 수 없는 경우에는 National Instruments/Shared/Domain 디렉토리에서 사용자가 직접 도메인을 삭제할 수 있습니다. National Instruments 디렉토리에서 파일을 삭제하려면 사용자가 관리자로서 컴퓨터에 로그인해야 합니다.



#### 관련 정보

[도메인 계정 관리자 사용하기\(Windows\)](#) ( 1534 페이지 참조)

[로컬 도메인 생성하기\(Windows\)](#) ( 1535 페이지 참조)

[도메인 계정 관리자를 사용하여 로컬 도메인 반출하기 및 반입하기 \(Windows\)](#) ( 1538 페이지 참조)

[네트워크 도메인 추가하기\(Windows\)](#) ( 1534 페이지 참조)

[Lookout 계정 반입하기\(Windows\)](#) ( 1539 페이지 참조)

[도메인 사용자 생성 및 설정하기\(Windows\)](#) ( 1537 페이지 참조)

[도메인 그룹 생성 및 설정하기\(Windows\)](#) ( 1536 페이지 참조)

[도메인에 로그인하기\(Windows\)](#) ( 1540 페이지 참조)

[도메인 계정 관리자 사용하기\(Windows\)](#) ( 1534 페이지 참조)

## 도메인 계정 관리자를 사용하여 로컬 도메인 반출하기 및 반입하기 (Windows)

**도메인 계정 관리자**를 사용하여 로컬 도메인을 **생성** 및 **삭제**하고, 로컬 도메인을 반출 및 반입하고, **네트워크 도메인**을 추가 및 제거하고, **Lookout 계정**을 반입하고, 도메인 **사용자**와 **그룹**을 관리합니다.

### 로컬 도메인 반출하기

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 로컬 도메인을 반출합니다. 로컬 도메인을 반출하려면 먼저 로컬 도메인에 **로그인**해야 합니다.

1. 도구 » 보안 » 도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.

2. 파일»반입과 반출»로컬 도메인을 파일로 반출을 선택하거나 로컬 도메인에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반입과 반출»로컬 도메인을 파일로 반출을 선택합니다. 파일 대화 상자가 나타납니다.
3. 로컬 도메인을 반출하려는 위치를 탐색합니다.
4. 이 도메인의 파일 이름을 입력한 후 저장 버튼을 클릭합니다.
5. 관리자 암호 입력 대화 상자에 암호를 입력하고 확인 버튼을 클릭합니다.

## 로컬 도메인 반입하기

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 로컬 도메인을 반입합니다.



### 노트

이미 네트워크에 있는 도메인은 반입할 수 없습니다.

1. 도구»보안»도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 파일»반입과 반출»로컬 도메인을 파일로부터 반입을 선택하거나 도메인에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 반입과 반출»로컬 도메인을 파일로부터 반입을 선택합니다. 파일 대화 상자가 나타납니다.
3. 반입하려는 도메인을 선택하고 열기 버튼을 클릭합니다.

노트 이미 로컬 도메인이 컴퓨터에 있다면 대화 상자가 나타나 기존 로컬 도메인의 복사본을 저장하도록 입력 요청합니다. 새 도메인을 반입하기 전에 기존 로컬 도메인의 복사본을 저장하려면 예 버튼을 클릭합니다. 기존 로컬 도메인을 새 도메인으로 대체하려면 아니오 버튼을 클릭합니다. 반입을 취소하려면 취소 버튼을 클릭합니다.

4. 관리자 암호 입력 대화 상자에 암호를 입력하고 확인 버튼을 클릭합니다.

## Lookout 계정 반입하기(Windows)

도메인 계정 관리자를 사용하여 로컬 도메인을 생성 및 삭제하고, 로컬 도메인을 반출 및 반입하고, 네트워크 도메인을 추가 및 제거하고, Lookout 계정을 반입하고, 도메인 사용자와 그룹을 관리합니다.

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 Lookout 계정을 반입합니다.

1. 도구»보안»도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 파일»새 파일»Lookout Security File 반입을 선택하거나 도메인 리스트에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 Lookout Security File 반입을 선택하여 도메인 프로퍼티 대화 상자를 디스플레이합니다.



### 노트

이미 로컬 도메인이 컴퓨터에 있으면, 대화 상자가 나타나 기존 로컬 도메인의 복사본을 저장하도록 입력 요청합니다. Lookout 계정을 반입하기 전에 기존 로컬 도메인의 복사본을 저장하려면 예 버튼을 클릭합니다. 기존 로컬 도메인을 Lookout 계정으로 대체하려면 아니오 버튼을 클릭합니다. 반입을 취소하려면 취소 버튼을 클릭합니다.

3. 일반 탭을 클릭한 후 도메인 텍스트 박스에서 도메인 이름을 입력합니다.
4. 암호 탭을 클릭한 후 최소 및 최대 암호 길이와 암호 만료 날짜를 설정합니다.



노트 지정한 암호 길이와 만료 날짜는 해당 도메인에 있는 모든 사용자 계정에 대한 기본값이 됩니다.

1. 컨트롤 접근 탭을 클릭하여 머신 또는 IP 주소를 허용 리스트 또는 거부 리스트에 추가합니다. 활성화할 리스트 선택은 도메인이 현재 사용 중인 리스트를 지정합니다.
2. 확인 버튼을 클릭하여 암호 변경 대화 상자를 디스플레이합니다.
3. 관리자 계정 암호를 입력하고 OK 버튼을 클릭합니다.
4. 모든 사용자에게 대한 기본 암호를 입력하고 확인 버튼을 클릭합니다.

## 도메인에 로그인하기(Windows)

도메인 계정 관리자를 사용하여 로컬 도메인을 생성 및 삭제하고, 로컬 도메인을 반출 및 반입하고, 네트워크 도메인을 추가 및 제거하고, Lookout 계정을 반입하고, 도메인 사용자와 그룹을 관리합니다.

다음 단계를 완료하여 도메인 계정 관리자를 사용하여 도메인에 로그인합니다.

1. 도구 >> 보안 >> 도메인 계정 관리자를 선택하여 도메인 계정 관리자를 디스플레이합니다.
2. 로그인하려는 도메인을 클릭하고 도메인 >> 로그인을 선택하거나 도메인에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 로그인을 선택하여 도메인에 로그인 대화 상자를 디스플레이합니다.
3. 사용자 이름과 암호를 입력하고 로그인 버튼을 클릭합니다.



### 노트

로그인 사용자는 관리자 또는 관리자 그룹의 구성원이어야 합니다.



도메인에서 로그아웃하려면 도메인을 클릭하고 도메인 >> 로그아웃을 선택하거나, 도메인에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 로그아웃을 선택합니다.



#### 관련 정보

[도메인 계정 관리자 사용하기\(Windows\)](#) ( 1534 페이지 참조)

[로컬 도메인 생성하기\(Windows\)](#) ( 1535 페이지 참조)

[로컬 도메인 삭제하기\(Windows\)](#) ( 1537 페이지 참조)

[도메인 계정 관리자를 사용하여 로컬 도메인 반출하기 및 반입하기 \(Windows\)](#) ( 1538 페이지 참조)

[네트워크 도메인 추가하기\(Windows\)](#) ( 1534 페이지 참조)

[Lookout 계정 반입하기\(Windows\)](#) ( 1539 페이지 참조)

[도메인 사용자 생성 및 설정하기\(Windows\)](#) ( 1537 페이지 참조)

[도메인 그룹 생성 및 설정하기\(Windows\)](#) ( 1536 페이지 참조)

## LabVIEW 측정 개념

### Overview of Virtual Instruments and Measurements

Virtual instrumentation combines hardware and software with industry-standard computer technologies to create user-defined instrumentation solutions. National Instruments specializes in developing plug-in and distributed hardware and driver software for data acquisition (DAQ), IEEE 488 (GPIB), PXI, serial, and industrial communications. The driver software is the application programming interface (API) to the hardware and is consistent across National Instruments application software, such as LabVIEW, LabWindows™/CVI™, and Measurement Studio. These platforms deliver the sophisticated display and analysis capabilities that virtual instrumentation requires.

You can use virtual instrumentation to create a complete and customized system for test, measurement, and industrial automation by combining different hardware and software components. If the system changes, you often can reuse the virtual instrument components without purchasing additional hardware or software.

#### Taking Measurements with Virtual Instruments

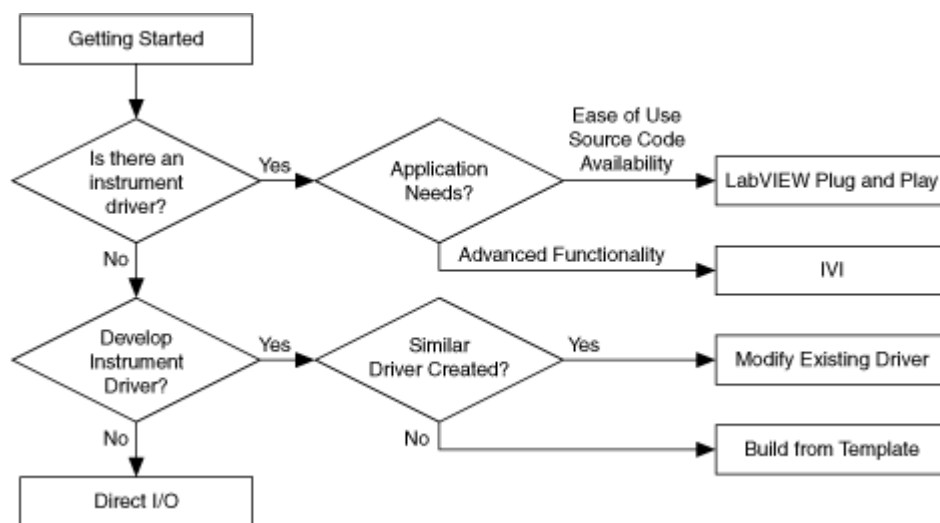
**Different hardware and software components** can make up a virtual instrumentation system. You can use a variety of hardware components to monitor or control a process or to test a device. If you can connect the hardware to the computer and understand how the hardware takes measurements, you can incorporate hardware into a virtual instrumentation system.

# Controlling Instruments

You control instruments by sending commands and data between the instrument and the computer. You can develop various LabVIEW applications to configure and control many [types of instruments](#).

## Choosing Method of Instrument Control

Due to the wide array of instruments and instrument control interfaces, it is important to choose the proper method of instrument control. The following flowchart can help you choose the appropriate method of instrument control.



## Using Instrument Drivers

Use [instrument drivers](#) to control and communicate with instrument hardware in a system. LabVIEW instrument drivers are sets of VIs that simplify instrument control and reduce test program development time by eliminating the need for you to learn the complex, low-level programming commands for each instrument.

When you need easy-to-use drivers with block diagram source code, use [LabVIEW Plug and Play](#) instrument drivers. LabVIEW Plug and Play instrument drivers include error handling, front panels, block diagrams, icons, and help. (Windows) When you need advanced functionality such as instrument interchangeability, simulation, and state-caching, use Interchangeable Virtual Instruments (IVI) instrument drivers. Refer to IVI on the National Instruments Web site for more information about IVI instrument drivers.

# 인스트루먼트 및 DAQ 디바이스

## General-Purpose DAQ

A general-purpose DAQ device is a device that acquires or generates data and can contain multiple channels. You also can use general-purpose DAQ devices to generate analog signals, such as a sine wave, and digital signals, such as a pulse. Typically, you connect these devices directly to the internal bus of a computer through a plug-in slot.

A general-purpose DAQ measurement system is different from other measurement systems because the software installed on the computer performs the actual measurements. The DAQ device only converts the incoming signal into a digital signal the computer can use. This means that the same DAQ device can perform a multitude of measurements simply by changing the software application that reads the data. In addition to acquiring the data, the application for a DAQ measurement system also uses the software that processes the data and displays the results. Although this flexibility allows you to have one hardware device for many types of measurements, you must spend more time developing the different applications for the different types of measurements. LabVIEW includes many acquisition and analysis functions to help you develop different applications.

### How Computers Communicate with DAQ Devices

Before a computer-based measurement system can measure a physical signal, such as temperature, a sensor or transducer must convert the physical signal into an electrical one, such as voltage or current. The plug-in DAQ device is only one system component of the entire measurement system. You cannot always directly connect signals to a plug-in DAQ device. In these cases, you must use signal conditioning accessories to condition the signals before the plug-in DAQ device converts them to digital information. The software controls the DAQ system by acquiring the raw data, analyzing, and presenting the results.

A DAQ system has the following options:

- The plug-in DAQ device resides in the computer. You can plug the device into the PCI slot of a desktop computer or the PCMCIA slot of a laptop computer for a portable DAQ measurement system.
- The DAQ device is external and connects to the computer through an existing port, such as the serial port or Ethernet port, which means you can quickly and easily place measurement nodes near sensors.

### Software and DAQ Devices

The computer receives raw data through the DAQ device. The application software you write presents and manipulates the raw data in a form you can understand. The software also controls the DAQ system by commanding the DAQ device when to acquire data from channels, and from which channels to acquire the data.

Typically, DAQ software includes drivers and application software. Drivers are unique to the device or type of device and include the set of commands the device accepts. Application

software, such as LabVIEW, sends the drivers commands, such as a command to acquire and return a thermocouple reading. The application software also displays and analyzes the acquired data.

NI measurement devices include NI-DAQ driver software and NI Measurements VIs and functions that you can use to configure, acquire data from, and send data to the measurement devices.

## NI-DAQ

NI-DAQ 7. x contains two NI-DAQ drivers—Traditional NI-DAQ (Legacy) and NI-DAQmx—each with its own application programming interface (API), hardware configuration, and software configuration. NI-DAQ 8.0 and later come with only NI-DAQmx, the replacement for Traditional NI-DAQ (Legacy).

- Traditional NI-DAQ (Legacy) is an upgrade to NI-DAQ 6.9. x, the earlier version of NI-DAQ. Traditional NI-DAQ (Legacy) has the same VIs and functions and works the same way as NI-DAQ 6.9. x. You can use Traditional NI-DAQ (Legacy) on the same computer as NI-DAQmx, which you cannot do with NI-DAQ 6.9. x. However, you cannot use Traditional NI-DAQ (Legacy) on Windows Vista.
- NI-DAQmx is the latest NI-DAQ driver with new VIs, functions, and development tools for controlling measurement devices. The advantages of NI-DAQmx over previous versions of NI-DAQ include the DAQ Assistant for configuring channels and measurement tasks for a device; increased performance, including faster single-point analog I/O and multithreading; and a simpler API for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.



### NOTE

(Windows) LabVIEW supports NI-DAQmx and the DAQ Assistant. (Mac OS X)  
LabVIEW supports NI-DAQmx Base but not the DAQ Assistant. (Linux)  
LabVIEW supports NI-DAQmx but not the DAQ Assistant.

Traditional NI-DAQ (Legacy) and NI-DAQmx support different sets of devices. Refer to Data Acquisition (DAQ) Hardware on the National Instruments Web site for the list of supported devices.

## Instrument I/O

Many instruments are external to the computer and do not rely on a computer to take a measurement. By connecting instruments to a computer, you can control and monitor the instruments programmatically and collect data that you can process further or store in files. You can install some instruments in a computer similar to general-purpose DAQ devices. These internal instruments are called modular instruments.

Regardless of how you connect to an instrument, the computer must use a specific protocol to communicate with the instrument. How the computer controls the instrument and acquires data from the instrument depends on the type of the instrument. GPIB, serial port, and PXI are common types of instruments.

Like general-purpose DAQ devices, instruments digitize data, but they have a special purpose or are designed for a specific type of measurement. For standalone instruments, you generally cannot modify the software that processes the data and calculates the result because the software usually is built into the instrument.

Because modular instrumentation uses software running on standard PC technology, you can modify the behavior of these instruments more easily. For example, with some digital multimeter modular instruments, you can program the instruments to acquire a buffer of data at a high rate of speed, much like an oscilloscope.

## How Computers Control Instruments

Computers control instruments by sending commands to the instruments over a bus, such as GPIB, PXI, or RS232. For example, you can send a command to the instrument to measure a signal and then send another command to send the resulting data over the bus back to the computer.

## Instrument Drivers

**Instrument drivers** are not required to use an instrument, but can help save you time as you develop an application.

Using instrument drivers provides the following advantages:

- You can use instrument drivers to quickly build complete systems. Instrument drivers receive, parse, and scale the response strings from instruments into scaled data that you can use in test programs.
- Instrument drivers can reduce software development costs because developers do not need to spend time understanding low-level command syntax needed to program instruments.
- Instrument drivers can help make test programs easier to maintain because instrument drivers contain all the I/O for an instrument within one library, separate from other code, which is easy to upgrade when or if the hardware changes.

## 웨이브폼 컨트롤과 디지털 웨이브폼 컨트롤

웨이브폼 컨트롤, 디지털 웨이브폼 컨트롤, 웨이브폼 그래프, 디지털 웨이브폼 그래프를 사용하여 사용자가 수집하거나 생성하는 웨이브폼과 디지털 웨이브폼을 나타냅니다. LabVIEW는 기본으로 웨이브폼 데이터 타입과 함께 사인파 또는 사각파와 같은 아날로그 웨이브폼을 나타냅니다. 웨이브폼 데이터 타입의 1D 배열은 여러 웨이브폼을 나타냅니다. LabVIEW는 기본으로 디지털 웨이브폼 데이터 타입과 함께 디지털 웨이브폼을 나타냅니다.

웨이브폼과 디지털 웨이브폼 컨트롤은 시작 시간, 델타  $t$ , 웨이브폼 데이터, 속성과 같은 구성요소로 구성되어 있습니다. 웨이브폼 VI와 함수를 사용하여 개별 구성요소에 접근하고 조작합니다.

시작 시간( $t_0$ )은 웨이브폼의 첫번째 측정 포인트와 관련된 타임스탬프입니다. 시작 시간을 사용하여 멀티플롯 웨이브폼 그래프 또는 디지털 웨이브폼 그래프의 플롯을 동기화할 수 있으며 웨이브폼 사이의 지연을 결정할 수 있습니다. 델타  $t(dt)$ 는 신호의 두 포인트 간의 시간 간격(초)입니다. 웨이브폼 데이터와 디지털 웨이브폼 데이터는 웨이브폼을 나타내는 값입니다. 고정 소수점 숫자형 데이터 타입 이외의 모든 숫자형 데이터 타입의 배열은 아날로그 웨이브폼 데이터를 나타낼 수 있습니다. 일반적으로 배열의 데이터 값 개수는 DAQ 디바이스로부터 스캔한 개수에 직접적으로 대응됩니다. 디지털 데이터 타입은 디지털 웨이브폼을 나타내며 테이블에서 디지털 데이터를 디스플레이합니다. 속성에는 신호의 이름 및 신호 수집 디바이스와 같은 신호에 대한 정보가 포함됩니다. NI-DAQ은 자동으로 일부 속성을 설정해 줍니다. [웨이브폼 속성 설정] 함수를 사용하여 속성을 설정하며 [웨이브폼 속성 얻기] 함수를 사용하여 속성을 읽습니다.



#### 노트

웨이브폼 데이터 타입을 **다이나믹 데이터 타입**으로 변환하는 경우, 다이나믹 데이터의 속성을 설정하고 읽을 때 각각 [다이나믹 데이터 속성 설정] 익스프레스 VI와 [다이나믹 데이터 속성 얻기] 익스프레스 VI를 사용할 수 있습니다. 이러한 익스프레스 VI는 사용자가 속성을 설정하고 읽기 전에 자동으로 다이나믹 데이터 타입을 웨이브폼 데이터 타입을 변환시킵니다. 다이나믹 데이터 타입은 익스프레스 VI와 함께 사용할 수 있습니다.

### 웨이브폼 디스플레이하기

프론트패널 윈도우에 웨이브폼 데이터를 나타내려면 웨이브폼 컨트롤 또는 웨이브폼 그래프를 사용하십시오. 디지털 웨이브폼 데이터를 나타내려면 디지털 웨이브폼 컨트롤 또는 디지털 웨이브폼 그래프를 사용합니다.

웨이브폼 컨트롤과 디지털 웨이브폼 컨트롤을 사용하여 웨이브폼의  $t_0$ ,  $dt$ ,  $Y$  성분을 조작하거나 이러한 성분을 인디케이터로 디스플레이합니다. 웨이브폼을 그래프에 연결하면  $t_0$  성분은  $x$ 축의 초기값입니다. 수집된 스캔의 개수와  $dt$  성분은  $x$ 축의 이후의 값을 결정합니다.  $Y$  성분의 데이터 원소는 그래프 플롯의 포인트로 구성됩니다.

사용자가  $dt$  성분과 같은 특정한 성분을 컨트롤 할 수 있도록 하려는 경우, 프론트패널 컨트롤을 생성하여 [웨이브폼 만들기(아날로그 웨이브폼)] 함수의 적당한 성분에 연결합니다.

디지털 웨이브폼 데이터를 디지털 웨이브폼 배열로 정리하고 싶은 경우, [디지털 신호 그룹화] 익스프레스 VI를 사용합니다.

### 웨이브폼 컨트롤 사용하기

여러 VI를 사용하여 웨이브폼을 받고, 작업하고, 반환할 수 있습니다. 또한 웨이브폼 데이터 타입을 직접 그래프, 차트, 숫자 컨트롤, 숫자 배열 컨트롤과 같은 많은 컨트롤에 연결할 수 있습니다.

[AI 웨이브폼 얻기] VI는 단일 입력 채널에서 특정한 시간에 지정된 샘플 속도로 지정된 샘플 개수를 수집하고 웨이브폼을 반환합니다. [디지털 IIR 필터] VI는 웨이브폼의 배열을 받으며 각 웨이브폼의 데이터를 필터합니다. [AI 샘플 채널] VI는 채널에서 단일 샘플을 수집하여 단일 포인트 웨이브폼을 반환합니다. [사인 웨이브폼] VI는 사인 웨이브폼을 생성하며 AO 웨이브폼 생성 VI는 웨이브폼을 디바이스에 보냅니다.

[웨이브폼 구성요소 얻기(아날로그 웨이브폼)] 함수를 사용하여 생성한 웨이브폼의 구성요소를 추출하고 조작합니다. [음수화] 함수는 웨이브폼 데이터를 음수화하고 그 결과를 그래프에 플롯합니다.

## 디지털 웨이브폼 컨트롤 사용하기

디지털 웨이브폼 VI와 함수를 사용하여 디지털 신호의 구성요소를 추출하고 편집하여 디지털 데이터를 조작합니다. 디지털 I/O 팔레트의 NI-DAQmx VI를 사용하여 디지털 신호를 수집하고 보냅니다. 디지털 웨이브폼 팔레트에는 아날로그 데이터를 디지털 신호로 변환하고, 디지털 신호에서 패턴을 검색하고, 디지털 신호를 다른 디지털 신호에 추가하고, 기타 디지털 태스크를 수행하는 VI가 포함되어 있습니다.

일반적인 DAQ 어플리케이션 생성에 대한 더 자세한 정보는 “LabVIEW에서 NI-DAQmx 측정 사용하기” 길라잡이를 참조하십시오.

## Data and String Manipulation Techniques

Communicating with instruments involves sending data to instruments and retrieving data from instruments. You rarely need to format the data sent to or retrieved from the instrument when using instrument drivers because the instrument driver handles the formatting for you. However, you might need to format the data when writing VIs to communicate with instruments.

When you communicate with a message-based instrument, you must format and build the correct command strings for the instrument to perform the appropriate operation or return a response.

No standards exist for register-based instrument communication. Each device operates differently, and the instrument documentation is the best resource for learning how to program the device.

Typically, a command string, or query, is a combination of text and numeric values. Some instruments require text-only command strings, requiring you to **convert the numeric values to text** and append them to the command string. Similarly, to use the data an instrument returns in LabVIEW, you must convert the data to a format a VI, function, or indicator can accept.

(Windows) You can use the Instrument I/O Assistant to send queries to an instrument and to format the data an instrument returns. Place the Instrument I/O Assistant Express VI on the block diagram to access the Instrument I/O Assistant.

You can use the Format Into String function to build command strings to send to an instrument or to append other strings or numeric data types to an initial string.

### Formatting Data Retrieved from an Instrument

Just as the commands strings you send to an instrument contain headers and trailers, most instruments return data with headers and/or trailers. The header usually contains information such as the number of data points returned or the instrument settings. Trailer



information often contains units or other instrument settings at the end of the data string. The instrument documentation should describe what header and trailer information to expect from each data transfer. You must first remove the header and trailer information before you can display or analyze the returned data in LabVIEW.

You can use String Subset functions to parse the header and trailer information in a string and display them in string indicators. You can use the Scan From String function to parse the data from the string.

## Waveform Transfers

Instruments also can return data in other formats, such as ASCII, 1-byte binary, and 2-byte binary formats. The instrument documentation describes available formats and how to convert each one to usable data.

### ASCII Waveforms

If an instrument returns data in ASCII format, you can view the data as a character string. However, if you need to manipulate the data in numeric format or if you need to graph the data, you must convert the string data to numeric data. As an example, consider a waveform composed of 1,024 points and in which each point has a value between 0 and 255. Using ASCII encoding, you need a maximum of 4 bytes to represent each data value (a maximum of 3 bytes for the value and 1 byte for the separator, such as a comma). You need a maximum of 4,096 bytes (4 bytes 1,024) plus any header and trailer bytes to represent the waveform as an ASCII string. You can use the Fract/Exp String To Number function to convert the ASCII string to a numeric array.

### 1-Byte Binary Waveforms

Some instruments do not have the option of sending data in ASCII format or send all waveform data in binary format. Because no standard exists for binary format, refer to the instrument documentation to determine exactly how the instrument stores the data values. One common binary format is 1-byte binary. With this type of data encoding, the instrument converts each piece of data to an 8-bit binary value before sending it.

When you read 1-byte binary data from the bus, the instrument returns the data as a character string. However, the characters do not appear to correspond to the expected data. LabVIEW interprets the binary numbers as ASCII character values, and displays the corresponding characters. For example, if you send the value of 65 as one data value, you read the character A from the bus. For a value of 13, no printable ASCII character exists because 13 corresponds to an invisible carriage return character.

You can display these invisible characters in a string indicator by right-clicking the indicator and selecting '\Codes Display from the shortcut menu. The carriage return character is \r in the string indicator.

To use the numeric data in an ASCII string with the Analysis VI or to display the numeric data in a graph or chart, you must convert the binary string to a numeric array. If the instrument sends a binary string that contains 1,024 1-byte binary encoded values, the waveform requires only 1,024 bytes plus any header information. Using binary encoding, you need only 1 byte to represent each data value, assuming each value is an unsigned 8-



bit integer. You can use String Subset functions and the String To Byte Array function to convert the binary string to an array of integers.



#### 노트

When acquiring binary data, extract the data using the data size rather than searching for the first character of the trailer information because that character might be part of the binary values.

## 2-Byte Binary Waveforms

When data is in 2-byte binary format, it is binary encoded and sent as ASCII characters similar to the 1-byte binary format. However, 16 bits of data, or two ASCII characters, represent each data value. Although this format uses twice as much space as the 1-byte binary format, it is more efficiently packed than ASCII-formatted data.

As an example, consider an oscilloscope that transfers waveform data in binary notation. For this example, assume the waveform consists of 1,024 data points where each value is a 2-byte signed integer. Therefore, the entire waveform requires 2,048 bytes plus a 5-byte header and a 2-byte trailer. After you remove the 5-byte header, use the Unflatten From String function to convert the waveform string to an array of 16-bit integers.

It is important to know the order of the bytes you receive when data is transferred in 2-byte binary format. The 2-byte combination qH has the corresponding integer value of 29,000, but the opposite byte order of Hq has the corresponding integer value of 18,545.



#### 노트

You can configure the byte order for some instruments, and some instruments have a fixed byte order. Refer to the instrument documentation for specific instrument byte information.

If you receive the high byte first, you must reverse the order of the bytes before you convert them to an integer value. Use the Unflatten From String to reverse the byte order.

## Comparing DAQ Devices and Computer-Based Instruments for Data Acquisition

The fundamental task of all measurement systems is the measurement and/or generation of real-world physical signals. Measurement devices help you acquire, analyze, and present the measurements you take.

Through data acquisition, you acquire and convert physical signals, such as voltage, current, pressure, and temperature, into digital formats and transfer them into the computer. Popular methods for acquiring data include plug-in **DAQ** and **instrument** devices, GPIB instruments, PXI (PCI eXtensions for Instrumentation) instruments, and RS232 instruments.

Through data analysis, you transform raw data into meaningful information by using curve fitting, statistical analysis, frequency response, or other numerical operations.

**노트**

Many of these operations are available only in the LabVIEW Full and Professional Development Systems.

Through data presentation, you display data in a graph, thermometer, table, or other visual display.

## Configuring DAQ Devices, Instruments, and Other Devices

You can configure DAQ devices, instruments, and other devices to control the device or instrument using an **I/O control**. After you configure the device or instrument, configure the I/O control to pass Traditional NI-DAQ (Legacy) virtual channels, NI-DAQmx resources (such as tasks, channels, and scales), IVI logical names, VISA resource names, FieldPoint I/O points, or Motion resource names to I/O VIs to communicate with an instrument or a measurement device.

The NI-DAQ driver software provides LabVIEW with a high-level interface to National Instruments DAQ devices. Driver software, such as NI-DMM, NI-SCOPE, NI-FGEN, and NI-SWITCH, provides LabVIEW with high-level interfaces to modular instrumentation. Use driver software such as NI-488.2, NI-VISA, and Interchangeable Virtual Instruments (IVI) to communicate with stand-alone instruments. The LabVIEW VIs call into the driver software, which communicates with the measurement devices. Refer to the DAQ Getting Started Guide for more information about installing NI-DAQmx.

### Configuring Devices and Instruments Using MAX and the DAQ Assistant (Windows)

You can use MAX to configure the following items:

- Traditional NI-DAQ (Legacy) virtual channels
- IVI logical names
- VISA resource names
- FieldPoint I/O points
- Motion resource names

**노트**

You can access MAX on multiple platforms, but LabVIEW supports the DAQ Assistant on Windows only. You cannot use Traditional NI-DAQ (Legacy) on Windows Vista. Refer to the NI-DAQ Readme for more information about platforms supported by NI-DAQ software.

To launch MAX, you can select Tools»Measurement & Automation Explorer or double-click the NI MAX icon on the desktop. You also can create a new Traditional NI-DAQ (Legacy) virtual channel from the front panel by right-clicking the traditional DAQ channel control and selecting New Channel from the shortcut menu.



#### 노트

The Measurement & Automation Explorer option is available only if you install Measurement & Automation Explorer.

You can right-click an IVI logical name, FieldPoint IO point, VISA resource name, or Motion resource name control and select Allow Undefined Names from the shortcut menu to enter logical names you have not configured in MAX or another configuration utility. You also can use the Allow Undefined Names property to programmatically configure the IVI logical name control to accept undefined IVI logical names.

You can right-click the VISA resource name or Motion resource name control and select Allow Multiple Names from the shortcut menu to configure the controls to allow the user to enter more than one resource name.

The DAQ Assistant allows you to create and configure the following for NI-DAQmx.

- Tasks
- Global channels
- Scales

Refer to the Using NI-DAQmx Name Controls book for more information about configuring tasks, channels, and scales for NI-DAQmx.

#### Configuring Devices and Instruments Using Other Utilities (Mac OS X and Linux)

You can use the configuration utilities for the instrument. Refer to the instrument documentation for information about the configuration utilities. You also can configure I/O controls to pass VISA resource names that have not been configured.

## 인스트루먼트 타입

### Types of Instruments

When you use a PC to automate a test system, you are not limited to the type of instrument you can control. You can mix and match instruments from various categories. The most common categories of instrument interfaces are [GPIB](#), [serial](#), and [modular instruments](#). Additional types of instruments include image acquisition, motion control, [USB](#), [Ethernet](#), parallel port, NI-CAN, and other devices.

When you use PCs to control instruments, you need to understand properties of the instrument, such as the communication protocols to use. Refer to the instrument documentation for information about the properties of an instrument.

## GPIB Communication

GPIB, or General Purpose Interface Bus, instruments offer test and manufacturing engineers the widest selection of vendors and instruments for general-purpose to specialized vertical market test applications. GPIB instruments are often used as stand-alone benchtop instruments where measurements are taken by hand. You can automate these measurements by using a PC to control the GPIB instruments.

Use **instrument drivers** and the VISA VIs to control GPIB instruments. When you use the VISA VIs, you need to know the instrument address. Use the VISA resource name control or MAX on Windows to verify communication with the instrument and determine the instrument address. You also need to know the command set for the instrument. Most instruments use SCPI, or Standard Commands for Programmable Instruments: a single, comprehensive command set suitable for all instruments. Refer to the NI Developer Zone for more information about common instrument commands and protocols.

The GPIB protocol categorizes devices as controllers, talkers, or listeners to determine which device has active control of the bus. Each device has a unique GPIB primary address between 0 and 30. The Controller defines the communication links, responds to devices that request service, sends GPIB commands, and passes/receives control of the bus. Controllers instruct Talkers to talk and to place data on the GPIB. You can address only one device at a time to talk. The Controller addresses the Listener to listen and to read data from the GPIB. You can address several devices to listen.

The GPIB is a digital, 24-conductor parallel bus. The GPIB uses an 8-bit parallel, byte-serial, asynchronous data transfer scheme, which means that whole bytes are sequentially handshaked across the bus at a speed that the slowest participant in the transfer determines. Because the unit of data on the GPIB is a byte, the messages transferred are frequently encoded as ASCII character strings. Refer to the GPIB documentation for more information about the hardware specifications of GPIB devices.

You can obtain faster data rates with HS488 devices and controllers. HS488 is an extension to GPIB that most NI controllers support.

Refer to the NI Developer Zone for more information on communicating with GPIB instruments.

## Modular Instruments

NI modular instruments are high-speed hardware products that include the functionality of many traditional benchtop instruments for test and measurement. NI modular instruments include high-speed digitizers, signal generators, high-speed digital I/O devices, high-frequency (RF) devices, digital multimeters (DMMs), programmable power supplies and source measure units (SMUs), and switches. NI modular instrument platforms include PCI, PCI Express, PXI, PXI Express, SCXI, and USB.

You can use the instrument drivers that ship with NI modular instruments products to create custom measurement applications and user interfaces. Each instrument driver includes a palette of LabVIEW VIs that you can use to control a modular instrument from LabVIEW.

Because NI modular instruments combine flexible, user-defined software and scalable, general-purpose hardware components, you can modify measurements as your needs change. For example, you can make modifications to your program or substitute one instrument for another. You also can create custom measurements and define the algorithms used to make those measurements.

Refer to Modular Instruments at [ni.com](http://ni.com) for more information about NI modular instruments.

## Serial Port Communication

Serial communication transmits data between a computer and a peripheral device, such as a programmable instrument or another computer. Serial communication uses a transmitter to send data one bit at a time over a single communication line to a receiver. Use this method when data transfer rates are low or you must transfer data over long distances. Most computers have one or more serial ports, so you do not need any extra hardware other than a cable to connect the instrument to the computer or to connect two computers to each other.

You must specify four parameters for serial communication: the baud rate of the transmission, the number of data bits that encode a character, the sense of the optional parity bit, and the number of stop bits. A character frame packages each transmitted character as a single start bit followed by the data bits.

Baud rate is a measure of how fast data moves between instruments that use serial communication.

Data bits are transmitted upside down and backwards, which means that inverted logic is used and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left and read 1 for negative voltage and 0 for positive voltage.

An optional parity bit follows the data bits in the character frame. The parity bit, if present, also follows inverted logic. This bit is included as a means of error checking. You specify ahead of time for the parity of the transmission to be even or odd. If you choose for the parity to be odd, the parity bit is set in such a way so the number of 1s add up to make an odd number among the data bits and the parity bit.

The last part of a character frame consists of 1, 1.5, or 2 stop bits that are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, begins with a start bit of positive (SPACE) voltage.

### Rate of Data Transfer

You can calculate the maximum transmission rate in characters per second for a given communication setting by dividing the baud rate by the bits per character frame.

## Serial Hardware Overview

The following examples are the most common recommended standards of serial port communication:

- RS232 (ANSI/EIA-232 Standard) is used for many purposes, such as connecting a mouse, printer, or modem. It also is used with industrial instrumentation. Because of improvements in line drivers and cables, applications often increase the performance of RS232 beyond the distance and speed in the standards list. RS232 is limited to point-to-point connections between PC serial ports and devices.
- RS422 (AIA RS422A Standard) uses a differential electrical signal as opposed to the unbalanced (single-ended) signals referenced to ground with RS232. Differential transmission, which uses two lines each to transmit and receive signals, results in greater noise immunity and longer transmission distances as compared to RS232.
- RS485 (EIA-485 Standard) is a variation of RS422 that allows you to connect up to 32 devices to a single port and define the necessary electrical characteristics to ensure adequate signal voltages under maximum load. With this enhanced multidrop capability, you can create networks of devices connected to a single RS485 serial port. The noise immunity and multidrop capability make RS485 an attractive choice in industrial applications that require many distributed devices networked to a PC or other controller for data collection and other operations.

## Connecting a Serial Device

If you have a serial device in a system, you first must obtain the pinout for that device and make sure you have the correct cable to connect the serial device to the computer. Determine if the device is data communications equipment (DCE) or data terminal equipment (DTE) and what settings it uses to communicate—baud rate, data bits, stop bits, parity, or handshaking (flow control).

## More Information About Serial Communication

Refer to the NI Developer Zone for more information on communicating with Serial instruments.

## Ethernet Communication

You can use [NI-VISA](#) to communicate with Ethernet devices, such as NI LXI-based instruments. Ethernet devices are used within measurement systems for tasks such as general networking and remote data storage.

Refer to the NI Developer Zone for information on communicating with Ethernet instruments using NI-VISA.

## USB Communication

You can use NI-VISA to communicate with a USB device. Universal Serial Bus (USB) is a message-based communication bus. Each USB device has its own command set. You can use VISA Read and VISA Write functions to send these commands to an instrument and read the response from an instrument.

Refer to the NI Developer Zone for information on communicating with USB instruments using NI-VISA.



### RELATED INFORMATION

[Using VISA in LabVIEW](#) on page 1565

## 기타 통신 타입

인스트루먼트 드라이버 외에, 인스트루먼트 I/O 어시스턴트를 사용하여 메시지 기반의 인스트루먼트와 통신할 수 있고 그래픽으로 그 응답을 분석할 수 있습니다. 예를 들어, 인스트루먼트 I/O 어시스턴트를 사용하여 그 인스트루먼트와의 통신을 확인하기 위해 쿼리를 인스트루먼트에 보낼 수 있습니다.

인스트루먼트 드라이버를 생성하지 않는 경우, VISA를 사용하여 GPIB, 시리얼, USB, 이더넷, LXI, PXI, 또는 VXI 인스트루먼트를 컨트롤합니다. VISA는 다양한 범위의 인스트루먼트를 컨트롤하는데 사용할 수 있는 표준 API입니다. VISA는 사용하는 인스트루먼트 타입에 따라 적절한 드라이버를 호출하므로 사용자는 인스트루먼트 고유의 통신 프로토콜을 배울 필요가 없습니다.

NI-488.2를 사용하여 어플리케이션 프로그램을 개발하고 디버그합니다. NI-488.2 드라이버는 자동으로 모든 버스 관리를 처리하는 상위 레벨 명령을 가지고 있으므로 GPIB 하드웨어 보드 또는 IEEE 488.2 프로토콜의 프로그래밍 세부사항을 배울 필요가 없습니다. 최대의 유연성과 성능을 위해 하위 레벨 명령도 사용할 수 있습니다.

National Instruments 디바이스 드라이버 DVD에 제공되는 인스트루먼트와 디바이스 드라이버를 사용하여 NI-CAN과 같은 산업 자동화를 위한 NI 모듈러 인스트루먼트와 디바이스를 컨트롤할 수 있습니다.



### 관련 정보

[Using VISA in LabVIEW](#) ( 1565 페이지 참조)

[GPIB Communication](#) ( 1552 페이지 참조)

[Serial Port Communication](#) ( 1553 페이지 참조)

[USB Communication](#) ( 1555 페이지 참조)

[Ethernet Communication](#) ( 1554 페이지 참조)

[Modular Instruments](#) ( 1552 페이지 참조)

# 인스트루먼트 드라이버

## Using Instrument Drivers

An instrument driver is a set of high-level functions that control and communicate with instrument hardware in a system. In LabVIEW, an instrument driver is a set of VIs that communicate with an instrument using LabVIEW built-in VISA I/O functions. Each VI corresponds to a programmatic operation, such as configuring, reading from, writing to, and triggering an instrument. LabVIEW instrument drivers simplify instrument control and reduce test program development time by eliminating the need for you to learn the complex, low-level programming commands for each instrument. You also can **control instruments directly**.

The Instrument Driver Network contains instrument drivers for a variety of programmable instruments that use the GPIB, Ethernet, serial, and other interfaces. You also can search for National Instruments drivers at NI Drivers and Updates on the National Instruments Web site. You can use an instrument driver for a particular instrument as is. However, LabVIEW Plug and Play instrument drivers are distributed with their block diagram source code, so you can customize them for a specific application. You can create instrument control applications and systems by programmatically linking instrument driver VIs on the block diagram. Use the NI Instrument Driver Finder to search for and install LabVIEW Plug and Play instrument drivers without leaving the LabVIEW development environment. Select Tools»Instrumentation»Find Instrument Drivers or Help»Find Instrument Drivers to launch the Instrument Driver Finder.

LabVIEW instrument drivers usually use VISA functions to communicate with instruments. You can use VISA for many different instrument types, such as GPIB, serial, USB, LXI, PXI, and VXI. Once you learn how to use VISA for one type of instrument, you do not have to learn a different way to communicate when you use another type of instrument. You do have to learn about the specific command set for the two instruments, but the method by which you send and receive the commands does not change.



### 노트

You can use the Instrument I/O Assistant to communicate with message-based instruments and graphically parse the response.

NI provides more than 8,000 instrument drivers from more than 275 vendors. Refer to the Instrument Driver Network on the National Instruments Web site for a list of available instrument drivers. You also can search for National Instruments drivers at NI Drivers and Updates on the National Instruments Web site.

You can **search for and install** existing LabVIEW instrument drivers.



(Windows) To view related topics, click the Locate button, shown at left, in the toolbar at the top of this window. The LabVIEW Help highlights this topic in the Contents tab so you can navigate the related topics.



## Types of Instrument Drivers

You can use Plug and Play and Interchangeable Virtual Instrument (IVI) driver types to control instruments in LabVIEW. National Instruments considers drivers Certified or Noncertified depending on if they meet specific requirements.

### Plug and Play Instrument Drivers

A LabVIEW Plug and Play instrument driver is a set of VIs that controls and communicates with a programmable instrument. Each VI corresponds to a programmatic operation, such as configuring, reading from, writing to, or triggering an instrument. LabVIEW Plug and Play instrument drivers include error handling, front panels, block diagrams, icons, and help. Because LabVIEW Plug and Play drivers maintain a common architecture and interface, you can quickly connect to and communicate with an instrument with very little or no code development. Project-style drivers take advantage of the LabVIEW project and library. You can use project-style drivers in the same way as previous LabVIEW Plug and Play drivers.

Use the NI Instrument Driver Finder to search for and install LabVIEW Plug and Play instrument drivers without leaving the LabVIEW development environment. Select **Tools»Instrumentation»Find Instrument Drivers** or **Help»Find Instrument Drivers** to launch the Instrument Driver Finder. If you want to download IVI and contributed drivers, use the Instrument Driver Network. You also can search for National Instruments drivers at **NI Drivers and Updates** on the National Instruments Web site.

### IVI Instrument Drivers

National Instruments IVI drivers are sophisticated, DLL-based drivers developed in LabWindows™/CVI™ that allow for simulation and instrument interchangeability. You do not have to rewrite an application to use it with a similar instrument type. For example, you can write a VI that works with several different brands of oscilloscopes, even if those oscilloscopes use different bus connections. To achieve interchangeability, the IVI Foundation, which develops IVI standards, defines specifications for the following instrument classes: DMM, oscilloscope, arbitrary waveform/function generator, DC power supply, switch, power meter, spectrum analyzer, and RF signal generator. Use IVI drivers for the following additional benefits: instrument state caching for improved performance, multithread safety, and instrument attribute access. Refer to IVI on the National Instruments Web site for more information about IVI instrument drivers.

To convert an instrument driver written in LabWindows/CVI, download the LabVIEW Instrument Driver Import Wizard from the National Instruments Web site.

## Finding and Installing Instrument Drivers

Use the NI Instrument Driver Finder to search for and install LabVIEW Plug and Play instrument drivers without leaving the LabVIEW development environment. Select **Tools»Instrumentation»Find Instrument Drivers** or **Help»Find Instrument Drivers** to launch the Instrument Driver Finder.

If you want to download an IVI driver or browse for noncertified drivers, you can download instrument drivers from the Instrument Driver Network. You also can download National Instruments drivers from NI Drivers and Updates on the National Instruments Web site.

LabVIEW installs instrument driver files to the `labview\instr.lib` directory. Access the instrument driver VIs installed to this directory on the Instrument Drivers palette.



#### NOTE

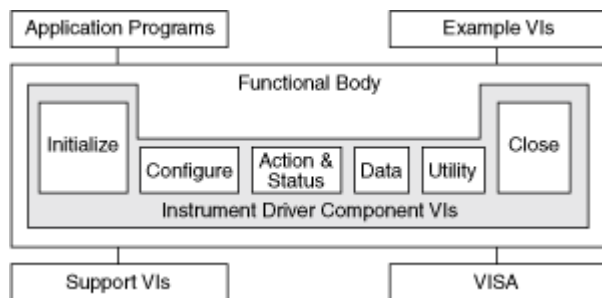
To install, create, or modify instrument drivers, the `labview\instr.lib` directory must have read and write access.

## Accessing Instrument Driver VIs

The Instrument Driver VIs are located on the Instrument Drivers palette.

### Organization of Instrument Driver

The following illustration shows the organization of a typical instrument driver.



### Typical Instrument Driver Components

Most instrument drivers have menu palettes that include the following components.

- The Initialize VI, the first instrument driver VI you call, establishes communication with the instrument. Additionally, it can perform any necessary actions to place the instrument in its default power on state or in another specific state. Generally, you need to call the Initialize VI only once at the beginning of an application.
- The Configuration, Action, Data, and Utility VIs provide numerous functionalities to **control and communicate with** the instrument.
- The Close VI terminates the software connection to the instrument and frees system resources. Generally, you need to call the Close VI only once at the end of an application or when you finish communication with an instrument. Make sure that for each successful call to the Initialize VI, you use a matching Close VI to avoid maintaining unnecessary memory resources.

Use the NI Example Finder to search or browse for project-style **instrument driver examples**. You can find non-project style instrument driver examples in the Instrument Drivers palette and subpalettes.

Many instrument drivers have a VI Tree VI you can use to view the entire instrument driver hierarchy. The VI Tree VI is a non-executable VI that illustrates the functional structure of the instrument driver.

You can **test instrument driver VIs** to interactively determine the appropriate instrument configuration settings.

## Getting Started with Instrument Drivers

To verify communication with an instrument and test a typical programmatic instrument operation, open an **instrument driver example VI**. Look over each of the controls and set them appropriately. Generally, with the exception of the resource name control, the defaults for most controls will be sufficient for the first run. You will need to set the resource name appropriately.

Use the NI Example Finder to search or browse for project-style instrument driver examples. You can find non-project style instrument driver examples in the Instrument Drivers palette and subpalettes.

After running the VI, check to see that the VI returned reasonable data and did not report an error in the error cluster. Instrument driver examples most commonly fail for one of the following reasons:

- NI-VISA is not installed. You install NI-VISA from the National Instruments Device Drivers DVD.
- The resource name is incorrect. The instrument driver example VI requires you to **specify the correct resource** for the instrument. Instruments that are powered on at LabVIEW launch time appear in the VISA resource name pull-down menu. If the instrument does not appear in this list, launch **Measurement & Automation Explorer** to configure the device.
- The instrument driver does not support the exact model you are using. You might need to double-check that the instrument driver supports the instrument model you are using.

After you have verified basic communication with the instrument using the example VI, you might want to customize instrument control for your application. If the application requirements are similar to the example VI, the simplest means of creating a customized VI is to save a copy of the example VI by selecting Save As from the File menu. You can change the default values in the front panel window by selecting Edit»Make Current Values Default. Block diagram changes might include changing the constants wired to the instrument driver subVIs.

## Interactively Testing Instrument Driver VIs

You might want to test **instrument driver VIs** interactively before you include them in an application. Testing instrument driver VIs helps to select appropriate instrument configuration settings. To run the driver VIs from their front panels, you must first run the Initialize VI. For subsequent VIs, you must first select the resource name from the VISA resource name pull-down menu.

After you have selected a resource, you can interactively run the driver VI from the front panel multiple times without resetting the resource selection.

In general, you should run the instrument driver VIs in the order you want to call them in the application. First you run the Initialize VI, followed by one or more configuration VIs. If you are using a trigger for the measurement, you might need to call an action VI to arm the trigger. Calls to data VIs then collect the measured value(s). When you are finished testing the instrument driver component VIs, you should run the Close VI to deallocate resources.

## Creating Instrument Control Applications

Instrument drivers include many component VIs for use in building an instrument control application. Wire the VIs together similar to a typical LabVIEW application.

### Establishing Communication

The Initialize VI, the first instrument driver VI you call, establishes communication with the instrument. Additionally, it can perform any necessary actions to place the instrument in its default power on state or in another specific state. Generally, you need to call the Initialize VI only once at the beginning of an application.



#### 노트

Use the resource name and resource name out terminals to connect the Initialize VI to subsequent VIs that interact with or close the instrument.

### Interacting with the Instrument

The Configuration VIs are a collection of software routines that configure the instrument to perform the operation you want. Numerous Configuration VIs can exist, depending on the particular instrument. After you call these VIs, the instrument is ready to take measurements or to simulate a system.

The Action VIs initiate or terminate test and measurement operations, such as arming the trigger system or generating a stimulus. Action VIs are different from Configuration VIs because they do not change the instrument settings but order the instrument to carry out an action based on its current configuration. The Status VIs obtain the current status of the instrument or the status of pending operations.

The Data VIs transfer data to or from the instrument. Examples include VIs for reading a measured value or waveform and VIs for downloading waveforms or digital patterns to a source instrument.

The Utility VIs perform a variety of operations that are auxiliary to the most often used instrument driver VIs. These VIs include the majority of the instrument driver template VIs, such as reset, self-test, revision, error query, and error message. The Utility VIs might also include other custom instrument driver VIs that perform operations such as calibration or storage and recall of setups.

## Closing the connection

The Close VI terminates the software connection to the instrument and frees system resources. Generally, you need to call the Close VI only once at the end of an application or when you finish communication with an instrument. Make sure that for each successful call to the Initialize VI, you use a matching Close VI to avoid maintaining unnecessary memory resources.

## Error Handling

It is important to perform **error handling** in instrument control applications because there are several potential sources for errors.

## Modifying an Existing Instrument Driver

You might want to modify an existing driver to optimize the code. In general, you should optimize only those VIs that are called repeatedly in a loop. Configuration VIs generally are called only once and have little effect on application speed.

Follow the **Instrument Driver Modification Instructions** to modify an existing instrument driver.

If you need to modify only one or two VIs, consider renaming each VI by selecting File»Save As and selecting the Rename option in the Save As dialog box. After you modify the name, you can modify the block diagram and front panel controls. Most changes to the block diagram should affect the String functions.

If you need to modify several VIs and the existing driver is a project-style driver, consider creating a new driver using the existing driver as the starting point. Select Tools»Instrumentation»Create Instrument Driver Project to launch the Create New Instrument Driver Project wizard and create a new driver using the existing driver as the source.

If you intend to submit the modified driver to National Instruments for certification, refer to the NI Developer Zone for more information on Instrument Driver Certification.

## Creating Instrument Drivers

Before you build an instrument driver, you can [check for existing drivers](#) for the instrument you are using. You can search for existing drivers using the NI Instrument Driver Finder, the manufacturer's Web site, and the Instrument Driver Network on the National Instruments Web site. You also can search for National Instruments drivers at NI Drivers and Updates on the National Instruments Web site.

Use the Create New Instrument Driver Project wizard to create a new instrument driver project. Select Tools»Instrumentation>Create Instrument Driver Project to launch the Create New Instrument Driver Project wizard. After creating the new instrument driver, follow the [Instrument Driver Modification Instructions](#) to complete the driver.

Use the Instrument Driver VI Wizard to create an instrument driver VI for an instrument driver project. Right-click a LabVIEW instrument driver project library `.lvlib` in the [Project Explorer](#) window and select New»Instrument Driver VI from the shortcut menu to launch the Instrument Driver VI Wizard.



### 노트

To install, create, or modify instrument drivers, the `labview\instr.lib` directory must have read and write access.

## Instrument Control Error Handling

It is important to perform error handling in instrument control applications because there are several potential sources for errors. Error handling with instrument driver VIs is similar to error handling with other I/O VIs in LabVIEW. Each instrument driver VI contains an error in input and an error out output for passing error clusters from one VI to another. The error cluster contains a Boolean flag that indicates if an error occurred, an error code number, and a string that contains the location of the VI where the error occurred.

- VISA functions can return errors because VISA or the underlying software or hardware is not properly installed. For example, when communicating with GPIB instruments, NI-488.2 must be installed correctly to use a National Instruments GPIB controller card. Similarly, if the board is not installed or is not correctly configured, the instrument driver VIs return an error.
- VISA functions can return errors if the device you are accessing is not responding to the commands you have sent. The instrument could be incorrectly addressed, malfunctioning, or unable to understand the commands that are being sent.

Use the Simple Error Handler or General Error Handler VIs to detect and report these errors in an application.

# 측정 어플리케이션

## Creating a Typical DAQ Application

Use the **I/O controls** on the I/O and Classic I/O palettes to specify the instrument or device resource you want to communicate with. Use the VIs on the Measurement I/O palette to develop DAQ applications.

Complete the following basic programmatic steps to create a DAQ application.

1. **Configure the measurement hardware.**
2. Create tasks and channels.
3. (Optional) Set timing.
4. (Optional) Set triggering.
5. Read or write data.
6. Clear.

Notice that timing and triggering are optional. Include the timing step if you want to specify hardware timing instead of software timing. If you are using NI-DAQmx, you can use the DAQ Assistant to set timing parameters for a task.



### 노트

(Windows) LabVIEW supports NI-DAQmx and the DAQ Assistant. (Mac OS X) LabVIEW supports NI-DAQmx Base but not the DAQ Assistant. (Linux) LabVIEW supports NI-DAQmx but not the DAQ Assistant.

Use triggering if you want the device to acquire samples only when certain conditions are met. For example, you might want to acquire samples if the input signal goes higher than 4 V. If you are using NI-DAQmx, you can use the DAQ Assistant to configure triggering for a task.

Many NI-DAQmx applications also can include steps to start, stop, and clear the task. For example, for applications that use a counter/timer to count edges or to measure period, use the Start VI to arm the counter.

In NI-DAQmx, LabVIEW clears the task automatically when the VI hierarchy that created the task finishes executing.

Traditional NI-DAQ (Legacy) and NI-DAQmx include VIs for timing, triggering, reading, and writing samples. Use the Data Acquisition VIs and Functions to acquire data from DAQ devices. To use these VIs, you must have the **NI-DAQ driver software and DAQ hardware** installed. After you acquire data, you can use the built-in Signal Processing, Report Generation, and Mathematics VIs and functions to analyze, generate reports, and perform mathematical operations on that data.

You can **use the NI-DAQmx properties** to extend the functionality of the NI-DAQmx VIs.

Many of the Traditional NI-DAQ (Legacy) and NI-DAQmx VIs are polymorphic and can accept or return data of various types, such as scalar values, arrays, or waveforms. You use other polymorphic NI-DAQmx VIs to configure various triggers and methods of sample timing, and to create virtual channels. By default, NI-DAQmx VIs appear with the **polymorphic VI** selector.

## Physical and Virtual Channels

A physical channel is a terminal or pin at which you can measure or generate an analog or digital signal. Every physical channel on a device that supports NI-DAQmx has a unique name.

A virtual channel is a collection of property settings that can include a name, a physical channel, input terminal connections, the type of measurement or generation, and scaling information. In Traditional NI-DAQ (Legacy) and earlier versions, configuring virtual channels is an optional way to record which channels are being used for different measurements, but virtual channels are integral to every NI-DAQmx measurement.

## Tasks

A task in NI-DAQmx is a collection of one or more virtual channels with timing, triggering, and other properties. A task represents a measurement or a generation you want to perform. You can set up and save all of the configuration information in a task and use the task in an application.

In NI-DAQmx, you can configure virtual channels as part of a task or separate from a task.

Complete the following steps to perform a measurement or a generation with a task.

1. Create a task and channels.
2. (Optional) Configure the channel, timing, and triggering properties.
3. Read or write samples.
4. Clear the task.

Repeat steps 2 and 3, if it is appropriate for the application. For example, after reading or writing samples, you can reconfigure the channel, timing, or triggering properties and then read or write additional samples based on this new configuration.

## Creating a Typical FieldPoint Application

Use the **I/O controls** on the I/O and Classic I/O palettes to specify the instrument or device resource with which you want to communicate. Use the VIs on the FieldPoint palette to develop FieldPoint distributed I/O applications. Most FieldPoint applications require only the FP Read VI and the FP Write VI. The FP Read VI returns data from the FieldPoint I/O channel or group of channels the FieldPoint IO Point function represents. The FP Write VI sends data to the FieldPoint I/O channel or group of channels the FieldPoint IO Point



function represents. You also can [use properties](#) to extend the functionality of FieldPoint VIs programmatically.

## Channels versus Items

FieldPoint modules consist of physical I/O points called channels. Items represent the channels or groups of channels. You can create items in Measurement & Automation Explorer (MAX) and use the FieldPoint I/O Point control to access the items in LabVIEW.

## Using the FieldPoint I/O Point Control

Use the FieldPoint I/O Point control to communicate with the FieldPoint items you create in MAX. Place the FieldPoint I/O Point control in the front panel window. Right-click the control and select the items you want to read from or write to from the shortcut menu. If you do not see the FieldPoint items you want, configure the items in MAX.

## Using Properties with Measurement Applications

You can write most measurement applications using only the VIs of the NI-DAQmx, NI-VISA, and IVI Instrument Driver APIs. You also can use properties with these APIs to extend the functionality to include less commonly used features. For example, you can use the VISA Configure Serial Port VI to set several commonly used serial port settings in a VISA Session, including the baud rate. However, if you want to change only the baud rate, you can use a Property Node.

Use the Property Node on the DAQmx palette to configure various low-level settings for NI-DAQmx. Use the Property Node on the VISA Advanced palette for any VISA property. Use the Property Nodes on the Modular Instrument palette and the IVI Instrument Drivers palette for these APIs, respectively.

## LabVIEW의 VISA

## Using VISA in LabVIEW

VISA is a standard I/O API for instrumentation programming.

### Versatility of VISA

VISA can control GPIB, serial, USB, Ethernet, PXI, or VXI instruments, making the appropriate driver calls depending on the type of instrument you use so you do not have to learn instrument-specific communication protocol. It is platform independent, bus independent, and environment independent. In other words, the same API is used regardless of device type, platform, or programming language.

Before you begin [using VISA](#), make sure you [choose the appropriate method of instrument control](#).

## Message-Based Communication

GPIB, serial, USB, Ethernet, and some VXI instruments use message-based communication. You program message-based instruments with high-level ASCII character strings. The instrument has a local processor that parses the command strings and sets the appropriate register bits to perform the operations you want. The Standard Commands for Programmable Instruments (SCPI) standardizes the ASCII command strings used to program compliant instruments. Similar instruments often use similar commands. Instead of learning different command messages for each type of instrument from each manufacturer, you need to learn only one command set. The most common message-based functions are VISA Read, VISA Write, VISA Assert Trigger, VISA Clear, and VISA Read STB.

## Register-Based Communication

PXI and many VXI instruments use register-based communication. You program register-based instruments at a low level using binary information that you write directly to the instrument control registers. Speed is the advantage of this type of communication because the instrument no longer needs to parse the command strings and convert the information to register-level programming. Register-based instruments literally communicate at the level of direct hardware manipulation. The most common register-based functions are VISA In, VISA Out, VISA Move In, and VISA Move Out.



(Windows) To view related topics, click the Locate button, shown at left, in the toolbar at the top of this window. The LabVIEW Help highlights this topic in the Contents tab so you can navigate the related topics.

## Selecting a Resource Name

The front panel of most instrument driver VIs includes a VISA resource name control and a VISA resource name out indicator. These **controls and indicators** pass session information between instrument driver subVIs. The VISA resource name identifies the resource on which the VI operates. It also differentiates between different sessions of the instrument driver.

VISA resource name is a unique identifier reference to a device I/O session. It identifies the device with which the VI communicates and passes all necessary configuration information required to perform the I/O.

VISA resource name out contains the same identifier information as VISA resource name. It passes the reference out of the VI and into other subsequent VIs that access the same instrument. Wiring VISA resource names together establishes data dependencies.

You select the VISA resource name in the Initialize VI from the VISA resource name pull-down menu. Instruments that are powered on and connected to your computer are visible in the VISA resource name pull-down menu. If the instrument does not appear in this list, enter the instrument descriptor using the following format:

Interface Type[board index]::Address::INSTR. Examples resource names: GPIB: GPIB::22::INSTR, GPIB board 0, primary address 22

```
Serial: COM1 TCP/IP Instr: TCPIP::10.0.32.48::INSTR TCP/IP
Socket: TCPIP::10.0.32.45::3800::SOCKET
```

(Windows) Use **MAX** to determine the available resources and instrument addresses and set aliases.

## Verifying VISA Communication

To verify that VISA and the device driver are installed correctly and VISA detects the instrument, first view the resource names that appear in the VISA resource name pull-down menu. If no resource names appear in this list, VISA or the device driver might not be installed correctly. (Windows) Use **Measurement & Automation Explorer** to configure and test instrument communication. If resource names appear in the VISA resource name pull-down menu, but a particular instrument does not appear in the list, you might have a cable connection problem.

If VISA is installed correctly and VISA detects the instrument, but you receive errors when you run the instrument control application, you must identify what sequence of VIs produces the error in LabVIEW. If you are using an instrument driver, try to **run each VI interactively** and verify that each control is set appropriately.

NI I/O Trace is a tool for monitoring instrument I/O communications while your application runs. You can use NI I/O Trace to capture instrument I/O calls and their results while LabVIEW VIs run. Capturing calls and their results can help you debug problems with instrument communications. Select Tools»Instrumentation»NI I/O Trace to launch NI I/O Trace. (Windows) You also can select Start»All Programs»National Instruments»NI I/O Trace to launch NI I/O Trace. (Mac OS X) You also can select Applications»National Instruments»NI I/O Trace and double-click the NI I/O Trace icon to launch NI I/O Trace. (Linux) You also can type `niotrace` in the command line to launch NI I/O Trace.

## Creating a Typical VISA Application

Use the **I/O controls** on the I/O and Classic I/O palettes to specify the instrument or device resource with which you want to communicate.

### VISA Functions and Properties

Use the VIs and functions on the VISA palette to build VIs that control instruments. Use VISA properties to extend the functionality of VISA VIs programmatically. Before you begin using VISA, make sure you **choose the appropriate method of instrument control**.

For most simple instrument applications, you need only two VISA functions, VISA Write and VISA Read. Refer to the Simple Serial VI in the `labview\examples\Instrument IO\Serial` directory for an example of how to use VISA functions.



Open example



Find related examples

Use the VIs in the VISA Advanced VIs and functions palette to build advanced VISA VIs. Refer to the folders in the `labview\examples\Instrument IO\VISA` directory for examples of using advanced VISA VIs.

Refer to the `labview\examples\Instrument IO\GPIB\GPIB.lvproj` for examples of using VISA VIs with GPIB devices.



Open example



Find related examples

## LabVIEW Shared Libraries

### Calling Shared Libraries

A shared library is a file containing executable program modules that any number of different programs can use to perform some function. You can call these program modules, or shared library functions, from LabVIEW by using the Call Library Function Node. Call a shared library function when you want to accomplish any of the following tasks:

- Using functions written in another programming language within LabVIEW
- Performing tasks that LabVIEW cannot perform with its own functions and VIs, such as calling system routines
- Completing tasks that are better suited for text-based languages

#### Shared Libraries on Different Platforms

(Windows) A shared library is called a DLL.

(OS X) A shared library is called a Framework.

(Linux) A shared library is called a Shared Library function.

#### General Approach to Calling Shared Libraries

Calling a shared library function in LabVIEW includes the following actions:

1. You obtain or create a shared library.
2. You configure a Call Library Function Node to call a function within the shared library.
3. LabVIEW calls the function when the Call Library Function Node executes.
4. LabVIEW passes input data from the block diagram to the executable code.
5. LabVIEW returns data from the executable code to the block diagram.

## Limitations on Calling Shared Libraries in LabVIEW

- Shared libraries must use either the stdcall or C calling convention. You can use any language to write shared libraries as long as the shared libraries can be called using one of these two calling conventions.
- Shared libraries execute synchronously. You cannot reset a VI that is running a shared library until execution completes. If you want to write a shared library that performs a long task, be aware that LabVIEW cannot perform other tasks in the same thread while the shared library executes.
- VIs that call shared libraries are platform specific. VIs that call functions within shared libraries execute on different platforms only if you have a corresponding library for each platform. If you move a VI that call a shared library with a Call Library Function Node to another platform, you must update the node to specify the version of the shared library that is compiled for the new platform.

Examples and troubleshooting information help you build and use shared libraries and successfully configure the Call Library Function Node in LabVIEW. The general methods described here for DLLs also apply to other types of shared libraries. Refer to the `labview\examples\dll` directory for examples of using shared libraries.

## Configuring the Call Library Function Node

Use the Call Library Function Node to directly call a 32-bit Windows DLL, a OS X Framework, or a Linux Shared Library function. With this node, you can create an interface in LabVIEW to call existing libraries or new libraries specifically written for use with LabVIEW. National Instruments recommends using the Call Library Function Node to create an interface to external code.



### 노트

Be aware when using the Call Library Function Node or writing code that is called by the Call Library Function Node that LabVIEW reserves Windows messages WM\_USER through WM\_USER+99 for internal use only.

Right-click the Call Library Function Node and select Configure from the shortcut menu to display the Call Library Function dialog box. Use the Call Library Function dialog box to specify the library, function, parameters, return value for the node, calling conventions, and function callbacks on Windows. When you click the OK button in the Call Library Function dialog box, LabVIEW updates the Call Library Function Node according to your settings, displaying the correct number of terminals and setting the terminals to the correct data types.



### 노트

If you want to run applications or shared libraries created with different versions of LabVIEW on the same computer, the computer **must have a version of the LabVIEW Run-Time Engine that is compatible** with each version of LabVIEW used to create the applications or shared libraries.

Configuring Library Name or Path

You must specify either the name of the shared library or the path to the shared library on disk. However, these two techniques produce different consequences when you distribute or relocate your VIs and applications.



Caution Do not specify the location of a system shared library, such as `kernel32.dll`, by path. Because LabVIEW automatically includes by-path shared libraries in build specifications, you might distribute a by-path system shared library to another computer inadvertently. Redistributing some shared libraries, such as `kernel32.dll`, might cause the target computer to crash. Therefore, always specify system shared libraries by name.

If you want to run applications or shared libraries created on different platforms, use the `*` wildcard to make the reference to the shared library platform independent. Use `*` for the file extension and `*` or `**` to the left of the file extension, depending on how you name 32-bit and 64-bit libraries.

The following example illustrates how to use the `*` wildcard.

| Example                   | Translation                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>myshared.*</code>   | LabVIEW replaces the reference with the appropriate file extension to match the platform that is running the Call Library Node, for example, <code>myshared.dll</code> , <code>myshared.so</code> , and <code>myshared.framework</code> .                                                                                                                                    |
| <code>myshared*.*</code>  | LabVIEW replaces the reference with <code>myshared32.*</code> on 32-bit platforms and <code>myshared64.*</code> on 64-bit platforms. You can place the <code>*</code> anywhere to the left of the file extension. For example, <code>my*shared.*</code> can translate to <code>my32shared.*</code> . LabVIEW replaces <code>.*</code> with the appropriate file extension.   |
| <code>myshared**.*</code> | LabVIEW replaces the reference with <code>myshared.*</code> on 32-bit platforms and <code>myshared_64.*</code> on 64-bit platforms. You can place the <code>**</code> anywhere to the left of the file extension. For example, <code>my**shared.*</code> can translate to <code>my_64shared.*</code> . LabVIEW replaces <code>.*</code> with the appropriate file extension. |



노트

Keep the C++ compiler from introducing platform dependence in exported function names through a process called name mangling by using the C++ compiler function export directive, `extern "C"{};` in your header file.

Related Information

Specifying the Location of Shared Libraries on Disk

## Configuring for Multiple Thread Operation

In a multithreaded operating system, you can make multiple calls to a DLL or shared library simultaneously. You can select the thread to execute the library call from the Thread section on the Function tab of the Call Library Function dialog box. The thread options are Run in UI thread and Run in any thread. If you select Run in UI thread, the Call Library Function Node switches from the thread the VI is currently executing in to the user interface thread. If you select Run in any thread, the Call Library Function Node continues in the currently executing thread. By default, all Call Library Function Nodes run in the user interface thread.

Before you configure a Call Library Function Node to run in any thread, make sure that multiple threads can call the function simultaneously. In a shared library, code can be considered thread-safe when:

- It does not store any global data, such as global variables, files on disk, and so on.
- It does not access any hardware. In other words, the code does not contain register-level programming.
- It does not make any calls to any functions, shared libraries, or drivers that are not thread safe.
- It uses semaphores or mutexes to restrict access to global resources.
- It is called by only one non-reentrant VI.



### 노트

All calls to LabVIEW-built shared libraries should specify Run in any thread. If you configure the Call Library Function Node using LabVIEW-built shared libraries and specify Run in UI thread, LabVIEW might hang and require you to restart.

## Related Information

### Multitasking, Multithreading, and Multiprocessing

#### Setting the Calling Convention

Calling conventions define the way to pass information from a piece of code to a function. Use the Calling convention control on the Function tab of the Call Library Function dialog box to select the calling convention for the function. The default calling convention is C. The C calling convention allows variable-length parameter lists.

(Windows) You also can use the standard Windows calling convention, `stdcall`. The number of parameters passed to the function is fixed.

Refer to the documentation for the DLL you want to call for the appropriate calling conventions.



**Caution** Using the incorrect calling convention can cause an irregular shutdown of LabVIEW.

## Configuring Parameters

This section discusses the return value and how to add parameters to the Call Library Function Node.

To configure parameters for the Call Library Function Node, navigate to the Parameters tab of the Call Library Function dialog box. Initially, the Call Library Function Node has no parameters and has a return type of Void.

As you configure parameters, the Function Prototype text box displays the C prototype for the function you are building. This text box is a read-only display.



### 노트

If a type library is found, the parameters are updated to match the parameters found in the type library for the selected function. The order of the parameters must match the prototype of the function found in the library.

The return type for the Call Library Function Node returns to the right terminal of the top terminal. If the return type is Void, the top terminal is unused. Each additional pair of terminals corresponds to a parameter in the Parameters list of the Call Library Function Node. To pass a value to the Call Library Function Node, wire to the left terminal of a terminal pair. To read the value of a parameter after the Call Library Function Node call, wire from the right terminal of a terminal pair. The following illustration shows a Call Library Function Node that has a return type of Void, a string parameter, and a numeric parameter.



## Configuring Return Type

For return type, you can set Type to Void, Numeric, or String. Void is only available for return type and is not available for other parameters. Use Void for the return type if your function does not return any values.

Even if the function you call returns a value, you can use Void for the return type. When the function returns a value and you select Void as the return type, the value returned by the function is ignored.



### 노트

The function you are calling can return a C string pointer. If you want to deallocate the pointer, you must do so explicitly as LabVIEW does not automatically deallocate the C string pointer for you.





Tip If the function you are calling returns a data type not listed, choose a return data type the same data size as the one returned by the function. For example, if the function returns a char data type, use an 8-bit unsigned integer. A call to a function in a DLL cannot return a pointer because there are no pointer types in LabVIEW. However, you can specify the return type as an integer that is the same size as the pointer. LabVIEW then treats the address as a simple integer, and you can pass it to future DLL calls.

## Related Information

### Numeric Return Type

### String Return Type

### C String Pointer

## Adding and Deleting Parameters

To add parameters to the Call Library Function Node, navigate to the Parameters tab of the Call Library Function dialog box. Click the Add a parameter button. To remove a parameter, click the Delete the selected parameter button. To change the order of the parameters, use the Move the selected parameter up one and Move the selected parameter down one buttons to the right of the parameter list.

## Editing Parameters

Select the parameter from the Parameters list to edit the data type or parameter name. You can edit the parameter name to something more descriptive, which makes it easier to distinguish between parameters. The parameter name does not affect the call, but it is propagated to output wires. Also, you can edit all fields in the Current parameter section for the selected parameter.

## Selecting the Parameter Type

Use the Type pull-down menu to indicate the data type of each parameter. You can select from the following parameter types:

- Numeric
- Array
- String
- Waveform
- Digital Waveform
- Digital Data
- ActiveX
- Adapt to Type

- Instance Data Pointer



#### 노트

If the library function expects a data type not included in the previous list, you must convert the LabVIEW data into the type the function expects. Common data types that require conversion include structures, arrays containing pointers to other data, and pointers to callback functions.

After you select an item from the Type pull-down menu, you see more items you can use to indicate details about the parameter and about how to pass the data to the library function. The Call Library Function Node has a number of different items for parameter types because of the variety of data types required by different libraries. Refer to the documentation for the library you call to determine which parameter types to use.

The following sections discuss the different parameter types available from the Type pull-down menu.

(Windows) Refer to the `labview\examples\Connectivity\Libraries and Executables\Libraries and Executables.lvproj` for an example of using data types in shared libraries.



Open example



Find related examples

#### Related Information

### Calling Functions That Expect Other Data Types

#### Numeric

For numeric data types, you must indicate the exact numeric type by using the Data Type pull-down menu. You can choose from the following data types:

- 8-, 16-, 32-, 64-bit, and pointer-sized signed and unsigned integers
- 4-byte, single-precision numbers
- 8-byte, double-precision numbers

If you use pointer-sized integers, the Call Library Function Node adapts to the specific operating system it is being executed on and passes data of the appropriate size to and from the library function. LabVIEW represents the data in 64 bits and, on 32-bit platforms, translates the numeric data types to 32-bit integer types.



#### 노트

You can pass extended-precision numbers and complex numbers by selecting Adapt to Type from the Type pull-down menu. However, standard libraries generally do not use extended-precision numbers and complex numbers.

Use the Pass pull-down menu to indicate whether you want to pass the value or a pointer to the value.

## Array

Use the Data Type pull-down menu to indicate the data type of the array. You can choose from the same data types available for numeric parameters.

Specify the dimensions of the array in Dimensions.

Use the Array Format pull-down menu to make one of the following choices:

- Array Data Pointer —passes a pointer to the array data, allowing the called library to access the array data as the specified data type of the array data.
- Array Handle —passes a pointer to a pointer that points to a four-byte value for each dimension, followed by the data.
- Array Handle Pointer —passes a pointer to an array handle.

Use the Minimum size control to have LabVIEW check at run time that the memory LabVIEW allocated for an array data pointer is at least the Minimum size. To indicate the Minimum size of a 1D array, you can enter a numeric value, or, if you configure an integer parameter in the Parameters list, you can select the parameter from the pull-down menu. This option is available only for array data pointers.



### 노트

If you pass in an array that is smaller than the Minimum size, LabVIEW enlarges the size of the array to the minimum. However, if you pass in an array that is bigger than the minimum, the array retains the larger size.



**Caution** Do not attempt to resize an array with system functions, such as `realloc`. Doing so might crash your system. Instead, use one of the memory manager functions, such as `NumericArrayResize`.

## Related Information

Memory Manager Functions

NumericArrayResize Function

## String

Use the String Format pull-down menu to indicate the string format. You can choose from the following string formats:

- C String Pointer —a string followed by a null character.

- Pascal String Pointer—a string preceded by a length byte.
- String Handle—a pointer to a pointer to four bytes for length information, followed by string data.
- String Handle Pointer—a pointer to an array of string handles.

Select a string format that the library function expects. Most standard libraries expect either a C string or a Pascal string. If the library function you are calling is written for LabVIEW, you might want to use the String Handle format. When configuring a Pascal string pointer, you must wire a value to the string input on the block diagram. That value must be initialized with enough characters to hold any new string that may be written to that Pascal string. When configuring a C string pointer, you have two options:

- Wire a value to the string input that is initialized with enough characters to hold any new string that may be written to that string.
- Specify the string size in the Minimum size pull-down menu on the Parameters tab of the Call Library Function dialog box.

Use the Minimum size control to have LabVIEW check at run time that the memory LabVIEW allocated for a C string pointer is at least the Minimum size. To indicate the Minimum size of a string, you can enter a numeric value, or, if you configure an integer parameter in the Parameters list, you can select the parameter from the pull-down menu. This option is available only for C string pointers.



#### 노트

If you pass in a string that is smaller than the Minimum size, LabVIEW enlarges the size of the string to the minimum. However, if you pass in a string that is bigger than the minimum, the string retains the larger size.



**Caution** Do not attempt to resize an array with system functions, such as `realloc`. Doing so might crash your system. Instead, use one of the memory manager functions, such as `NumericArrayResize`.

## Waveform

When you call a shared library that includes a waveform data type, you do not have to specify a numeric value from the Data Type pull-down menu; the default is 8-byte Double. However, you must specify Dimensions. If the parameter is a single waveform, specify Dimensions as 0. If the parameter is an array of waveforms, specify Dimensions as 1. LabVIEW does not support an array of waveforms greater than one-dimensional.



#### 노트

Pointer-sized signed and unsigned integers are not available in the Data Type pull-down menu for waveforms.

## Digital Waveform

Specify Dimensions as 0 if the parameter is a single digital waveform. Specify Dimensions as 1 if the parameter is an array of digital waveforms. LabVIEW does not support an array of digital waveforms greater than one-dimensional.

## Digital Data

Specify Dimensions as 1 if the Parameter is an array of digital data. Otherwise, specify Dimensions as 0. LabVIEW does not support an array of digital data greater than one-dimensional.



### 노트

You can pass waveforms, digital waveforms, and digital data through shared libraries, but you cannot access the data inside the shared libraries.

## ActiveX

Select one of the following items from the Data Type pull-down menu:

- ActiveX Variant Pointer —passes a pointer to ActiveX data.
- IDispatch\* Pointer —passes a pointer to the IDispatch interface of an ActiveX Automation server.
- IUnknown\* Pointer —passes a pointer to the IUnknown interface of an ActiveX Automation server.

## Related Information

### Using ActiveX with LabVIEW

## Adapt to Type

Use Adapt to Type to pass arbitrary LabVIEW data types to DLLs in the following ways:

- Scalars are passed by reference. A pointer to the scalar is passed to the library.
- Arrays and strings are passed according to the Data Format setting. You can choose from the following Data Format settings:
  - Handles by Value passes the handle to the library. The handle is not NULL.
  - Pointers to Handles passes a pointer to the handle to the library. If the handle is NULL, treat the handle as an empty string or array. To set a value when the handle is NULL, you must allocate a new handle.
  - Array Data Pointer passes a pointer to the first element of the array, allowing the called library to access the array data as the data type of the array data.
  - Interface to Data allows you to inspect LabVIEW data and data type information in C++ code. For more information about this setting, refer to the

ILVDataInterface.idl and ILVTypeInterface.idl files in the labview\cintools directory.

- Clusters are passed by reference.
- Scalar elements in arrays or clusters are in line. For example, a cluster containing a numeric is passed as a pointer to a structure containing a numeric.
- Clusters within arrays are in line.
- Strings and arrays within clusters are referenced by a handle.



#### 노트

When one or more of the parameters of the function you want to call in a DLL are of types that do not exist in LabVIEW, ensure that each parameter is passed to the function in a way that allows the DLL to correctly interpret the data. Create a skeleton .c file from the current configuration of the Call Library Function Node. By viewing the .c file, you can determine whether LabVIEW will pass the data in a manner compatible with the DLL function. You then can make any necessary adjustments.

### Related Information

#### Completing the.c File

### Instance Data Pointer

Use Instance Data Pointer to access data allocated for each instance of the Call Library Function Node. The Instance Data Pointer references a pointer sized allocation that you may use at your own discretion. This allocation is also passed to each of the callback functions on the Callbacks tab.

### Configuring Callbacks

When you configure a Call Library Function Node to call a function, you can use the Callback tab to specify other functions within the same library to call at the following times:

- Reserve time—When the top-level VI that causes the Call Library Function Node to execute begins executing. Specify a Reserve callback when you need to perform initialization tasks before the primary function executes.
- Unreserve time—When the top-level VI that caused the Call Library Function Node to execute stops executing. Specify an Unreserve callback when you save or analyze information or carry out clean-up operations.
- Abort—The specified function executes if the VI that called the primary function aborts.

## Instance Data Pointer Instance Data Pointer



### 노트

You cannot use the Callback tab to pass callback functions as parameters to library functions. The Call Library Function Node cannot directly call library functions that require a callback function as a parameter, but you can [create a wrapper library](#) to implement such calls.

## Calling Functions That Expect Other Data Types

You might encounter a function that expects data in a form that the Call Library Function Node cannot pass. Specifically, the Call Library Function Node does not support structures or arrays containing a pointer to other data or structures containing flat arrays that can be variably sized. You can call a function that expects an unsupported data type in the following ways:

- If the data contains no pointers, you might be able to use the Flatten To String function to create a string containing the binary image of the data required and pass this string as a C string pointer. You will probably want to use the byte order input to Flatten To String to specify that the data be flattened in native byte order.
- Write a library function that accepts the data in the form used by LabVIEW and builds the data structure expected by the other library. This function then can call the other library and retrieve any returned values before returning. Your function will probably accept the data from the block diagram as Adapt to Type, so that any block diagram data type can be passed.
- If the function expects a callback function as a parameter, complete the following steps to call the function from LabVIEW.
  1. Write a wrapper library that defines at least two new functions:
    - A wrapper function that accepts from LabVIEW any data required by the original function. This wrapper function must call the original function, passing the data from LabVIEW as well as a callback function that you define in the same wrapper library.
    - The callback function that you want to pass to the original function.
  2. Call the wrapper function with the Call Library Function Node.

## Related Information

### Flatten To String Function

## Configuring Error Checking

Use error checking to ensure no errors occur if you call a DLL or shared library with the Call Library Function Node.

The Maximum and Default controls on the Error Checking tab of the Call Library Function dialog box allow LabVIEW to recover from unhandled exceptions that occur in the configuration of the Call Library Function Node or during a call to a shared library or DLL.

The Disabled control on the Error Checking tab disables error checking but improves the execution speed of the Call Library Function Node.

If you use the Call Library Function Node to call a shared library generated by LabVIEW, the node automatically calls the LVDLLStatus function embedded in the shared library. For these LabVIEW-generated shared libraries, this function detects certain errors, including the following:

- A VI inside the shared library uses licensed features that are not installed on the target computer.
- A VI inside the shared library uses a Call Library Function Node whose associated shared library is not installed on the target computer.
- The VIs inside the shared library were compiled using the SSE2 optimizations but the target computer does not support SSE2 instructions.

In these cases, the Call Library Function Node returns error 1003.

## Specifying the Location of Shared Libraries on Disk

To call a shared library with the Call Library Function Node, you must specify either the name of the shared library or the path to the shared library on disk. However, these two techniques produce different consequences when you distribute or relocate your VIs and applications. Use the following table to determine which technique addresses your particular use case.

| Technique for Specifying Location | Relevant use cases                                                                                                                                                                                       | Basis of Search Algorithm | Effects on Distribution                                                            |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|------------------------------------------------------------------------------------|
| By name                           | The shared library is in the system search path. Common examples include system shared libraries, such as <code>kernel32.dll</code> , and third party shared libraries designed for componentized reuse. | System search algorithm   | LabVIEW does not include the shared library automatically in build specifications. |
| By path                           | The shared library is private to your VI. Common examples include shared libraries that you wrote specifically for a particular VI.                                                                      | The path you specify      | LabVIEW automatically includes the shared library in build specifications.         |

To specify the location of a shared library, you can enter the name or path in the Library name or path text box of the Call Library Function dialog box. You can also wire the name or path to the path in input of the Call Library Function Node.



**CAUTION**

Do not specify the location of a system shared library, such as `kernel32.dll`, by path. Because LabVIEW automatically includes by-path shared libraries in build specifications, you might distribute a system shared library to another computer inadvertently. Redistributing some shared libraries, such as `kernel32.dll`, might cause the target computer to crash. Therefore, always specify system shared libraries by name.

### Common Problems Loading a Shared Library by Name

If LabVIEW cannot locate a shared library by name, consider the following possible causes:

- To locate shared libraries that you specify by name, LabVIEW uses a modified version of the system search algorithm defined by your operating system. Refer to the KnowledgeBase at [ni.com](http://ni.com) for more information about how LabVIEW modifies the search algorithm on specific platforms.
- Specifying a shared library by name is not the same as specifying a relative path to a shared library within the same directory as the calling VI. To specify the latter, enter `./shared_library_name` as the path to the shared library.

### Common Problems Loading a Shared Library by Path

If LabVIEW cannot locate a shared library by path, consider the following possible causes:

- If you use a relative path to specify the location of the shared library, LabVIEW interprets the path relative to the location of the VI on disk.
- If you use an absolute path to specify the location of the shared library, LabVIEW saves the relative location of the shared library rather than the absolute path. Therefore, if you change the location of the shared library relative to the VI, LabVIEW cannot locate the shared library and **must search for it**.

### How LabVIEW Locates Missing Shared Libraries

If LabVIEW cannot locate a shared library that you specified either by name or by path, LabVIEW searches the VI search path for the missing shared library. If LabVIEW finds the shared library, it updates the path to the shared library within the Call Library Function dialog box.

Searching for missing shared libraries requires extra time, displays an unavoidable search dialog box, and causes unsaved changes in the calling VI that you must save yourself. Therefore, configure VIs with the correct name or path information for shared libraries whenever possible.

**NOTE**

You can configure stand-alone applications to search for shared libraries in specific locations on a target computer by using a `.ini` file. Refer to the KnowledgeBase at [ni.com](http://ni.com) for more information about using a `.ini` file to help a stand-alone application locate a shared library.

## Calling External APIs

You might need to access external APIs from within LabVIEW code. Most often, you access external APIs to obtain functionality that the operating system provides. Normally, you can use the LabVIEW **Call Library Function Node** to accomplish this goal. You must provide the following information to the Call Library Function Node to access external APIs from within LabVIEW code:

- Function name as it appears in the library
- **Function prototype**
- Library or module in which the function resides
- Calling conventions of the function
- Thread-safe status of the function



### CAUTION

Do not **specify the location** of a system shared library, such as `kernel32.dll`, by path. Because LabVIEW automatically includes by-path shared libraries in build specifications, you might distribute a system shared library to another computer inadvertently. Redistributing some shared libraries, such as `kernel32.dll`, might cause the target computer to crash. Therefore, always specify system shared libraries by name.

### Common Pitfalls with the Call Library Function Node

The function reference documentation for any API should provide most of the information that the Call Library Function Node requires. However, you should keep in mind the common errors listed in this section.



### NOTE

Configure the Call Library Function Node to return an error when some of the pitfalls described below occur by selecting the Maximum or Default control on the Error Checking tab of the Call Library Function dialog box.

### Incorrect Function Name

Your library call can fail when the name of the function as it appears in the library is different than is expected. Usually this error occurs due to function name redefinition, or to function name decoration, as in the following examples:

- **Redefinition** —This pitfall appears when an API manufacturer uses a `define` mechanism, such as `#define` directive in ANSI C, to define an abstracted function name to one of many functions present in the library, based on some external condition such as language or debug mode. In such cases, you can look in the header `.h` file for the API to determine whether a `#define` directive redefined the name of a function you want to use.
- **Function Name Decoration** —This pitfall appears when certain functions have their names decorated when they are linked. A typical C compiler tracks name

decoration, and when it looks for a function in a shared library, it recognizes the decorated name. However, because LabVIEW is not a C compiler, it does not recognize decorated names. If you suspect that function name decoration is causing difficulty, inspect the shared library's exported functions. If the function name that appears in the function prototype section has characters such as @ appended to it, the function was decorated when the DLL was built. This is most common with C++ compilers. In LabVIEW, the Function name control in the Call Library Function dialog box is a pull-down menu where you can access a list of all functions within the library you have selected. In addition, most operating systems have a utility you can use to view a library's exports, for example, QuickView on the Windows operating system and the nm command on most Linux systems. If the Function name list contains entries but the function you want to call does not appear in the list, the most likely reason is that the function has not been exported. Refer to the documentation for your compiler for information about how to mark functions for export.

## Data Types

Your library call can fail when you do not use the correct data types. LabVIEW only supports basic numeric data types and C strings. Also, you can select Adapt to Type from the Type pull-down menu of the Call Library Function dialog box and direct LabVIEW to pass its own internal data types for a given parameter. You might encounter the following specific problems:

- **Non-Standard Data Type Definitions** —Frequently, other APIs use non-standard definitions for data types. For example, instead of using `char`, `short`, and `long`, the Windows API uses `BYTE`, `WORD`, and `DWORD`. If an API that you are using makes use of such data types, you need to [find the equivalent basic C data type](#) so that you can properly configure the Call Library Function Node.
- **Structure and Class Data Types** —Some APIs have structure and, in the case of C++, class data types. LabVIEW cannot use these data types. If you need to use a function that has a structure or class as an argument, you should write a shared library wrapper function that takes as inputs the data types that LabVIEW supports and that appropriately packages them before LabVIEW calls the desired function.
- **ActiveX Objects** —If you want to call a shared library that contains [ActiveX](#) objects, use the Automation Open function with the Property Node and the Invoke Node instead of the Call Library Function Node.

(Windows) Refer to the `labview\examples\Connectivity\Libraries and Executables\Libraries and Executables.lvproj` for an example of using data types in shared libraries.



Open example



Find related examples

## Constants

Your library call can fail when your external code uses identifiers in place of constants. Many APIs define identifiers for constants to make the code easier to read. LabVIEW must receive the actual value of the constant rather than the identifier that a particular API uses.

Constants are usually numeric, but they might also be strings or other values. To identify all constants, inspect the header file for the API to [find the definitions](#). The definition might either be in `#define` statements or in enumerations, which use the `enum` keyword.

## Calling Conventions

Your library call can fail when certain operating systems use calling conventions other than the C calling convention and the Standard `__stdcall` calling convention. The calling convention defines how data is passed to a function and how clean up occurs after the function call is complete. The documentation for the API should say which calling convention(s) you must use. The Standard `__stdcall` calling convention is also known as the WINAPI convention, or the Pascal convention.

Use of calling conventions other than the C or Standard calling conventions frequently causes the failure of library calls in LabVIEW because those other calling conventions use an incompatible method for maintaining the stack.

## Example 1: Call a Shared Library That You Built

This section describes the tasks necessary to complete the Array Average VI you started building in [Building a Shared Library to Call from LabVIEW](#) so the VI can call the `avg_num` function in `myshared.dll`.



### NOTE

(Linux) The shared library file has a `.so` extension. You cannot call functions in static libraries `.a` files) on Linux. You can only call shared libraries `.so` files) dynamically.

You must complete the following tasks before the Array Average VI can call the `avg_num` function in `myshared.dll`.

- Complete configuration of the Call Library Function Node.
- Build the front panel window.
- Complete the block diagram.

## Complete Configuration of the Call Library Function Node



### NOTE

If you want to run applications or shared libraries created with different versions of LabVIEW on the same computer, the computer [must have a version of the LabVIEW Run-Time Engine that is compatible](#) with each version of LabVIEW used to create the applications or shared libraries.

Complete the following steps to complete the configuration of the Call Library Function Node.

1. Open the Array Average VI block diagram.
2. Right-click the Call Library Function Node and select Configure from the shortcut menu to open the Call Library Function dialog box.
3. Click the Browse button to the right of the Library Name or Path control to display a file dialog box.
4. Navigate to the location of your `myshared.dll` file.
5. Select `myshared.dll` and click the Open button. The file path to `myshared.dll` appears in the Library Name or Path control. The Library Name or Path control specifies the library name or path for the function you are calling. Enter only the library name if the library is in the search path of the system. Enter the entire path if the library is not in the search path of the system.



#### NOTE

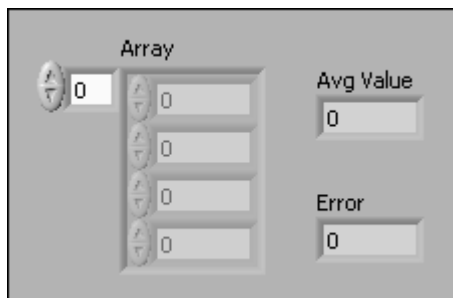
To make the reference platform independent, use the **\* wildcard**. Keep the C++ compiler from introducing platform dependence in exported function names through a process called name mangling by using the C++ compiler function export directive, `extern "C"{};` in your header file.

### Build the Front Panel Window

Complete the following steps to create the front panel of the Array Average VI.

1. Place an array control on the front panel window and label it `Array`.
2. Place a numeric control in the array shell and resize the array to contain four elements.
3. Right-click the Array control and select Representation»Single Precision from the shortcut menu.
4. Place a numeric indicator on the front panel window and label it `Avg Value` to display the result of your averaging calculation.
5. Right-click the Avg Value indicator and select Representation»Single Precision from the shortcut menu.
6. Place a numeric indicator on the front panel and label it `Error` to display any errors that your VI generates.
7. Right-click the Error indicator and select Representation»Long from the shortcut menu.

The following front panel belongs to the Array Average VI.

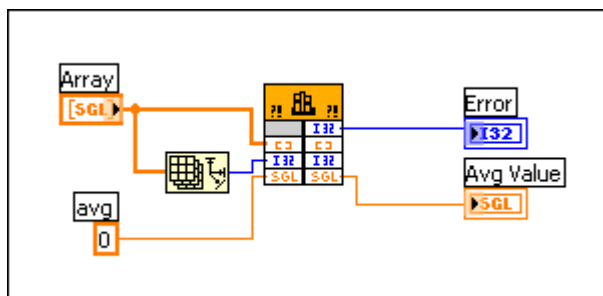


### Complete the Block Diagram

Complete the following steps to complete the block diagram of the Array Average VI.

1. Wire Array to the a input of the Call Library Function Node.
2. Place an Array Size function on the block diagram.
3. Wire Array to the input of the Array Size function.
4. Wire the Array Size function output to the size input of the Call Library Function Node.
5. Right-click the avg input of the Call Library Function Node and select Create»Constant from the shortcut menu. Set the constant value to zero.
6. Wire the avg output of the Call Library Function Node to Avg Value.
7. Wire the error output of the Call Library Function Node to Error.

The following block diagram shows the completed Array Average block diagram.



### Run the VI

On the front panel, enter values in Array and run the VI to calculate the average of those values. Save your work and close the VI.

If your shared library **returns incorrect results or crashes**, verify the data types and wiring to see if you wired the wrong type of information.

## Example 2: Call a Hardware Driver API

You might want to access an API associated with hardware you have purchased. In this example, you call a hypothetical interface card for a databus called X-bus.



### NOTE

You do not need to use the Call Library Function Node to gain access to the APIs of National Instruments hardware. All National Instruments products come with LabVIEW interfaces.

### Configure the Call Library Function Node

The X-bus interface card comes with a software driver for your operating system. The X-bus documentation provides the following standard information:

- A listing of all functions you can use to access the hardware
- Description of the shared library file `xbus.dll` that contains these functions
- Instructions on including a header file `xbus.h`



### NOTE

Although LabVIEW does not permit you to include such header files, you can open header files and extract information about function prototypes and constants.

- (Windows) A statement about the Standard `__stdcall` calling convention that the X-bus library uses

One of the functions you want to use with this hypothetical hardware is `XBusRead16`, which reads a 16-bit integer from a certain address. The documentation describes `XBusRead16` as follows:

```
long XBusRead16(unsigned
long offset,
short*data);
```

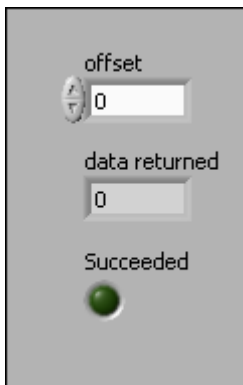
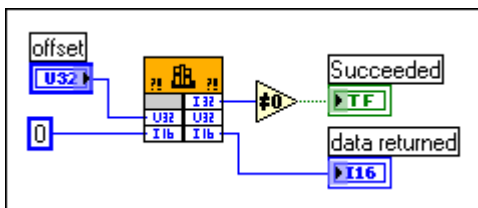
Puts 16 bits from the register at "offset" into the memory location pointed to by "data." Returns 1 if successful, or 0 if it fails.

Given the preceding information from the X-bus documentation, complete the following steps to configure the LabVIEW Call Library Function Node.

1. Open a new, blank VI and save the VI as Read Data.
2. Place a Call Library Function Node on the block diagram.
3. Right-click the Call Library Function Node object and select Configure from the shortcut menu.
4. Make the following settings in the Call Library Function dialog box:
  1. Enter `XbusRead16` in the Function name control.
  2. Select the `stdcall (WINAPI)` control from the Calling Convention field.

3. Navigate to the Parameters page of the Call Library Function dialog box. Select Numeric from the Type pull-down menu for return type.
4. Select Signed 32-bit Integer from the Data Type pull-down menu for return type.
5. Add a parameter and name it offset.
6. Select Numeric from the Type pull-down menu.
7. Select Unsigned 32-bit Integer from the Data Type pull-down menu.
8. Add a parameter and name it data.
9. Select Numeric from the Type pull-down menu.
10. Select Signed 16-bit Integer from the Data Type pull-down menu.
11. Select Pointer to Value from the Pass pull-down menu.
5. Inspect the function prototype that appears in the Function Prototype field. If the prototype you see does not match the definition of the function in the API you are calling, you must change your settings in the Call Library Function dialog box.

Place a numeric control, numeric indicator, and round LED indicator on the front panel window. Label the control and indicators and complete the block diagram as shown in the following block diagram and front panel.



### Example 3: Call the Win32 API

You might want to access the 32-bit Windows platform API (Win32 API). In Win32 environments, various DLLs permit your application to interact with the operating system and with the graphical user interface. Because the API offers thousands of functions, programmers must rely on the documentation for the Microsoft Software Development Kit



(SDK). Microsoft Visual Studio products give you access to the SDK documentation. You also can access this information at the Microsoft Developer Network (MSDN) website.



#### TIP

Instead of using the Windows DLL as described in this example, you could easily create this message box in LabVIEW.

In this example, you call the Windows `MessageBox` function, a function which illustrates several of the typical complexities of the Win32 API. `MessageBox` is a simple SDK function that presents a small dialog box with a message, and has the following prototype:

```
int MessageBoxHWND hWnd, // handle to owner window
LPCTSTR lpText, // text in message box
LPCTSTR lpCaption, // message box title
UINT uType // message box style;
```

Notice the non-standard data types, such as `HWND` and `LPCTSTR`. The Win32 API uses hundreds of data types in the SDK, and very few of them are standard C data types. However, many of the non-standard data types are merely aliases for standard C data types. The API uses the aliases to identify the context of a particular data type. The following table lists the data types in the preceding prototype and the corresponding standard C data types.

| WIN32 SDK Data Type  | Standard C Data Type      |
|----------------------|---------------------------|
| <code>HWND</code>    | <code>int **</code>       |
| <code>LPCTSTR</code> | <code>const char *</code> |
| <code>UINT</code>    | <code>unsigned int</code> |

To properly call the `MessageBox` function in LabVIEW, you need to identify the equivalent LabVIEW data types, which you can usually infer from the C data types. Mapping `LPCTSTR` and `UINT` to LabVIEW is straightforward: `LPCTSTR` is a C String and `UINT` is a `U32`.

Mapping `HWND` is more complex. The previous table lists `HWND` as a double pointer to an integer. However, inspection of the function shows that `MessageBox` uses `HWND` merely as a reference number that identifies the owner of the window. Because of this fact, you do not need to know the integer value for which the `HWND` is a handle. Instead, you need to know the value of the `HWND` variable itself. Because it is a double pointer, and hence a pointer, you can be treat it as a 32-bit unsigned integer, or in LabVIEW terms, a `U32`. Handles such as `HWND` are common in the Win32 SDK. In LabVIEW, you are almost always interested in the handle itself and not the data to which it points. Therefore, you can usually treat handles as `U32`. Handle names always begin with the letter H in the Win32 API.

If the SDK documentation does not make clear what C data type corresponds to a Win32 type, search `windef.h` for the appropriate `#define` or `typedef` statement.

The following table lists the Win32 SDK data types from the previous table and their corresponding LabVIEW data types.

| WIN32 SDK Data Type | LabVIEW Data Type       |
|---------------------|-------------------------|
| HWND                | uInt32                  |
| LPCTSTR             | CStr (C string pointer) |
| UINT                | uInt32                  |

(Windows) Refer to the `labview\examples\Connectivity\Libraries and Executables\Libraries and Executables.lvproj` for a list of more Win32 API data types.

 Open example  Find related examples

Constants

This section presents methods for finding the numerical values of constants in the Win32 API, using `MessageBox` constants as examples. The following table lists selected constants for `MessageBox`.

| Constant             | Description                                                                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MB_ABORTRETRYIGNORE  | An Abort, Retry, Ignore message box.                                                                                                                                                 |
| MB_CANCELTRYCONTINUE | A Cancel, Try Again, Continue message box on Windows 2000. An alternative to MB_ABORTRETRYIGNORE.                                                                                    |
| MB_HELP              | A Help button to add to a message box on Windows XP and later. The system sends a <code>WM_HELP</code> message to the owner whenever the user clicks the Help button or presses<F1>. |
| MB_OK                | A message box with an OK button. This is the default message box.                                                                                                                    |

In Visual Studio, programmers do not use the actual values of constants. In LabVIEW, however, you need to pass the actual numeric value of the constant to the function. You find these values in the header files that come with the SDK. The SDK documentation normally lists the relevant header file at the bottom of the help topic for a given function. For `MessageBox`, the SDK documentation specifies the header file as `winuser.h`.

The header file named in the preceding statement usually declares the constants. Searching through that header file, you should be able to find a `#define` statement or an

enumeration that assigns the constant text a value. `winuser.h` defines values for some of the `MessageBox` constants as follows:

```
#define MB_OK 0x00000000L

#define MB_ABORTRETRYIGNORE 0x00000002L

#define MB_ICONWARNING MB_ICONEXCLAMATION
```

Thus, `MB_OK` has the decimal value 0. `MB_ABORTRETRYIGNORE` has the decimal value 2. `MB_ICONWARNING` is defined as `MB_ICONEXCLAMATION`. Elsewhere in `winuser.h` you find the following statement defining `MB_ICONEXCLAMATION`:

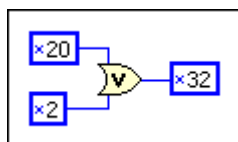
```
#define MB_ICONEXCLAMATION 0x00000030L
```

A hexadecimal value of 30 translates to a decimal value of 48.

Keep in mind that constants in the SDK often are used in bitfields. A bitfield is usually a single integer in which each bit controls a certain property. The `uType` parameter in `MessageBox` is an example of a bitfield. Often, you can combine multiple constants in order to set multiple properties through one parameter. In order to combine these constants, you use a bitwise OR operation `|`. For example, to set the `MessageBox` to have a warning icon and the buttons Abort, Retry, and Ignore, you pass the following value of `uType` to `MessageBox`:

```
MB_ABORTRETRYIGNORE | MB_ICONEXCLAMATION = 0x32
```

In LabVIEW, you combine multiple constants by wiring integer types to the OR operator, as shown in the following block diagram.



### Determining the Proper Library and Function Name

Before you can configure the call to the Win32 API, you must identify the DLL that contains `MessageBox` and the specific name of `MessageBox` within the DLL. Refer to the description of `MessageBox` in the documentation that comes with your SDK or search for `MessageBox Function ()` on the Microsoft Web site. The Requirements section of the documentation on the Microsoft Web site contains the following information:

|                          |                               |
|--------------------------|-------------------------------|
| Minimum supported client | Windows 2000 Professional     |
| Minimum supported server | Windows 2000 Server           |
| Header                   | Winuser.h (include Windows.h) |

|                        |                                              |
|------------------------|----------------------------------------------|
| Library                | User32.lib                                   |
| DLL                    | User32.dll                                   |
| Unicode and ANSI names | MessageBoxW (Unicode) and MessageBoxA (ANSI) |

The DLL line specifies `user32.dll` as the DLL that contains `MessageBox`. The Unicode and ANSI names line specifies the name of the `MessageBox` function for each type of character encoding.

## Unicode Versions and ANSI Versions of Functions

`MessageBox` uses two string arguments. The SDK implements two versions of functions that use string arguments, a Unicode version and an ANSI version. One of the items in the Requirements section of the `MessageBox` documentation says, “Unicode: Implemented as Unicode and ANSI version on Windows.” You can distinguish the two versions in the DLL because each has a W (Unicode) or an A (ANSI) appended to the end of the function name. `winuser.h` contains the following code:

```
#ifdef UNICODE

#define MessageBox MessageBoxW

#else

#define MessageBox MessageBoxA

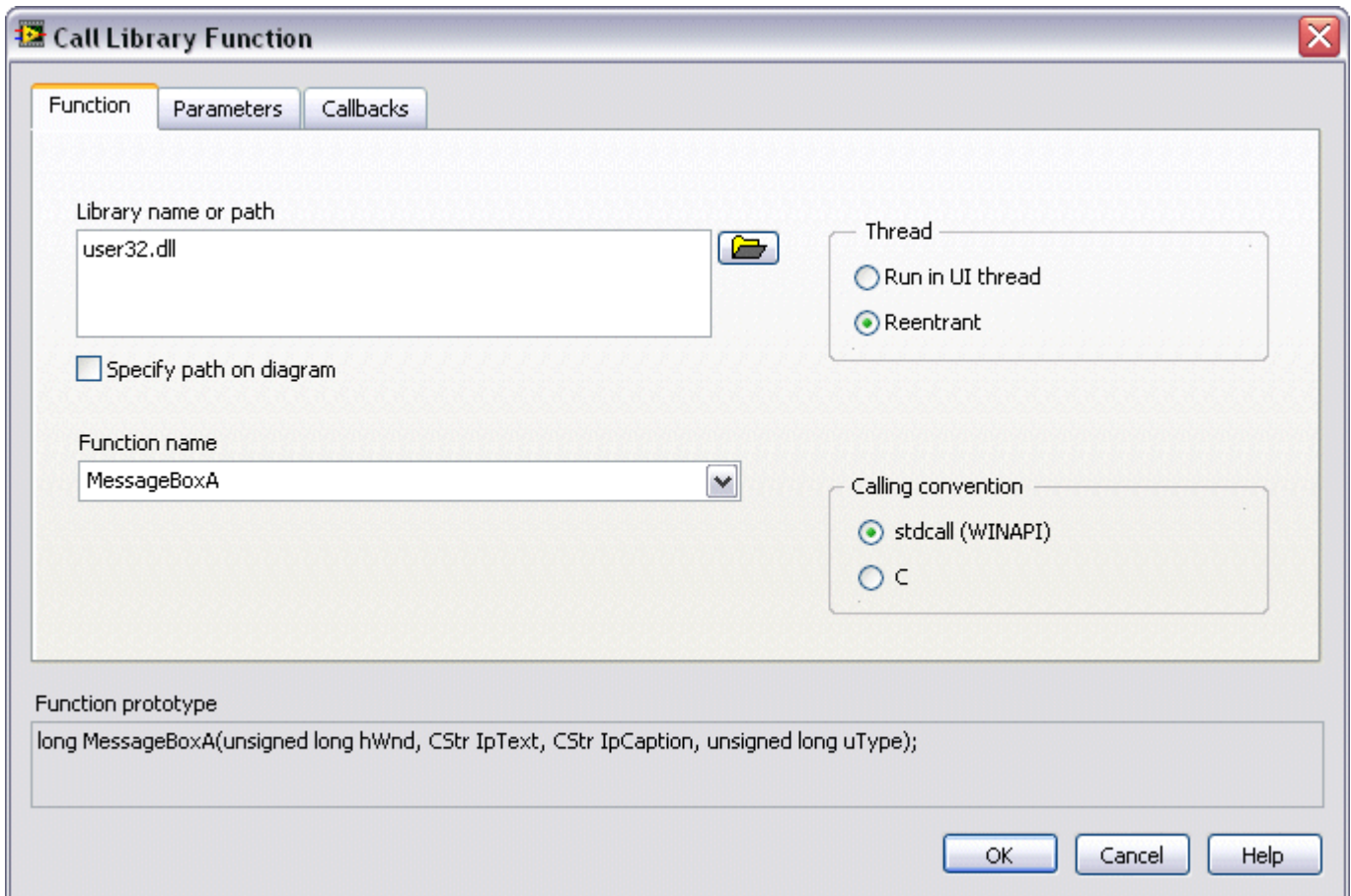
#endif // !UNICODE
```

The preceding code defines `MessageBox` to be either `MessageBoxA` or `MessageBoxW`, depending on whether the application is a Unicode application. In effect, a `MessageBox` function does not exist in `user32.dll`. Instead, `user32.dll` contains a function `MessageBoxA` and a function `MessageBoxW`. In most cases in LabVIEW, you use the ANSI version of the function because the LabVIEW strings are based on ANSI, not Unicode. For this example, you use the `MessageBoxA` function.

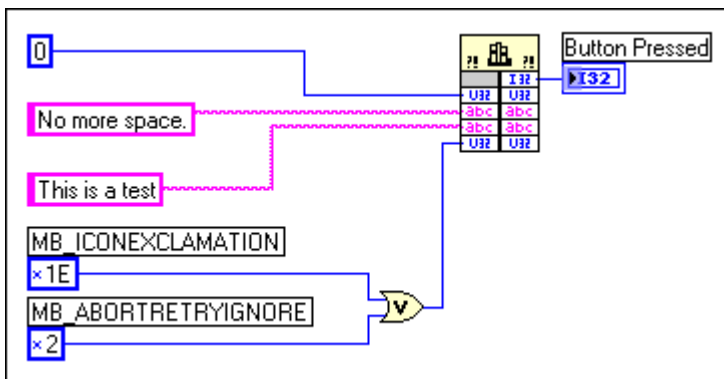
### Configuring a Call to the Win32 API

Now that you are familiar with many aspects of the Win32 API, you can configure a LabVIEW **Call Library Function Node** to call the `MessageBox` function. Remember that you must use the Standard `__stdcall` calling convention in calls to any function in the Windows SDK.

The following front panel shows a **correctly configured instance** of the Call Library Function Node. Make your Call Library Function dialog box match the settings in the graphic.



The following block diagram shows a VI designed to call the Win32 API. Configure your block diagram to match the following block diagram.



The VI generates the following message box.



## Additional Windows Examples of LabVIEW Calls to DLLs

(Windows) The following examples can help you learn more about calling DLLs from LabVIEW:

- The Call DLL VI located in `labview\examples\Connectivity\Libraries and Executables\Libraries and Executables.lvproj` allows you to browse examples of C and C++ external code data types and how they interface with LabVIEW.  Open example  Find related examples
- The Sound Player VI located in the `labview\examples\Graphics and Sound\Sound` directory lets you play Windows .WAV sound files on your computer from LabVIEW, if you have a sound card with Windows sound drivers installed on your system.  Open example  Find related examples
- If you do not have a sound card you can generate a sound in your PC speaker by calling the `MessageBeep` function in `User32 . DLL`. The function prototype is as follows:

```
VOID MessageBeep(UINT uType);
```

- You can programmatically position your cursor anywhere on your monitor using the `SetCursorPos` function in `User32 . DLL`. The function prototype is as follows:

```
BOOL SetCursorPos(INT x, INT y);
```

`x` and `y` are the coordinates you want, referenced from the upper left corner of the screen. The return value is `TRUE` if the function was successful and `FALSE` if it was unsuccessful. Remember that the value returned is type `BOOL`, which is defined in the Win32 API as a 32-bit signed integer with values 0 = `FALSE` and 1 = `TRUE`.

## Debugging Shared Libraries and Calls to Shared Libraries

When you debug your LabVIEW calls to shared libraries, you must be prepared to trace problems in the shared library you are calling and in the implementation of the Call Library Function Node in LabVIEW.

### Troubleshooting the Call Library Function Node

When the LabVIEW calls to shared libraries generate errors, check for the following problems in your use of the Call Library Function Node:

- Verify the Call Library Function Node has the correct name or path for the shared library. Also ensure that you specify the location on disk appropriately for your use case. Some use cases require you to specify the shared library by path, and other use cases require you to specify the shared library by name.

- For LabVIEW-built shared libraries, make sure the computer has a version of the LabVIEW Run-Time Engine that is compatible with each version of LabVIEW used to create the shared libraries.
- If you receive the error message `Function not found in library`, verify the Call Library Function Node has the correct spelling, syntax, and case sensitivity for the function name that you are calling.
- Be sure that the compiler has not decorated the function.
- If you receive an error message that a secondary shared library cannot be found, yet you properly specified the path to the primary shared library in the Call Library Function Node, the primary shared library needs additional functions from one or more other shared libraries. You need to find the other shared libraries and place them in the same directory as the shared library that needs them or in a directory that is in the search path. Refer to the KnowledgeBase at ni.com for more information.
- If your VI crashes, verify that return types and data types of arguments for functions in the Call Library Function Node exactly match the data types your function uses. Also, verify the Call Library Function Node uses the proper calling convention (C or `__stdcall`) using the Error Checking Level tab in the Call Library Function dialog box.
- Verify that all the parameters are defined to be passed by the correct method, such as value or pointer. Also, verify the Call Library Function Node passes arguments to the function in the correct order.
- For arrays or strings of data, always pass a buffer or array that is large enough to hold any results that the function places in the buffer. However, if you are passing them as LabVIEW handles, use LabVIEW Manager functions to resize them under Visual C++ or Xcode compilers.
- If you receive a message, the cause is almost always an error in the code of the shared library, such as writing past the end of the memory allocated for an array. Notice that these kinds of crashes might or might not occur at the time the shared library call actually executes on the block diagram.
- Use the Automation Open function with the Property Node and the Invoke Node instead of the Call Library Function Node if you want to call a shared library that contains ActiveX objects.
- All calls to LabVIEW-built shared libraries should specify Run in any thread. If you configure the Call Library Function Node using LabVIEW-built shared libraries and specify Run in UI thread, LabVIEW might hang and require you to restart.
- Pascal strings do not exceed 255 characters in length.
- Resizing of arrays and strings can take place only when the Call Library Function Node passes a LabVIEW array handle or LabVIEW string handle. Resize arrays and strings using functions exported through `labviewv.lib` to a Visual C++ project, and `liblveports.a` to an Xcode project.
- Remember that C strings are NULL terminated. If your DLL function returns numeric data in a binary string format, for example, through GPIB or the serial port, it might return NULL values as part of the data string.

## Related Information

### Calling Shared Libraries in LabVIEW

## Configure the Call Library Function Node

Call Library Function Node

Call Library Function Dialog Box

LabVIEW Manager Functions

## Specifying the Location of Shared Libraries on Disk

## Using the LabVIEW Run-Time Engine

## Building a Function Prototype

Automation Open Function

Property Node

Invoke Node

## ActiveX Container Overview

Troubleshooting Your Shared Library

Check for the following problems in your shared library when LabVIEW calls to shared libraries generate errors:

- Verify that shared library functions that other applications call appear in the module definition file EXPORTS section, or you include the `_declspec (dllexport)` keyword in the function declaration. Keep the C++ compiler from introducing platform dependence in exported function names through a process called name mangling by using the C++ compiler function export directive, `extern "C"{}` , in your header file, as shown in the following example code:

```
extern "C" {

 /* your function prototypes here */

}
```

- Remember that you need to declare the function with the `_declspec (dllexport)` keyword in the header file and the source code, or define it in the EXPORTS section of the module definition file.
- When you use the `_declspec (dllexport)` keyword and you are also using the `__stdcall` calling convention, you must declare the shared library function name in the EXPORTS section of the module definition `.def` file. In the absence of a `.def` file, `__stdcall` might truncate function names in an unpredictable pattern, so the actual function name is unavailable to applications that call the shared library.



- When a function has not been properly exported, you must recompile the shared library. Never recompile the shared library while the shared library is loaded into memory by another application, for example, by your VI. Before recompiling a shared library, make sure that all applications making use of the shared library are unloaded from memory. This ensures that the shared library itself is not loaded into memory during a recompile. The shared library might fail to rebuild correctly if you forget this point and your compiler does not warn you. However, most compilers warn you when the shared library is in use by an application.
- Try to debug the shared library by using the source level debugger provided with your compiler. Using the debugger of your compiler, you can set breakpoints, step through your code, watch the values of the variables, and so on. Debugging using conventional tools can be extremely beneficial. Refer to the appropriate manual for your compiler for more information about debugging.
- Calling the shared library from another C program is also another way to debug the shared library. By calling the shared library from another C program, you have a means of testing the shared library independent of LabVIEW, thus helping you to identify any problems, sooner.
- When calling a LabVIEW shared library that passes a 2D array, you must first declare the handler variable and initialize the variable to NULL, as shown in the following C code:

```
main

{

/*LabVIEW data handler variable for the array */

TD1Hd1myArray = NULL;

...

/* Call to the LabVIEW shared library function */

DLLFunctionalCall(&myArray);

...

}
```

If you do not initialize the handler variable to NULL, the code produces a `General Protection Fault` when you call the shared library.

- To allow LabVIEW shared library functions to execute without interruption, LabVIEW delays the processing of operating system messages until the end of any calls to shared library functions or until you load a modal window from the shared library. Delaying the operating system messages, such as keyboard messages from the users, is useful in order to avoid calling the same shared library file while a shared library function runs. For example, if the shared library function is called in response to the user pressing a button, the user should not be able to press the button again until the shared library function has completed. When LabVIEW delays the processing of operating system messages, all function calls to the same LabVIEW shared library are put into a queue. That queue will be executed after the original

shared library function call completes. A modal window is a type of window that remains active or remains on top of all other LabVIEW windows until you close the window or open another modal window. If you load a modal window from the shared library, LabVIEW overrides the process delay and processes the messages to allow the modal window to become active. You cannot interact with other windows while a modal window is open. Most dialog boxes in LabVIEW are modal windows. You cannot open a non-modal window from a LabVIEW callback VI nor a shared library while any other process is running. If you want your callback VI or shared library to call a non-modal window, you must do so programmatically by completing the steps outlined in the Callback VIs topic.

- You can choose whether to delay operating system messages in shared libraries that you build. Before building the shared library, navigate to the Advanced page of the Shared Library Properties dialog box and remove the checkmark from the Delay operating system messages in shared library checkbox to process operating system messages while shared library functions run.
- The following scenarios might cause the `No debuggable applications or runtime libraries found` error to appear.
  - The debuggable shared library was not loaded or was unloaded by the application.
  - The configuration `ini` file of the shared library, created by the Application Builder, was not distributed with the shared library.

## Related Information

### Callback VIs

### Shared Library Properties Dialog Box

## Module Definition Files

When you **build a shared library**, you configure LabVIEW to use the C calling convention in the `.c` source file you built with the **Call Library Function Node**. In contrast, you use the `__stdcall` calling convention when you call the Win32 API. When you build a shared library with `__stdcall`, you normally use a module definition `.def` file to export the functions in your shared library. In the absence of a `.def` file, `__stdcall` might truncate function names in an unpredictable pattern, so the actual function name would be unavailable to applications that call the shared library.

You can associate a `.def` file with a shared library. The `.def` file contains the statements for defining a shared library, such as the name of the shared library and the functions that it exports, as shown in the following example code:

```
LIBRARY myshared
```

```
EXPORTS
```

```
avg_num
```

The preceding code example demonstrates the following key requirements for .def files:

- The only mandatory entries in the .def files are the `LIBRARY` statement and the `EXPORT` statement.
- The `LIBRARY` statement must be the first statement in the file.
- The name you specify in the `LIBRARY` statement identifies the library in the import library of the shared library.
- The names you specify in the `EXPORTS` statement identify the functions that the shared library exports.



#### NOTE

Instead of a .def file, many Windows programmers use the `LINK` option in Project Settings of the Visual C++ compiler to obtain equivalent command-line options for most module definition statements.

## Using Arrays and Strings in the Call Library Function Node

This section reviews important concepts regarding array and string data in the [Call Library Function Node](#).

(Windows) Refer to the `labview\examples\Connectivity\Libraries and Executables\Libraries and Executables.lvproj` for an example of using arrays and strings in shared libraries (DLLs).



Open example  Find related examples

### Arrays of Numeric Data

Arrays of numeric data can be comprised of any type of integers or floating point numbers with single (4-byte) or double (8-byte) precision. When you pass an array of data to a shared library function, you can pass the data as an array data pointer, as a LabVIEW array handle, or as a LabVIEW array handle pointer.

Array Data Pointers have the following characteristics whether you pass the Array Data Pointers in the Windows API or in another API:

- You can set the number of dimensions in the array, but you must not include information about the size of the array dimension(s). Instead, you must pass the size of the array dimension(s) information to your shared library in a separate variable.
- Never resize an array or perform operations that might change the length of the array data passed from LabVIEW. Resizing might cause a crash because the pointer sent is not an allocated block but points into the middle of an allocated block.
- To return an array of data, you should allocate an array of sufficient size in LabVIEW, pass the array to your function, and have this array act as the buffer. If the data

takes less space, you can return the correct size as a separate parameter and then, on the calling diagram, use array subset to extract the valid data.

Remember that the Windows API does not use LabVIEW array handles, so with functions that are part of the Windows API you can use only Array Data Pointers.

If you pass the array data as a LabVIEW array handle, you can use LabVIEW Manager functions, such as `NumericArrayResize`, to resize the handle. If you need to resize a handle to an array of strings or clusters, consider alignment on different platforms.

Related Information

LabVIEW Manager Functions

NumericArrayResize Function

Alignment on Different Platforms

String Data

The types of your string pointers must match the types of string pointers that your function uses, or errors occur. The Call Library Function Node offers the following choices:

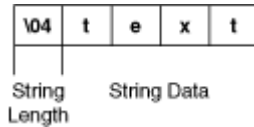
- C String Pointer is a pointer to the string, followed by a NULL character. Most Win32 API functions use this C-style string pointer.
- Pascal String Pointer is a pointer to the string, preceded by a length byte.
- LabVIEW String Handle is a pointer to a pointer to the string, preceded by four bytes of length information.
- LabVIEW String Handle Pointer is a pointer to a handle for a string, preceded by four bytes of length information.

You can think of a string as an array of characters. Assembling the characters in order forms a string. LabVIEW stores a string in a special format in which the first four bytes of the array of characters form a 32-bit signed integer that stores how many characters appear in the string. Thus, a string with *n* characters requires *n* + 4 bytes to store in memory. For example, in the following illustration, the string `text` contains four characters.

|               |     |     |     |             |   |   |   |
|---------------|-----|-----|-----|-------------|---|---|---|
| \00           | \00 | \00 | \04 | t           | e | x | t |
| String Length |     |     |     | String Data |   |   |   |

When LabVIEW stores the string, the first four bytes contain the value 4 as a 32-bit signed number, and each of the following four bytes contains a character of the string. The advantage of this type of string storage is that NULL characters are allowed in the string. Strings are virtually unlimited in length, up to 2<sup>31</sup> characters. This method of string storage is shown in the previous illustration. If you pass a LabVIEW string handle from the Call Library Function Node to the shared library, then you can use the LabVIEW Manager functions, such as `DSSetHandleSize`, to resize the LabVIEW string handle.

The Pascal string format is nearly identical to the LabVIEW string format, but instead of storing the length of the string as a 32-bit signed integer, the string length is stored as an 8-bit unsigned integer. Storing the string length as an 8-bit unsigned integer limits the length of a Pascal-style string to 255 characters. A Pascal string that is  $n$  characters long will require  $n + 1$  bytes of memory to store. The following illustration shows a Pascal string.



C strings are probably the type of strings you will deal with most commonly. The similarities between the C-style string and normal numeric arrays in C becomes much more clear when you see that C strings are declared as `char*`, where `char` is typically an 8-bit unsigned integer. Unlike LabVIEW and Pascal strings, C strings do not contain any information that directly gives the length of the string. Instead, C strings use a special character called the NULL character to indicate the end of the string. NULL is defined to have a value of zero in the ASCII character set. Notice that NULL is the number zero and not the character “0.” Thus, in C, a string that contains  $n$  characters requires  $n + 1$  bytes of memory to store:  $n$  bytes for the characters in the string and one additional byte for the NULL termination character. The following illustration shows how a C-style string is stored in memory.



The advantage of C-style strings is that they are limited in size only by available memory. However, if you are acquiring data from an instrument that returns numeric data as a binary string, as is common with serial or GPIB instruments, values of zero in the string are possible. If you treat the string as a C-style string, your program incorrectly assumes that the end of the string has been reached, when in fact your instrument is returning a numeric value of zero. For binary data that might contain NULL values, consider using an array of 8-bit unsigned integers.

Observe the following guidelines when passing string data to a shared library:

- Never resize a string, concatenate a string, or perform operations that might increase the length of string data passed from LabVIEW if you are using the C or Pascal string pointers.
- If you must return data as a string, allocate a string of the appropriate length in LabVIEW and pass this string into the shared library to act as a buffer. You can specify the string size and allocate the correct amount of memory with the Minimum size control on the Parameters tab of the Call Library Function dialog box. If you pass in a string that is smaller than the Minimum size, LabVIEW enlarges the size of the string to the minimum.
- If you pass a LabVIEW string handle from the Call Library Function Node to the shared library, you can use the LabVIEW Manager functions, such as `DSSetHandleSize`, to resize the LabVIEW string handle. Consider alignment on different platforms before resizing the string handle.

## Related Information

### Resizing the LabVIEW String Handle

Call Library Function Dialog Box

LabVIEW Manager Functions

DSSetHandleSize Function

### Alignment on Different Platforms

#### Determining When to Resize Array and String Handles in Shared Libraries

When a shared library function increases the size of a string or array that you passed by handle, you need to adjust the size of the handle that references the data. You can adjust the handle size using the memory manager routine `DSSetHandleSize`. Because this routine requires you to calculate the size of the new memory space, you must account for platform-dependent requirements about how you align and pad memory. A piece of data is aligned if it falls on a boundary equal to a multiple of its size. For example, a 2-byte value is at a memory address that is a multiple of 2 and a 4-byte value is at a memory address that is a multiple of 4. On Intel processors the maximum alignment is 4.

The following examples highlight alignment differences on various platforms for a three-dimensional array of clusters, with each cluster containing a double-precision, floating-point number and a 1-byte Boolean:

- (Windows) The first twelve bytes describe the size of each dimension of the array (four bytes per dimension). Cluster elements follow, with each cluster element consisting of an 8-byte double-precision floating-point number followed by a 1-byte Boolean. LabVIEW ignores alignment on Windows.
- (64-bit LabVIEW running on Windows) The first twelve bytes describe the size of each dimension of the array (four bytes per dimension) followed by four bytes of padding. Cluster elements follow, with each cluster element consisting of an 8-byte double-precision floating-point number followed by a 1-byte Boolean and seven bytes of padding.
- (OS X and Linux) The first twelve bytes describe the size of each dimension of the array (four bytes per dimension). Cluster elements follow, with each cluster element consisting of an 8-byte double-precision floating-point number followed by a 1-byte Boolean and three bytes of padding.



#### NOTE

Although you must use `DSSetHandleSize` to resize a string or an array of arbitrary data type, you can use the `NumericArrayResize` manager routine to resize arrays of numeric data. `NumericArrayResize` automatically accounts for platform-dependent alignment requirements.

## Related Information

## Memory Manager

DSSetHandleSize Function

NumericArrayResize Function

## How LabVIEW Stores Data in Memory

### How LabVIEW Stores Clusters in Memory

#### Array and String Tip

When you are not passing LabVIEW handles and your shared library function must create an array, resize an array, or resize a string of data, you should break the shared library function into the following two shared library functions:

- Determine the number of elements that the array requires or the length of the string to be returned. Have the first shared library function return the desired size to LabVIEW.
- In LabVIEW, initialize an array or string with default values and pass this array to the second shared library function in your shared library, which actually places the data into the array. If you are working with string-based instrument control, it might be easier to pass an array of 8-bit integers than C strings because of the possibility of having NULL values in the string. When you are passing a LabVIEW array handle or LabVIEW string handle from the Call Library Function Node to your shared library, you can use the LabVIEW Manager functions to resize or create an array or string.

#### Related Information

### Completing the.c File

## Importing Shared Libraries

This book describes the Import Shared Library wizard you can use to create or update a LabVIEW project library of wrapper VIs for functions in a Windows .dll file, an OS X .framework file, or a Linux .so file.



(Windows) To view related topics, click the Locate button, shown at left, in the toolbar at the top of this window. The LabVIEW Help highlights this topic in the Contents tab so you can navigate the related topics.

## Advantages of the Import Shared Library Wizard Compared to the Call Library Function Node

Although you can use either the Call Library Function Node or the Import Shared Library wizard to call shared library functions, the Import Shared Library wizard can accelerate the process of integrating the shared library functions and reduce the potential for errors when calling these libraries.

If you provide the name of a shared library file and a header .h file, the wizard parses the header file and generates a wrapper VI for each function in the shared library, which the wizard saves in a LabVIEW project library .lvlib. The VIs that the wizard generates each contain a Call Library Function Node that accesses the selected function from the shared library.

The Import Shared Library wizard has the following advantages compared to using the Call Library Function Node directly:

- The wizard automatically configures the Call Library Function Nodes generated in the wrapper VIs by the wizard. Optionally, you also can customize the configuration for this Call Library Function Node.
- You can use the wizard to configure calls to multiple functions or entire libraries at once. With the Call Library Function Node, you must configure each node to call a function in a shared library individually.
- Because the wizard generates the wrapper VIs in the shared library you specify, you can easily locate and update those VIs. For example, if you want to add or remove some of the functions you already imported from the shared library, you can run the Import Shared Library wizard again to update the previously generated wrapper VIs. You can run the wizard multiple times on the same shared library file.
- The wizard converts the data types in the shared library to LabVIEW data types. For example, the wizard automatically generates cluster data types for any structure data types in the shared library functions. With the Call Library Function Node, you must manually convert the structure data type.

If you want to individually configure each call to a shared library or if you want to customize configurations for calling shared libraries, use the Call Library Function Node to call libraries directly. Otherwise, use the Import Shared Library wizard to automate this process.



### TIP

If you want to import a shared library file for an instrument driver, you can download the LabVIEW Instrument Driver Import Wizard from [ni.com](http://ni.com).

### Related Information

#### Using the Import Shared Library Wizard

#### Importing Functions from a Shared Library File



## Example: Importing Functions from a Shared Library File (Windows)

### Troubleshooting the Import Shared Library Wizard

#### Call Library Function Node

## Supported Data Types for the Import Shared Library Wizard

The LabVIEW Import Shared Library wizard converts the following shared library data types to the indicated LabVIEW data types.

- numeric and numeric pointer
- string and string pointer
- void and void pointer
- Boolean and Boolean pointer
- cluster and cluster pointer

For return values in the shared library functions, the Input/Output type can only be Output. The Configure VIs and Controls page of the Import Shared Library wizard displays an exclamation point  icon to indicate that LabVIEW cannot process the specified data type. For data types that LabVIEW does not support, [create a custom control and link to it](#). Refer to the [Example: Importing Functions from a Shared Library File](#) topic for more details.

#### Numeric

| C Data Type | Icon                                                                                | I/O Type | LabVIEW Control Type | Pass Type | LabVIEW Representation | Unsigned Representation (unsigned int) |
|-------------|-------------------------------------------------------------------------------------|----------|----------------------|-----------|------------------------|----------------------------------------|
| __int8      |  | Input    | Numeric              | Value     | I8                     | U8                                     |
| __int16     |  | Input    | Numeric              | Value     | I16                    | U16                                    |
| __int32     |  | Input    | Numeric              | Value     | I32                    | U32                                    |
| __int64     |  | Input    | Numeric              | Value     | I64                    | U64                                    |
| char        |  | Input    | Numeric              | Value     | I8                     | U8                                     |
| short       |  | Input    | Numeric              | Value     | I16                    | U16                                    |
| int         |  | Input    | Numeric              | Value     | I32                    | U32                                    |
| long        |  | Input    | Numeric              | Value     | I32                    | U32                                    |
| float       |  | Input    | Numeric              | Value     | SGL                    |                                        |

| C Data Type | Icon                                                                              | I/O Type | LabVIEW Control Type | Pass Type | LabVIEW Representation | Unsigned Representation (unsigned int) |
|-------------|-----------------------------------------------------------------------------------|----------|----------------------|-----------|------------------------|----------------------------------------|
| double      |  | Input    | Numeric              | Value     | DBL                    |                                        |

Numeric Pointer

| C Type         | Icon                                                                                | I/O Type     | I/O Type (const int*) | LabVIEW Control Type | Pass Type | LabVIEW Representation | Unsigned Representation (unsigned int*) |
|----------------|-------------------------------------------------------------------------------------|--------------|-----------------------|----------------------|-----------|------------------------|-----------------------------------------|
| <u>int</u> 8*  |    | Input/Output | Input                 | Numeric              | Pointer   | I8                     | U8                                      |
| <u>int</u> 16* |    | Input/Output | Input                 | Numeric              | Pointer   | I16                    | U16                                     |
| <u>int</u> 32* |    | Input/Output | Input                 | Numeric              | Pointer   | I32                    | U32                                     |
| <u>int</u> 64* |  | Input/Output | Input                 | Numeric              | Pointer   | I64                    | U64                                     |
| short *        |  | Input/Output | Input                 | Numeric              | Pointer   | I16                    | U16                                     |
| int *          |  | Input/Output | Input                 | Numeric              | Pointer   | I32                    | U32                                     |
| long *         |  | Input/Output | Input                 | Numeric              | Pointer   | I32                    | U32                                     |
| float *        |  | Input/Output | Input                 | Numeric              | Pointer   | SGL                    |                                         |
| double *       |  | Input/Output | Input                 | Numeric              | Pointer   | DBL                    |                                         |

String and String Pointer

| C Type        | Icon                                                                                | I/O Type     | LabVIEW Control Type |
|---------------|-------------------------------------------------------------------------------------|--------------|----------------------|
| char*         |  | Input/Output | String               |
| Char []       |  | Input/Output | String               |
| const char *  |  | Input        | String               |
| const char [] |  | Input        | String               |

## Void and Void Pointer

| C Type      | Icon                                                                              | I/O Type                                                                  | LabVIEW Control Type | Pass Type | LabVIEW Representation |
|-------------|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------|----------------------|-----------|------------------------|
| void        |  | void means no control is needed and can appear only as the return control |                      |           |                        |
| void*       |  | Input/Output                                                              | Numeric              | Value     | U32                    |
| const void* |  | Input                                                                     | Numeric              | Value     | U32                    |

## Boolean and Boolean Pointer

| C Type       | Icon                                                                                | I/O Type     | LabVIEW Control Type | Pass Type |
|--------------|-------------------------------------------------------------------------------------|--------------|----------------------|-----------|
| bool         |    | Input        | Boolean              | Value     |
| bool *       |   | Input/Output | Boolean              | Pointer   |
| const bool * |  | Input        | Boolean              | Pointer   |

## Cluster and Cluster Pointer

| C Type         | Icon                                                                                | I/O Type     | LabVIEW Control Type | Pass Type |
|----------------|-------------------------------------------------------------------------------------|--------------|----------------------|-----------|
| struct         |  | Input        | Cluster              | Value     |
| Struct *       |  | Input/Output | Cluster              | Pointer   |
| const struct * |  | Input        | Cluster              | Pointer   |

## Using the Import Shared Library Wizard

Use the Import Shared Library wizard to **create or update a LabVIEW project library of wrapper VIs** for functions in a Windows .dll file, an OS X .framework file, or a Linux .so file.

The Import Shared Library wizard supports most C and C++ header files. The wrapper VIs the wizard creates use the Call Library Function Node, which does not support the C++ `this` pointer or calling methods in C++ classes.

**NOTE**

If you want to import a shared library file for an instrument driver, you can download the LabVIEW Instrument Driver Import Wizard from [ni.com](http://ni.com).

The Import Shared Library wizard parses the header file, lists the functions in the shared library, [converts data types in the shared library to LabVIEW data types](#), and generates a wrapper VI for each function. The wizard saves the VIs in a LabVIEW project library that you can edit and creates an HTML report about the generated library that you can launch when you complete the wizard.

In the wizard, you can specify include paths and preprocessor definitions, configure the individual VIs that wrap each function, and configure memory allocation and error handling. The wizard also creates [custom controls](#) for structure elements in the original functions and adds the controls to the project library. You can use the custom controls to modify all the VIs in the library that contain the corresponding data type.

You can run the wizard multiple times on the same shared library file. If you select the Update VIs for a shared library option on the Specify Create or Update Mode page, the wizard overwrites the previous version of the project library file and the existing VIs within that file. If you choose not to re-import generated VIs within the project library file, the VIs remain unchanged in the directory. The wizard retains the most recent settings for each individual function in a particular shared library. For example, if you have a shared library that contains three functions, you might update only the second function. The next time you run the wizard on that shared library file, it retains the original settings for functions one and three and the new settings for function two.

Select Tools»Import»Shared Library to launch the Import Shared Library wizard. Follow the prompts to create wrapper VIs for shared library files. You must provide the name of a shared library file and a header .h file for the wizard to parse.

Refer to the [Importing Functions from a Shared Library File](#) topic for step-by-step instructions for importing a shared library and creating wrapper VIs.

## LabVIEW Manager Functions in Shared Libraries

### Using LabVIEW Manager Functions in Shared Libraries

LabVIEW has a suite of functions that you can call from shared libraries. This suite of functions performs user-specified routines using the appropriate instructions for a given platform. These routines, which manage the functions of a specific operating system, are grouped into the following categories:

- Memory manager
- File manager
- Support manager

## Portability of LabVIEW Manager Functions

External code written using the LabVIEW managers is portable, that is, you can compile it without modification on any platform that supports LabVIEW. This portability has the following advantages:

- The LabVIEW application is built on top of the manager functions. Except for the manager functions, the LabVIEW source code is identical across platforms.
- The analysis VIs rely mainly on manager functions. Therefore, their source code is the same for all platforms.



### NOTE

When you call the LabVIEW manager functions from a shared library, use `#include extcode.h` in any files that use manager functions and link to `labviewv.lib`. Set the structure alignment of the compiler to 1 byte.

To achieve platform independence, data types should not depend on the peculiarities of various compilers. For example, the C language does not define the size of an integer. Without an explicit definition of the size of each data type, you have difficulty creating code that works identically across multiple compilers. LabVIEW managers use data types that explicitly indicate the size of the data type. For example, if a routine requires a 4-byte integer as a parameter, you define the parameter as an `int32`. The managers define data types in terms of the fundamental data types for each compiler. Thus, on one compiler, the managers might define an `int32` as an `int`, while on another compiler, the managers might define an `int32` as a `long int`. When you write external code, use the manager data types instead of the host computer data types so your code is more portable and has fewer errors.

## Pointers as Parameters

Some manager functions have a parameter that is a pointer. These parameter type descriptions are identified by a trailing asterisk, such as the `ph` parameter of the `DSCopyHandle` allocating and releasing function, or are type defined as such, such as the `name` parameter of the `FNamePtr` function. In most cases, the manager function writes a value to pre-allocated memory. In some cases, such as `FStrFitsPath` or `GetALong`, the function reads a value from the memory location, so you do not have to pre-allocate memory for a return value.

The following functions have parameters that return a value for which you must pre-allocate memory.

|                           |                               |
|---------------------------|-------------------------------|
| <code>DSMemStats</code>   | <code>FNamePtr</code>         |
| <code>DSCopyHandle</code> | <code>FNewRefNum</code>       |
| <code>DateToSecs</code>   | <code>FPathToArr</code>       |
| <code>DSMemStats</code>   | <code>FPathToAZString</code>  |
| <code>FCreate</code>      | <code>FPathToDSSString</code> |

|                  |                    |
|------------------|--------------------|
| FCreateAlways    | FPathToPath        |
| FFlattenPath     | FRefNumToFD        |
| FGetAccessRights | FStringToPath      |
| FGetEOF          | FTextToPath        |
| FGetInfo         | FUnflattenPath     |
| FGetPathType     | GetAlong           |
| FMOpen           | SetAlong           |
| FMRead           | RandomGen          |
| FMTell           | SecsToDate         |
| FMWrite          | NumericArrayResize |

You must allocate space for this return value. The following examples illustrate incorrect and correct ways to call one of these functions from within a generic function foo.

Incorrect example:

```
foo(Path path) {

 PStr p; /* an uninitialized pointer */ File *fd; /* an
 uninitialized pointer */

 MgErr err;

 err = FNamePtr(path, p);

 err = FMOpen(fd, path, openReadOnly denyWriteOnly);

}
```

In the incorrect example, p is a pointer to a Pascal string, but the pointer is not initialized to point to any allocated buffer. FNamePtr expects its caller to pass a pointer to an allocated space and writes the name of the file referred to by path into that space. Even if the pointer does not point to a valid place, FNamePtr writes its results there, with unpredictable consequences. Similarly, FMOpen writes its results to the space to which fd points, which is not a valid place because fd is uninitialized.

Correct example:

```
foo(Path path) {

 Str255 buf; /* allocated buffer of 256 chars */
```

```

File fd;

MgErr err;

err = FNamePtr(path, buf);

err = FMOpen(&fd, path, openReadOnly, denyWriteOnly);

}

```

In the correct example, `buf` contains space for the maximum-sized Pascal string, whose address is passed to `FNamePtr`. `fd` is a local variable (allocated space) for a file descriptor.



#### RELATED INFORMATION

[Memory Manager](#) on page 1620

[File Manager](#) on page 1624

[Support Manager](#) on page 1628

[LabVIEW Manager Data Types](#) on page 1611

## LabVIEW Manager Data Types

**LabVIEW Manager** data types include the following data types:

- Scalar
- Char
- Dynamic
- Memory-related
- Constants

### Scalar

Scalar data types include Boolean and numeric.

### Boolean

External code modules work with two kinds of Boolean scalars—those existing in LabVIEW block diagrams and those passing to and from manager routines. The manager routines use a conventional Boolean form where 0 is FALSE and 1 is TRUE. This conventional Boolean form is called a `Bool32` and is stored as a 32-bit value.

LabVIEW block diagrams store Boolean scalars as 8-bit values. The value is 0 if FALSE and 1 if TRUE. This Boolean form is called an `LVBoolean`.

The following table describes the two forms of Boolean scalars.

| Name      | Description                           |
|-----------|---------------------------------------|
| Bool32    | 32-bit integer, 0 if FALSE, 1 if TRUE |
| LVBoolean | 8-bit integer, 0 if FALSE, 1 if TRUE  |

Numeric

The numeric data types support 8-, 16-, 32-, and 64-bit signed and unsigned integers.

LabVIEW supports the following single, double, and extended floating-point data types.

| Type     | Description  |
|----------|--------------|
| Single   | 32-bit       |
| Double   | 64-bit       |
| Extended | Up to 80-bit |

LabVIEW supports complex numbers containing two floating-point numbers, with different complex numeric types for each of the floating-point data types.

LabVIEW supports the following basic data types for numbers:

- Signed integers
  - `int8` 8-bit integer
  - `int16` 16-bit integer
  - `int32` 32-bit integer
  - `int64` 64-bit integer
- Unsigned integers
  - `uint8` 8-bit integer
  - `uint16` 16-bit integer
  - `uint32` 32-bit integer
  - `uint64` 64-bit integer
  - `size_t` 32-bit integer
- Floating-point numbers
  - `float32` 32-bit floating-point number
  - `float64` 64-bit floating-point number
  - `floatExt` extended-precision floating-point number



The following table explains how various platforms store extended-precision numbers.

| Platform          | Storage Format                                                                                                                                       |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Windows and Linux | 80-bit structure with two <code>int32</code> components, <code>mhi</code> and <code>mlo</code> , and an <code>int16</code> component, <code>e</code> |
| Mac Intel         | 64-bit double-precision floating-point number                                                                                                        |

## Complex Numbers

The complex data types are structures with two floating-point components, `re` and `im`. As with floating-point numbers, complex numbers can have 32-bit, 64-bit, and extended-precision components. The following table contains the code for the type definitions for each of these complex data types.

| Complex Number Type | Code                                                        |
|---------------------|-------------------------------------------------------------|
| 32-bit              | <pre>typedef struct {     float32 re, im; } cmplx64;</pre>  |
| 64-bit              | <pre>typedef struct {     float64 re, im; } cmplx128;</pre> |
| Extended-precision  | <pre>typedef struct {     floatExtre, im; } cmplxExt;</pre> |

## char

The `char` data type is defined by C to be an 8-bit signed integer. LabVIEW defines an unsigned `char` data type, with the following type definition:

```
typedef uInt8 uChar;
```

Dynamic

LabVIEW defines a number of data types you must allocate and deallocate dynamically. Arrays, strings, and paths have data types you must allocate using [memory manager](#) and [file manager](#) routines.

Arrays

LabVIEW supports arrays of any of the basic data types described in this section. You can construct more complicated data types using clusters, which can in turn contain scalars, arrays, and other clusters.

The first four bytes of a LabVIEW array indicate the number of elements in the array. The elements of the array follow the length field.

Strings

LabVIEW supports C- and Pascal-style strings, lists of strings, and LStr, a special string data type you use for string parameters to external code modules. The [support manager](#) contains routines for manipulating strings and converting them among the different types of strings.

C-Style Strings (CStr)

A C-style string CStr is a series of zero or more unsigned characters, terminated by a zero. C strings have no effective length limit. Most manager routines use C strings, unless you specify otherwise. The following code is the type definition for a C string:

```
typedef uChar *CStr;
```

Pascal-Style Strings (PStr)

A Pascal-style string PStr is a series of unsigned characters. The value of the first character indicates the length of the string. A PStr can have a range of 0 to 255 characters. The following code is the type definition for a Pascal string:

|                 |                         |
|-----------------|-------------------------|
| typedef uChar   | Str255[256], Str31[32], |
| *StringPtr,     |                         |
| **StringHandle; |                         |
| typedef uChar   | *PStr;                  |

## LabVIEW Strings (LStr)

The first four bytes of a LabVIEW string `LStr` indicate the length of the string. The specified number of characters follow the length of the string. `LStr` is the string data type used by LabVIEW block diagrams. The following code is the type definition for an `LStr` string:

```
typedef struct {
 int32 cnt;

 /* number of bytes that follow */

 uChar str[1];

 /* cnt bytes */

} LStr, *LStrPtr, **LStrHandle;
```

## Concatenated Pascal String (CPStr)

Many algorithms require manipulation of lists of strings. Arrays of strings are usually the most convenient representation for lists. However, arrays of strings can place a burden on the memory manager because of the large number of dynamic objects it must manage. To make working with lists more efficient, LabVIEW supports the concatenated Pascal string `CPStr` data type, which is a list of Pascal-style strings concatenated into a single block of memory. Using the `CPStr` data structure, you can use support manager routines to create and manipulate lists.

The following code is the type definition for a `CPStr` string:

```
typedef struct {
 int32 cnt;

 /* number of pascal strings that follow */

 uChar str[1];

 /* cnt concatenated pascal strings */

} CPStr, *CPStrPtr, **CPStrHandle;
```

## Paths

A path (pathname) indicates the location of a file or directory in a file system. LabVIEW has a separate data type for a path, represented as `Path`, which the file manager defines in a

platform-independent manner. The actual data type for a path is private to the file manager and subject to change. You can create and manipulate Path data types using file.

Memory-Related

LabVIEW uses pointers and handles to reference dynamically allocated memory. The memory-related data types have the following type definitions:

```
typedef uChar *UPtr;

typedef uChar **UHandle;
```

Constants

The manager data types define the following constant for use with external code modules:

```
NULL 0(uInt32)
```

The following constants define the possible values of the Bool32 data type:

```
FALSE 0 (int32)

TRUE 1 (int32)
```

The following constants define the possible values of the LVBoolean data type:

```
LVFALSE 0 (uInt8)

LVTRUE 1 (uInt8)
```

LabVIEW Manager Function Errors

If you receive errors from LabVIEW Manager functions, most error names correspond to LabVIEW error codes. The error names correspond to the following LabVIEW error codes:

| Error Name | LabVIEW Error Code Number |
|------------|---------------------------|
| mgArgErr   | 1                         |
| mFullErr   | 2                         |
| fEOF       | 4                         |
| fIsOpen    | 5                         |
| fIOErr     | 6                         |

| Error Name        | LabVIEW Error Code Number |
|-------------------|---------------------------|
| fNotFound         | 7                         |
| fNoPerm           | 8                         |
| fDiskFull         | 9                         |
| fDupPath          | 10                        |
| ftMFOpen          | 11                        |
| fNotEnabled       | 12                        |
| rfNotFound        | 13                        |
| rAddFailed        | 14                        |
| rNotFound         | 15                        |
| iNotFound         | 16                        |
| iMemoryErr        | 17                        |
| dPenNotExist      | 18                        |
| cfgBadType        | 19                        |
| cfgTokenNotFound  | 20                        |
| cfgParseError     | 21                        |
| cfgAllocError     | 22                        |
| ecLVSBFormatError | 23                        |
| ecLVSBSubrError   | 24                        |
| ecLVSBNoCodeError | 25                        |
| wNullWindow       | 26                        |
| wDestroyMixup     | 27                        |
| menuNullMenu      | 28                        |
| pAbortJob         | 29                        |
| pBadPrintRecord   | 30                        |
| pDriverError      | 31                        |
| pWindowsError     | 32                        |
| pMemoryError      | 33                        |
| pDialogError      | 34                        |

| Error Name          | LabVIEW Error Code Number |
|---------------------|---------------------------|
| pMiscError          | 35                        |
| dvInvalidRefnum     | 36                        |
| dvDeviceNotFound    | 37                        |
| dvParamErr          | 38                        |
| dvUnitErr           | 39                        |
| dvOpenErr           | 40                        |
| dvAbortErr          | 41                        |
| bogusError          | 42                        |
| cancelError         | 43                        |
| OMObjLowErr         | 44                        |
| OMObjHiErr          | 45                        |
| OMObjNotInHeapErr   | 46                        |
| OMOHeapNotKnownErr  | 47                        |
| OMBadDPIdErr        | 48                        |
| OMNoDPinTabErr      | 49                        |
| OMMsgOutOfRangeErr  | 50                        |
| OMMethodNullErr     | 51                        |
| OMUnknownMsgErr     | 52                        |
| mgNotSupported      | 53                        |
| ncBadAddressErr     | 54                        |
| ncInProgress        | 55                        |
| ncTimeOutErr        | 56                        |
| ncBusyErr           | 57                        |
| ncNotSupportedErr   | 58                        |
| ncNetErr            | 59                        |
| ncAddrInUseErr      | 60                        |
| ncSysOutOfMem       | 61                        |
| ncSysConnAbortedErr | 62                        |

| Error Name                     | LabVIEW Error Code Number |
|--------------------------------|---------------------------|
| ncConnRefusedErr               | 63                        |
| ncNotConnectedErr              | 64                        |
| ncAlreadyConnectedErr          | 65                        |
| ncConnClosedErr                | 66                        |
| amInitErr                      | 67                        |
| occBadOccurrenceErr            | 68                        |
| occWaitOnUnBoundHdlrErr        | 69                        |
| occFunnyQOverflowErr           | 70                        |
| fDataLogTypeConflict           | 71                        |
| ecLVSBCannotBeCalledFromThread | 72                        |
| amUnrecognizedType             | 73                        |
| mCorruptErr                    | 74                        |
| ecLVSBErrorMakingTempDLL       | 75                        |
| ecLVSBOldCIN                   | 76                        |
| fmtTypeMismatch                | 81                        |
| fmtUnknownConversion           | 82                        |
| fmtTooFew                      | 83                        |
| fmtTooMany                     | 84                        |
| fmtScanError                   | 85                        |
| lvOLEConvertErr                | 87                        |
| rtMenuErr                      | 88                        |
| pwdTampered                    | 89                        |
| LvVariantAttrNotFound          | 90                        |
| LvVariantTypeMismatch          | 91                        |
| axEventDataNotAvailable        | 92                        |
| axEventStoreNotPresent         | 93                        |
| axOccurrenceNotFound           | 94                        |
| axEventQueueNotCreated         | 95                        |

| Error Name              | LabVIEW Error Code Number |
|-------------------------|---------------------------|
| axEventInfoNotAvailable | 96                        |
| oleNullRefnumPassed     | 97                        |
| iviInvalidDowncast      | 102                       |
| iviInvalidClassSesn     | 103                       |
| ncSockNotMulticast      | 108                       |
| ncSockNotSinglecast     | 109                       |
| ncBadMulticastAddr      | 110                       |
| ncMcastSockReadOnly     | 111                       |
| ncMcastSockWriteOnly    | 112                       |
| ncDatagramMsgSzErr      | 113                       |
| dataCorruptErr          | 116                       |
| requireFullPathErr      | 117                       |
| folderNotExistErr       | 118                       |
| ncBtInvalidModeErr      | 119                       |
| ncBtSetModeErr          | 120                       |
| mgBtInvalidGUIDStrErr   | 121                       |
| rVersInFuture           | 122                       |

## Memory Manager

Most applications need routines for allocating and deallocating memory on request. The LabVIEW memory manager is the set of platform-independent memory manager functions you can use to dynamically allocate, manipulate, and deallocate memory. The memory manager supports dynamic allocation of both nonrelocatable and relocatable blocks, using pointers and handles.

If you need to perform dynamic memory allocation or manipulation from external code modules, use the memory manager. If your external code operates on data types other than scalars, you should understand how LabVIEW manages memory and know which utilities manipulate data.

The following code shows how the memory manager defines generic handle and pointer data types:

```
typedef uChar *UPtr;
```



```
typedef uChar **UHandle;
```

All data passed to or from a shared library is allocated using the memory manager. You should only use file manager functions, not the memory manager routines, to manipulate paths. Thus, your shared library should use the memory routines when working with parameters passed from the block diagram.

## Memory Allocation

Applications use the following types of memory allocation:

- Static
- Dynamic



### NOTE

You can allocate memory for variables in C code using `malloc`. However, do not assign the resulting pointer to any variable you pass back to the LabVIEW block diagram. Use the memory manager functions if you want to create or resize memory associated with a variable passed from the LabVIEW block diagram.

## Static Memory Allocation

With static memory allocation, the compiler determines memory requirements when you create an application. When you launch the application, LabVIEW creates memory for the known global memory requirements of the application. The memory LabVIEW creates remains allocated while the application runs. Static memory allocation is simple to work with because the compiler handles all the details.

However, static memory allocation cannot address the memory management requirements of most real-world applications because you cannot determine most memory requirements until run time. Also, statically declared memory might result in larger memory requirements because the memory is allocated for the duration of the application.

## Dynamic Memory Allocation

With dynamic memory allocation, you reserve memory when you need it and free memory when you are no longer using it. Dynamic allocation requires more work than static memory allocation because you have to determine memory requirements and allocate and deallocate memory as necessary.

You can use the LabVIEW memory manager to dynamically allocate memory in the following ways:

- Using pointers to allocate memory
- Using handles to allocate memory

## Using Pointers for Dynamic Memory Allocation

The more conventional method uses pointers to allocate memory. With pointers, you request a block of memory of a certain size. The routine returns the address of the block of memory to your shared library. When you no longer need the block of memory, you call a routine to free the block of memory. You can use the block of memory to store data. You reference the data stored in the block of memory by using the address the manager routine returned when you created the pointer. You can make copies of the pointer and use them in multiple places in your application to refer to the same data.

Pointers in the LabVIEW memory manager are nonrelocatable, which means the manager never moves the memory block to which a pointer refers while that memory is allocated for a pointer. Because other references to the memory block do not become out of date, not moving the memory block allocated to a pointer avoids problems that occur when you need to change the amount of memory allocated to a pointer. If you need more memory, sufficient memory might not exist to expand the memory space of the pointer without moving the memory block to a new location. If an application had multiple references to the pointer, moving the memory block to a new location causes problems because each pointer refers to the old memory address of the data. Using invalid pointers can cause severe problems.

## Using Handles for Dynamic Memory Allocation

A second form of memory allocation uses handles. As with pointers, when you allocate memory using handles, you request a block of memory of a certain size. The memory manager allocates the memory and adds the address of the memory block to a list of master pointers. The memory manager returns a handle that is a pointer to the master pointer. If you reallocate a handle and it moves to another address, the memory manager updates the master pointer to refer to the new address. If you look up the correct address using the handle, you access the correct data.

Use handles to perform most memory allocation in LabVIEW. Pointers are available because in some cases they are more convenient and simpler to use.

## Using Pointers and Handles

Create a handle using `DSNewHandle`, which allows you to specify the size of the memory block, or `DSNewAlignedHandle`, which allows you to specify the size, alignment, and alignment offset of the memory block. Create a pointer using `DSNewPtr`. Create a handle or pointer and set it to all zeros using `DSNewHClr`, `DSNewAlignedHClr`, and `DSNewPClr`.

When you are finished with the handle or pointer, release it using `DSDisposeHandle` or `DSDisposePtr`.

If you need to resize an existing handle, use the `DSSetHandleSize` function, which determines the size of an existing handle. If you need to resize and realign an existing handle, use the `DSSetAlignedHandleSize` function, which determines the size, alignment, and alignment offset of an existing handle. You also can resize and realign an existing handle and set it to all zeros using `DSSetAlignedHSzClr`. Because pointers are not relocatable, you cannot resize them.

A handle is a pointer to a pointer. In other words, a handle is the address of an address. The second pointer, or address, is a master pointer, which means it is maintained by the memory manager. Languages that support pointers provide operators for accessing data by its address. With a handle, you use this operator twice—once to get to the master pointer, and a second time to get to the actual data.

Additional routines make it easy to copy and concatenate handles and pointers to other handles, check the validity of handles and pointers, and copy or move blocks of memory from one place to another.

### Simple Example of Using Pointers and Handles

This simple example demonstrates how to work with pointers and handles in C.

The following code shows how to work with a pointer to an `int32`:

```
int32 *myInt32P;

myInt32P = (int32 *)DSNewPtr(sizeof(int32));

*myInt32P = 5;

x = *myInt32P + 7;

...

DSDisposePtr(myInt32P);
```

The first line declares the variable `myInt32P` as a pointer to, or the address of, a 32-bit signed integer. The first line does not actually allocate memory for the `int32`. The first line creates memory for an address and associates the name `myInt32P` with that address. The P at the end of the variable name is a convention used in this example to indicate the variable is a pointer.

The second line creates a block of memory in the data space large enough to hold a single 32-bit signed integer and sets `myInt32P` to refer to this memory block.

The third line places the value 5 in the memory location to which `myInt32P` refers. The `*` operator refers to the value in the address location.

The fourth line sets `x` equal to the value at address `myInt32P` plus 7.

The last line frees the pointer.

The following code is the same example using handles instead of pointers:

```
int32 **myInt32H;
```

```

myInt32H =(int32**)DSNewHandle(sizeof(int32));

**myInt32H = 5;

x = **myInt32H + 7;

...

DSDisposeHandle(myInt32H);

```

The first line declares the variable `myInt32H` as a handle to a 32-bit signed integer. Strictly speaking, the first line declares `myInt32H` as a pointer to a pointer to an `int32`. As with the previous example, the first line does not allocate memory for the `int32`. The first line creates memory for an address and associates the name `myInt32H` with that address. The H at the end of the variable name is a convention used in this example to indicate the variable is a handle.

The second line creates a block of memory in the data space large enough to hold a single `int32`. `DSNewHandle` places the address of the memory block as an entry in the master pointer list and returns the address of the master pointer entry. Finally, the second line sets `myInt32H` to refer to the master pointer.

The third line places the value 5 in the memory location to which `myInt32H` refers. Because `myInt32H` is a handle, you use the `*` operator twice to get to the data.

The fourth line sets `x` equal to the value referenced by `myInt32H` plus 7.

The last line frees the handle.

This example shows only the simplest aspects of how to work with pointers and handles in C. Other examples show different aspects of using pointers and handles. Refer to a C manual for a list of other operators you can use with pointers and for more information about how to work with pointers.

## File Manager

The file manager supports routines for opening and creating files, reading data from and writing data to files, and closing files. In addition, you can manipulate the end-of-file mark of a file and position the current read or write mark to an arbitrary position in the file. You also can move, copy, and rename files, determine and set file characteristics, and delete files.

The file manager contains a number of routines for directories, with which you can create and delete directories. You also can determine and set directory characteristics and obtain a list of a directory's contents.

LabVIEW supports concurrent access to the same file, so you can have a file open for both reading and writing simultaneously. When you open a file, you can indicate whether you

want the file to be read from and written to concurrently. You also can lock a range of the file, if you need to make sure a range is nonvolatile at a given time.

The file manager also provides many routines for manipulating paths, or path names, in a platform-independent manner. The file manager supports the creation of path descriptions, which are either relative to a specific location or absolute, that is, the full path. With file manager routines you can create and compare paths, determine characteristics of paths, and convert a path between platform-specific descriptions and the platform-independent form.

Applications that manipulate files can use the functions in the file manager. The file manager routines support basic file operations such as creating, opening, and closing files, writing data to files, and reading data from files. In addition, you can use file manager routines to create directories, determine characteristics of files and directories, and copy files. File manager routines use a LabVIEW data type for file path names, called `Path s`, that indicates a file or directory path independent of the platform. You can translate a `Path` to and from the conventional format a host platform uses for describing a file pathname.

## Identifying Files and Directories

When you perform operations on files and directories, you need to identify the target of the operation. The platforms LabVIEW supports use a hierarchical file system, meaning files are stored in directories, possibly nested several levels deep. These hierarchical file systems support the connection of multiple discrete storage media, called volumes. For example, DOS-based systems support multiple drives connected to the system. For most of these hierarchical file systems, you must include the volume name to specify the location of a file. On other systems, such as Linux, you do not need to specify the volume name because the physical implementation of the file system is hidden from the user.

How you identify a target depends upon whether the target is an open or closed file. If a target is a closed file or a directory, specify the target using the path of the target. The path describes the volume containing the target, the directories between the top level and the target, and the name of the target. If the target is an open file, use a file descriptor to specify that LabVIEW should perform an operation on the open file. The file descriptor is an identifier the file manager associates with the file when you open it. When you close the file, the file manager dissociates the file descriptor from the file.

## Path Specifications

LabVIEW uses the following types of file path specifications:

- Conventional
- Empty
- LabVIEW specifications

Conventional

All platforms have a method for describing the paths for files and directories. These path specifications are similar, but they are usually incompatible from one platform to another. You usually specify a path as a series of names separated by separator characters. Typically, the first name is the top level of the hierarchical specification of the path. The last name is the file or directory the path identifies.

A path can be one of the following types:

- Relative path
- Absolute path

A relative path describes the location of a file or directory relative to an arbitrary location in the file system. An absolute path describes the location of a file or directory starting from the top level of the file system.

A path does not necessarily go from the top of the hierarchy down to the target. You can often use a platform-specific tag in place of a name that indicates the path should go up a level from the current location.

(OS X 64-bit and Linux) You specify the path of a file or directory as a series of names separated by the slash (/) character. If the path is an absolute path, you begin the specification with a slash. Indicate the path should move up a level using two periods in a row (..). Thus, the following path specifies a file `readme` relative to the top level of the file system, where `username` is the user directory:

`/usr/home/ username /myapps/readme`

The following paths are two relative paths to the same file:

|                                      |                                                                       |
|--------------------------------------|-----------------------------------------------------------------------|
| <code>username /myapps/readme</code> | relative to <code>/usr/home</code>                                    |
| <code>../myapps/readme</code>        | relative to a directory inside of the <code>username</code> directory |

(Windows) You separate names in a path with a backslash (\) character. If the path is an absolute path, you begin the specification with a drive designation, followed by a colon (:), followed by the backslash. Indicate the path should move up a level using two periods in a row (..). Thus, the following path specifies a file `readme` relative to the top level of the file system, on a drive named `C` :

`c:\home\ username \myapps\readme`

The following paths are two relative paths to the same file:

|                                      |                                                                       |
|--------------------------------------|-----------------------------------------------------------------------|
| <code>username \myapps\readme</code> | relative to the <code>home</code> directory                           |
| <code>..\myapps\readme</code>        | relative to a directory inside of the <code>username</code> directory |

(OS X 32-bit) You separate names in a path with the colon (:) character. If the path is an absolute path, you begin the specification with the name of the volume containing the file. If an absolute path consists of only one name, that is, it specifies a volume, the path must end with a colon. If the path is a relative path, it begins with a colon. The colon is optional for a relative path consisting of only one name. Indicate the path should move up a level using two colons in a row (::). Thus, the following path specifies a file `readme` relative to the top level of the file system, on a drive named `hard drive`:

`hard drive:home: username :myapps:readme`

The following paths are two relative paths to the same file:

|                                          |                                                          |
|------------------------------------------|----------------------------------------------------------|
| <code>: username :myapps : readme</code> | relative to the home directory                           |
| <code>: :myapps : readme</code>          | relative to a directory inside of the username directory |

## Empty

You can define a path with no names, called an empty path. An empty path is either absolute or relative. An absolute path contains the complete address of a file's location within the file hierarchy. An empty absolute path refers to the highest point in the file hierarchy. A relative path contains the address of a file's location relative to the position of the user within the file hierarchy. An empty relative path refers to the current position of the user within the file hierarchy.

(OS X 64-bit and Linux) A slash (/) represents the empty absolute path. The slash specifies the root of the file hierarchy. A period (.) represents the empty relative path.

(Windows) You represent the empty absolute path as an empty string. The empty absolute path specifies the set of all volumes on the system. A period (.) represents the empty relative path.

(OS X 32-bit) The empty absolute path is represented as an empty string. The absolute path specifies the set of all volumes on the system. A colon (:) represents the empty relative path.

## LabVIEW

In LabVIEW, you specify a path using a special LabVIEW data type, represented as `Path`. The exact structure of the `Path` data type is private to the file manager. You create and manipulate the `Path` data type using file manager routines.

A `Path` is a dynamic data structure. Just as you use memory manager routines to allocate and deallocate handles and pointers, you use file manager routines to create and deallocate a `Path`. Just as with handles, declaring a `Path` variable does not actually create a `Path`. Before you can use the `Path` to manipulate a file, you must dynamically allocate the `Path` using file manager routines. When you are finished using the `Path` variable, you should release the `Path` using file manager routines.

In addition to providing routines for the creation and elimination of a Path, the file manager provides routines for comparing, duplicating, determining Path characteristics, and converting Path to and from other formats, such as the platform-specific format for the system on which LabVIEW is running.

File Descriptors

When you open a file, LabVIEW returns a file descriptor associated with the file. A file descriptor is a data type LabVIEW uses to identify open files. All operations performed on an open file use the file descriptor to identify the file.

A file descriptor is valid only while the file is open. If you close the file, the file descriptor is no longer associated with the file. If you open the file again, the new file descriptor is most likely different from the previous file descriptor.

File Refnums

In the file manager, LabVIEW accesses open files using file descriptors. However, on the front panel and block diagram, LabVIEW accesses open files using file refnums. A file refnum contains a file descriptor for use by the file manager and additional information used by LabVIEW.

LabVIEW specifies file refnums using the LVRefNum data type, the exact structure of which is private to the file manager. To pass references to open files into a shared library, convert file refnums to file descriptors with the FRefNumToFD LabVIEW Manager function. To pass references to open files out of a shared library, convert file descriptors to file refnums with the FNewRefNum LabVIEW Manager function.

Support Manager

The support manager contains a collection of constants, macros, basic data types, and functions, such for bit or byte manipulation of data, string manipulation, mathematical operations, sorting, searching, and determining the current time and date.

The string functions contain much of the functionality of the string libraries supplied with standard C compilers, such as string concatenation and formatting. You can use variations of many of these functions with LabVIEW strings, Pascal strings, and C strings. The following table describes the different string types.

| String Type | Description                                                        |
|-------------|--------------------------------------------------------------------|
| LabVIEW     | 4-byte length field followed by data, generally stored in a handle |
| Pascal      | 1-byte length field followed by data                               |
| C           | data terminated by a null character                                |



With the utility functions, you can sort and search on arbitrary data types, using quicksort and binary search algorithms. You also can convert numeric error codes to the associated text description.

The support manager also contains transcendental functions for many complex and extended floating-point operations.

Certain routines specify time as a data structure. The following example code illustrates specifying time as a data structure:

```
typedef struct {
 int32sec; /* 0:59 */
 int32min; /* 0:59 */
 int32hour; /* 0:23 */
 int32mday; /* day of the month, 1:31 */
 int32mon; /* month of the year, 1:12 */
 int32year; /* year, 1600:3000 */
 int32wday; /* day of the week, 1:7 for Sun:Sat */
 int32yday; /* day of year (julian date), 1:366 */
 int32isdst; /* 1 if daylight savings time */
} DateRec;
```

## Signal Processing and Analysis

### Signal Generation in LabVIEW

The Signal Generation VIs create many common signals required for network analysis and simulation. You also can use the Signal Generation VIs in conjunction with National Instruments hardware to generate analog output signals.

The names of most of the Signal Generation VIs contain the word wave or pattern. These names indicate whether a VI can keep track of the phase of the signal it generates each time the VI is called.



Note You can use the Signal Generation VIs only in the LabVIEW Full and Professional Development Systems.

When you use a VI that requires the normalized frequency as an input, you must convert the frequency units to the normalized units of cycles per sample. You must use normalized units of cycles per sample with the following Signal Generation VIs :

- Sine Wave VI
- Square Wave VI
- Sawtooth Wave VI
- Triangle Wave VI
- Arbitrary Wave VI
- Chirp Pattern VI

## IIR Filter Characteristics in LabVIEW

IIR filters in LabVIEW have the following characteristics:

- The IIR Filter VI interprets values at negative indexes in the following impulse equations as zero the first time you call the VI.  

$$x_0 = 1$$

$$x_i = 0$$
 for all  $i \neq 0$ .
- A transient response, or delay, proportional to the filter order occurs before the filter reaches a steady state.
- The number of elements in the filtered sequence equals the number of elements in the input sequence.
- The filter retains the internal filter state values when the filtering process finishes.

## Curve Fitting in LabVIEW

Some Fitting VIs return only the coefficients for the curve that best describe the input data while other Fitting VIs return the fitted curve. Using the VIs that return only coefficients allows you to further manipulate the data. The VIs that return the fitted curve also return the coefficients and the mean squared error (MSE). MSE is a relative measure of the residuals between the expected curve values and the actual observed values. Because the input data represents a discrete system, the VIs use the following equation to calculate MSE:

$$\text{MSE} = \frac{1}{n} \sum_{i=0}^{n-1} (f_i - y_i)^2$$

where  $f$  is the sequence representing the fitted values,  $y$  is the sequence representing the observed values, and  $n$  is the number of observed sample points.

The Fitting VIs automatically set up and solve the Jacobian system and return the set of coefficients that best describes the data set.

Related information Fitting VIs

## Spline Interpolation

Spline interpolation avoids the disadvantages of polynomial interpolation because functions require more data points and the line interpolant is similar to the original function.

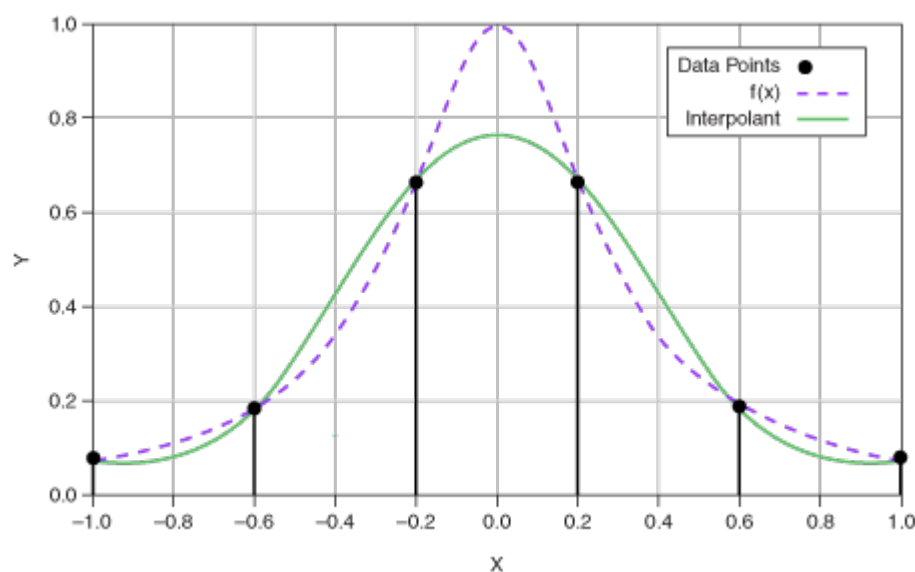
Cubic spline interpolation is a common type of spline interpolation that uses a third-order polynomial for each interval between two adjacent points. Third-order polynomials must meet the following conditions:

- The first and second derivatives at interior points are continuous.
- The polynomials pass through all data points.
- For one-dimensional interpolation, the second derivatives for the beginning and end points are zero. For spline interpolant, the initial boundary and final boundary inputs specify the first derivatives at the beginning and end points. For spline interpolation, a line with the first derivatives behaves the same as the spline interpolant. For spline interpolation, a line with a natural spline behaves the same as one-dimensional interpolation.

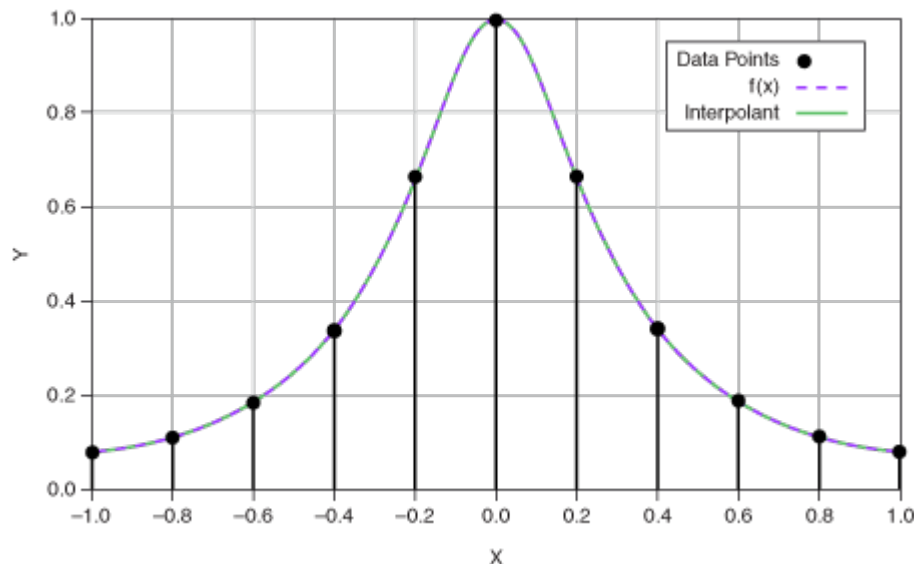
For example, the equation

$$y = f(x) = \frac{1}{1+12x^2}, \quad x \in [-1, 1]$$

with natural cubic spline interpolation is shown in the following figure.



When you have more data points of  $f(x)$ , the interpolant of the cubic spline interpolation is similar to  $f(x)$ , as shown in the following figure.



Related information

Interpolate 1D VI Spline Interpolant VI Spline Interpolation VI

## Basic Linear Algebra Subroutines

### Introduction to Basic Linear Algebra Subroutines

Basic Linear Algebra Subroutines (BLAS) are routines that provide standard functions for basic vector and matrix operations. You can classify these routines in the following three levels:

- BLAS level 1, which calculates vector operations.
- BLAS level 2, which calculates matrix-vector operations.
- BLAS level 3, which calculates matrix-matrix operations.

For example, dgemm is a BLAS level 3 function that calculates the general real matrix-matrix product using the following expression:

$$\alpha * op(A) * op(B) + \beta * op(C)$$

where  $\alpha$  and  $\beta$  are scalars, A, B, and C are matrices, and opA is A or  $A^T$ .

Refer to BLAS (Basic Linear Algebra Subprograms) for more information on the BLAS functions.

Related information [BLAS \(Basic Linear Algebra Subprograms\)](#)

### Choosing Between BLAS VIs and Linear Algebra VIs

While LabVIEW provides basic Linear Algebra VIs, you can use the Basic Linear Algebra Subroutines VIs for advanced linear algebra algorithms. Use the Basic Linear Algebra Subroutines VIs to combine multiple linear algebra operations, reduce data copying in

LabVIEW, and optimize execution. Therefore, if you have limited memory, use the Basic Linear Algebra Subroutines VIs.



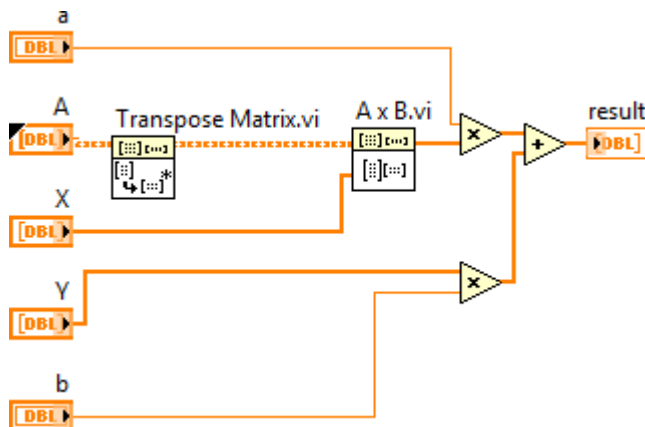
Note You can use the Basic Linear Algebra Subroutines VIs only in the LabVIEW Full and Professional Development Systems.

For example, the following expression combines linear algebra functions:

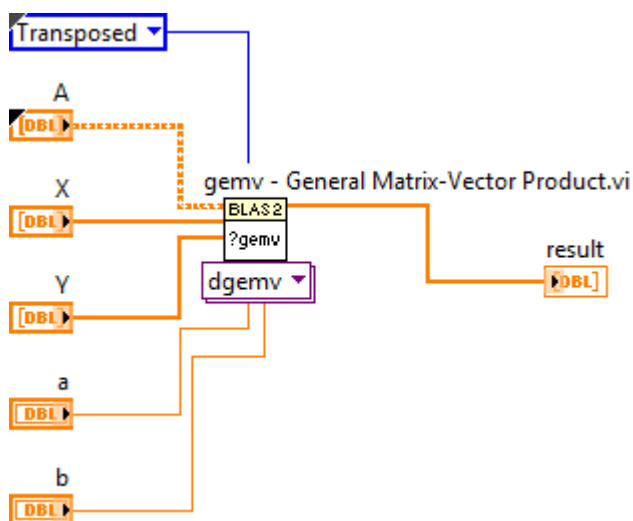
$$\alpha A^T X + bY$$

where  $\alpha$  is scalar, A is a matrix, and X and Y are vectors.

With linear algebra VIs, you need five different VIs and functions to perform this calculation, as shown in the following block diagram.



To simplify the code on a block diagram, you can replace multiple VIs and functions with the Basic Linear Algebra Subroutines VIs. For the current example, the gemv - General Matrix-Vector Product VI simplifies the code, as shown in the following block diagram.



Related information Linear Algebra VIs  
Basic Linear Algebra Subroutines VIs gemv - General Matrix-Vector Product VI

# Solving ODEs in LabVIEW

Use the ODE Solver VI to specify the ODE solver to use to evaluate an ordinary differential equation. You can use the ODE Solver VI only in the LabVIEW Full and Professional Development Systems.

LabVIEW includes the following ODE solvers:

- Runge-Kutta 1 (Euler) —A fixed step-size, single-step explicit Runge-Kutta ODE solver of first order.
- Runge-Kutta 2 —A fixed step-size, single-step explicit Runge-Kutta ODE solver of second order.
- Runge-Kutta 3 —A fixed step-size, single-step explicit Runge-Kutta ODE solver of third order.
- Runge-Kutta 4 —A fixed step-size, single-step explicit Runge-Kutta ODE solver of fourth order.
- Runge-Kutta 23 —A variable step-size, single-step explicit Runge-Kutta ODE solver of third order.
- Runge-Kutta 45 —A variable step-size, single-step explicit Runge-Kutta ODE solver of fifth order, which uses the Dormand-Prince coefficients.
- BDF —A variable step-size, variable order (orders 1 through 5) implementation of the multi-step backwards difference formula (BDF), also known as Gear's Method. This method is adequate for moderately stiff problems.
- Adams-Moulton —A variable step-size, multi-step variable order (orders 1 through 12) implementation of the Adams-Moulton predictor-corrector pair in predict-evaluate-correct-evaluate (PECE) mode.
- Rosenbrock —A variable step-size, single-step explicit solver. This method is adequate for some stiff problems.

# Solving PDEs in LabVIEW

Use the Partial Differential Equations VIs to solve partial differential equations (PDE), including the Helmholtz, heat, and wave equations. The following table provides details about these equations.

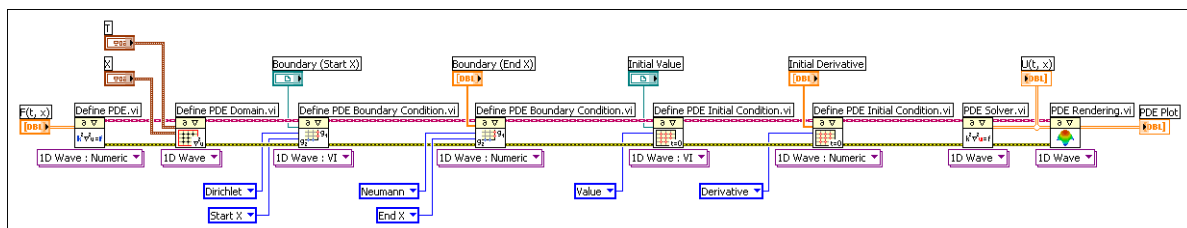
|                                                                                     |                                                                                                                          |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
|  | Note The Partial Differential Equations VIs are available only in the LabVIEW Full and Professional Development Systems. |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|

| Type of Equation   | Dimension and Domain                                                                      | Boundary Condition                                                                                                                | Method                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Helmholtz equation | The equation is two-dimensional and defined in a rectangular or polygonal domain.         | The polygonal domain supports only the Dirichlet condition. The rectangular domain supports the Dirichlet and Neumann conditions. | The finite element method solves the equation defined on the polygonal domain. The finite difference method solves the equation defined on the rectangular domain. |
| Heat equation      | The equation is one-dimensional or two-dimensional and defined on the rectangular domain. | The equation supports the Dirichlet and Neumann conditions.                                                                       | The finite difference method solves this equation.                                                                                                                 |
| Wave equation      | The equation is one-dimensional or two-dimensional and defined on the rectangular domain. | The equation supports the Dirichlet and Neumann conditions.                                                                       | The finite difference method solves this equation.                                                                                                                 |

The following block diagram examples show a common order for solving PDEs with the VIs. While you can add the VIs to the block diagram in the order they appear in the two examples, you do not need to follow that order when you use the PDE VIs. The following list provides an example of an order you can use when you want to solve PDEs with the VIs.

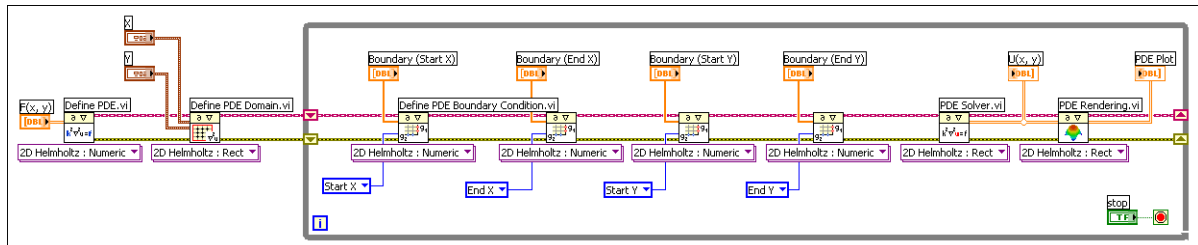
- Define PDE
- Define PDE Domain
- Define PDE Boundary Condition
- Define PDE Initial Condition
- PDE Solver
- PDE Rendering

The following block diagram illustrates an example of how to solve a 1D wave equation.



Note The polymorphic instance you choose must match the type of equation you need to solve. The type of instance also needs to be consistent from step to step. Otherwise, LabVIEW returns a type conflict error.

The following block diagram illustrates an example of how to solve a Helmholtz equation in a loop. The coefficient, the right side of the equation, and the domain of the equation do not change during the execution. While you do not need to repeatedly define the equation and domain in the loop, the boundary values vary over time. Therefore, you only need to place the definition of the boundary condition in the loop. In the following block diagram example, a shift register of a While Loop implements the PDE class to transfer data from one iteration to the next.



Related information [Partial Differential Equations VI](#) [Define PDE VI](#) [Define PDE Domain VI](#) [Define PDE Boundary Condition VI](#) [Define PDE Initial Condition VI](#) [PDE Solver VI](#) [PDE Rendering VI](#)

## Point-By-Point Analysis

## Point-by-Point Analysis and Array-Based Analysis in

The approach used for most point-by-point analysis operations in LabVIEW remains the same as array-based analysis. You can use filters, integration, mean value algorithms, and so on, in the same situations and for the same reasons that you can use these operations in array-based data analysis. In contrast, the computation of zeros in polynomial functions is not relevant to point-by-point analysis, and point-by-point versions of these array-based VIs are not necessary.

The following table compares array-based LabVIEW analysis to point-by-point analysis.

| Data Analysis Technology                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Array-Based Analysis (traditional paradigm)</p> <ul style="list-style-type: none"> <li>• Prepare a buffer unit of data.</li> <li>• Analyze data.</li> <li>• Produce a buffer of analyzed data.</li> <li>• Generate report.</li> </ul> | <p>Point-By-Point Analysis (newer paradigm)</p> <ul style="list-style-type: none"> <li>• Receive continuous stream of data.</li> <li>• Filter and analyze data continuously.</li> <li>• Generate real-time events and reports continuously.</li> </ul> |

The following table presents other comparisons between array-based and point-by-point analysis.



| Characteristic              | Array-Based Analysis                                                         | Data Acquisition and Analysis with Point By Point VIs                                              |
|-----------------------------|------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| Compatibility               | Limited compatibility with real-time systems                                 | Compatible with real-time systems; backward compatible with array-based systems                    |
| Data typing                 | Array-oriented                                                               | Scalar-oriented                                                                                    |
| Interruptions               | Interruptions critical                                                       | Interruptions tolerated                                                                            |
| Operation                   | You observe, offline                                                         | You control, online                                                                                |
| Performance and programming | Compensate for startup data loss (4-5 seconds) with complex “state machines” | Startup data loss does not occur; initialize the data acquisition system once and run continuously |
| Point of view               | Reflection of a process, like a mirror                                       | Direct, natural flow of a process                                                                  |
| Programming                 | Specify a buffer                                                             | No explicit buffers                                                                                |
| Results                     | Output a report                                                              | Output a report and an event in real time                                                          |
| Run-time behavior           | Delayed processing                                                           | Real time                                                                                          |
| Run-time behavior           | Stop                                                                         | Continue                                                                                           |
| Run-time behavior           | Wait                                                                         | Now                                                                                                |
| Work style                  | Asynchronous                                                                 | Synchronous                                                                                        |

When you perform point-by-point analysis, keep in mind the following concepts:

- **Initialization** —You must initialize the point-by-point analysis application to prevent interference from settings you made in previous sessions of data analysis.
- **Reentrant Execution** —You must enable LabVIEW reentrant execution for point-by-point analysis. Reentrant execution allocates fixed memory to a single analysis process, guaranteeing that two processes that use the same analysis function never interfere with each other.

---

**Note** If you create custom VIs to use in a point-by-point application, be sure to enable reentrant execution. Reentrant execution is enabled by default in almost all Point By Point VIs.

---

- **Deterministic Performance** —Point-by-point analysis is the natural companion to many deterministic systems, because it efficiently integrates with the flow of a real-time data signal.

Related information [Reentrant Execution](#)

## Performing Analysis without Buffers of Data

Analysis functions yield solutions that characterize the behavior of a data set. In array-based data acquisition and analysis, you might analyze a large set of data by dividing the data into 10 smaller buffers. Analyzing those 10 sets of data yields 10 solutions. You can further resolve those 10 solutions into one solution that characterizes the behavior of the entire data set.

In point-by-point analysis, you analyze an entire data set in real-time. A sample unit of a specific length replaces a buffer. The point-by-point sample unit can have a length that matches the length of a significant event in the data set that you are analyzing. For example, the application in a case study acquires a few thousand samples per second to detect defective train wheels. The input data for the train wheel application comes from the signal generated by a train that is moving at 60 km to 70 km per hour. The sample length corresponds to the minimum distance between wheels.

A typical point-by-point analysis application analyzes a long series of sample units, but you are likely to have interest in only a few of those sample units. To identify those crucial samples of interest, the point-by-point application focuses on transitions, such as the end of the relevant signal.

The train wheel detection application in the case study uses the end of a signal to identify crucial samples of interest. The instant the application identifies the transition point, it captures the maximum amplitude reading of the current sample unit. This particular amplitude reading corresponds to the complete signal for the wheel on the train whose signal has just ended. You can use this real-time amplitude reading to generate an event or a report about that wheel and that train.

Related concepts [Case Study of Point-By-Point Analysis](#)

## Case Study of Point-by-Point Analysis

### Case Study of Point-By-Point Analysis Application Built in LabVIEW

This case study shows a complete point-by-point analysis application built in LabVIEW with Point By Point VIs.



Note You can use the Point By Point VIs only in the LabVIEW Full and Professional Development Systems.

Refer to the Train Wheel PtByPt VI in the `labview\examples\ptbypt\PtByPt_No_HW.11b` for an example of using the Point By Point VIs.

The Train Wheel PtByPt VI is a real-time data acquisition application that detects defective train wheels and demonstrates the simplicity and flexibility of point-by-point data analysis.

When acquiring data with real-time performance, point-by-point analysis helps you analyze data in real time. Point-by-point analysis occurs continuously and instantaneously. While you acquire data, you filter and analyze it, point by point, to extract the information you need and to make an appropriate response. This case study demonstrates the effectiveness of the point-by-point approach for generation of both events and reports in real time.

Related information Point By Point VIs

## Point-By-Point Analysis of Train Wheels

In this example, the maintenance staff of a train yard must detect defective wheels on a train. The current method of detection consists of a railroad worker striking a wheel with a hammer and listening for a different resonance that identifies a flaw. Automated surveillance must replace manual testing, because manual surveillance is too slow, too prone to error, and too crude to detect subtle defects. An automated solution also adds the power of dynamic testing, because the train wheels can be in service during the test, instead of standing still.

The automated solution to detect potentially defective train wheels needs to have the following characteristics:

- Detect even subtle signs of defects quickly and accurately.
- Gather data when a train travels during a normal trip.
- Collect and analyze data in real time to simplify programming and to increase speed and accuracy of results.

The Train Wheel PtByPt VI offers a solution for detecting defective train wheels.



**Note** This example focuses on implementing a point-by-point analysis program in LabVIEW. The issues of ideal sampling periods and approaches to signal conditioning are beyond the scope of this example.

## Overview of the LabVIEW Point-By-Point Solution

As well as Point By Point VIs, the Train Wheel PtByPt VI requires standard LabVIEW programming objects, such as Case structures, While Loops, numeric controls, and numeric operators.

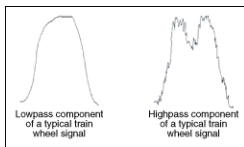
The data the Train Wheel PtByPt VI acquires flows continuously through a While Loop. The process carried out by the Train Wheel PtByPt VI inside the While Loop consists of five analysis stages that occur sequentially. The following list reflects the order in which the five analysis stages occur, briefly describes what occurs in each stage, and corresponds to the labeled portions of the previous block diagram.

1. In the data acquisition stage (DAQ), waveform data flows into the While Loop.
2. In the Filter stage, separation of low- and high-frequency components of the waveform occurs.
3. In the Analysis stage, detection of the train, wheel, and energy level of the waveform for each wheel occurs.
4. In the Events stage, responses to signal transitions of trains and wheels occurs.
5. In the Report stage, the logging of trains, wheels, and trains that might have defective wheels occurs.

## Related information While Loop Case Structure

## Characteristics of a Train Wheel Waveform

The characteristic waveform that train wheels emit determines how you analyze and filter the waveform signal point-by-point. A train wheel in motion emits a signal that contains low- and high-frequency components. If you mount a strain gauge in a railroad track, you detect a noisy signal similar to a bell curve. The following illustration shows the low- and high-frequency components of this curve.



The low-frequency component of train wheel movement represents the normal noise of operation. Defective and normal wheels generate the same low-frequency component in the signal. The peak of the curve represents the moment when the wheel moves directly above the strain gauge. The lowest points of the bell curve represent the beginning and end of the wheel, respectively, as the wheel passes over the strain gauge.

The signal for a train wheel also contains a high-frequency component that reflects the quality of the wheel. In operation, a defective train wheel generates more energy than a normal train wheel. In other words, the high-frequency component for a defective wheel has greater amplitude.

## Analysis Stages of the Train Wheel

### Five Analysis Stages of the Train Wheel

The waveform of all train wheels, including defective ones, falls within predictable ranges. This predictable behavior allows you to choose the appropriate analysis parameters. These parameters apply to five analysis stages.



**Note** You must adjust parameters for any implementation of the Train Wheel PtByPt VI because the characteristics of each data acquisition system differ.

## DAQ Stage of the Train Wheel PtByPt VI

Data moves into the Point By Point VIs through the input data parameter. The point-by-point detection application operates on the continuous stream of waveform data that comes from the wheels of a moving train. For a train moving at 60 km to 70 km per hour, a few hundred to a few thousand samples per second are likely to give you sufficient information to detect a defective wheel.

## Filter Stage of the Train Wheel PtByPt VI

The Train Wheel PtByPt VI must filter low- and high-frequency components of the train wheel waveform. Two Butterworth Filter PtByPt VIs perform the following tasks:

- Extract the low-frequency components of the waveform.
- Extract the high-frequency components of the waveform.

In the Train Wheel PtByPt VI, the Butterworth Filter PtByPt VIs use the following parameters:

- order specifies the amount of the waveform data that the VI filters at a given time and is the filter resolution. 2 is acceptable for the Train Wheel PtByPt VI.
- fl specifies the low cut-off frequency, which is the minimum signal strength that identifies the departure of a train wheel from the strain gauge. 0.01 is acceptable for the Train Wheel PtByPt VI.
- fh specifies the high cut-off frequency, which is the minimum signal strength that identifies the end of high-frequency waveform information. 0.25 is acceptable for the Train Wheel PtByPt VI.

## Related information Butterworth Filter PtByPt VI

## Analysis Stage of the Train Wheel PtByPt VI

The point-by-point detection application must analyze the low- and high-frequency components separately. The Array Max & Min PtByPt VI extracts waveform data that reveals the level of energy in the waveform for each wheel, the end of each train, and the end of each wheel.

Three separate Array Max & Min PtByPt VIs perform the following discrete tasks:

- Identify the maximum high-frequency value for each wheel.
- Identify the end of each train.
- Identify the end of each wheel.



Note The name Array Max & Min PtByPt VI contains the word array only to match the name of the array-based form of this VI. You do not need to allocate arrays for the Array Max & Min PtByPt VI.

In the Train Wheel PtByPt VI, the Array Max & Min PtByPt VIs use the following parameters and functions:

- sample length specifies the size of the portion of the waveform that the Train Wheel PtByPt VI analyzes. To calculate the ideal sample length, consider the speed of the train, the minimum distance between wheels, and the number of samples you receive per second. 100 is acceptable for the Train Wheel PtByPt VI. The Train Wheel PtByPt VI uses sample length to calculate values for all three Array Max & Min PtByPt VIs.
- The Multiply function sets a longer portion of the waveform to analyze. When this longer portion fails to display signal activity for train wheels, the Array Max & Min PtByPt VIs identify the end of the train. 4 is acceptable for the Train Wheel PtByPt VI.
- threshold provides a comparison point to identify when no train wheel signals exist in the signal that you are acquiring. threshold is wired to the Greater? function. 3 is an acceptable setting for threshold in the Train Wheel PtByPt VI.

Related information Multiply Function Greater? Function Array Max and Min PtByPt VI

## Events Stage of the Train Wheel PtByPt VI

After the Analysis stage identifies maximum and minimum values, the Events stage detects when these values cross a threshold setting.

The Train Wheel PtByPt VI logs every wheel and every train that it detects. Two Boolean Crossing PtByPt VIs perform the following tasks:

- Generate an event each time the Array Max & Min PtByPt VIs detect the transition point in the signal that indicates the end of a wheel.
- Generate an event every time the Array Max & Min PtByPt VIs detect the transition point in the signal that indicates the end of a train.

The Boolean Crossing PtByPt VIs respond to transitions. When the amplitude of a single wheel waveform falls below the threshold setting, the end of the wheel has arrived at the strain gauge. For the Train Wheel PtByPt VI, 3 is a good threshold setting to identify the end of a wheel. When the signal strength falls below the threshold setting, the Boolean Crossing PtByPt VIs recognize a transition event and pass that event to a report.

Analysis of the high-frequency signal identifies which wheels, if any, might be defective. When the Train Wheel PtByPt VI encounters a potentially defective wheel, the VI passes the information directly to the report at the moment the end-of-wheel event is detected.

In the Train Wheel PtByPt VI, the Boolean Crossing PtByPt VIs use the following parameters:

- initialize resets the VI for a new session of continuous data acquisition.
- direction specifies the kind of Boolean crossing.

## Related information Boolean Crossing PtByPt VI

## Report Stage of the Train Wheel PtByPt VI

The Train Wheel PtByPt VI reports on all wheels for all trains that pass through the data acquisition system. The Train Wheel PtByPt VI also reports any potentially defective wheels.

Every time a wheel passes the strain gauge, the Train Wheel PtByPt VI captures its waveform, analyzes it, and reports the event. The following table describes the components of a report on a single train wheel.

| Information Source                        | Meaning of Results                                                                                                  |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Counter mechanism for waveform events     | Stage One: Wheel number four has passed the strain gauge.                                                           |
| Analysis of highpass filter data          | Stage Two: Wheel number four has passed the strain gauge and the wheel might be defective.                          |
| Counter mechanism for end-of-train events | Stage Three: Wheel number four in train number eight has passed the strain gauge, and the wheel might be defective. |

The Train Wheel PtByPt VI uses point-by-point analysis to generate a report, not to control an industrial process. However, the Train Wheel PtByPt VI acquires data in real time, and you can modify the application to generate real-time control responses, such as stopping the train when the Train Wheel PtByPt VI encounters a potentially defective wheel.

# 스캔 I/O 데이터ter 접근하기

## I/O 변수 사용하기 (Real-Time, Windows)

I/O 변수는 I/O 데이터를 전송하는 **공유 변수**의 한 종류입니다. 로컬에서 접근할 때 I/O 변수는 **NI 스캔 엔진**을 사용해 I/O 데이터에 단일 포인트 접근합니다. 로컬에서 접근할 때 I/O 변수는 **NI Publish-Subscribe Protocol (NI-PSP)**을 사용해 I/O 데이터에 단일 포인트 접근하거나 버퍼 접근합니다. I/O 변수는 물리적인 I/O 채널과 직접 통신하거나, 사용자 I/O 데이터를 동일한 타겟 상에서 실행 중인 FPGA VI와 리얼타임(RT) VI 간에 전송시킬 수 있습니다.



주의  
NI 스캔 엔진은 데이터를 버퍼하지 않습니다. I/O 변수를 로컬에서 접근할 때 데이터 손실을 방지하려면, 다음 포인트가 도착하기 전에 각 데이터 포인트를 읽어야 하며, **NI 스캔 엔진 주기**보다 빨리 데이터를 쓰는 것을 피하십시오.

다음 테이블에는 I/O 변수를 사용하는 경우가 요약되어 있습니다:

| 용도                                                                                             | I/O 변수 타입                      |
|------------------------------------------------------------------------------------------------|--------------------------------|
| LabVIEW FPGA Module을 사용하지 않고 I/O 하드웨어와 LabVIEW VI 사이에서 스케일된 I/O 데이터를 전송함                       | 스케일된 I/O 데이터를 위한 I/O 변수        |
| 같은 하드웨어 디바이스 상에서 실행 중인 LabVIEW FPGA VI와 LabVIEW Real-Time VI 사이에서 FPGA 처리된 사용자 정의 I/O 데이터를 전송함 | 사용자 정의 FPGA I/O 데이터를 위한 I/O 변수 |



노트  
NI 스캔 엔진 지원에 대한 정보는 관련 하드웨어 디바이스 문서를 참조하십시오.

스케일된 I/O 데이터를 위한 I/O 변수

스케일된 I/O 데이터를 위해 I/O 변수를 사용하려면, 먼저 지원되는 I/O 디바이스를 NI 스캔 엔진이 설치된 타겟에 연결한 다음에 LabVIEW 프로젝트에 있는 타겟에서 해당 I/O 디바이스를 **추가**합니다.

프로젝트 탐색기 윈도우에서 지원되는 타겟에 지원되는 I/O 디바이스를 추가하면, LabVIEW는 자동으로 각 I/O 채널에 대해 I/O 변수를 추가합니다. 현재 LabVIEW 프로젝트에 포함된 타겟에 물리적으로 I/O 디바이스를 추가하는 경우, 이 I/O 디바이스를 프로젝트에 추가하고 이에 대응하는 I/O 변수를 먼저 **배포**해야 사용할 수 있습니다.

현재 LabVIEW 프로젝트의 일부인 타겟에서 물리적으로 I/O 디바이스를 제거하는 경우, 프로젝트에서 I/O 디바이스를 삭제하고 해당 I/O 디바이스를 포함하고 있었던 타겟을 다시 배포해야 합니다. 타겟을 재배포하면 타겟에서의 설정이 프로젝트 설정을 반영하도록 보장할 수 있습니다. 프로젝트 및 시스템 비교 대화 상자를 사용하면 프로젝트와 타겟 설정이 일치하는지 확인할 수 있습니다.

사용자 FPGA I/O 데이터를 위한 I/O 변수 (FPGA Module 또는 Real-Time Module)

프로젝트가 NI 스캔 엔진을 지원하는 FPGA 타겟을 포함하고 있는 경우, 동일한 하드웨어 디바이스 상에서 실행 중인 FPGA VI와 RT VI 사이에서 사용자 I/O 데이터를 전송할 **사용자 정의 I/O 변수를 생성**할 수 있습니다.

사용자 정의 I/O 변수를 호스트하는 새시에서 실행 중인 FPGA VI와 RT VI의 블록다이어그램에 사용자 정의 I/O 변수를 추가할 수 있습니다. 하지만, 모든 I/O 변수는 한 방향으로만 동작하기 때문에 각 사용자 정의 I/O 변수의 방향을 FPGA에서 호스트로 또는 호스트에서 FPGA로 중어서 하나로 설정해야 합니다. 예를 들어, FPGA VI에서 아날로그 I/O 데이터를 수집하고 데이터에 FFT를 수행하고, 이렇게 처리된 데이터를 FPGA에서 호스트로 I/O 변수를 사용하여 RT VI에 있는 컨트롤 루프에 전송한 후, 호스트에서 FPGA로 I/O 변수를 사



용하여 RT 컨트롤 루프로부터의 출력 데이터를 다시 FPGA에 전송하여 물리적 I/O 채널에 출력할 수 있습니다.

사용자 정의 I/O 변수를 생성하려면 LabVIEW FPGA Module을 설치해야 합니다. 그러나 Real-Time Module만 있는 경우, LabVIEW FPGA Module을 설치하지 않아도 RT VI에서 사용자 정의 I/O 변수를 사용할 수 있습니다. 예를 들어, LabVIEW FPGA Module을 보유한 개발자는 사용자 정의 I/O 변수를 생성하고, 이 변수를 사용하는 FPGA 비트 파일을 컴파일할 수 있습니다. 이후, 개발자는 FPGA Module이 없는 다른 개발자에게 해당 비트 파일과 사용자 정의 I/O 변수를 포함하는 LabVIEW 프로젝트를 배포할 수 있습니다. 그러면, 이 프로젝트를 사용하여 CompactRIO 새시에서 해당 비트 파일을 실행할 수 있습니다. 이 경우, 두번째 개발자는 사용자 정의 I/O 변수를 생성할 수는 없지만 Real-Time Module을 사용하여 첫번째 개발자가 생성한 사용자 정의 I/O 변수에 접근할 수 있습니다.

(Windows) 사용자 정의 I/O 변수에서 **네트워크 공개하기**가 활성화되어 있는 경우, 같은 LabVIEW 프로젝트 내에 있는 모든 RT VI 또는 Windows 기반의 VI에서 이 변수를 사용할 수 있습니다. 예를 들어, 네트워크에 공개된 I/O 변수를 사용하여 Windows에서 실행되는 사용자 인터페이스 VI를 생성할 수 있습니다.



#### 노트

사용자 정의 I/O 변수는 오직 **스캔 접근**만 지원합니다. 사용자 정의 I/O 변수에 대해 **다이렉트 접근**은 사용할 수 없습니다.

## I/O 변수 설정하기

편집 시나 실행 시에 **대화식으로** 또는 실행 시에 **프로그램적으로** I/O 변수를 설정할 수 있습니다. 또한 이 방법을 섞어서 사용할 수 있습니다. 다음 테이블에서는 각 방법의 장점과 단점이 요약되어 있습니다:

| 방법  | 용도                                              | 장점            | 단점                                                                                                                                 |
|-----|-------------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------|
| 대화식 | 프로젝트 탐색기 윈도우에 대한 접근 권한이 있을 때 I/O 변수에 대한 셋팅 설정하기 | 프로그래밍이 덜 필요함. | <ul style="list-style-type: none"> <li>블록 다이어그램을 볼 때 설정 셋팅이 보이지 않음.</li> <li>변수를 호스트하는 타겟에서만 I/O 변수에 대한 셋팅을 설정할 수 있습니다.</li> </ul> |

| 방법    | 용도                                                                                                                                                                                                                                                                                                                                                            | 장점                                                                                                          | 단점            |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|---------------|
| 프로그램적 | <ul style="list-style-type: none"> <li>프로젝트 탐색기 윈도우에 대한 접근 권한이 없을 때 I/O 변수에 대한 셋팅 설정하기.<br/>예를 들어, I/O 변수를 사용하는 독립 어플리케이션을 생성할 계획이 있고 어플리케이션 실행 중 I/O 변수에 대한 설정 셋팅을 변경할 필요가 있을 경우, I/O 변수를 프로그램적으로 설정합니다.</li> <li>I/O 변수를 호스트하는 타겟 이외의 컴퓨터에서 I/O 변수에 대한 셋팅 설정하기<br/>예를 들어, 호스트 컴퓨터에서 CompactRIO 타겟의 I/O 변수를 설정할 필요가 있을 경우, I/O 변수를 프로그램적으로 설정합니다.</li> </ul> | <ul style="list-style-type: none"> <li>블록다이어그램을 볼 때 설정 셋팅이 보임.</li> <li>I/O 변수를 원격으로 설정할 수 있습니다.</li> </ul> | 프로그래밍이 더 필요함. |

## 대화식으로 I/O 변수 설정하기

공유 변수 프로퍼티 대화 상자를 사용하여 단일 I/O 변수를 설정하거나 여러 변수 편집기 윈도우를 사용하여 동시에 여러 I/O 변수를 설정합니다.

I/O 변수를 사용하는 VI가 실행 중인 동안에도 I/O 변수의 설명, 네트워크, 스케일링 옵션을 업데이트할 수 있습니다. 그러나 실행 중에 I/O 변수의 이름이나 데이터 타입은 업데이트할 수 없습니다.



### 노트

새 I/O 변수를 추가하거나 I/O 변수의 설정을 변경할 때, 새 설정을 활성화하려면 변수를 **배포**해야 합니다.

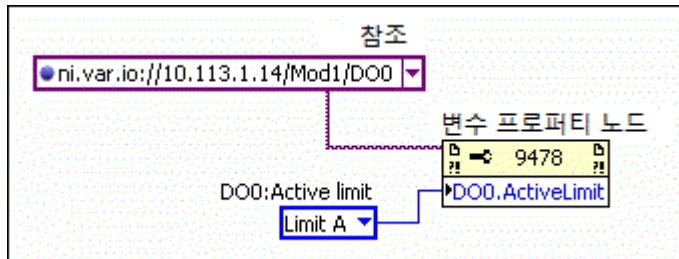
## 프로그램적으로 I/O 변수 설정하기

다음 단계를 따라 변수 프로퍼티를 사용하여 I/O 변수를 프로그램적으로 설정합니다:

1. 블록다이어그램에 변수 프로퍼티 노드를 추가합니다.
2. 참조 입력에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 생성>> 상수를 선택합니다.
3. 공유 변수 참조 번호 상수의 오른쪽에 있는 화살표 버튼을 클릭한 후 탐색을 선택합니다.
4. 변수 객체 탐색 대화 상자를 사용하여 설정하려는 I/O 변수를 선택합니다.
5. 공유 변수 참조 번호 상수에서 마우스 오른쪽 버튼을 클릭한 후 변수 클래스 선택 바로 가기 메뉴를 통해 사용하려는 프로퍼티의 클래스를 선택합니다.

6. 변수 프로퍼티 노드의 흰색 영역을 클릭한 후 사용하려는 프로퍼티를 선택합니다.
7. (옵션) 위치 도구를 사용해서 변수 프로퍼티 노드의 크기를 조정하면, 한 개 이상의 프로퍼티를 동시에 디스플레이할 수 있습니다.

다음 그림은 원격 타겟의 I/O 변수를 프로그램적으로 설정하는 예를 보여줍니다:



이 예에서, 변수 프로퍼티 노드는 CompactRIO 모듈 채널에 대응하는 I/O 변수를 설정하여 해당 채널에 대한 전류 리미트를 지정합니다.



#### 노트

5초 후에도 변수 프로퍼티 노드가 원격 I/O 변수에 연결될 수 없으면, 설정 호출이 타임아웃됩니다. 다음 단계를 따라 타임아웃 기간을 변경합니다:

1. labview\LabVIEW.ini를 텍스트 파일로 엽니다.
2. 새 라인에서, RemoteIOVConfigurationTimeout=을 입력하고 이어서 새로운 타임아웃 기간의 초 단위 수를 입력합니다. 예를 들어, 타임아웃 기간을 10초로 연장하려면, RemoteIOVConfigurationTimeout=10을 입력합니다.
3. 파일을 저장하고 닫습니다.

## I/O 변수 배포하기

I/O 변수를 배포하려면, 프로젝트 탐색기 윈도우에서 해당 I/O 변수를 포함하는 프로젝트 아이템에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 배포를 선택합니다.

I/O 변수 노드를 포함하는 VI를 실행할 때, LabVIEW는 해당 변수가 속한 I/O 변수 컨테이너를 자동으로 배포합니다. 이로써 컨테이너에 있는 모든 I/O 변수가 배포됩니다.



#### 노트

I/O 드라이버가 초기화에 실패하는 경우, LabVIEW는 프로젝트에서 모든 I/O 변수에 대한 로드 작업을 강제 종료합니다. 이 경우에 I/O 변수가 여전히 프로젝트에 나타날 수도 있지만, 어플리케이션에서 해당 변수에 접근하려고 시도하면 런타임 에러가 발생합니다. I/O 드라이버 에러 디버깅에 대한 정보는 해당 타겟 I/O 드라이버 문서를 참조하십시오.

## I/O 변수 값 스케일하기

공유 변수 프로퍼티 대화 상자의 스케일링 페이지에 있는 I/O 변수 또는 **가명**에서 선형 스케일링을 활성화할 수 있습니다. 하지만, I/O 변수는 비선형 스케일링을 지원하지 않습니다.



### 노트

일부 I/O 버스는 하드웨어 스케일링을 구현합니다. 이 경우, LabVIEW I/O 변수 스케일링은 추가적인 스케일링 레이어를 제공합니다. LabVIEW는 하드웨어 스케일링을 덮어쓰지 않습니다. LabVIEW는 I/O 버스로부터의 하드웨어 스케일된 값을 원시 값으로 사용하여 소프트웨어 스케일된 값을 계산합니다.

## I/O 가명 생성하기

I/O 변수의 가명을 생성하여, 다른 상황에서 I/O 데이터의 이름과 스케일을 서로 다르게 사용할 수 있습니다. 예를 들어, 열전쌍에 연결된 I/O 변수를 갖고 있는 경우에 섭씨와 화씨라는 가명을 생성할 수 있습니다. 그런 다음에 각 가명을 각자 스케일하여, 두 종류의 측정 단위에서 온도를 디스플레이하는데 사용할 수 있습니다.

I/O 가명의 값은 직상위의 값에 양방향으로 링크되어 있기 때문에 직상위 값을 업데이트하면 I/O 가명의 값이 업데이트되고, I/O 가명의 값을 업데이트하면 직상위의 값이 업데이트됩니다. 또한, I/O 가명 연결 링크가 업데이트 되면 연결된 다른 링크도 모두 업데이트 됩니다.

## 단일 I/O 가명 생성하기

I/O 가명을 생성하려면, 프로젝트 탐색기 윈도우에서 I/O 변수를 포함하는 타겟에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 새로 만들기 >> 변수를 선택하여 공유 변수 프로퍼티를 디스플레이합니다. I/O 가명의 이름을 입력하고, 변수 타입 풀다운 메뉴에서 I/O 가명을 선택합니다. 이후, 탐색 버튼을 클릭하고 새 I/O 가명과 연계시키려는 기존의 I/O 변수 또는 I/O 가명을 선택합니다.



### 노트

가명을 I/O 변수에 연계시킬 때, 가명의 데이터 타입이 기본 I/O 변수의 데이터 타입과 일치하는지 점검하십시오.

프로젝트 탐색기 윈도우를 사용하여 I/O 가명의 이름, 설명, 네트워크 공개 셋팅, 스케일링을 편집할 수 있습니다. 또한, 여러 변수 편집기 윈도우를 사용하여 여러 I/O 가명을 동시에 편집할 수 있습니다.



### 노트

I/O 가명 및 다른 타입의 공유 변수를 모두 포함하는 라이브러리를 배포할 수 없습니다. I/O 가명은 **별도의 라이브러리**에서 배포해야 합니다.

## 여러 I/O 가명 동시에 생성하기

일괄 변수 생성 대화 상자를 사용하여 여러 I/O 가명을 동시에 생성할 수 있습니다. 다음 단계에 따라 일괄적으로 여러 I/O 가명을 생성하십시오:

1. 단일 I/O 가명을 생성하여, 일괄 생성 프로세서에서 템플릿으로 사용합니다.
2. I/O 가명을 포함하는 변수 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 바로 가기 메뉴에서 변수 생성을 선택하여 일괄 변수 생성 대화 상자를 디스플레이합니다.
3. 프로퍼티 복사를 선택하고, 탐색 버튼을 클릭한 후에 템플릿으로 사용하려면 기존의 I/O 가명을 선택합니다.
4. 생성하려는 I/O 가명의 개수를 생성할 개수 필드에 입력하고 확인 버튼을 클릭하여 새 I/O 가명을 생성합니다.

이 단계를 모두 마치면, LabVIEW는 자동으로 여러 변수 편집기 윈도우를 열고 이를 통해 사용자가 새로 만든 변수를 편집할 수 있습니다. 예를 들어, 각 I/O 가명의 가명 경로를 편집하여 특정 I/O 변수에 연계시킬 수도 있습니다.

### I/O 변수 값 강제 변환하기

I/O 변수의 값을 **대화식** 또는 **프로그램적**으로 강제 변환할 수 있습니다. I/O 변수를 강제 변환하면, 이와 연계된 I/O 데이터는 변수의 강제 변환을 해제하거나, 타겟을 재부팅하거나, 변수가 다른 값을 취하도록 강제 변환할 때까지 사용자가 지정한 값을 취하게 됩니다. I/O 변수에 대한 강제 변환을 해제하면, I/O 변수에 대한 통제 권한이 NI 스캔 엔진에게 되돌아갑니다.



#### 노트

I/O 변수가 강제 변환될 때, 각 I/O 변수에 대한 접근 시간은 강제 변환하지 않는 경우에 비해 다소 길어집니다. 이로 인해 루프 주기가 강제 변환 오버헤드를 처리할 정도로 길지 않을 경우에, 루프의 실행이 느려질 수 있습니다.

### 대화식으로 I/O 변수 값 강제 변환하기

NI 배포된 시스템 관리자를 사용하여 I/O 채널을 디버깅하거나 수동으로 컨트롤하는 동안에 I/O 변수의 값을 **강제 변환** 또는 강제 변환을 해제합니다. LabVIEW에서 도구 >> 배포된 시스템 관리자를 선택하여 NI 배포된 시스템 관리자를 시작합니다. 또한, 프로젝트 탐색기 윈도우에 있는 RT 타겟에서 마우스 오른쪽 버튼을 클릭하고 유틸리티 >> 시스템 관리자에서 보기를 선택하여, 배포된 시스템 관리자를 시작할 수 있습니다.



#### 노트

NI 배포된 관리자의 보안 설정을 사용하여, 사용자가 NI 배포된 시스템 관리자 내에서 I/O 변수를 강제 변환하는 것을 방지할 수 있습니다.

### 프로그램적으로 I/O 변수 값 강제 변환하기

강제 변환 VI를 사용하여 I/O 변수를 프로그램적으로 강제 변환 및 강제 변환 해제합니다.

## I/O 가명 강제 변환하기

강제 변환은 표준 I/O 변수뿐만 아니라 가명에도 적용됩니다. I/O 변수를 강제 변환하면, 이와 연결된 가명도 모두 강제 변환하는 것입니다. 가명을 강제 변환하면, 직상위 I/O 변수와 이와 연결된 다른 가명도 모두 강제 변환하는 것입니다.



### 노트

LabVIEW는 I/O 채널의 실제 값처럼 강제 변환된 값에 스케일링을 적용합니다. 따라서, 가명 연결 고리에서 어느 한 링크를 강제 변환하면 그 연결 고리에 있는 모든 링크가 적절하게 스케일링됩니다.

## 동시에 타겟에 있는 모든 I/O 변수를 강제 변환 및 강제 변환 해제하기

타겟의 글로벌 강제 변환 상태를 사용하여, 타겟에서 모든 I/O 변수를 강제 변환 및 강제 변환 해제할 수 있습니다. 대화식 또는 프로그램적으로 타겟에서 글로벌 강제 변환을 활성화 및 비활성화시킬 수 있습니다.

NI 배포된 시스템 관리자에 있는 강제 변환 활성화와 강제 변환 비활성화 버튼을 사용하여, 대화식으로 타겟에서 글로벌 강제 변환을 활성화하고 비활성화합니다. 변수 강제 변환 활성화와 변수 강제 변환 비활성화 버튼을 사용하여, 프로그램적으로 타겟에서 강제 변환을 활성화하고 비활성화합니다.

다음 단계에 따라 타겟에 있는 모든 I/O 변수를 동시에 강제 변환합니다:

1. 개별적으로 I/O 변수를 개별적으로 강제 변환시킬 필요가 없이, 원하는 I/O 변수의 강제 변환 값을 개별적으로 설정할 수 있도록 타겟에서 강제 변환을 비활성화합니다.
2. 타겟에 있는 I/O 변수 모두에 대해 원하는 강제 변환 값을 설정합니다.
3. 타겟에서 강제 변환을 활성화하여, 타겟에 있는 모든 I/O 변수를 동시에 강제 변환합니다.

## 원격으로 I/O 변수에 접근하기

I/O 변수에서 네트워크 공개가 활성화되어 있는 경우, I/O 변수를 호스트하는 타겟과 같은 네트워크에 연결되어 있는 원격 컴퓨터에서 그 I/O 변수에 접근할 수 있습니다. NI 배포된 시스템 관리자에서 **I/O 변수 프로브** 보기를 사용하여 대화식으로 원격 I/O 변수에 접근할 수 있습니다. 또한, **VI에서** 프로그램적으로 원격 I/O 변수에 접근할 수 있습니다.

공유 변수 프로퍼티 대화 상자를 사용하여 I/O 변수에서 네트워크 공개를 활성화 또는 비활성화합니다. 호스트 컴퓨터에서 I/O 값을 모니터링해야 하거나, 원격 타겟에서 I/O 변수에 접근할 필요가 있는 경우에는 네트워크 공개를 활성화합니다. 스캔 엔진 페이지를

사용하여 타겟에 있는 모든 I/O 타겟에 대해 글로벌 네트워크 공개 속도를 설정할 수 있습니다.



#### 노트

I/O 변수에 원격으로 접근할 계획이 없는 경우, 네트워크 공개를 비활성화하여 CPU 오버헤드를 최소화하십시오. I/O 변수 또는 I/O 가명에서 네트워크 공개를 비활성화하면, 다른 컴퓨터에서 실행 중인 VI에 있는 변수에 접근할 수 없습니다. 하지만, NI 배포된 시스템 관리자에서 변수를 강제 변환할 수 있습니다. 사용자가 I/O 변수를 강제 변환하지 못하도록 방지하려면, 시스템 관리자의 보안 설정을 사용합니다.

### 로컬에서 I/O 변수에 접근하기

LabVIEW는 I/O 변수를 글로벌 스캔 엔진 메모리 맵에 추가하고 동시에 모든 I/O 변수의 값을 업데이트합니다. 하지만, I/O 변수를 호스트하는 로컬 타겟에서의 I/O 변수 접근에 대해 각 노드가 스캔 접근 또는 다이렉트 접근 중에 하나를 사용하도록 설정할 수 있습니다.

다음 단계에 따라 I/O 변수 노드의 로컬 접근 모드를 선택합니다:

1. 블록 다이어그램에 있는 I/O 변수 노드에서 마우스 오른쪽 버튼을 클릭합니다.
2. 바로 가기 메뉴에서 로컬 접근 모드를 선택합니다.
3. 노드에 대해 사용하려는 접근 모드를 선택합니다.



#### 노트

각 변수에 대해 사용하는 접근 모드와는 상관 없이, LabVIEW는 모든 I/O 변수를 글로벌 스캔 엔진 메모리 맵에 추가합니다.

### 로컬 접근 모드 선택하기

일반적으로 스캔 접근은 업데이트 속도가 비슷한 I/O 채널 그룹에 적합하고, 다이렉트 접근은 스캔 주기에 대해 비동기적으로 업데이트되는 개별 I/O 채널에 적합합니다. 다음 테이블은 각 I/O 접근 방법을 사용하는 경우에 대한 요약 정보를 보여줍니다:

| 일반적인 용도                                                                                           | 접근 방법 |
|---------------------------------------------------------------------------------------------------|-------|
| 공통된 업데이트 속도를 갖는 I/O 채널에 대한 단일 포인트 접근                                                              | 스캔    |
| 긴급 정지 출력에 대한 단일 쓰기와 같이 (사용자 정의된 I/O 변수에서는 지원되지 않음), NI 스캔 엔진 주기에 대해 비동기적인 로컬 I/O 채널에 대한 단일 포인트 접근 | 다이렉트  |

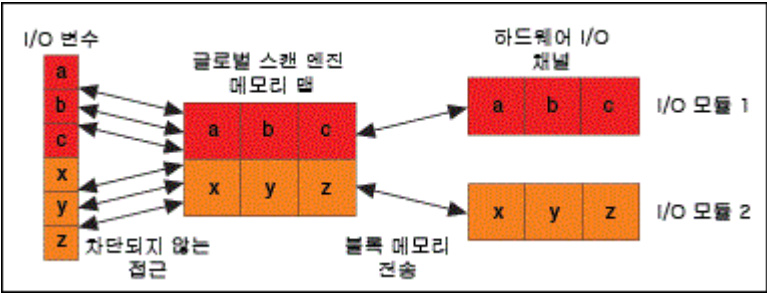


#### 노트

NI 스캔 엔진 주기보다 업데이트 속도가 느린 I/O 채널에 대해 다이렉트 접근을 사용할 수 있습니다. 그러나 이런 경우에는 일반적으로 스캔 접근이 다이렉트 접근보다는 같은 I/O 동작을 더 효율적으로 제공합니다.

스캔 접근

LabVIEW는 I/O 변수 노드를 설정하여 스캔 접근을 사용하도록 기본설정됩니다. 같은 속도로 업데이트 되는 I/O 채널 세트 및 확장 I/O 채널에 대해 스캔 접근을 사용합니다. 스캔 접근은 스캔 엔진 메모리 맵을 사용하여, 다음 그림에서 보이는 것처럼, 무차단형 I/O 읽기와 쓰기를 수행합니다:



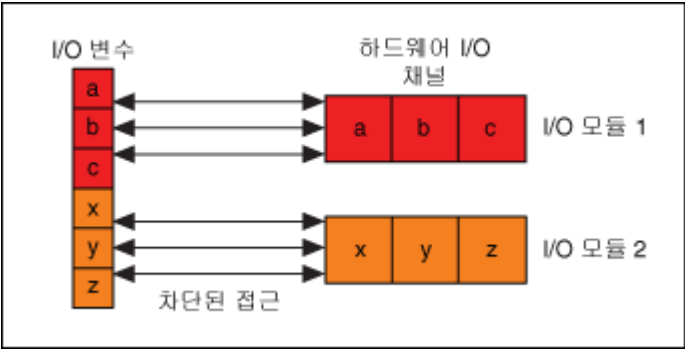
스캔 접근을 사용하여 I/O 변수에서 읽을 때마다, NI 스캔 엔진은 새 값을 기다리거나 차단하지 않고 메모리 맵에 저장된 최신 값을 즉시 반환합니다. 각 스캔에서, LabVIEW는 최신 I/O 값을 읽고 이 값을 NI 스캔 엔진 메모리 맵에 씁니다.

스캔 접근을 사용하여 I/O 변수에 대해 쓰기 작업을 취할 때마다, 스캔 엔진 메모리 맵에 저장된 이전 값을 덮어쓰게 됩니다. 각 스캔이 진행될 때마다, LabVIEW는 메모리 맵에 저장된 값을 물리적인 I/O 채널에 밀어넣습니다. 데이터 손실을 막으려면, 모든 I/O 변수 쓰기 작업을 스캔 주기에 동기화해야 합니다.

다이렉트 접근

스캔 주기에 대해 비동기적으로 단일 포인트 로컬 I/O 채널을 읽거나 쓰려면 다이렉트 접근을 사용합니다. 어플리케이션이 가능한 빨리 값을 써야 하는 경우 다이렉트 접근을 사용할 수 있습니다. 다음 그림에서 보이는 것처럼, 다이렉트 접근은 스캔 엔진 메모리 맵을 건너뛰고 I/O 디바이스 드라이버와 직접 통신하여 비동기식(Non-Blocking) I/O 읽기와 쓰기 작업을 수행합니다:

 **노트**  
사용자 정의 I/O 변수는 다이렉트 접근을 지원하지 않습니다.





**노트**

NI 스캔 엔진을 RIO Scan Interface와 함께 사용하는 경우, 하드웨어 스캔 신호가 I/O에 대한 업데이트를 제어합니다. 하드웨어 스캔 신호의 유효 상태에서 다이렉트 접근을 통해 값을 쓰면, 이 값이 하드웨어 스캔 신호가 끝난 후에 하드웨어로 전달됩니다.

**노트**

컨트롤러, I/O 모듈, 통신 프로토콜에 따라 다이렉트 접근의 속도는 다릅니다. I/O 접근 속도에 대한 추가적인 정보는 관련 하드웨어 문서를 참조하십시오.

## I/O 변수 노드 아이콘 해석하기

블록다이어그램에 있는 각 I/O 변수 노드는 다음에 나열된 접근 방법 중 한 개를 나타내는 문양을 가집니다.

| 접근 방법  | 노드 모양 |
|--------|-------|
| NI-PSP |       |
| 스캔     |       |
| 다이렉트   |       |

**데이터 통신 방법**

## 스캔 I/O 데이터 접근하기 (Real-Time, Windows)

NI 스캔 엔진을 사용하여 스캔 I/O 데이터에 대한 단일 포인트에 접근합니다.

## NI 스캔 엔진 사용하기

NI 스캔 엔진은 글로벌 메모리 맵에 데이터를 저장하고 스캔 주기라고 하는 단일 속도로 모든 값을 업데이트하는 스캔을 사용하여, I/O 채널과 같은 데이터 채널 세트에 대한 효율적 **단일 포인트** 접근을 가능하게 합니다. NI 스캔 엔진은 기본적으로 Windows용 LabVIEW에 설치됩니다. 또한, RT Series PXI와 CompactRIO 타겟과 같은 지원되는 LabVIEW 타겟에 NI 스캔 엔진을 설치할 수 있습니다. NI 스캔 엔진 지원에 대한 정보는 관련 하드웨어 디바이스 문서를 참조하십시오.

(Real-Time Module) LabVIEW가 CPI 사용 통계를 리포트할 때 스캔 스레드를 시간 결정적인 항목에 포함시키지만, 기본적으로 RT 타겟에서 NI 스캔 엔진은 시간 결정적인 우선순위보다 높은 스레드에서 실행됩니다. NI 스캔 엔진을 시간 결정적인 우선순위보다 높게 실행할 때, 어플리케이션의 확정적(deterministic)인 섹션을 스캔 주기와 **동기화**해야 합니다. 이를 통해 스캔 스레드가 어플리케이션의 확정성에 영향을 미치지 않도록 보장할 수 있습니다. I/O가 가장 우선순위가 높은 태스크가 아닌 어플리케이션에 대해서도 스캔 엔진의 우선 순위가 시간 결정적인 우선 순위와 Timed 구조의 우선 순위 사이에 있도록 설정할 수 있습니다. 타겟에서 I/O 변수를 사용할 계획이 없는 경우, 타겟에 NI 스캔 엔진

을 설치하지 마십시오. 이미 타겟에 NI 스캔 엔진이 설치되어 있는 경우, NI Measurement & Automation Explorer (MAX)를 사용하여 NI 스캔 엔진을 제거할 수 있습니다.



#### 노트

(Windows) Windows에서 사용 가능한 NI 스캔 엔진 우선순위는 표준 우선순위가 유일합니다. Windows OS의 비확정적(non-deterministic)인 특성 때문에, Windows 상에서 실행 중일 때 NI 스캔 엔진의 타이밍은 보장되지 않습니다.

## NI 스캔 엔진 셋팅 설정하기

스캔 엔진 페이지를 사용하여 NI 스캔 엔진의 스캔 주기, 네트워크 공개 속도, 우선 순위 레벨을 포함하는 스캔 엔진 셋팅을 설정할 수 있습니다.

NI 배포된 시스템 관리자를 사용하여 스캔 엔진의 **결합**과 **모드**를 모니터하고 관리합니다. 도구 >> 배포된 시스템 관리자를 선택하여 NI 배포된 시스템 관리자를 디스플레이합니다. 또한, NI 스캔 엔진 VI를 사용하여 프로그램적으로 스캔 엔진 셋팅을 보고 설정할 수 있습니다.

## NI 스캔 엔진 배포하기 및 실행하기

프로젝트 탐색기 윈도우에서 NI 스캔 엔진 셋팅을 설정한 다음, 내 컴퓨터나 RT 타겟처럼 NI 스캔 엔진을 사용하는 프로젝트 아이템을 배포해야 이 셋팅이 적용됩니다.

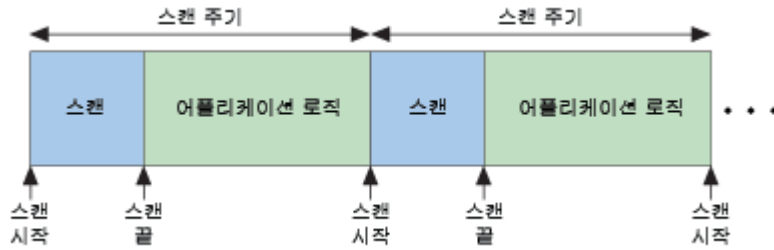
(Windows) 사용자가 타겟을 처음으로 배포할 때 NI 스캔 엔진이 실행을 시작하고, 사용자가 프로젝트를 닫으면 실행을 멈춥니다. 빌드 어플리케이션이 실행될 때 NI 스캔 엔진과 I/O 변수를 자동으로 배포하려면, 어플리케이션을 만들기 전에 어플리케이션 프로퍼티 대화 상자의 고급 페이지에 있는 타겟 하드웨어 설정 포함 확인란에 확인 표시를 해야 합니다.

(Real-Time Module) RT에서 NI 스캔 엔진을 실행하면, 사용자가 타겟을 부팅할 때 NI 스캔 엔진은 자동으로 시작하며 사용자가 타겟을 종료할 때까지 계속 실행됩니다.

하나의 컴퓨터는 한 번에 오직 하나의 NI 스캔 엔진 인스턴스만을 실행할 수 있습니다. 예를 들어, NI 스캔 엔진을 사용하는 Windows 빌드 어플리케이션을 생성하는 경우에는 어플리케이션을 만든 프로젝트를 닫아야만 이후에 같은 컴퓨터에서 어플리케이션을 실행할 수 있습니다. 여러 LabVIEW 프로젝트에서 여러 NI 스캔 엔진 인스턴스를 설정할 수 있지만, NI 스캔 엔진 인스턴스가 실행 중인 경우에는 NI 스캔 엔진을 사용하는 다음 프로젝트의 배포가 실패하게 됩니다. 다른 NI 스캔 엔진 인스턴스를 실행하려면, 먼저 현재의 인스턴스를 종료해야 합니다.

## 스캔 엔진 타이밍

NI 스캔 엔진은 사용자가 스캔 엔진 페이지에서 지정한 스캔 주기에 따라 결정되는 일정한 간격으로 실행됩니다. 다음 그림에서 보이는 것처럼, 스캔 자체와 어플리케이션 로직을 모두 처리할 수 있을 정도로 충분한 주기를 선택하십시오:



#### 노트

스캔에 소요되는 시간은 타겟에 배포된 I/O 아이템의 개수 및 타입에 따라 달라집니다. 스캔 엔진의 성능을 최대화하려면, 어플리케이션에서 사용할 계획이 없는 모든 I/O 아이템의 배포를 취소하십시오.

스캔 주기를 프로그램적으로 읽으려면, [스캔 엔진 주기 얻기] VI를 사용합니다. 스캔 주기를 프로그램적으로 설정하려면, [스캔 엔진 주기 설정하기] VI를 사용합니다.



#### 노트

(Real-Time Module) 일반적으로 리얼타임 루프는 한두번의 예비 반복을 수행한 후에야 확정적(deterministic)으로 실행을 시작할 수 있습니다. 어플리케이션이 타이밍 요구조건을 충족하는지 점검하기 이전에, 각각의 시간 결정적인 루프가 예비 반복을 실행할 수 있도록 허용해야 합니다.

### 스캔 엔진에 동기화하기

(Real-Time Module) NI 스캔 엔진이 시간 결정적인 우선순위보다 낮은 우선순위로 실행 되도록 설정하지 않는 한, 시간 결정적인 코드를 스캔 주기에 동기화시켜야 어플리케이션의 확정성에 영향을 미치는 충돌을 방지할 수 있습니다.

Timed 구조의 실행을 스캔 엔진과 동기화하려면, 스캔 엔진에 동기화 타이밍 소스를 사용합니다. Timed 구조를 사용하지 않으려는 경우, [스캔 엔진에 동기화] VI를 사용하여 스캔 엔진에 동기화할 수 있습니다. 두 동기화 방법 모두, 앞의 그림에서 스캔 끝이라는 라벨이 붙은 시간에 실행을 트리거합니다. I/O 변수를 일관성 있는 데이터 세트로 사용하려면, 다음 스캔 반복 이전에 동기화된 코드가 실행을 마치도록 보장해야 합니다. 하지만, 코드가 일관성 있는 데이터 세트에 의존하지 않는 경우에는 스캔 반복을 건너뛸 수 있습니다.



#### 노트

동기화된 코드가 다음 스캔 반복 이전에 실행을 완료하지 않는 경우, I/O 변수의 에러 클러스터에 의해 리포트된 정보는 I/O 값과의 동기화 상태를 지속하지 못할 수 있습니다.



#### 노트

NI 스캔 엔진 동기화와 관련된 VI에서 강제 종료 버튼을 클릭할 때, 현재의 스캔 반복이 완료될 때까지 VI는 강제 종료하지 않습니다. 따라서, 스캔 주기가 길은 경우에는 VI가 일시적으로 지연되는 것처럼 보일 수 있습니다.

NI 스캔 엔진 모드



**노트**  
오직 특정 타겟과 디바이스가 NI 스캔 엔진 모드를 사용합니다. 로컬 I/O 모듈을 가진 CompactRIO 타겟은 NI 스캔 엔진 모드를 사용하지 않습니다. 하드웨어 특정의 모드 동작에 대한 정보는 해당 I/O 하드웨어 문서를 참조하십시오.

LabVIEW가 정의하는 NI 스캔 엔진 모드는 4가지입니다:

|   |        |                                                            |
|---|--------|------------------------------------------------------------|
| 0 | 초기화 모드 | 시작 중일 때에만 잠시 발생합니다.                                        |
| 1 | 설정 모드  | NI 스캔 엔진 모드를 사용하는 하드웨어에서 스캔 엔진 셋팅을 설정할 때 반드시 사용해야하는 모드입니다. |
| 2 | 활성 모드  | 스캔 엔진이 실행되고 값을 업데이트하는 모드입니다.                               |
| 3 | 결함 모드  | 복구 불가능하거나 주요 결함이 발생할 때 트리거되는 모드입니다.                        |



**노트**  
타겟에 따라 결함 모드의 동작이 달라집니다. 결함 모드 동작에 대한 정보는 특정 타겟 하드웨어 문서를 참조하십시오.

NI 배포된 시스템 관리자를 통해 스캔 엔진 모드를 보고 설정합니다. NI 배포된 시스템 관리자는 EngineState 아이টে을 사용하여 스캔 엔진 모드의 값을 디스플레이합니다. 스캔 엔진 모드를 프로그램적으로 읽으려면, [스캔 엔진 모드 읽기] VI를 사용합니다. 스캔 엔진 모드를 프로그램적으로 설정하려면, [스캔 엔진 모드 설정하기] VI를 사용합니다.

 데이터 통신 방법 **훅**

NI 스캔 엔진 결함 (Real-Time, Windows)

NI 스캔 엔진이 설치된 타겟은 결함 기능을 사용하여 비동기화 에러 조건을 처리합니다. LabVIEW는 3가지 단계로 결함을 정의합니다: 보조, 주요, 회복할 수 없음. LabVIEW는 모든 결함을 타겟의 메모리에 로그합니다. NI 배포된 시스템 관리자에서 **스캔 엔진 프로브** 보기를 사용하여 타겟의 NI 스캔 엔진 결함을 보고 지웁니다.

보조 결함은 경고와 같은 레벨이며, 이로 인해 결함 처리 동작이 시작되지는 않습니다. 보조 결함의 예로 시작 에러를 들 수 있는데, 이는 컨트롤러를 시작할 때 저장된 설정을 적용할 수 없을 때 발생하는 결함입니다.

주요 결함은 에러와 같은 레벨입니다. 예를 들어, 스캔 엔진이 늦게 실행되었음을 LabVIEW가 감지하는 경우, LabVIEW는 주요 결함 -66460을 트리거합니다. 10개의 스캔 반복이 연속적으로 늦게 실행되었음을 LabVIEW가 감지하는 경우, LabVIEW는 주요 결함 -66461을 트리거하고 NI 스캔 엔진은 실행을 정지합니다.



**노트**  
일부 I/O 하드웨어 드라이버는 주요 결함에 대한 응답으로 결함 처리 동작을 시작합니다. 결함 처리 동작에 대한 정보는 특정 하드웨어 문서를 참조하십시오.

복구할 수 없음 결함은 하드웨어 오류 또는 소프트웨어 충돌로 인해 발생할 수 있습니다. 복구할 수 없음 결함이 발생한 경우, 컨트롤러를 재부팅하고 National Instruments에 연락하십시오.

## 결함 보기 및 삭제하기

NI 배포된 시스템 관리자를 사용하여 **결함을 보고 삭제**할 수 있습니다. 또한, [결함 리스트 열기] VI와 [결함 지우기] VI를 사용하여 프로그램적으로 결함을 보고 지울 수 있습니다. 결함을 삭제하면 메모리에서도 지워지며, 마치 그 결함이 전혀 발생하지 않은 것처럼 됩니다.

## 시스템 결함

LabVIEW는 일반적인 결함 세트를 정의합니다. LabVIEW 에러 클러스터를 기반으로, 사용자가 추가적인 결함을 로그할 수 있습니다. 다음 테이블에서 보이는 것처럼, LabVIEW는 타입별로 결함을 분류합니다:



### 노트

결함 처리 동작에 대한 정보는 특정 하드웨어 문서를 참조하십시오.

| 결함 타입          | 결함 코드의 범위       |
|----------------|-----------------|
| I/O 스캔 드라이버 에러 | -66000에서 -66099 |
| I/O 변수         | -66200에서 -66299 |
| NI 스캔 엔진 VI    | -66300에서 -66399 |
| NI 스캔 엔진       | -66400에서 -66499 |
| 사용자 결함         | LabVIEW 에러 코드   |

## 결함 트리거하기

[결함 셋팅] VI를 사용하여 LabVIEW 에러 코드를 기반으로 보조 결함과 주요 결함을 트리거할 수 있습니다. **사용자 에러 코드를 정의**한 다음에 그 코드를 [결함 셋팅] VI에 전달하여, 사용자 결함을 트리거할 수 있습니다.

LabVIEW는 최대 100개의 고유한 결함 코드를 로그할 수 있는 정도의 충분한 메모리 풀을 확보해놓고 있습니다. 고유 결함 코드의 개수가 100 개를 초과하는 경우, LabVIEW는 보조 결함 -66420을 트리거하고 추가 결함 로그 동작을 중지합니다. 하지만, 사용자 결함의 최대 개수에 도달하더라도, 여전히 주요 결함 또는 복구할 수 없음 결함은 하드웨어 드라이버가 결함 처리 동작을 시작하도록 트리거할 수 있습니다.

## 결함 설정하기

사용자 결함 설정을 통해 특정 결함에 대한 레벨 및 트리거 동작을 사용자 정의할 수 있습니다. 대화식 또는 프로그램적으로 사용자 결함 설정을 적용할 수 있습니다.

사용자 결합 설정을 대화식으로 적용하려면 스캔 엔진 페이지의 결합 설정 섹션을 사용합니다. 프로그램적으로 사용자 결합 설정을 적용하려면, [결합 설정] VI를 사용합니다.

일부 결합은 설정할 수 없습니다. 하지만, 설정 가능한 결합의 리스트는 설치된 NI 스캔 엔진 I/O 소프트웨어에 따라 달라집니다. 대화식으로 설정 가능한 결합의 리스트를 보려면, 스캔 엔진 페이지의 결합 설정 섹션에 있는 설정 가능한 결합 리스트를 사용합니다.

## I/O 변수 컨테이너를 배열로 읽고 쓰기(Real-Time, Windows)

I/O 변수 컨테이너를 단일 배열로 접근하면, 성능을 최적화하고, 더 깔끔한 블록다이어그램을 만들 수 있습니다. 공유 변수 함수를 사용하여 로컬 I/O 변수 컨테이너를 배열로 **읽고 쓸 수** 있습니다.

### I/O 변수 컨테이너 지정하기

I/O 변수 컨테이너는 LabVIEW 프로젝트에서 **I/O 변수**를 가지고 있는 아이템입니다. 다음의 방법 중 하나를 택해 블록다이어그램에서 읽거나 쓸 I/O 변수 컨테이너를 지정합니다:

- 다음 방법 중 하나를 사용해서 변수 참조 번호 컨트롤이나 상수로 I/O 변수 컨테이너를 지정합니다.
  - I/O 변수 컨테이너 **프로그램적으로** 찾기
  - 블록다이어그램에서 **URL 문자열** 만들기
  - 상수에 컨테이너의 URL 입력하기
  - 변수 객체 탐색 대화 상자를 사용하여 컨테이너를 대화식으로 찾기
- I/O 변수 컨테이너를 **프로그램적으로** 찾기.

### I/O 변수 컨테이너 읽기와 쓰기

I/O 변수 컨테이너에 대한 참조를 다음 함수 중 하나의 공유 변수 참조 번호 입력 입력에 연결할 수 있습니다:

- 변수 읽기
- 변수 쓰기
- 직접 변수 읽기
- 직접 변수 쓰기
- 스캔된 변수 읽기
- 스캔된 변수 쓰기

앞의 리스트에 있는 읽기 함수 중 한 개를 사용하여 I/O 변수 컨테이너를 읽으면, 그 함수는 I/O 컨테이너에 있는 각 채널의 데이터를 가진 배열을 반환합니다. 배열 안의 채널 순서는 모든 변수 프로퍼티에 의해 반환되는 채널의 순서와 일치합니다. 이전의 리스트에 있는 쓰기 함수를 사용하여 I/O 변수 컨테이너를 쓸 때는 반드시 배열이 이 순서와 같게 인덱스되도록 연결해야 합니다.

예를 들어, 다음의 블록다이어그램은 [스캔된 변수 읽기] 함수를 사용하여 아날로그 입력 모듈 Mod1의 모든 I/O 채널을 배정도의 배열로 읽습니다:



### 지원되지 않는 기능

이 토픽에서 설명된 기법은 다음에 열거된 기능을 지원하지 않습니다.

| 지원되지 않는 기능                         | 세부사항                                                                                                   |
|------------------------------------|--------------------------------------------------------------------------------------------------------|
| 여러 데이터 타입을 포함하는 I/O 변수 컨테이너 읽고, 쓰기 | I/O 변수 컨테이너의 데이터 타입은 한 개만 지정할 수 있습니다. 그러므로, 여러 데이터 타입을 가진 채널로 I/O 변수 컨테이너를 읽고, 쓰면, 에러가 발생합니다.          |
| PSP 변수 컨테이너 읽기와 쓰기                 | 이 기법은 로컬 I/O 변수 컨테이너에만 적용 가능합니다. 이 기법을 사용하여 원격으로 I/O 변수 컨테이너에 접근하거나, 네트워크 공유 변수 라이브러리에 접근하는 것은 불가능합니다. |
| 타임스탬프 읽기                           | 이 기법을 사용하는 동안에는 타임스탬프를 활성화할 수 없습니다.                                                                    |

## 데이터 통신 방법 홈

# PID와 퍼지 로직

## PID and Fuzzy Logic

Use PID and Fuzzy Logic VIs for Proportional-Integral-Derivative (PID) and fuzzy logic control. These VIs appear under the Control & Simulation palette. You can use the PID VIs with input/output (I/O) functions such as data acquisition (DAQ) to implement control of physical processes.

### PID Control

The Proportional-Integral-Derivative (PID) algorithm is the most common control algorithm used in industry. A PID controller is a feedback controller you can use to control processes such as heating and cooling systems, fluid level monitoring, flow control, pressure control, and so on. You can use the PID VIs with National Instruments hardware such as a DAQ device, FieldPoint I/O module, or GPIB board, to connect your PC to a system

and develop LabVIEW control applications. You also can use the PID VIs with LabVIEW math and logic functions to create block diagrams for real control strategies.

## Fuzzy Logic

Fuzzy logic is a method of rule-based decision making for expert systems and process control. Fuzzy logic differs from traditional Boolean logic in that fuzzy logic allows for partial membership in a set. You can use fuzzy logic to control processes represented by subjective, linguistic descriptions. You can wire acquired data to a fuzzy controller to implement real-time decision making or control of a physical system. You also can use outputs of the fuzzy controller with DAQ analog output hardware to implement real-time process control.

Use the Fuzzy System Designer to design a fuzzy system interactively. Use the Fuzzy Logic VIs to design and control fuzzy systems programmatically.



### RELATED INFORMATION

[PID Control](#) on page 1693

[Fuzzy Logic](#) on page 1685

[PID Control](#) on page 1693

[Fuzzy Logic](#) on page 1685

## Tutorial: Using the Fuzzy System Designer

You can use the PID and Fuzzy Logic VIs to design a [fuzzy system](#) in one of two ways: using the Fuzzy System Designer or using the Fuzzy Logic VIs. This tutorial teaches you to design a fuzzy system interactively using the Fuzzy System Designer.



Refer to the [Modifying a Fuzzy System with the Fuzzy Logic VIs](#) tutorial to learn how to design and modify fuzzy systems using the Fuzzy Logic VIs.

In this tutorial, you use the Fuzzy System Designer to design a fuzzy system similar to the vehicle maneuvering example described in the topics on [fuzzy system design](#). As a reference, you can find the complete fuzzy system you design in this tutorial, `Truck - backward.fs`, in the `labview\examples\control\fuzzy\Car Parking` directory.

Next [1. Creating Input Linguistic Variables](#)



## Creating Input Linguistic Variables

The first step in designing a fuzzy system with the Fuzzy System Designer is to create the input and output [linguistic variables](#) for the system. In the vehicle maneuvering example



described in the topics on [fuzzy system design](#), the fuzzy system has two linguistic input variables, Vehicle Position  $x$  and Vehicle Orientation  $\beta$ , and one linguistic output variable, Steering Angle  $\phi$ .



#### TIP

The graphs in the [Creating Membership Functions](#) topic illustrate the range and membership functions for each of these linguistic variables.

Complete the following steps to create an input linguistic variable that corresponds to the Vehicle Position  $x$  input linguistic variable of the vehicle maneuvering example.

1. Select Tools»Control Design and Simulation»Fuzzy System Designer to launch the Fuzzy System Designer and display the Variables page.
2. Click the Add Input Variable button to the right of the Input variables list to display the Edit Variable dialog box.
3. In the Name text box, enter `vehicle-position`.
4. Specify the Range by entering 0 in the minimum field and 10 in the maximum field. This specifies that the position of the vehicle relative to the destination ranges from 0.0 to 10.0 meters.
5. Click the Add Membership Function button under the Membership functions control to enable the control and create a new membership function for the vehicle-position linguistic variable.
6. In the Name text box, enter `left`.
7. Select Trapezoid from the Shape pull-down menu to specify that a trapezoid function determines the degrees of membership for the linguistic variable.
8. Click the Color field to select the color to use for the membership function in the Membership functions graph.
9. In the Points array, enter the following numbers, one for each element of the array: 0, 0, 1, 4. These points specify the values of the linguistic variable corresponding to the base and top points, in order from left to right and base to top, of the membership function.
10. Repeat steps 5 through 9 for each of the following membership functions for Vehicle Position  $x$ , using the shape and points shown in the following table:

| Membership Function | Shape        | Points |
|---------------------|--------------|--------|
| Triangle            | 3, 4, 5      |        |
| Triangle            | 4.5, 5, 5.5  |        |
| Triangle            | 5, 6, 7      |        |
| Trapezoid           | 6, 9, 10, 10 |        |



#### NOTE

The Points array requires that each point for a given membership function be greater than or equal to all previous points. If the array automatically changes your entries to fit this requirement, try entering points from left to right.

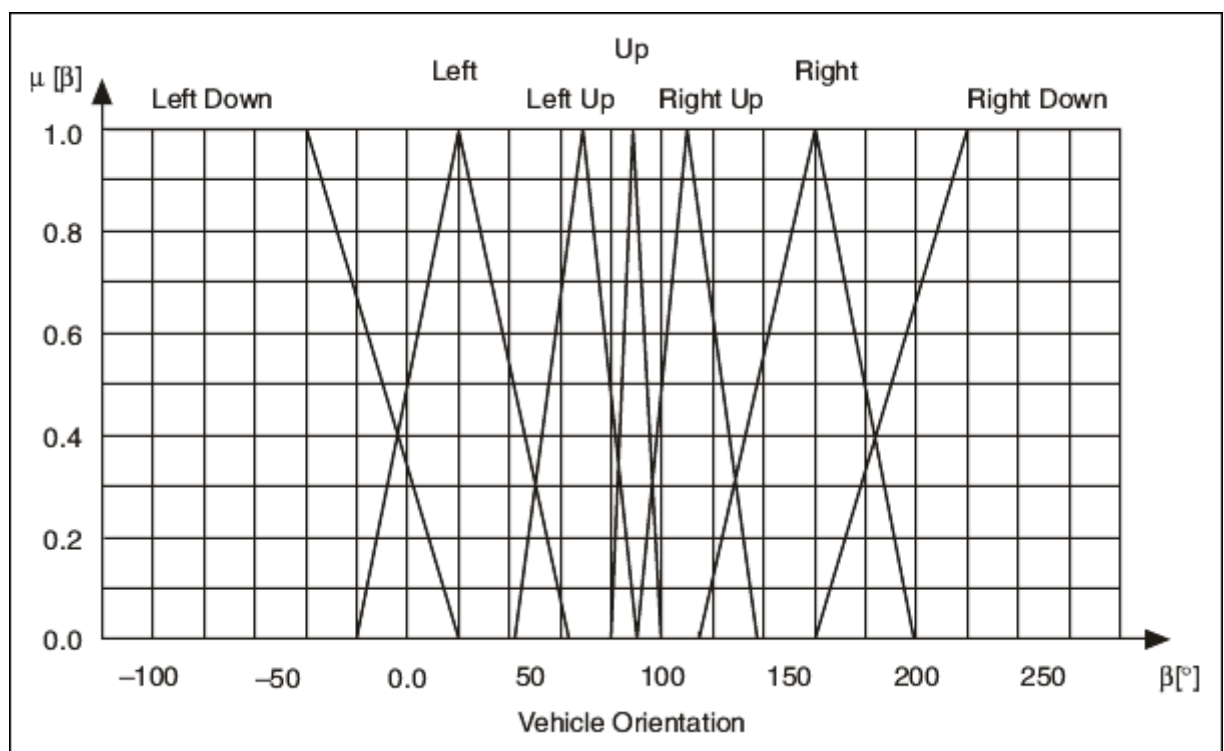
You can compare the Membership functions graph to the [Vehicle Position](#) graph in the Creating Membership Functions topic to confirm your entries.

11. Click the OK button to save your changes and return to the Variables page. The vehicle-position input variable appears in the Input variables list and the corresponding membership functions appear in the Input variable membership functions graph.
12. Repeat steps 2 through 11 to create a vehicle-orientation input linguistic variable that corresponds to the Vehicle Orientation  $\beta$  variable of the vehicle maneuvering example. The following graph shows the range and membership functions for the variable.



#### NOTE

For the purposes of this tutorial, you can use approximate values for the points of the membership functions.

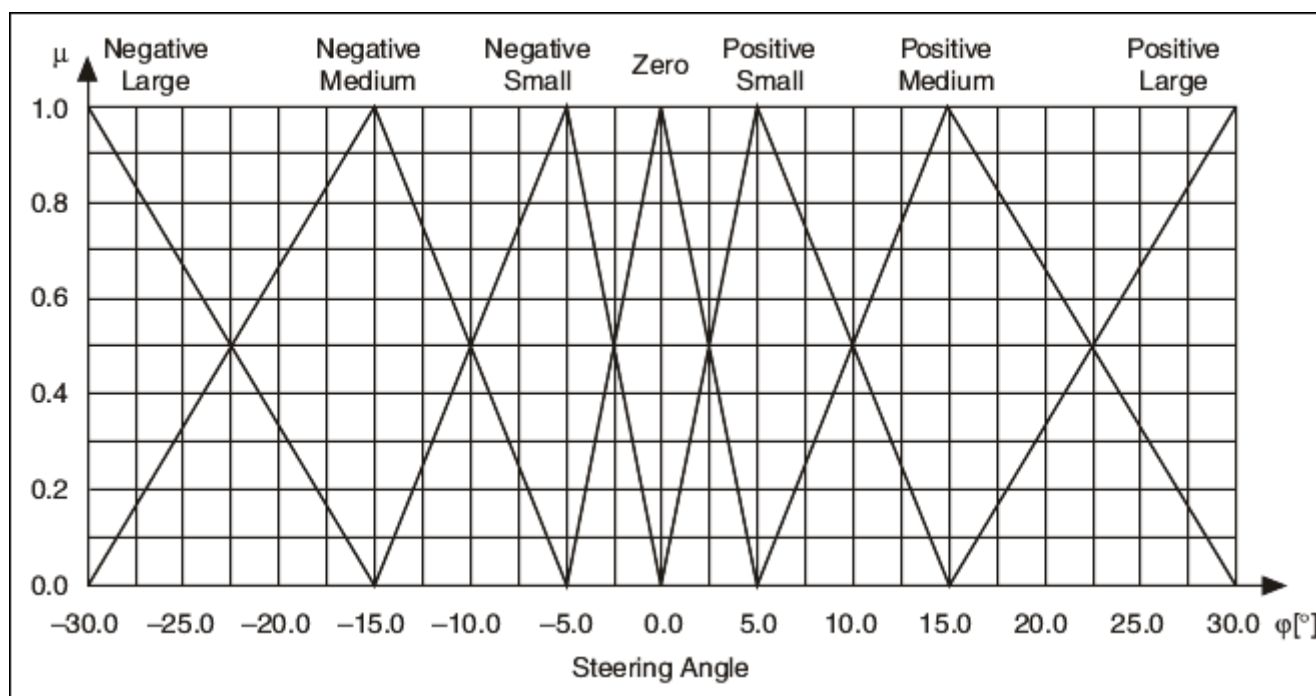


## Creating Output Linguistic Variables

You create output linguistic variables similarly to how you create input linguistic variables in the Fuzzy System Designer. Complete the following steps to create an output linguistic variable that corresponds to the Steering Angle  $\phi$  output linguistic variable of the vehicle maneuvering example.

1. Click the Add Output Variable button to the right of the Output variables list to display the Edit Variable dialog box.
2. Enter steering-angle in the Name text box.

3. Enter a minimum value of -30 and a maximum value of 30. This range specifies that the amount that the steering wheel can turn ranges from -30.0 to 30.0 degrees.
4. Click the Add Membership Function button to create a new membership function for the steering-angle output variable.
5. Enter neg-large in the Name text box.
6. Select Triangle from the Shape pull-down menu to specify that a triangle function determines the degrees of membership for the linguistic variable.
7. Select the color you want to use for the membership function in the Membership functions graph from the Color picker.
8. Enter -30, -30, and -15 in the fields of the Points array. These points specify the values of the linguistic variable corresponding to the base and top points, in order from left to right and base to top, of the membership function.
9. Repeat steps 4 through 8 to create each of the membership functions shown in the following graph.



10. Click the OK button to save your changes.
11. Select File»Save As to save the fuzzy system as `vehicle_maneuvering.fs` in an easily accessible location.

## Creating a Rule Base with Pre-Generated Rules

Rules describe, in words, the relationships between input and output **linguistic variables** based on their linguistic terms. The **rule base** of a fuzzy system determines the output values of the fuzzy system based on the input values.

The vehicle-position input linguistic variable you created in [step 1](#) has five membership functions, and the vehicle-orientation input linguistic variable has seven. Therefore, you can construct rules to associate 35 possible combinations of the linguistic

terms of these input linguistic variables with the linguistic terms of the steering-angle output linguistic variable you created in [step 2](#).



#### NOTE

The table in the [Complete Rule Bases](#) topic illustrates this rule base.

Complete the following steps to create the complete rule base for the vehicle maneuvering example. You must have created [both input linguistic variables](#) and the [output linguistic variable](#) before creating this rule base.

1. Click the Rules tab of the Fuzzy System Designer to display the Rules page.
2. Select Operate»Pre-Generate Rules to display the Pre-Generate Rules dialog box.
3. Select AND (Minimum) from the Antecedent connective pull-down menu. This [antecedent connective](#) specifies that the smallest degree of membership of the antecedents determines the truth value of the aggregated rule antecedent for each rule.
4. Enter a Degree of support of 1 to specify the weight that you want to apply to each rule. The final rule weight for each rule is equal to the Degree of support multiplied by the truth value of the aggregated rule antecedent.
5. Select Minimum from the Consequent implication pull-down menu to specify that the fuzzy logic controller truncates the output membership functions at the value of the corresponding rule weights before performing [defuzzification](#). Defuzzification is the process of converting the degrees of membership of output linguistic variables into numerical values.
6. Click the OK button to return to the Rules page. The Fuzzy System Designer populates the Rules list with all 35 combinations of linguistic terms of the input variables. Each rule uses the same antecedent connective, degree of support, and implication method that you specified in the Pre-Generate Rules dialog box.

Notice that the consequent of each rule in the Rules list is THEN 'steering-angle' IS 'neg-large'. However, the linguistic term of the steering-angle output variable depends on the combination of linguistic terms of the input variables. Therefore, you must modify the pre-generated rules to use the correct consequents.

## Modifying Pre-Generated Rules

Complete the following steps to modify the pre-generated rules to use the correct consequents.

1. Select the first rule in the Rules list to display an IF THEN diagram that shows the Antecedents and Consequents of the rule.
2. In the THEN column, select neg-small as the linguistic term for the steering-angle output linguistic variable. Notice the consequent of the rule in the Rules list changes to THEN 'steering-angle' IS 'neg-small'.
3. Repeat steps 1 and 2 for each rule using the consequents in the following table:

| AND                             |            | Vehicle Position $x$ [m] |                 |                 |                 |                 |
|---------------------------------|------------|--------------------------|-----------------|-----------------|-----------------|-----------------|
|                                 |            | Left                     | Left Center     | Center          | Right Center    | Right           |
| Vehicle Orientation $\beta$ [°] | Left Down  | Negative Small           | Negative Medium | Negative Medium | Negative Large  | Negative Large  |
|                                 | Left       | Positive Small           | Negative Small  | Negative Medium | Negative Large  | Negative Large  |
|                                 | Left Up    | Positive Medium          | Positive Small  | Negative Small  | Negative Medium | Negative Large  |
|                                 | Up         | Positive Medium          | Positive Medium | Zero            | Negative Medium | Negative Medium |
|                                 | Right Up   | Positive Large           | Positive Medium | Positive Small  | Negative Small  | Negative Medium |
|                                 | Right      | Positive Large           | Positive Large  | Positive Medium | Positive Small  | Negative Small  |
|                                 | Right Down | Positive Large           | Positive Large  | Positive Medium | Positive Medium | Negative Small  |

4. Select File»Save to save the fuzzy system.

The fuzzy system now has a **complete linguistic rule base** with which to analyze input data and produce output data. However, the rule base returns the output data as linguistic terms. The fuzzy controller must defuzzify the output data before the output can apply to the control structure that the fuzzy system automates. You therefore must specify a defuzzification method for the fuzzy controller to use.

## Specifying a Defuzzification Method

After you create the rule base for a fuzzy system, you must specify how a fuzzy controller performs **defuzzification** for the system.

In the vehicle maneuvering example, you must supply a continuous output signal to control the steering angle of the vehicle. Therefore, you must use a defuzzification method that calculates the best compromise between any rules that apply at a given time. According to the guidelines for **selecting a defuzzification method** the **Center of Maximum (CoM)**, **Center of Area (CoA)**, and **Center of Sums (CoS)** defuzzification methods all fit this requirement.

Complete the following steps to specify the defuzzification method.

1. On the Rules page of the Fuzzy System Designer, select the method you want to use from the Defuzzification method pull-down menu.
2. Select File»Save to save the fuzzy system.

The fuzzy system is complete. However, before you integrate the fuzzy system into the control structure you want to automate, you must test the functionality of the system.

## Testing the Fuzzy System

You can test the relationship between the input and output values of a fuzzy system to validate the rule base of the fuzzy system.

Complete the following steps to test the vehicle maneuvering fuzzy system you created in the previous steps.

1. Click the Test System tab of the Fuzzy System Designer to display the Test System page.
2. Enter an Input value of 5 for the vehicle-position input linguistic variable. Recall from creating the input linguistic variables in [step 1](#) that a value of 5 for vehicle-position corresponds to the center linguistic term.
3. Enter an Input value of -30 for the vehicle-orientation input linguistic variable, which corresponds to the left-down linguistic term.
4. In the Invoked Rules table at the bottom of the page, notice that the fuzzy system invokes rule 15: IF 'vehicle-position' IS 'center' AND 'vehicle-orientation' IS 'left-down' THEN 'steering-angle' IS 'neg-medium'. Recall from creating the output linguistic variable in [step 2](#) that the neg-medium linguistic term for the steering-angle variable corresponds to the range between -30 and -5. If you configured the fuzzy system correctly, the Output value indicator for the steering-angle displays a value within that range. The fuzzy controller calculates the Output value based on the weight of the rule, the consequent implication method, and the defuzzification method.
5. (Optional) Use the Input variable 1 and Input variable 2 slide controls to sweep a range of values for the input linguistic variables and observe the corresponding changes in the Input/Output relationship graph. You can use this graph to verify that the rule base is reasonable and complete. For example, if the Input/Output relationship graph displays 0 at some points, the rule base might be incomplete.
6. Fix any errors you find while testing and save the fuzzy system.
7. Click the Close button to exit the Fuzzy System Designer.

## Controlling the Fuzzy System

After you create a fuzzy system in the Fuzzy System Designer, you can use the FL Fuzzy Controller VI to implement a controller for the fuzzy system. The FuzzyEx Car Backward

Parking VI, located in the `labview\examples\control\fuzzy\Car Parking` directory, illustrates how to implement a controller for a fuzzy system.



Open example



Find related examples

This example uses the FL Load Fuzzy System VI to load both the `Truck - forward.fs` and `Truck - backward.fs` files. These files represent the fuzzy systems for moving the vehicle forward and backwards, respectively. Each FL Load Fuzzy System VI passes a `.fs` file to a FL Fuzzy Controller VI, which implements a fuzzy controller for the fuzzy system.

The fuzzy system you created in this tutorial is similar to the `Truck - backward.fs` file. You can implement a fuzzy controller for the fuzzy system you created in the same fashion that the FuzzyEx Car Backward Parking VI implements a controller for the `Truck - backward.fs` file.

## Tutorial: Modifying a Fuzzy System with the Fuzzy Logic VIs

In the [Using the Fuzzy System Designer](#) tutorial, you design a fuzzy system interactively using the Fuzzy System Designer and then implement a fuzzy controller for that system using the FL Fuzzy Controller VI. You also can design a fuzzy system programmatically using the Fuzzy Logic VIs. However, in most cases, it is easier to use the Fuzzy System Designer to design a fuzzy system first and then use the Fuzzy Logic VIs to make programmatic modifications to the system. In this tutorial, you modify a fuzzy system programmatically using the Fuzzy Logic VIs.

This tutorial uses the example of a greenhouse control system to illustrate how to modify a fuzzy system using the Fuzzy Logic VIs. The `greenhouse.fs` file, located in the `labview\examples\control\fuzzy\Dynamic greenhouse controller` directory, represents the fuzzy system that you modify in this chapter.

Next [1. Observing the Fuzzy System](#)



## Observing the Fuzzy System

Before you modify a fuzzy system, you must familiarize yourself with the design of the system. Complete the following steps to open and observe the greenhouse fuzzy system in the Fuzzy System Designer.

1. Select **Tools»Control Design and Simulation»Fuzzy System Designer** to launch the Fuzzy System Designer.
2. Select **File»Open** and navigate to `greenhouse.fs` in the `labview\examples\control\fuzzy\Dynamic greenhouse controller` directory.



3. Click the OK button to open the greenhouse fuzzy system in the file dialog box. The Variables page displays details about the input and output variables of the fuzzy system.
4. Observe the input and output linguistic variables. The greenhouse fuzzy system has two input variables, Temperature and Humidity, and two output variables, Electric Roof and Water Spills.
5. Observe the ranges and membership functions for the linguistic variables. For example, the Temperature input variable represents the temperature, in degrees Celsius, inside the greenhouse, with a range of 0 to 50 and three membership functions: Cold, Normal, and Warm. You can see on the Input variable membership functions graph that a trapezoid function represents the shape of the Cold membership function. The left base, left top, right top, and right base points of this membership function are 0, 0, 13, and 21, respectively.
6. Click the Rules tab to display the Rules page. The Rules list displays nine rules for the fuzzy system, which correspond to all possible combinations of the membership functions of the two input linguistic variables. Also notice that the Defuzzification method for the fuzzy system is **Center of Area (CoA)**.
7. Select the first rule in the Rules list: IF 'Temperature' IS 'Cold' AND 'Humidity' IS 'Dry' THEN 'Electric Roof' IS 'Closed' ALSO 'Water Spills' IS 'Moderate'. Notice the characteristics of this rule. For example, the Antecedent connective is AND (Minimum), the Degree of support is 1, and the Consequent implication is Product.
8. Continue exploring the fuzzy system in the Fuzzy System Designer. When you are familiar with the system, you can click the Close button to exit the Fuzzy System Designer.

## Loading the Fuzzy System

Before modifying the greenhouse fuzzy system, you must load the.fs file. Complete the following steps to load the greenhouse.fs file.

1. Select File»New VI to create a new, blank VI.
2. Press the <Ctrl-E> keys to display the block diagram.
3. Add the FL Load Fuzzy System VI to the block diagram.
4. Right-click the file path input of the FL Load Fuzzy System VI and select Create»Constant from the shortcut menu.
5. Enter the absolute path to the greenhouse . fs file in the path constant.



### NOTE

You also can use the File I/O VIs and functions to create a relative path to wire to the file path input.

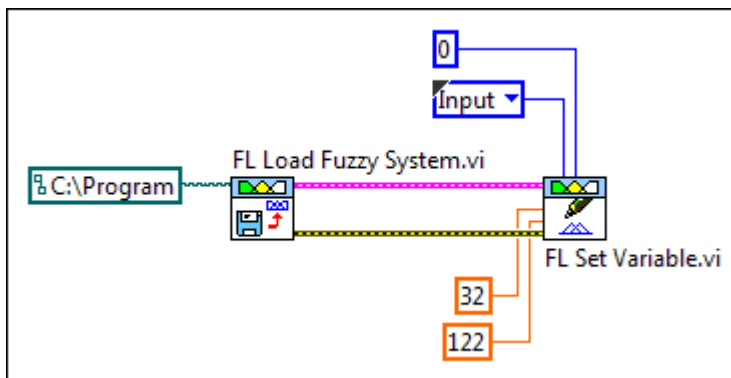
6. Select File»Save and save the VI as **Modified Greenhouse Fuzzy System . vi** in an easily accessible location.



## Modifying a Linguistic Variable

You can use the Variables VIs to modify the linguistic variables in a fuzzy system. Complete the following steps to modify the range of the Temperature input linguistic variable of the greenhouse fuzzy system to use degrees Fahrenheit instead of degrees Celsius.

1. Add the FL Set Variable VI to the block diagram. This VI modifies the name, range, or membership functions of a linguistic variable in a fuzzy system.
2. Wire the fuzzy system output of the FL Load Fuzzy System VI to the fuzzy system input of the FL Set Variable VI.
3. Right-click the input/output input of the FL Set Variable VI and select Create»Constant from the shortcut menu.
4. Select Input from the input/output constant to specify that you want to modify an input linguistic variable.
5. Right-click the variable index input of the FL Set Variable VI and select Create»Constant from the shortcut menu.
6. Enter 0 in the variable index constant to specify that you want to modify the first linguistic variable. The index is zero-based and corresponds to the order in which the linguistic variable was created. Recall that the Temperature variable is the first input linguistic variable in the fuzzy system.
7. Create constants for the minimum and maximum inputs of the FL Set Variable VI and set them to 32 and 122, respectively. This new range represents the possible temperature, in degrees Fahrenheit, inside the greenhouse. The block diagram now resembles the following image:



8. Save the VI.

This VI sets the new range of the Temperature variable when it runs. However, you still must modify the membership functions of the Temperature variable to complete the Celsius to Fahrenheit conversion.

## Modifying Membership Functions

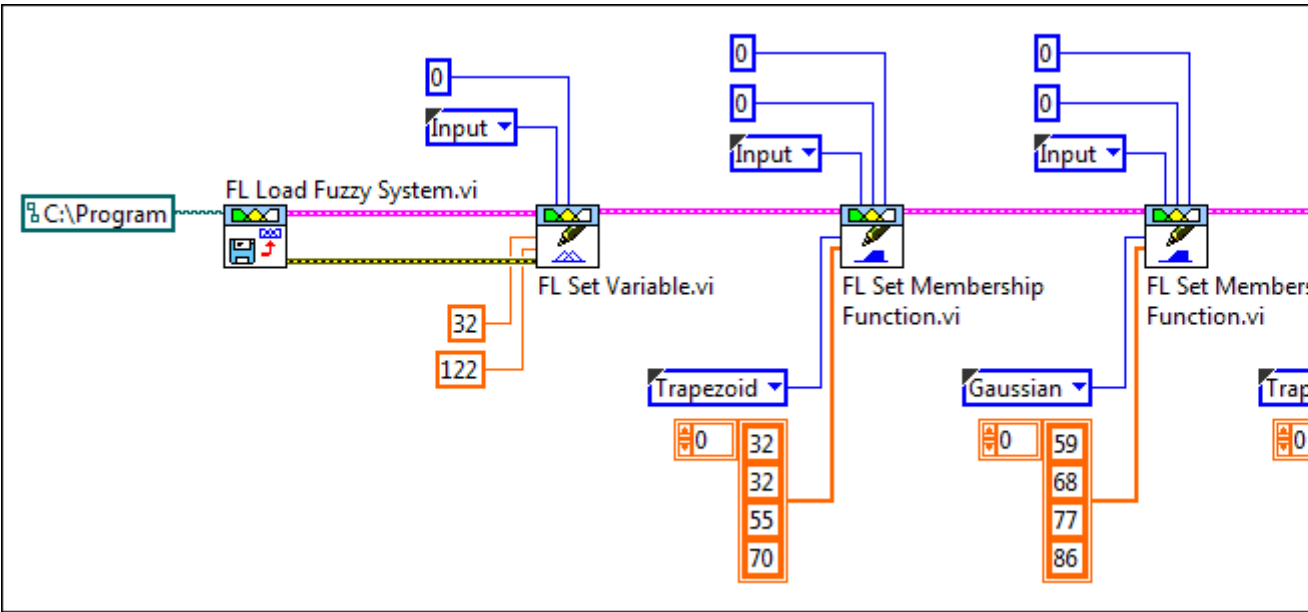
You can use the Membership VIs to modify the membership functions for linguistic variables in a fuzzy system. Complete the following steps to modify the membership

functions of the Temperature input linguistic variable to use degrees Fahrenheit instead of degrees Celsius.

- 1. Add the FL Set Membership Function VI to the block diagram.
- 2. Wire the fuzzy system out output of the FL Set Variable VI to the fuzzy system in input of the FL Set Membership Function VI.
- 3. Create constants for the membership function index, variable index, and input/output inputs of the FL Set Membership Function VI and set them to 0, 0, and Input, respectively. These inputs specify that you want to modify the first membership function, Cold, of the Temperature input linguistic variable.
- 4. Create a constant for the shape input of the FL Set Membership Function VI.
- 5. Select Trapezoid from the shape constant to specify the shape of the function that determines the degrees of membership for the linguistic variable.
- 6. Create a constant for the points input of the FL Set Membership Function VI. This input specifies the values of the linguistic variable corresponding to the base and top points, in order from left to right and base to top, of the membership function.
- 7. In the first four elements of the points array constant, enter the values 32, 32, 55, and 70, respectively.
- 8. Repeat steps 1 through 5 for the Normal and Warm membership functions. For each membership function, add a new FL Set Membership Function VI to the block diagram and wire the fuzzy system out output of the previous VI to the fuzzy system in input of the next VI. Use the shapes and points shown in the following table:

| Membership Function | Shape | Points           |
|---------------------|-------|------------------|
| Gaussian            |       | 59, 68, 77, 86   |
| Trapezoid           |       | 75, 90, 122, 122 |

The block diagram now resembles the following image:



- 9. Save the VI.

## Modifying a Rule

You can use the Rules VIs to modify the rules for a fuzzy system. This section demonstrates how to recreate the first rule of the greenhouse fuzzy system:

---

IF 'Temperature' IS 'Cold' AND 'Humidity' IS 'Dry' THEN 'Electric Roof' IS 'Closed' ALSO 'Water Spills' IS 'Moderate'

---

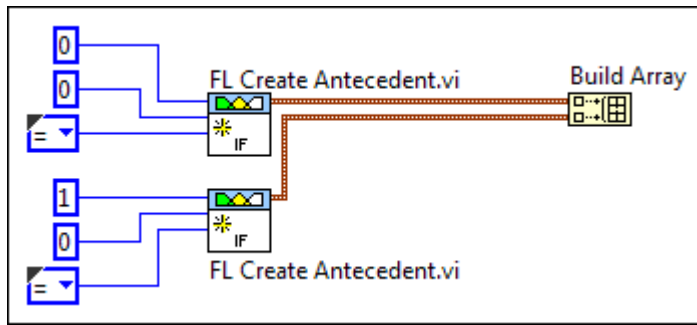
and replace the existing first rule of the fuzzy system with the rule you create.

When you use the Rules VIs to create rules, you must create the rules in parts. First, you create the antecedents. Then, you create the consequents. Finally, you combine the antecedents and consequents for a specific rule. The next three steps of this tutorial walk you through each of these actions.

## Creating Antecedents

The first step in creating a rule is to create the antecedents of the rule. Complete the following steps to create the 'Temperature' IS 'Cold' and 'Humidity' IS 'Dry' antecedents.

1. Add the FL Create Antecedent VI to the block diagram. This VI creates an antecedent, or IF portion, of a rule for a fuzzy system.
2. Create constants for the variable index and membership function index inputs of the FL Create Antecedent VI and set them both to 0. These inputs specify that you want to associate the Temperature input linguistic variable with the Cold membership function.
3. Create a constant for the condition input of the FL Create Antecedent VI.
4. Select = from the condition constant to calculate the degree of membership of the linguistic variable within the membership function. FL Create Antecedent VI now creates the 'Temperature' IS 'Cold' antecedent.
5. Repeat steps 1 through 4 to create the 'Humidity' IS 'Dry' antecedent. In the greenhouse fuzzy system, Dry is the first membership function of the second input linguistic variable, so use 1 as the variable index and 0 as the membership function index.
6. Add a Build Array function to the block diagram.
7. Resize the Build Array function to display two input elements.
8. Wire the antecedent output of the first FL Create Antecedent VI to the first element input of the Build Array function.
9. Wire the antecedent output of the second FL Create Antecedent VI to the second element input of the Build Array function. The section of block diagram you just created resembles the following image:



10. Save the VI.

## Creating Consequents

You create rule consequents similarly to how you create rule antecedents. Complete the following steps to create the 'Electric Roof' IS 'Closed' and 'Water Spills' IS 'Moderate' consequents.

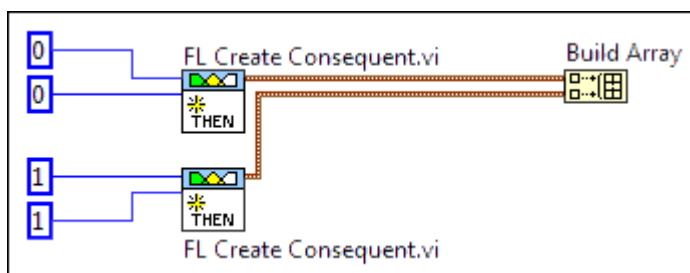
1. Add the FL Create Consequent VI to the block diagram. This VI creates a consequent, or THEN portion, of a rule for a fuzzy system.
2. Create constants for the variable index and membership function index inputs of the FL Create Consequent VI and set them both to 0. These inputs specify that you want to associate the Electric Roof output linguistic variable with the Closed membership function.



### NOTE

Unlike the FL Create Antecedent VI, the FL Create Consequent VI does not have a condition input. The FL Create Consequent VI always returns the consequent corresponding to the degree of membership of the output variable within the membership function.

3. Repeat steps 1 and 2 to create the 'Water Spills' IS 'Moderate' consequent. In the greenhouse fuzzy system, Moderate is the second membership function of the second output linguistic variable Water Spills, so set both the variable index and membership function index to 1.
4. Add a Build Array function to the block diagram.
5. Resize the Build Array function to display two input elements.
6. Wire the consequent output of the first FL Create Consequent VI to the first element input of the Build Array function.
7. Wire the consequent output of the second FL Create Consequent VI to the second element input of the Build Array function. The section of block diagram you just created resembles the following image:



8. Save the VI.

## Combining the Antecedents and Consequents for a Rule

Complete the following steps to combine the antecedents and consequents you created in the previous steps to form a specific rule.

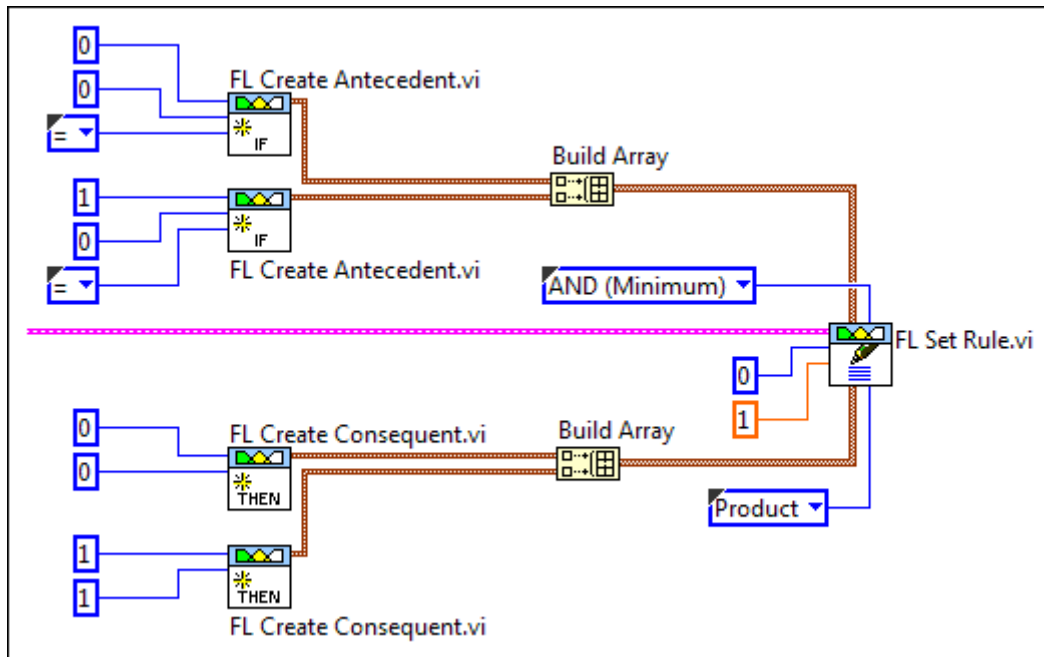
1. Add an FL Set Rule VI to the block diagram. This VI modifies the antecedents, consequents, or relationships of a rule in a fuzzy system.
2. Wire the fuzzy system output of the last FL Set Membership Function VI to the fuzzy system input of the FL Set Rule VI.
3. Create a constant for the rule index input of the FL Set Rule VI and set it to 0 to specify that you want to modify the first rule in the fuzzy system.
4. Wire the appended array output of the first Build Array function to the antecedents input of the FL Set Rule VI.
5. Wire the appended array output of the second Build Array function to the consequents input of the FL Set Rule VI.
6. Create a constant for the antecedent connective input of the FL Set Rule VI to specify how the VI calculates the truth value of the aggregated rule antecedent.
7. Select AND (Minimum) from the antecedent connective constant. The aggregated rule antecedent now is IF 'Temperature' IS 'Cold' AND 'Humidity' IS 'Dry', where this VI uses the smallest degree of membership of the individual antecedents to calculate the truth value of the aggregated rule antecedent.



### NOTE

You cannot specify how this VI calculates the truth value of an aggregated rule consequent. This VI always uses the greatest degree of membership of the individual consequents to calculate the truth value of the aggregated rule consequent.

8. Create a constant for the consequent implication input of the FL Set Rule VI. This input specifies the implication method this VI uses to scale the membership functions of the output linguistic variables based on the rule weight.
9. Select Product from the consequent implication constant to specify that this VI uses the **Product implication method**.
10. Create a constant for the degree of support input and set it to 1. This input specifies the weight that you want to apply to the rule. The final rule weight is equal to the degree of support multiplied by the truth value of the aggregated rule antecedent. The section of block diagram you just created resembles the following image:



11. Save the VI.

## Saving the Fuzzy System

In previous sections of this tutorial, you modified the linguistic variables, membership functions, and rules of the greenhouse fuzzy system. You now can save the fuzzy system to a .fs file. Complete the following steps to save the modified fuzzy system to a .fs file.

1. Add the FL Save Fuzzy System VI to the block diagram.
2. Wire the fuzzy system output of the FL Set Rule VI to the fuzzy system input of the FL Save Fuzzy System VI.
3. Right-click the file path input of the FL Save Fuzzy System VI and select Create»Constant from the shortcut menu.
4. Enter the absolute path to a modified greenhouse.fs file in the path constant. The FL Save Fuzzy System VI creates the .fs file when it runs.



### 노트

You also can use the File I/O VIs and functions to create a relative path to wire to the file path input of the FL Save Fuzzy System VI.

5. Save the Modified Greenhouse Fuzzy System VI.
6. Run the VI. The VI loads the original greenhouse.fs file, modifies the linguistic variable and membership functions you specified, creates new antecedents and consequents for a rule, and saves all the changes to the modified greenhouse.fs file.

You now can use the Fuzzy System Designer to open the modified greenhouse.fs file and observe the modifications you made to the greenhouse fuzzy system.

You also can use the FL Fuzzy Controller VI to implement a fuzzy logic controller for the modified greenhouse fuzzy system. The FuzzyEx Dynamic Fuzzy Controller for a greenhouse example VI in the `labview\examples\control\fuzzy\Dynamic greenhouse controller` directory implements a controller for the greenhouse fuzzy system. This example VI uses the FL Load Fuzzy System VI to load the `greenhouse . fs` file.



Open example



Find related examples

You can load the `modified greenhouse . fs` file instead and then run the example VI to implement a controller for the modified greenhouse fuzzy system.

## 퍼지 시스템

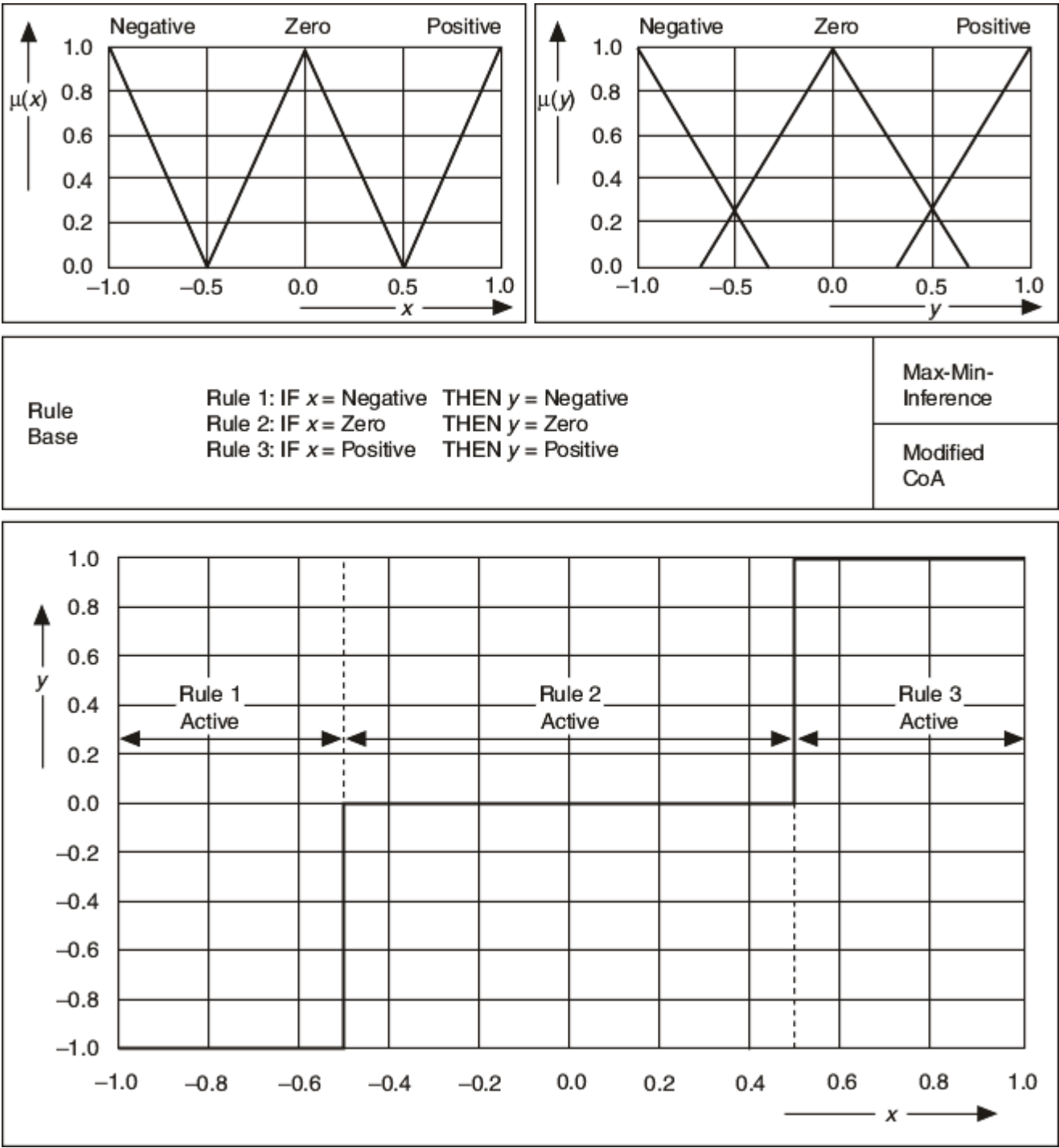
### Multi-Loop Control

Most of the PID VIs are polymorphic VIs for use in multiple control-loop applications. For example, you can design a multi-loop PID control application using the PID VI and DAQ functions for input and output. A DAQ analog input function returns an array of data when you configure it for multiple channels. You can wire this array directly into the process variable input of the PID VI. The polymorphic type of the PID VI automatically switches from DBL to DBL Array, which calculates and returns an array of output values corresponding to the number of values in the process variable array. You also can switch the type of the polymorphic VI manually by right-clicking the VI icon and selecting **Select Type** from the shortcut menu.

When the polymorphic type is DBL Array, other inputs automatically change to array inputs as well. For example, the PID VI inputs setpoint, PID gains, and output range all become array inputs. Each of these inputs can have an array length ranging from 1 to the array length of the process variable input. If the array length of any of these inputs is less than the array length of the process variable input, the PID VI reuses the last value in the array for other calculations. For example, if you specify only one set of PID gains in the PID gains array, the PID VI uses these gains to calculate each output value corresponding to each process variable input value. Other polymorphic PID and Fuzzy Logic VIs operate in the same manner.

### Non-Overlapping Input Terms

The following image shows the controller I/O characteristic that results when non-overlapping antecedent terms describe the input variable.



Only one rule is active for each input situation that leads to the stepped controller characteristic.



RELATED INFORMATION

[I/O Characteristics of Fuzzy Controllers](#) on page 1690

[Specifying an Antecedent Connective](#) on page 1702

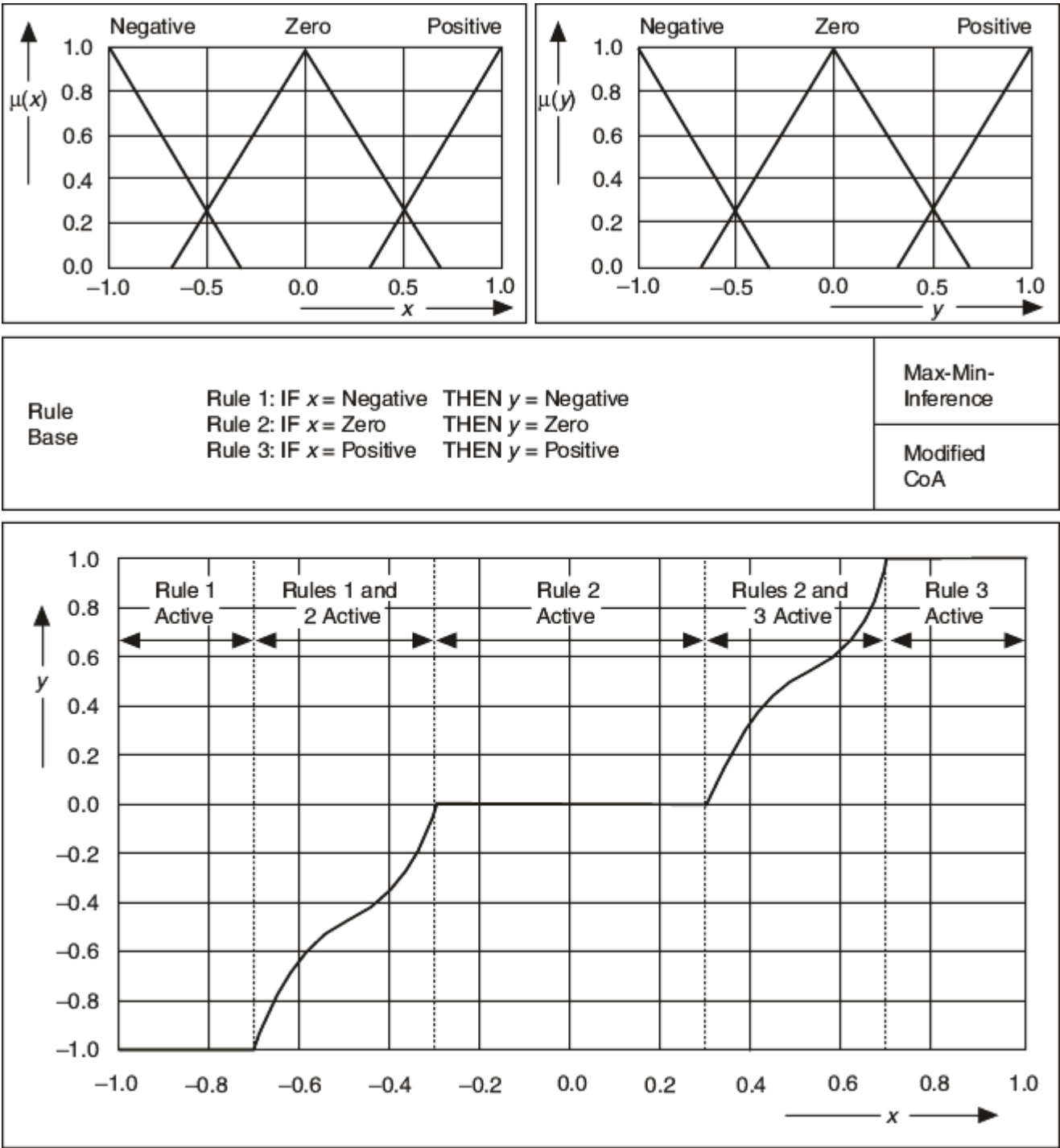
[Creating a Rule Base](#) on page 1736



## Partially Overlapping Input Terms

To illustrate how the **I/O characteristics of a fuzzy controller** depend on design parameters such as the **rule base** and **membership function** specifications of the fuzzy system, consider a single-input fuzzy controller. Many of the characteristics of a single-input fuzzy controller apply to fuzzy controllers with two or more inputs.

The following image shows the I/O characteristic of a fuzzy controller for a fuzzy system that has only three **linguistic terms** for the input variable  $x$  and the output variable  $y$ . The rule base consists of three rules, which indicate that the increasing input values cause the output value to increase.



The resulting controller characteristic shows nonlinear behavior. You obtain different intervals within the controller characteristic because the input linguistic terms partially overlap. The rule base has only one valid rule outside of the overlapping regions. The output therefore has a constant value determined by the output linguistic term of the output linguistic variable, which is independent of the degree of truth for that rule.

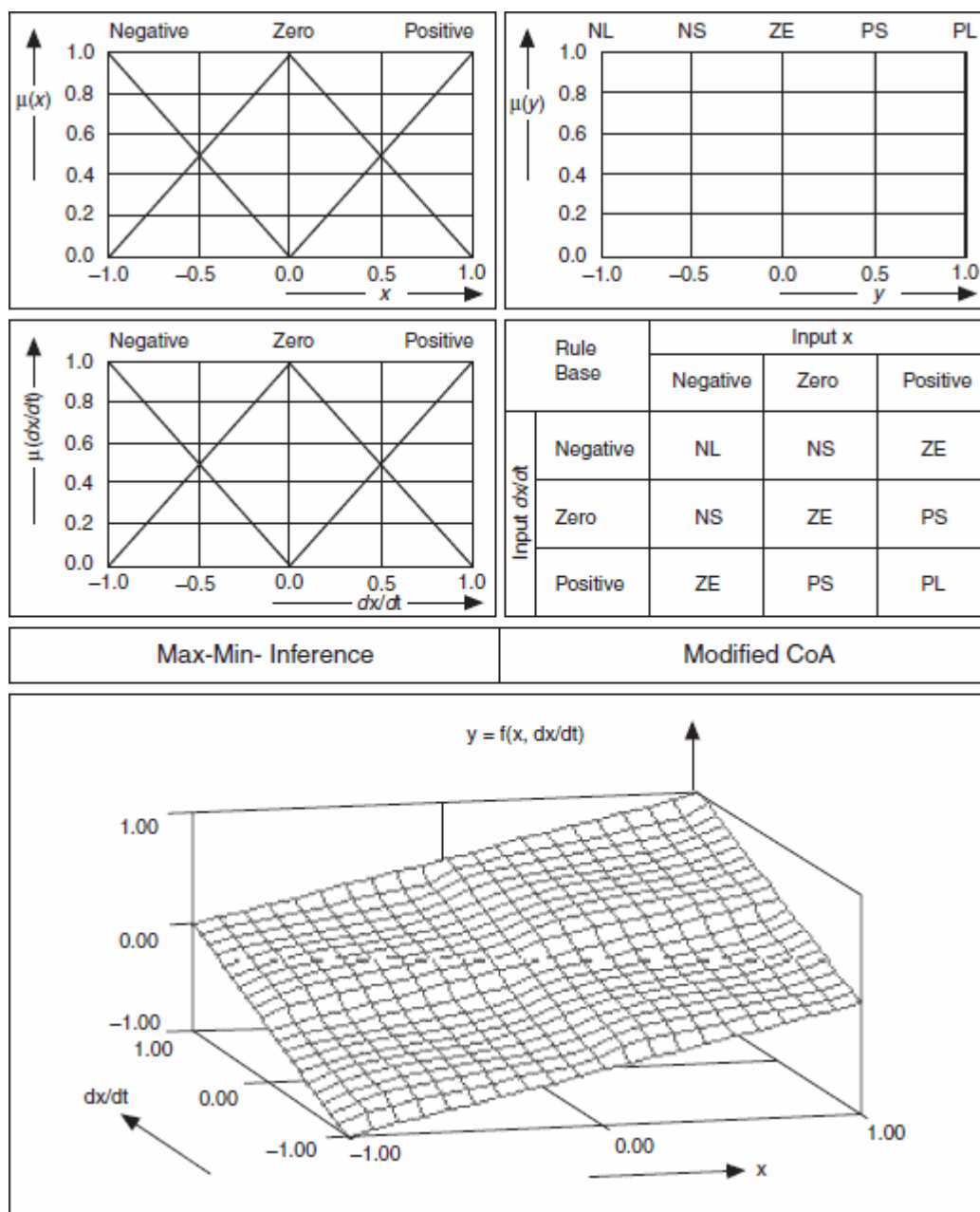
The overlapping sections of the **antecedent terms** lead to the rising intervals of the controller characteristic. Within these parts, two rules are simultaneously active. The different consequent terms, weighted by the degrees of truth of the different active rules, determine the output value. Notice that the overlapping triangular consequent terms cause the rising edges of the controller characteristic to be nonlinear.

# 퍼지 컨트롤러

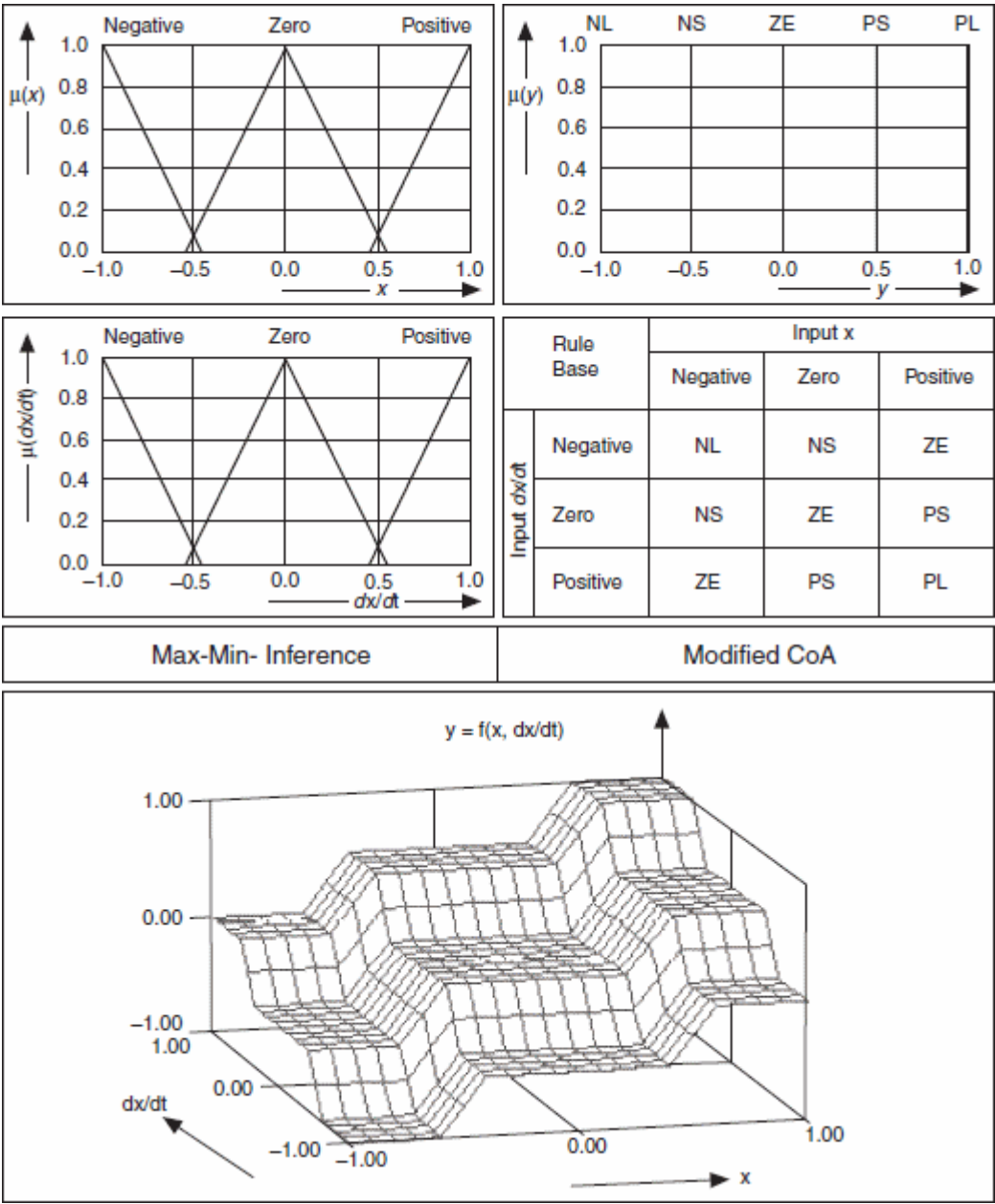
## Dual-Input Fuzzy Controllers

In principle, the conclusions about [I/O characteristics](#) for single-input fuzzy controllers also are valid for controllers with two or more inputs. However, using the AND (Minimum) [antecedent connective](#) to combine the different input conditions raises an additional nonlinear effect. The fuzzy controller considers the antecedent of the rule with the lowest degree of truth as shown in the [Reproducing a Given I/O Characteristic](#) example.

The following image shows the I/O characteristic field for a dual-input fuzzy controller.

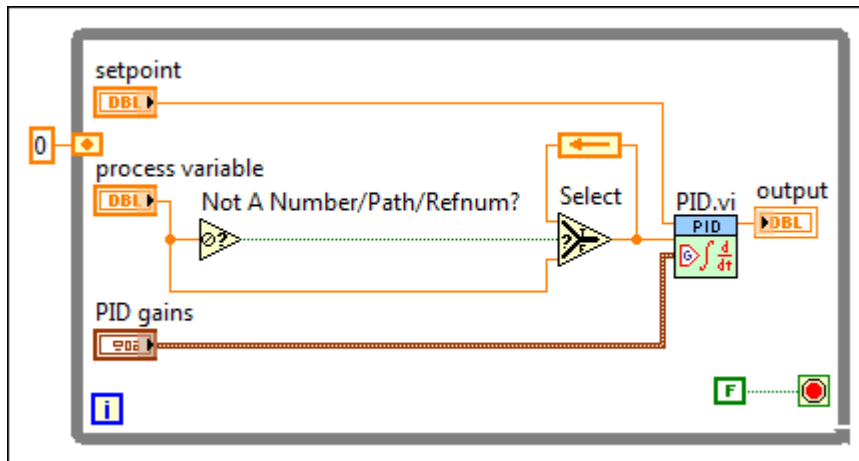


Because the AND (Minimum) antecedent connective is nonlinear, the characteristic field is not exactly linear despite the membership functions that overlap entirely for both input variables. Non-overlapping membership functions yield a stepped characteristic field with constant planes, as shown in the following image.



## Fault Protection

If a PID VI receives an invalid input, such as NaN (Not a Number), the VI outputs NaN until you reinitialize the VI. However, you can use the Not A Number/Path/Refnum? function to check for invalid inputs and respond in a way that is appropriate to the application. The following block diagram uses the Not A Number/Path/Refnum? function with a Select function and a Feedback Node to reuse the last valid input when the VI receives an invalid input.



Refer to the General PID with Fault Protection VI in the `labview\examples\control\PID` directory for an example of a single-channel PID implementation that includes fault protection.

 Open example  Find related examples

## Filtering Control Inputs

You can use the PID Control Input Filter VI to filter unwanted, high-frequency noise from measured values in a control application. For example, if you are measuring process variable values using a DAQ device, you can use this VI to filter noise from input values in the control loop before passing the values to control functions such as the PID VI.

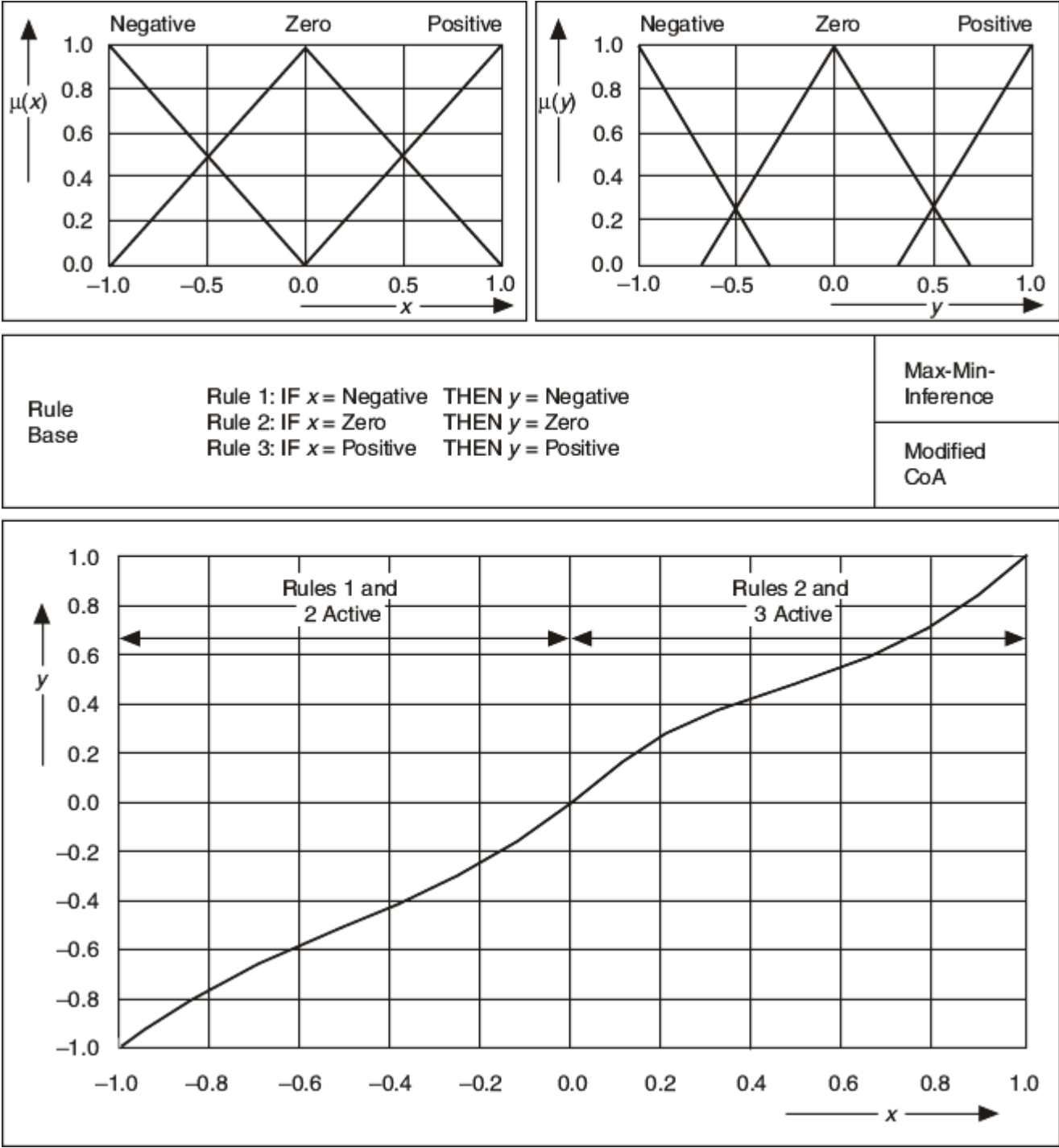
When filtering noise, set the sampling rate of the control system to at least 10 times the fastest time constant of the physical system. This ensures that any frequency components of the measured signal greater than 1/10 of the sampling frequency are a result of noise in the measured signal. Gains in the PID controller can amplify this noise and produce unnecessary wear on actuators and other system components.

The PID Control Input Filter VI uses a lowpass fifth-order finite impulse response (FIR) filter. The cutoff frequency of the lowpass filter is 1/10 of the sampling frequency, regardless of the actual sampling frequency value.

You can use the Filters PtByPt VIs or the LabVIEW Digital Filter Design Toolkit to perform additional filtering tasks. Refer to the National Instruments website for more information about or to purchase the Digital Filter Design Toolkit.

## Fully Overlapping Input Terms

The following image shows the resulting controller I/O characteristic for antecedent terms that overlap entirely. The consequent term distribution and the rule base remain unchanged for this case.



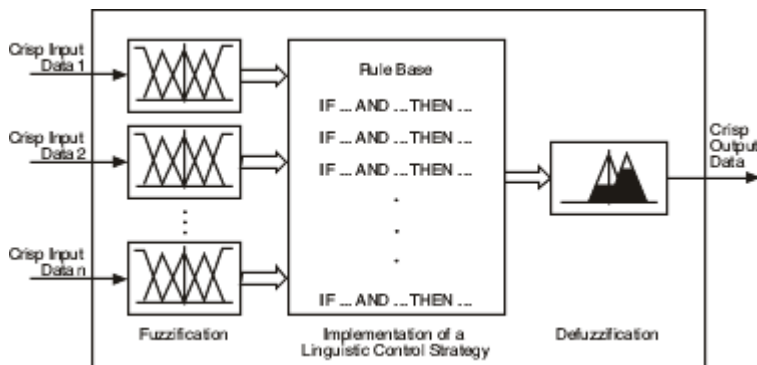
RELATED INFORMATION

- [I/O Characteristics of Fuzzy Controllers](#) on page 1690
- [Specifying an Antecedent Connective](#) on page 1702
- [Creating a Rule Base](#) on page 1736

## Fuzzy Controllers

You can use fuzzy controllers to control **fuzzy systems**. Most traditional control algorithms require a mathematical model of the system you want to control. However, many physical systems are difficult or impossible to model mathematically. In addition, many processes are either nonlinear or too complex for you to control with traditional strategies. However, if you can describe a control strategy qualitatively, you can use fuzzy logic to create a fuzzy controller that emulates a heuristic rule-of-thumb strategy.

The following image illustrates the process of a fuzzy controller.



### Fuzzification

Fuzzification is the process of associating crisp, or numerical, input values with the **linguistic terms** of the corresponding input **linguistic variables**.

For example, a fuzzy controller might associate the temperature reading from a thermometer with the linguistic terms cold, moderate, and hot for the current temperature linguistic variable. Depending on the **membership functions** for the linguistic terms, the temperature value might correspond to one or more of the linguistic terms.

### Implementing a Linguistic Control Strategy

After a fuzzy controller fuzzifies the input values of a fuzzy system, the fuzzy controller uses the corresponding input linguistic terms and the **rule base** to determine the resulting linguistic terms of the output linguistic variables.

For example, suppose the current temperature of a room is 50 degrees, which corresponds to a linguistic term of cold with a degree of membership of 0.4. Also suppose the desired temperature is 70, which corresponds to a linguistic term of moderate with a degree of membership of 0.8. The fuzzy controller invokes the following rule of the fuzzy system: IF current temperature is cold AND desired temperature is moderate, THEN heater setting is low.

Notice that this rule consists of two antecedents, "current temperature is cold" and "desired temperature is moderate". The truth value of each antecedent is equal to the degree of membership of the linguistic variable within the corresponding linguistic term. The fuzzy logic controller uses an antecedent connective to determine how to calculate the

truth value of the aggregated rule antecedent. Suppose the invoked rule in this example uses the AND (Minimum) antecedent connective, which specifies to use the smallest degree of membership of the antecedents as the truth value of the aggregated rule antecedent. Therefore, the truth value of the aggregated rule antecedent is 0.4.

You can specify a degree of support for each rule of a fuzzy system. The weight of a rule is equal to the degree of support multiplied by the truth value of the aggregated rule antecedent. The fuzzy controller uses an implication method to scale the membership functions of an output linguistic variable based on the rule weight before performing defuzzification.

## Defuzzification

Defuzzification is the process of converting the degrees of membership of output linguistic variables within their linguistic terms into crisp numerical values. A fuzzy controller can use one of several mathematical methods to perform defuzzification. The most accurate **defuzzification method** for a fuzzy controller varies based on the control application.

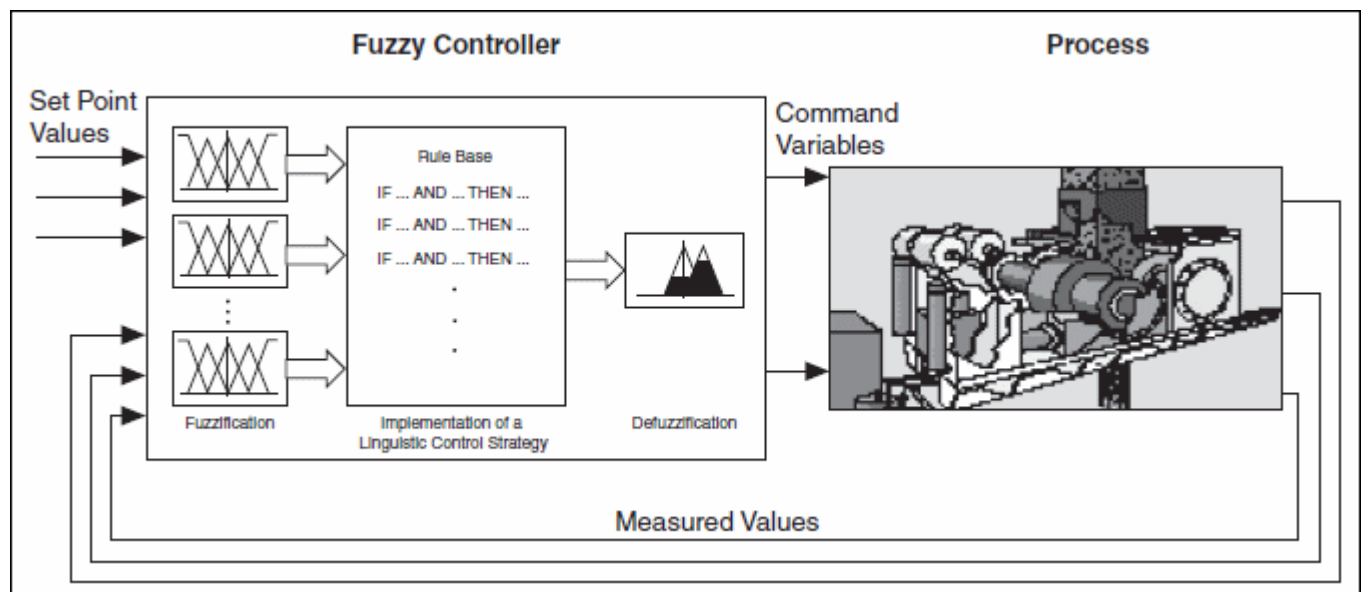
## Related Information

### I/O Characteristics of Fuzzy Controllers

### Fuzzy Controllers in Closed-Loop Control Structures

## Fuzzy Controllers in Closed-Loop Control Structures

The most common use case for fuzzy controllers is in closed-loop control structures. The most basic structure of closed-loop control applications uses sensor signals as input signals for the system and the outputs as command values to drive the actuators of the process. The following image shows a simple closed-loop control structure with a fuzzy controller.





Pure fuzzy control applications are more the exception than the rule. In most cases the fuzzy controller outputs serve as reference parameters, such as gains, that you provide to a conventional controller instead of directly to driving actuators.

Because you can regard a fuzzy controller as a nonlinear characteristic field controller, it has no internal dynamic aspects. Thus, you must implement any dynamic property by an appropriate preprocessing of the measured input data.

PID and Fuzzy Logic VIs enable you to integrate fuzzy controllers and PID control. For example, you can use fuzzy controllers with conventional PID controllers or create hybrid fuzzy-PI controllers. Use either the Fuzzy System Designer or the Fuzzy Logic VIs to design a fuzzy system. Then use the FL Fuzzy Controller VI to implement a fuzzy controller for the fuzzy system. Finally, you can integrate the fuzzy controller into a control structure you create using the PID VIs.



#### RELATED INFORMATION

[PID Control](#) on page 1693

[Using Fuzzy Controllers with PID Controllers](#) on page 1706

[Fuzzy-PI Controllers](#) on page 1687

[Fuzzy-PI Controllers](#) on page 1687

[Using Fuzzy Controllers with PID Controllers](#) on page 1706

## Fuzzy Logic

Fuzzy logic is a method of rule-based decision making used for expert systems and process control. Fuzzy logic differs from traditional Boolean logic in that fuzzy logic allows for partial membership in a set.

Traditional Boolean logic is two-valued in the sense that a member either belongs to a set or does not. Values of one and zero represent the membership of a member to the set with one representing absolute membership and zero representing no membership. Fuzzy logic allows for partial membership, or a degree of membership, which might be any value along the continuum of zero to one.

You can think of fuzzy logic in terms of fuzzy systems, which are systems of variables associated with fuzzy logic, and fuzzy controllers, which use defined rules to control a fuzzy system based on the current values of input variables. You can use the Fuzzy Logic VIs to design and control fuzzy systems. You also can use the Fuzzy System Designer to design fuzzy systems interactively.

### Related Information

Fuzzy Systems

Fuzzy Controllers

Tutorial: Using the Fuzzy System Designer

## Tutorial: Modifying a Fuzzy System with the Fuzzy Logic VIs



### RELATED INFORMATION

[Fuzzy Systems](#) on page 1686

[Fuzzy Controllers](#) on page 1683

[Fuzzy Systems](#) on page 1686

[Fuzzy Controllers](#) on page 1683

## Fuzzy System Design Overview

The topics in this section use an example fuzzy system that automates a vehicle to park itself to illustrate the basics of fuzzy system design.



### 노트

The topics in this section describe the general process of designing a fuzzy system. You also can refer to the tutorials [Using the Fuzzy System Designer](#) and [Modifying a Fuzzy System with the Fuzzy Logic VIs](#) for examples with step-by-step instructions of how to design a fuzzy system with the PID and Fuzzy Logic VIs.

A fuzzy system consists of three main parts: linguistic variables, membership functions, and rules. Therefore, fuzzy system design is a three-step process:

1. [Creating Linguistic Variables](#)
2. [Creating Membership Functions](#)
3. [Creating a Rule Base](#)

## Fuzzy Systems

A fuzzy system is a system of variables that are associated using fuzzy logic. A **fuzzy controller** uses defined rules to control a fuzzy system based on the current values of input variables. You can use the Fuzzy System Designer and the Fuzzy Logic VIs to design and control fuzzy systems.

Fuzzy systems consist of three main parts: linguistic variables, membership functions, and rules.

### Linguistic Variables

**Linguistic variables** represent, in words, the input variables and output variables of the system you want to control. For a heater, you might have two input linguistic variables, current temperature and desired temperature, and one output linguistic variable, heater setting. Each linguistic variable has a range of expected values. For example, the range of current temperature might be 0 to 100 degrees. The range of desired temperature might be 50 to 80 degrees.

A fuzzy controller requires at least one input linguistic variable and one output linguistic variable.

## Linguistic Terms and Membership Functions

**Linguistic terms** represent, in words, categories for the values of a linguistic variable. The linguistic variables current temperature and desired temperature each might include the linguistic terms cold, moderate, and hot. The linguistic variable heater setting might include the linguistic terms off, low, and high.

**Membership functions** are numerical functions corresponding to linguistic terms. A membership function represents the degree of membership of linguistic variables within their linguistic terms. The degree of membership is continuous between 0 and 1, where 0 is equal to 0% membership and 1 is equal to 100% membership. For example, the linguistic variable current temperature might have full membership (1) within the linguistic term hot at 100 degrees, no membership (0) within that term at 70 degrees or less, and partial membership at all temperatures between 70 and 100 degrees.

## Rules

Rules describe, in words, the relationships between input and output linguistic variables based on their linguistic terms. For example, you might define the following rule:

---

IF current temperature is cold AND desired temperature is moderate, THEN heater setting is low.

---

The clauses "current temperature is cold" and "desired temperature is moderate" are the antecedents of this rule. The AND connective specifies how the fuzzy logic controller relates the two antecedents to determine the truth value for the aggregated rule antecedent. The clause "heater setting is low" is the consequent of this rule.

A **rule base** is the set of rules for a fuzzy system. The rule base is equivalent to the control strategy of the controller.

## Related Information

### Fuzzy System Design Overview

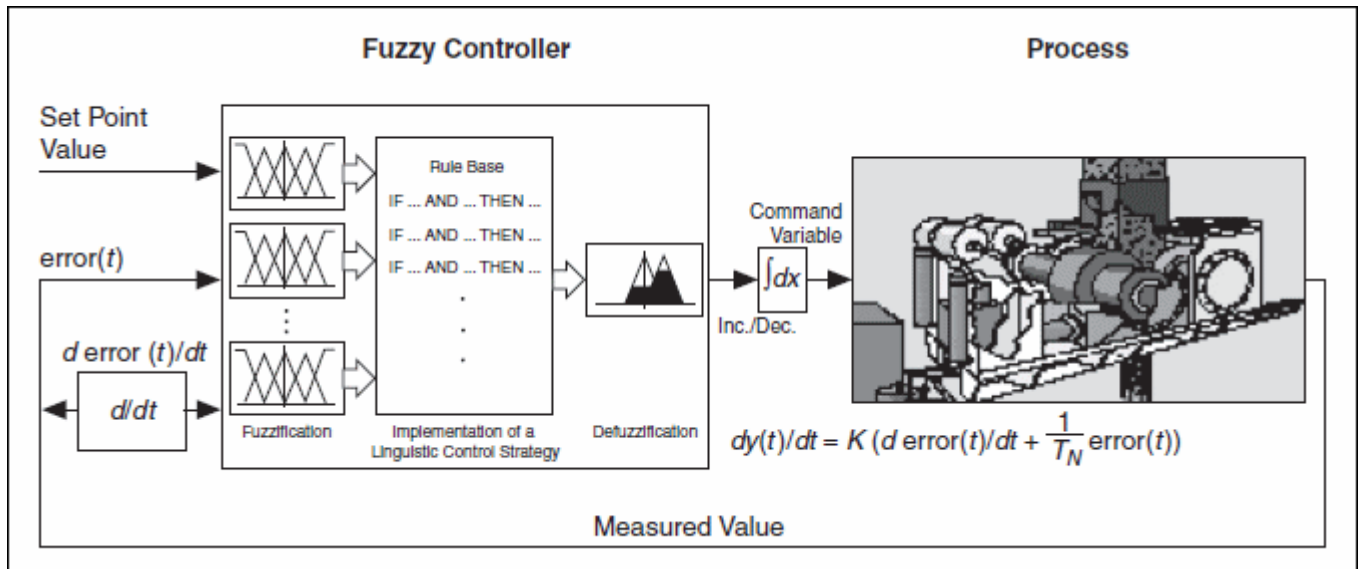
### Defuzzification Methods

## Fuzzy-PI Controllers

A PI controller is a controller that produces proportional plus integral control action. The PI controller has only one input and one output. The output value increases when the input value increases.

A fuzzy-PI controller is a generalization of the conventional PI controller that uses an error signal and its derivative as input signals. Fuzzy-PI controllers have two inputs and one output. Multiple inputs allow for greater control diversity for a fuzzy-PI controller over a conventional PI controller.

The fuzzy-PI controller shown in the following image uses the error signal  $e(t)$  and its derivative  $de(t)/dt$  from the measured data preprocessing step as inputs. If the output signal describes the necessary difference toward the current output value, you need a subsequent integrator device to build up the command variable value.



### Advantages of Fuzzy-PI Controllers

The benefit of the fuzzy-PI controller is that it does not have a special operating point. The rules evaluate the difference between the measured value and the set value, which is the error signal. The rules also evaluate the tendency of the error signal to determine whether to increase or decrease the control variable. The absolute value of the command variable has no influence.

Another advantage of a fuzzy-PI controller over a conventional PI controller is that it can implement nonlinear control strategies and that it uses linguistic rules. With fuzzy-PI controllers, you can consider the error tendency by itself when the error becomes small.

### Gain Scheduling

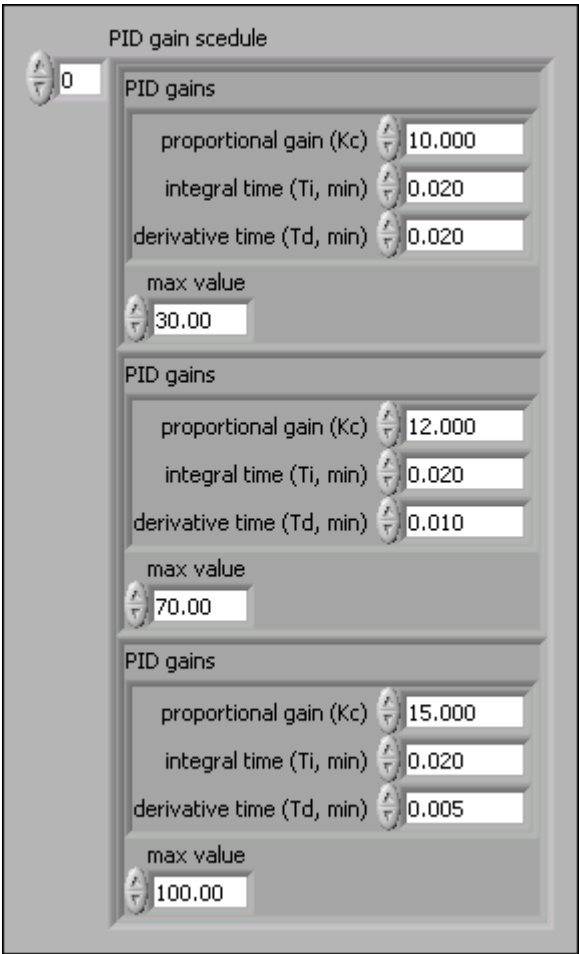
Gain scheduling refers to a system where you change controller parameters based on measured operating conditions. For example, the scheduling variable can be the setpoint, the process variable, a controller output, or an external signal. For historical reasons, the term gain scheduling is used even if other parameters such as derivative time,  $T_d$ , or integral time,  $T_i$  change. Gain scheduling effectively controls a system whose dynamics change with the operating conditions.

You can use the PID Gain Schedule VI, to apply different sets of PID parameters for different regions of operation of your controller. Because most processes are nonlinear, PID

parameters that produce a desired response at one operating point might not produce a satisfactory response at another operating point. The PID Gain Schedule VI selects and outputs one set of PID gains from a gain schedule based on the current value of its gain scheduling value input. For example, to implement a gain schedule based on the value of the process variable, wire the process variable value to the gain scheduling value input and wire the PID gains out output to the PID gains input of the PID VI.

The PID gain schedule input of the PID Gain Schedule VI is an array of clusters of PID gains and corresponding maximum values. Each set of PID gains corresponds to the range of input values from the max value of the previous element of the array to the max value of the same element of the array. The input range of the PID gains of the first element of PID gain schedule is all values less than or equal to the corresponding max value.

The following front panel shows a VI that uses the setpoint value as the gain scheduling variable with a default range of 0 to 100.



The following table summarizes parameter ranges specified in the previous image:

| Range                | Parameters                           |
|----------------------|--------------------------------------|
| $0 \leq SP \leq 30$  | $K_c = 10$ $T_i = 0.02$ $T_d = 0.02$ |
| $30 \leq SP \leq 70$ | $K_c = 12$ $T_i = 0.02$ $T_d = 0.01$ |

| Range                 | Parameters                            |
|-----------------------|---------------------------------------|
| $70 \leq SP \leq 100$ | $K_c = 15 \ T_i = 0.02 \ T_d = 0.005$ |

## I/O Characteristics of Fuzzy Controllers

You can consider a fuzzy controller to be a nonlinear characteristic field controller. The rule base and membership functions of the fuzzy system determine the behavior of the controller. Because the controller has no internal dynamic aspects, the I/O characteristics can describe the transient response of the controller completely.

This section includes examples that show the I/O characteristics of a fuzzy controller under various of conditions. Select an item from the list for information about I/O characteristics of a fuzzy controller with the described conditions.

- Partially Overlapping Input Terms
- Fully Overlapping Input Terms
- Non-Overlapping Input Terms
- Undefined Input Term Intervals
- Singletons as Output Terms with Fully Overlapping Input Terms
- Different Overlapping Degrees of Membership Functions for Output Terms
- Wide and Small Membership Functions for the Output Terms
- Using the Mean of Maximum (MoM) Defuzzification Method
- Reproducing a Given I/O Characteristic
- Dual-Input Fuzzy Controllers

These examples show that you can use a fuzzy controller to perform arbitrary I/O operations. The number of input and output linguistic terms depends on the desired characteristic type and the precision to which you approximate the given I/O characteristic.



#### RELATED INFORMATION

[Fuzzy Controllers](#) on page 1683

[Creating a Rule Base](#) on page 1736

[Creating Membership Functions](#) on page 1739

[Partially Overlapping Input Terms](#) on page 1677

[Fully Overlapping Input Terms](#) on page 1681

[Non-Overlapping Input Terms](#) on page 1675

[Undefined Input Term Intervals](#) on page 1704

[Singletons as Output Terms with Fully Overlapping Input Terms](#) on page 1699

[Different Overlapping Degrees of Membership Functions for Output Terms](#) on page 1750

[Wide and Small Membership Functions for Output Terms](#) on page 1757

[Using the Mean of Maximum \(MoM\) Defuzzification Method](#) on page 1754

[Reproducing a Given I/O Characteristic](#) on page 1723

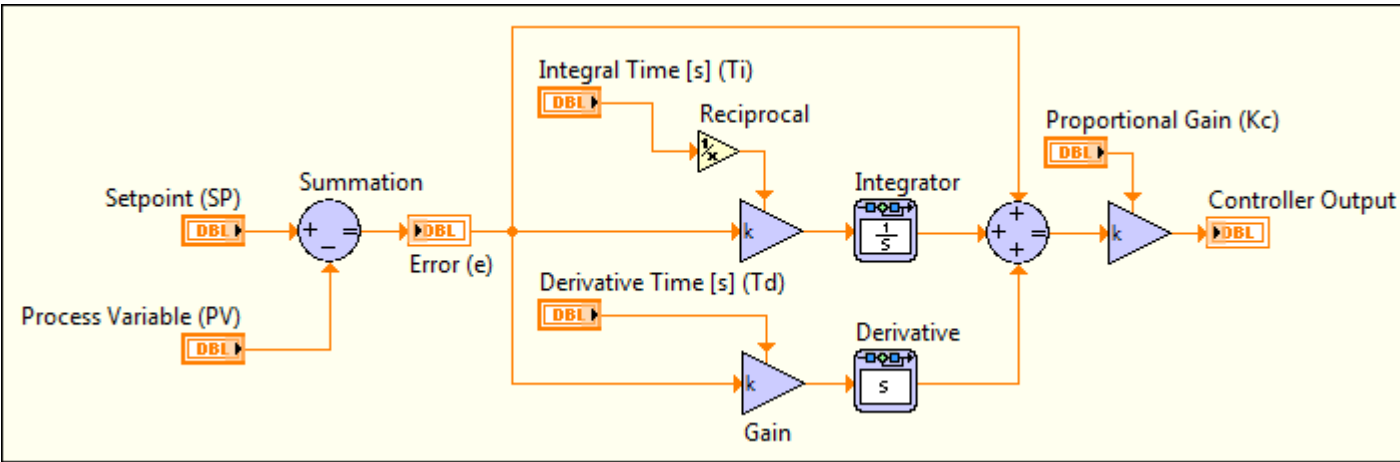
[Dual-Input Fuzzy Controllers](#) on page 1679

## PID Algorithms

The following equations describe the basic PID algorithm. This section also provides information about [how the PID VIs implement this algorithm](#), as well as the [advanced PID algorithm](#).

### Basic PID Controller

The following simulation diagram represents the basic PID (Proportional-Integral-Derivative) controller:



**NOTE**  
Constructing a simulation diagram like the one in the previous image requires the LabVIEW Control Design and Simulation Module. However, the PID VIs implement PID controllers for you. You do not need the Control Design and Simulation Module to build PID controllers. Refer to the National Instruments website at [ni.com](http://ni.com) for more information about the Control Design and Simulation Module.

Error Calculation (e)

A PID controller compares the setpoint SP to the process variable PV to obtain the error e, as follows:

$$e = SP - PV$$

|                            |                |
|----------------------------|----------------|
| where                      | e is the error |
| SP is the setpoint         |                |
| PV is the process variable |                |

Controller Action (u(t))

The PID controller calculates the controller action, u(t) as follows:

$$u(t) = K_c \left( e + \frac{1}{T_i} \int_0^t e dt + T_d \frac{de}{dt} \right)$$

|                                                                    |                          |
|--------------------------------------------------------------------|--------------------------|
| where                                                              | $K_c$ is controller gain |
| $T_i$ is the integral time in minutes, also called the reset time  |                          |
| $T_d$ is the derivative time in minutes, also called the rate time |                          |



This equation involves three actions that simultaneously apply to the output of the controller:

### Proportional Action (P)

Proportional action P is proportional to the error  $e$ . This term defines how quickly the controller must respond to changes in the error. The following formula represents the proportional action:

$$u_P(t) = K_c e$$

### Integral Action (I)

Integral action I is proportional to the integral of the error  $e$ . This term is generally used to correct steady-state error and to avoid disturbances in the system. The following formula represents the integral action:

$$u_I(t) = \frac{K_c}{T_i} \int_0^t e dt$$

### Derivative Action (D)

Derivative action D is proportional to the derivative of the error  $e$ . This term works as a brake to the response, avoiding the system to overshoot. The following formula represents the derivative action:

$$u_D(t) = K_c T_d \frac{de}{dt}$$

### Related Information

#### PID Topologies

#### Implementing the PID Algorithm with the PID VIs

#### Advanced PID Algorithm

## PID Control

Proportional-Integral-Derivative (PID) controllers are common feedback controllers. In PID control, you specify a process variable and a setpoint. The process variable is the system parameter you want to control, such as temperature, pressure, or flow rate, and the setpoint is the desired value for that system parameter. A PID controller determines a controller output value, for example the heater power or valve position, and applies the

controller output value to the system to drive the process variable toward the setpoint value.

Use the PID VIs to develop the following control applications based on PID controllers:

- Proportional (P); proportional-integral (PI); proportional-derivative (PD); and proportional-integral-derivative (PID) algorithms
- Gain-scheduled PID
- PID autotuning
- Error-squared PID
- Lead-Lag compensation
- Setpoint profile generation
- Multi-loop cascade control
- Feedforward control

## Related Information

### PID Algorithms

### Designing a Control Strategy

### Implementing PID Controllers with the PID VIs

### Autotuning PID Controllers

### Using PID on FPGA Targets



#### RELATED INFORMATION

[Implementing the PID Algorithm with the PID VIs](#) on page 1710

[Gain Scheduling](#) on page 1688

[Autotuning PID Controllers \(PID and Fuzzy Logic\)](#) on page 1721

[Advanced PID Algorithm](#) on page 1717

[Lead/Lag](#) on page 1714

[Setpoint Profile Generation](#) on page 1698

[Multi-Loop Control](#) on page 1675

[Lead/Lag](#) on page 1714

[PID Algorithms](#) on page 1691

[Designing a Control Strategy](#) on page 1747

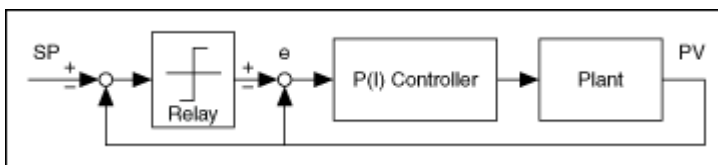
[Implementing PID Controllers with the PID VIs](#) on page 1708

[Autotuning PID Controllers \(PID and Fuzzy Logic\)](#) on page 1721

## PID Relay Autotuning Technique

The PID Relay **autotuning technique**, which is used by the **Autotuning Wizard** and the PID Relay instance of the PID Online Autotuning VI, uses the setpoint relay experiment to determine the information needed to tune the controller.

The following image illustrates the autotuning procedure excited by the setpoint relay experiment, which connects a relay and an extra feedback signal with the setpoint. The setpoint relay steps the setpoint up or down, exciting the autotuning procedure and allowing the controller to respond to the change and tune the system for an acceptable response. The PID Autotuning VI and the PID Relay instance of the PID Online Autotuning polymorphic VI directly implement this process. The existing controller remains in the loop.



For most systems, the nonlinear relay characteristic generates a limiting cycle, from which the autotuning algorithm identifies the relevant information needed for PID tuning. If the existing controller is proportional only, the autotuning algorithm identifies the ultimate gain  $K_u$  and ultimate period  $T_u$ . If the existing model is PI or PID, the autotuning algorithm identifies the dead time  $\tau$  and time constant  $T_p$ , which are two parameters in the integral-plus-deadtime model.

$$G_P(s) = \frac{e^{-\tau s}}{T_p s}$$

## PID Topologies

Standard PID literature provides several ways to define a PID controller. In general, PID controllers are classified using the following forms.

### Academic Form

In the Academic form, the proportional parameter P actuates on the integral I and derivative D terms, as represented by the following equation:

$$\frac{U(s)}{Y(s)} = K_c \left( 1 + \frac{1}{sT_i} + sT_d \right)$$

The **algorithms** implemented by the PID VIs are based on the Academic form with time expressed in minutes.

## Parallel Form

In the Parallel form, the three parameters P, I, and D are independent of each other and control each interaction. This form also is referred to as non-interacting PID. The Parallel form is represented by the following equation:

$$\frac{U(s)}{Y(s)} = K_p + \frac{K_i}{s} + sK_d$$

## Series Form

In the Series form, the parameters are coupled and in series with each component, as represented by the following equation:

$$\frac{U(s)}{Y(s)} = K_C \left(1 + \frac{1}{sT_I}\right) (1 + sT_D)$$

## Converting Topologies

You can use the PID Structure Conversion VI to convert a PID controller from one topology, or form, to another.

# Programmatically Autotuning a Previously Implemented PID Controller

You can use the PID Online Autotuning VI to tune a controller you implement or previously implemented with the PID Advanced VI.



### NOTE

You must use the PID Advanced VI to implement the controller. You can implement PID controllers with various PID VIs, but PID Advanced is the only VI that supports putting the controller under manual control, which is necessary for using the PID Online Autotuning VI. You also can use the PID Advanced Autotuning VI to **implement and autotune a PID controller** from a single VI.

Complete the following steps autotune a previously implemented PID controller.

1. If you have not already done so, **implement a PID controller** using the PID Advanced VI.
2. Put the controller in manual mode by setting the auto? input of the PID Advanced VI to FALSE.
3. Add the PID Online Autotuning VI to the block diagram within the control loop but to the left of the PID Advanced VI.
4. **Manually select the polymorphic instance** of the PID Online Autotuning VI.

5. Without deleting the controls, constants, or shift registers that contain the values, delete the wires to the setpoint, process variable, and PID gains inputs of the PID Advanced VI.
6. Wire the values to the Setpoint in, Process Variable, and PID gains in inputs of the PID Online Autotuning VI.
7. Create a Boolean control with latched mechanical action and a default value of FALSE, and wire the control to the Autotune? input of the PID Online Autotuning VI.
8. Wire the setpoint out, Process Variable Out, and PID gains out outputs of the PID Online Autotuning VI to the setpoint, process variable, and PID gains inputs of the PID Advanced VI.
9. (Optional) If you are specifying a value for the dt (s) input, wire the same value to both VIs.
10. Run the VI.
11. Set Autotune? to TRUE to begin autotuning.

Refer to the Autotuning PID Online and Inline VI in the `labview\examples\control\PID` for an example of implementing autotuning with the PID Online Autotuning VI and the PID Advanced VI.



Open example



Find related examples

## Programmatically Implementing and Autotuning a PID Controller

You can use the PID Advanced Autotuning VI both to implement and autotune a PID controller.



### NOTE

The PID Advanced Autotuning VI incorporates the functionality of the PID Advanced VI and the PID Online Autotuning VI. You also can use the PID Online Autotuning VI to **programmatically autotune a previously implemented PID controller**.

Complete the following steps to implement a tuned PID controller with the PID Advanced Autotuning VI.

1. Add the PID Advanced Autotuning VI to the block diagram within a control loop.
2. Wire the inputs you want to use to implement the controller. Minimally, wire the setpoint, process variable, and PID gains inputs.
3. Create a control for the autotuning parameters input and select the controller type and control specification you want to use.
4. Set the technique component of the autotuning parameters input to the **autotuning technique** you want to use.
5. Create a Boolean control with latched mechanical action and a default value of FALSE for the Autotune? input.

6. Run the VI.
7. Set Autotune? to TRUE to begin autotuning.

Refer to the Autotuning PID Online VI in the `labview\examples\control\PID` for an example of implementing autotuning with the PID Advanced Autotuning VI.

 Open example  Find related examples

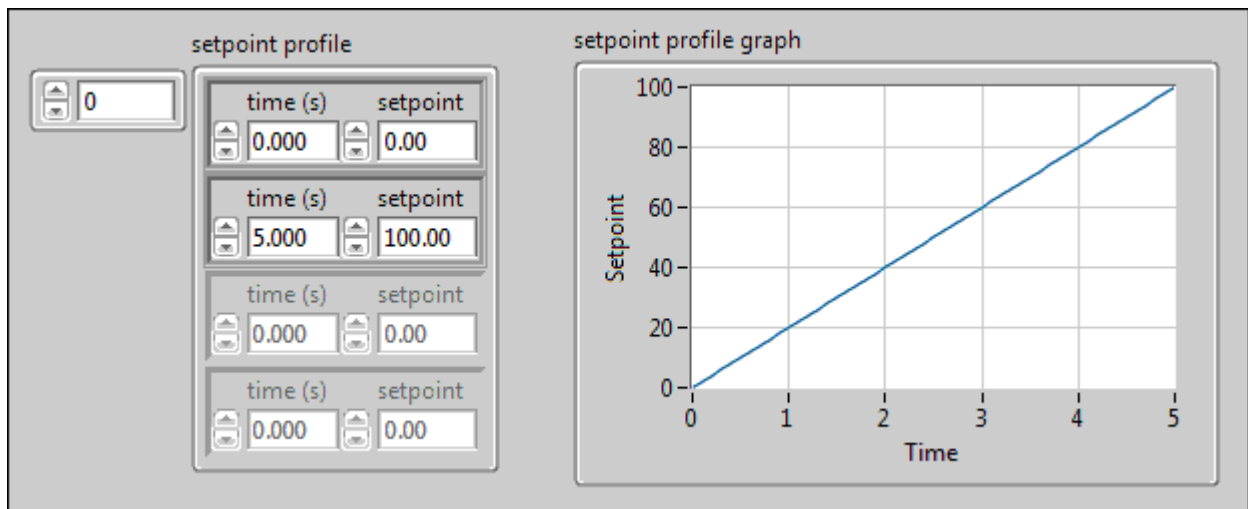
## Setpoint Profile Generation

You can use the PID Setpoint Profile VI inside a control loop to generate a profile of setpoint values over time for a ramp and soak type PID application. For example, you might want to ramp the setpoint temperature of an oven control system over time, and then hold, or soak, the setpoint at a certain temperature for another period of time. You can use the PID Setpoint Profile VI to implement any arbitrary combination of ramp, hold, and step functions.

Specify the setpoint profile as an array of pairs of time and setpoint values with the time values in ascending order.

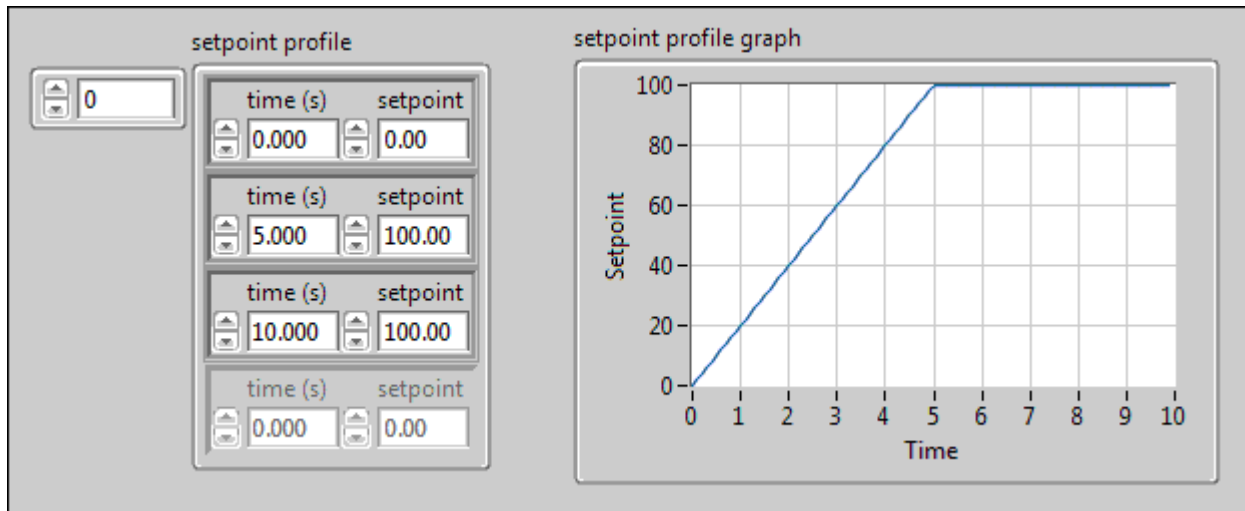
### Ramp Setpoint Profile

You can specify a ramp setpoint profile with two setpoint profile array values, as shown in the following image.



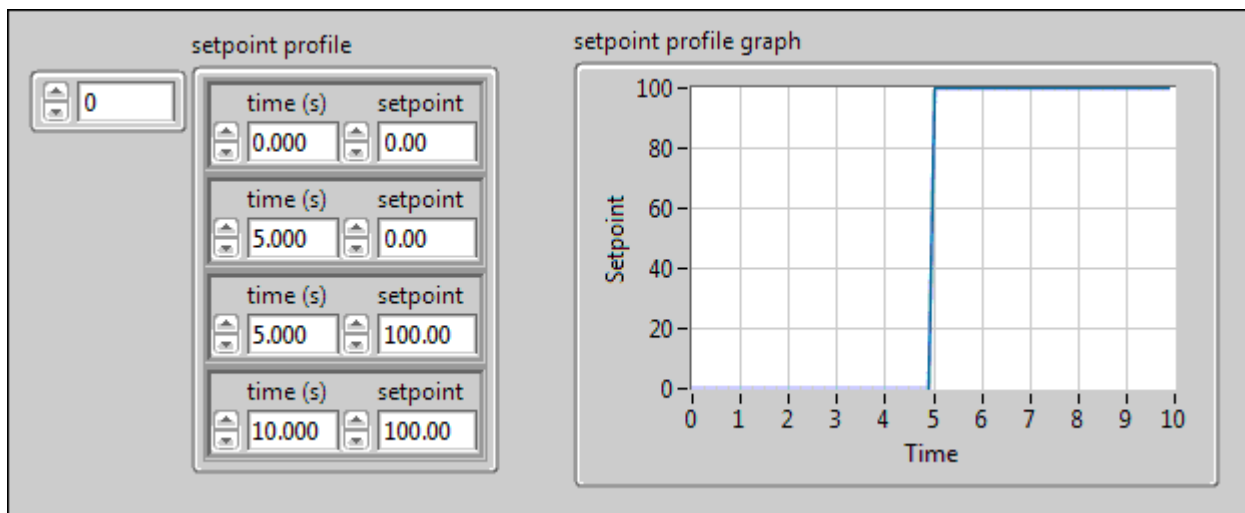
### Ramp and Hold Setpoint Profile

A ramp and hold setpoint profile also can have two successive array values with the same setpoint value, as shown in the following image.



### Step Setpoint Profile

A step setpoint profile can have two successive array values with the same time value but different setpoint values, as shown in the following image.

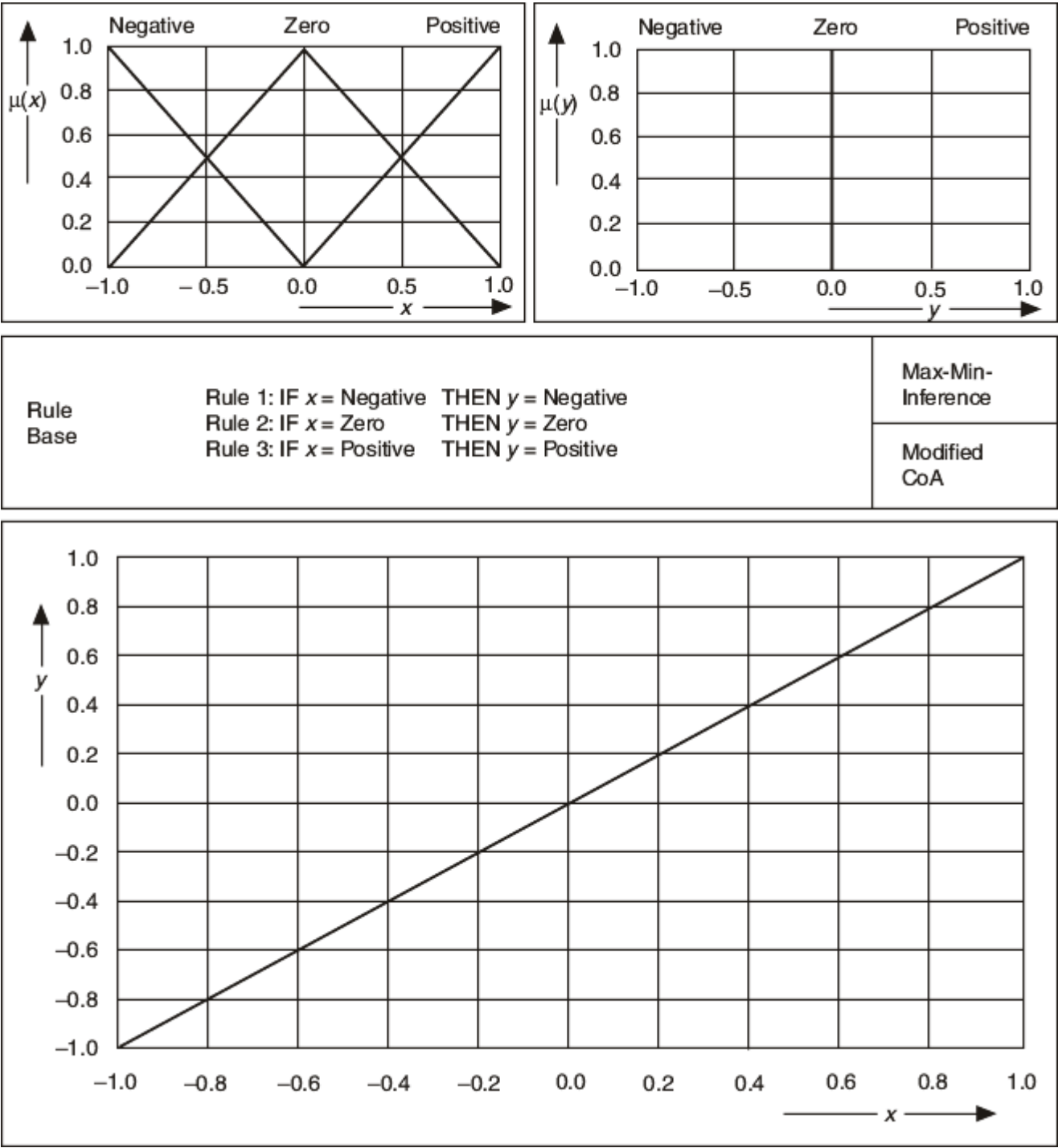


The PID Setpoint Profile VI returns a single setpoint value determined from the current elapsed time. The first call to the VI initializes the current time in the setpoint profile to 0. On subsequent calls, the VI determines the current time from the previous time and the dt input value. If you reinitialize the current time to 0 by passing a value of TRUE to the reinitialize? input, you can repeat the specified setpoint profile.

### Singletons as Output Terms with Fully Overlapping Input Terms

The simplest way to obtain a linear **controller characteristic** is to use singletons as consequent terms with entirely overlapping input terms. Singletons are normalized rectangular **membership functions** with an infinitely small width. Using singleton membership functions for consequent terms makes the **Center of Area (CoA)**

defuzzification method identical to the **Center of Maximum (CoM)** method. The following image shows the controller for the CoA method using singleton membership functions.



The controller characteristic remains relatively unchanged when you leave the input terms entirely overlapped to vary the overlapping degree of the membership functions for the consequent terms, especially if all the consequent terms are equal in width. Then only the typical values of the consequent terms are significant.

Therefore, in most closed-loop control applications, you can use singleton membership functions to model the output terms sufficiently rather than using triangular or other membership function types.



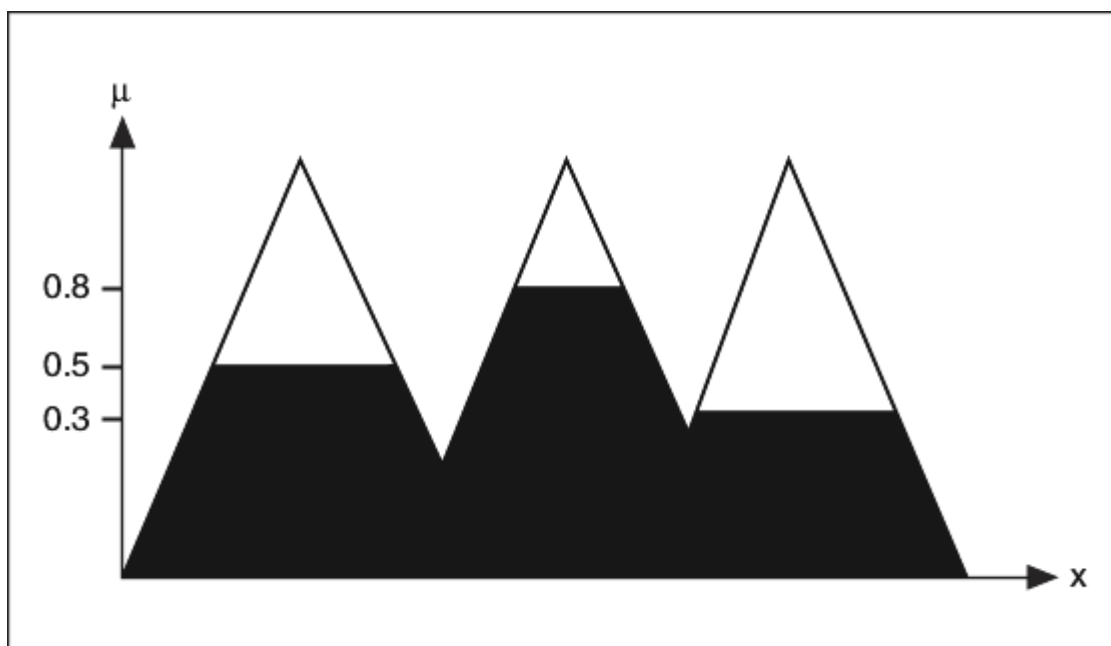
## Specifying a Consequent Implication Method

A fuzzy controller uses a consequent implication method to scale the **membership functions** of each output **linguistic variable** based on the corresponding rule weight before performing defuzzification.

With PID and Fuzzy Logic VIs, you can use either the Minimum or Product consequent implication method.

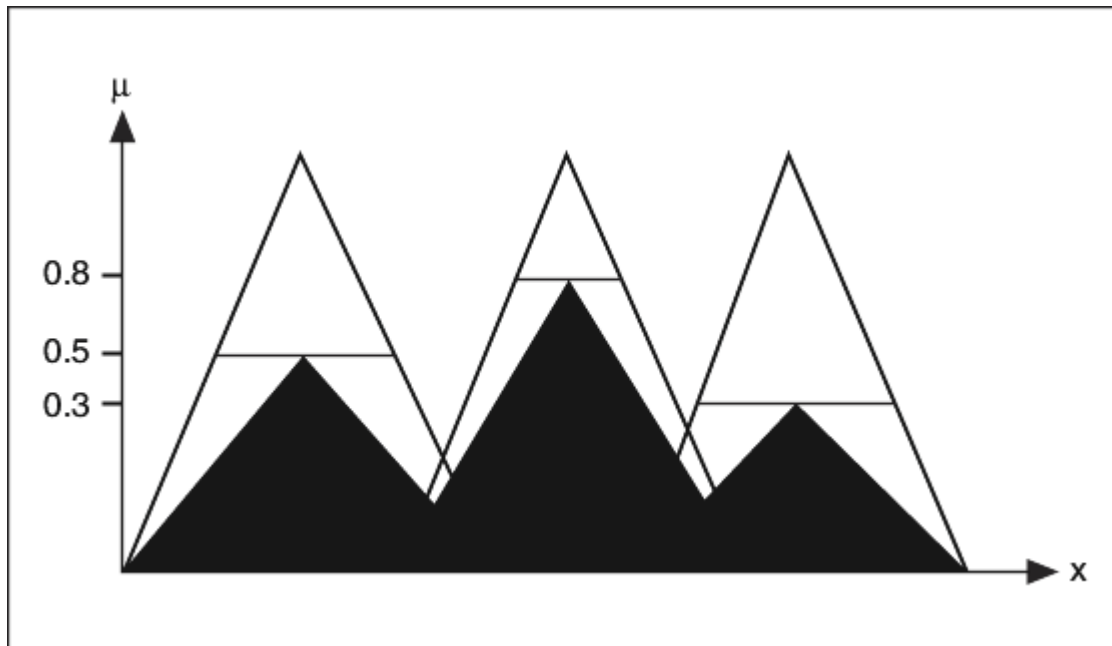
### Minimum Consequent Implication Method

In the Minimum implication method, the fuzzy logic controller truncates the output membership functions at the value of the corresponding rule weights. For example, if an output linguistic variable has three membership functions with rule weights 0.5, 0.8, and 0.3, respectively, the scaled membership functions might appear similar to the following image.



### Product Consequent Implication Method

In the Product implication method, the fuzzy logic controller scales the output membership functions at the value of the corresponding rule weights. For example, if an output linguistic variable has three membership functions with rule weights 0.5, 0.8, and 0.3, respectively, the scaled membership functions might appear similar to the following image.



## Specifying an Antecedent Connective

When **creating a rule base**, if a rule has more than one antecedent, you must specify an antecedent connective to determine how to calculate the truth value of the aggregated rule antecedent.

Because **linguistic variables** can have partial degrees of membership within **linguistic terms**, you cannot use Boolean operators from conventional dual logic as antecedent connectives. PID and Fuzzy Logic VIs use the following antecedent connectives instead.

AND (Minimum)

$$\mu A * B = \min (\mu A, \mu B)$$

AND (Product)

$$\mu A * B = (\mu A, \mu B)$$

OR (Maximum)

$$\mu A + B = \max (\mu A, \mu B)$$

OR (Probabilistic)

$$A + B = (A + B - AB)$$

Notice that these definitions agree with the logical operators used in Boolean logic. A truth table uses conventional operators to yield equivalent results.

The AND (Minimum) antecedent connective specifies to use the smallest degree of membership of the antecedents as the truth value of the aggregated rule antecedent, while the AND (Product) specifies to use the product of the degrees of membership of the antecedents. The OR (Maximum) antecedent connective specifies to use the largest degree of membership of the antecedents. The OR (Probabilistic) antecedent connective specifies to use the probabilistic sum of the degrees of membership of the antecedents.

### Antecedent Connectives for a Vehicle Maneuvering Fuzzy System

Assume the following rules are invoked for the input values of a vehicle maneuvering fuzzy system.

|                                               |                                                                      |               |                                                                           |
|-----------------------------------------------|----------------------------------------------------------------------|---------------|---------------------------------------------------------------------------|
| (1)                                           | IF Vehicle Position $x$ is Center (degree of membership = 0.8)       | AND (Minimum) | Vehicle Orientation $\beta$ is Left Up (degree of membership = 1.0) = 0.8 |
| THEN Steering Angle $\phi$ is Negative Small  |                                                                      |               |                                                                           |
| (2)                                           | IF Vehicle Position $x$ is Right Center (degree of membership = 0.1) | AND (Minimum) | Vehicle Orientation $\beta$ is Left Up (degree of membership = 1.0) = 0.1 |
| THEN Steering Angle $\phi$ is Negative Medium |                                                                      |               |                                                                           |

Notice that each rule uses the AND (Minimum) antecedent connective. In Rule 1, Vehicle Position  $x$  has a degree of membership of 0.8 within the linguistic term Center and Vehicle Orientation  $\beta$  has a degree of membership of 1.0 within the linguistic term Left Up. Because the antecedent connective is AND (Minimum), the fuzzy controller for this fuzzy system uses the smallest degree of membership of the antecedents, or 0.8, as the truth value of the aggregated rule antecedent. Similarly, the smallest degree of membership of the antecedents in Rule 2 is 0.1. Therefore, the fuzzy controller uses 0.1 as the truth value of the aggregated rule antecedent. The truth value of the aggregated rule antecedent is equivalent to the degree of truth of the rule.

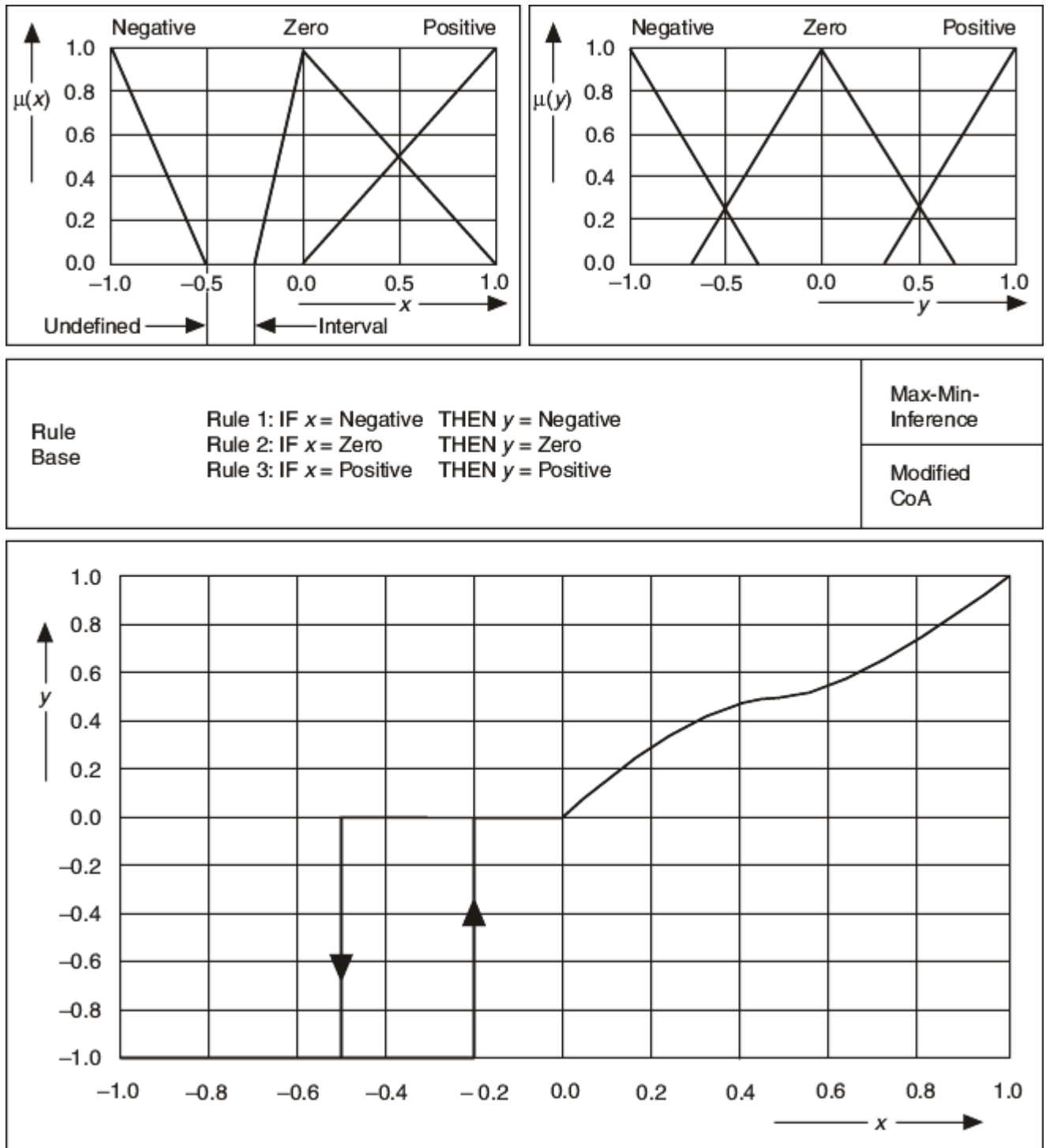
If these two rules are the only rules invoked for a given set of input values, the other linguistic terms for the Steering Angle  $\phi$  output linguistic variable have a truth value of 0. The following list describes the final truth values for each of the linguistic terms.

|                                                                                                              |                                                                                                                |                                   |
|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|-----------------------------------|
| Negative Large Negative<br>Medium Negative Small<br>Zero Positive Small<br>Positive Medium Positive<br>Large | to a degree of to a degree of to a degree of to a<br>degree of to a degree of to a degree of to a<br>degree of | 0.0 0.1 0.8<br>0.0 0.0 0.0<br>0.0 |
|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|-----------------------------------|

If a rule has more than one consequent, the fuzzy logic controller must evaluate the aggregated rule consequent. When you implement a fuzzy controller with PID and Fuzzy Logic VIs, the fuzzy controller always considers only the consequent that has the largest degree of membership.

## Undefined Input Term Intervals

If the **rule base** has undefined intervals within input and output linguistic terms, or if the rule base is incomplete, you must specify the output of the fuzzy controller. If no rule is available for a certain situation, the output value remains undefined. One way to avoid this problem is to leave the current output value unchanged until the controller encounters a situation that is covered by the rules. The following image shows the resulting effect on the controller characteristic.



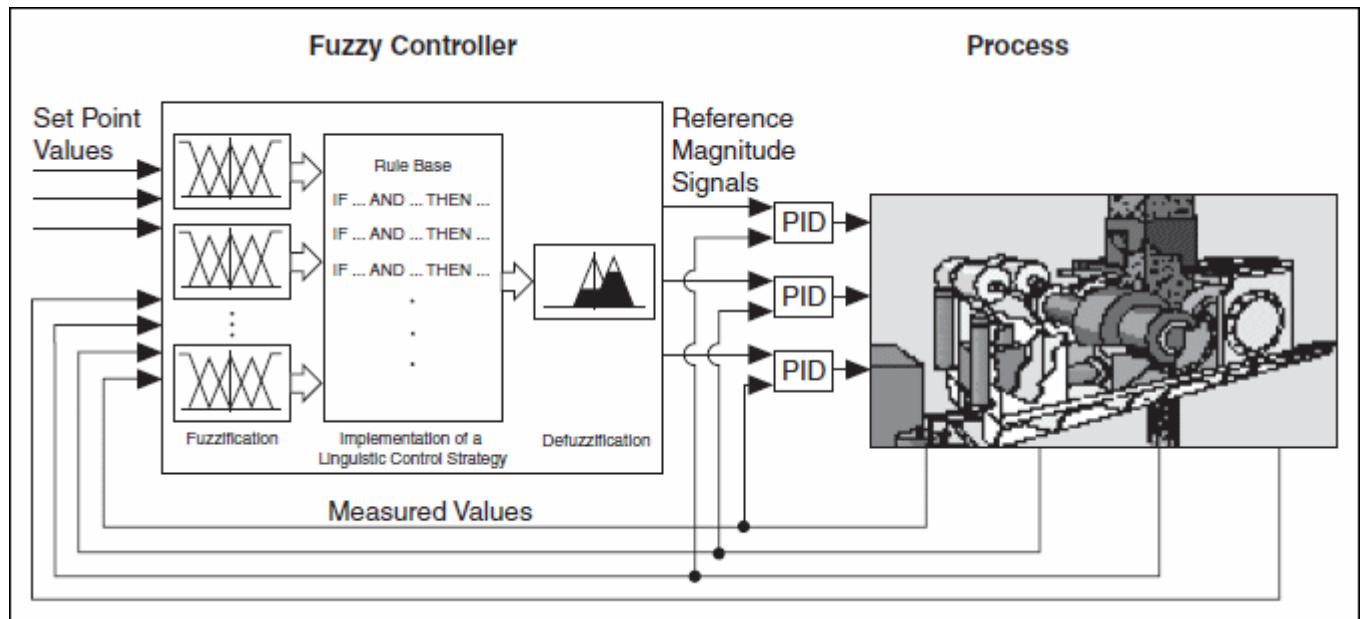
If you use an old output value as a default value, undefined intervals or incomplete rule bases can lead to hysteretic effects on the [controller characteristic](#).

You can use non-overlapping, rectangular-shaped consequent terms to obtain an exact linear [controller characteristic](#) for a single-input controller. In this case, both the area and momentum vary linearly with the degree of truth, and overlapping regions of the output linguistic terms do not cause any distortion.

## Using Fuzzy Controllers with PID Controllers

The chemical industries and process technologies often use fuzzy controllers with underlying PID control loops. The fuzzy controllers these industries use are PID fuzzy controllers that control single-process parameters. Usually, people supervise the operating point of the entire process.

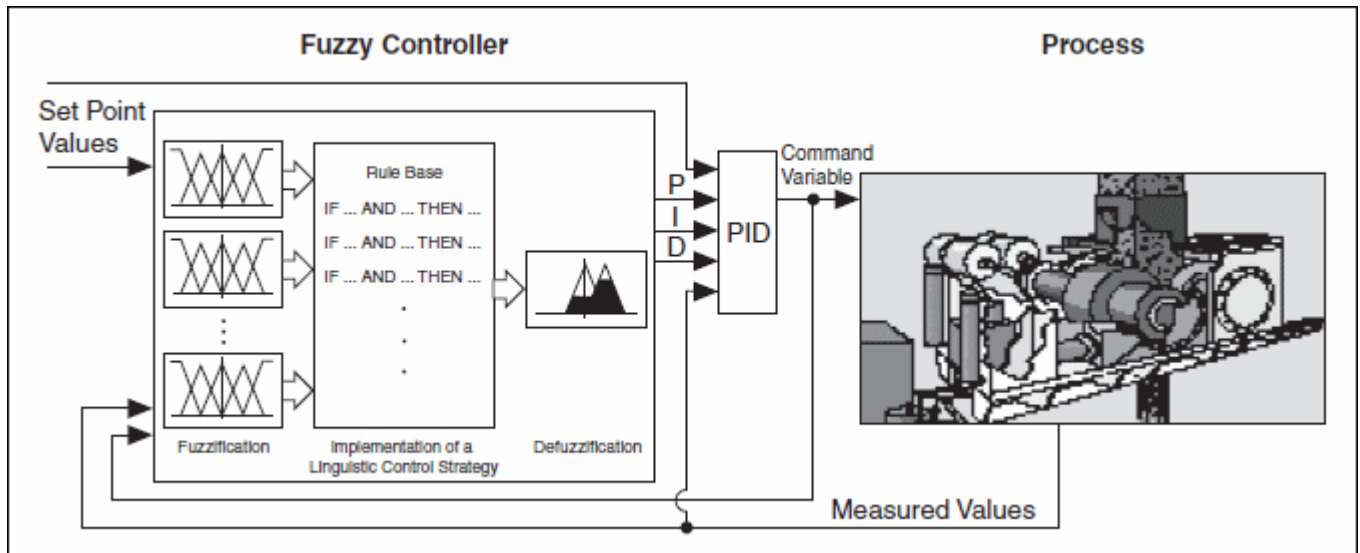
The following image shows the controller structure of the fuzzy controller with underlying PID control loops.



For automatic operation of such multi-variable control problems, you must build a model-based controller. However, for most applications, either the process is too complex to model adequately, or the mathematical modeling task requires too much time.

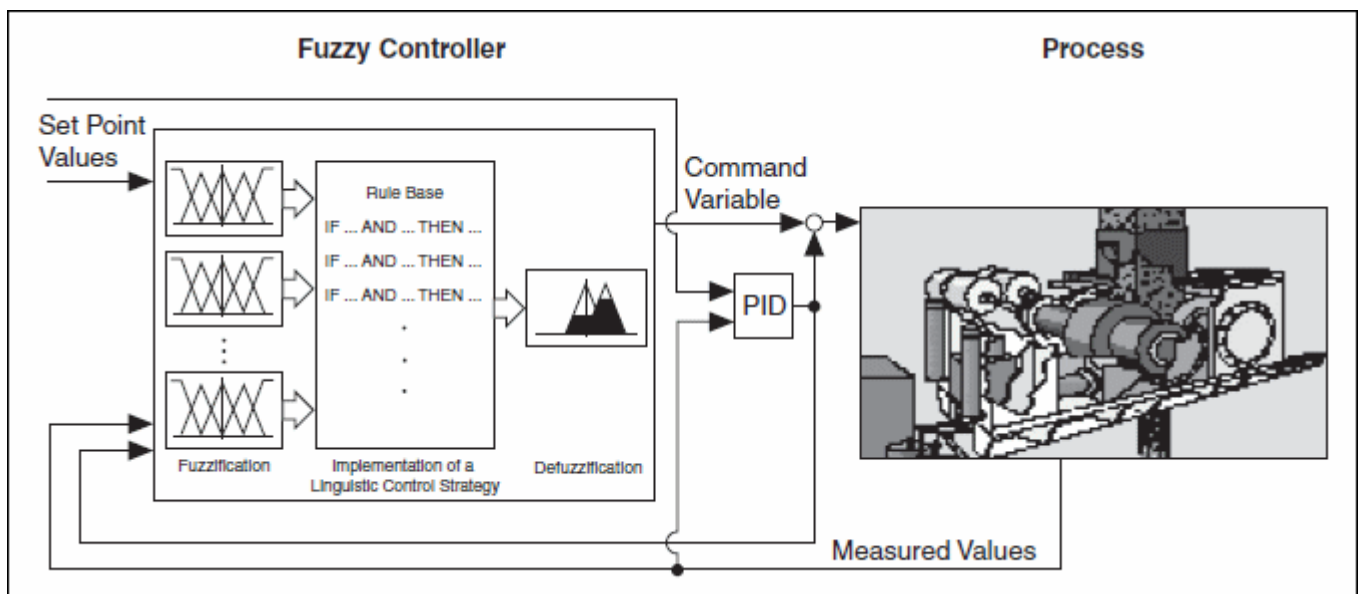
### Using a Fuzzy Controller to Tune PID Controller Parameters

The following image shows how to use a fuzzy controller to tune the parameters of a conventional PID controller automatically. This fuzzy controller constantly interprets the process reaction and calculates the optimal P, I, and D gains. You can apply this control structure to processes that change their characteristics over time.



### Using a Fuzzy Controller to Correct PID Controller Output

In the following image, both the fuzzy controller and the PID controller work in parallel. The process adds the output signals from both controllers, but the output signal from the fuzzy controller is zero under normal operating conditions. The PID controller output leads the process. The fuzzy controller intervenes only when it detects abnormal operating conditions, such as strong disturbances.



## PID 자동튜닝

### Implementing Autotuning with the Autotuning Wizard

To use the Autotuning Wizard to improve your controller performance, you must first create your control application and determine PID parameters that produce stable control of the

system. You can develop the control application using either the PID VI, the PID Gain Schedule VI, or the PID Autotuning VI. Because the PID Autotuning VI has inputs and outputs consistent with the other PID VIs, you can replace many PID VIs with it.



#### NOTE

The PID Autotuning VI cannot directly replace the PID Advanced VI or the more advanced autotuning VIs, such as PID Advanced Autotuning or PID Online Autotuning. These VIs provide advanced functionality that the PID Autotuning VI does not support.

### Autotuning-Specific Inputs

The PID Autotuning VI has several inputs and outputs specific to the autotuning procedure, including the following inputs:

- **autotuning parameters** —A cluster of parameters that the PID Autotuning VI uses for the autotuning process. The Autotuning Wizard allows you to specify all of these parameters manually, so wiring this input is optional.
- **autotune?** —Specifies whether to invoke the Autotuning Wizard and begin autotuning. Wire a Boolean control, such as a button, to this input so a user can press the button to open the Autotuning Wizard. The Autotuning Wizard steps the user through the autotuning process.



#### NOTE

**Set the Boolean control mechanical action** to Latch When Released so the Autotuning Wizard does not open repeatedly when the user presses the button.

### Autotuning-Specific Outputs

The following outputs are specific to the autotuning procedure:

- **tuning completed?** —Indicates when the autotuning process is complete. You can use this output to determine when to update the PID gains values.
- **PID gains out** —Returns the tuned PID gains. Before autotuning completes, PID gains out returns the same values as the PID gains input. PID gains out updates only when autotuning completes and tuning completed? returns TRUE.

## Implementing PID Controllers with the PID VIs

The PID VIs are all variations of the basic PID VI, and each variation provides additional functionality as described in the following table. You can use these VIs interchangeably because they use consistent inputs and outputs where possible. Refer to the help topic for the VI you want to use for specific information about the VI.



| Use Case                                                                                                                                                                                                                                                                                                               | VI Name                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| <ul style="list-style-type: none"><li>Simple systems, or when an efficient algorithm is required.</li><li>Implements the basic PID algorithm</li></ul>                                                                                                                                                                 | PID VI                          |
| <ul style="list-style-type: none"><li>Complex systems that require options for manual or automatic operation, setpoint weighting, filtering of derivative action, and so on.</li><li>Implements the advanced PID algorithm and provides extra input parameters to expand on the functionality of the PID VI.</li></ul> | PID Advanced VI                 |
| Complex systems that require the same advanced options as the PID Advanced VI and the ability to autotune parameters.                                                                                                                                                                                                  | PID Advanced Autotuning VI      |
| Systems that contain significant dead time.                                                                                                                                                                                                                                                                            | PID Autotuning (Temperature) VI |
| Systems that require lead/lag function.                                                                                                                                                                                                                                                                                | PID Lead-Lag VI                 |



(Windows) To view related topics, click the Locate button, shown at left, in the toolbar at the top of this window. The LabVIEW Help highlights this topic in the Contents tab so you can navigate the related topics.

Related Information

Fault Protection

Multi-Loop Control

Setpoint Profile Generation

Gain Scheduling

Control Output Rate Limiting

Lead/Lag

Converting Between Percentage of Full Scale and Engineering Units

## Filtering Control Inputs



## RELATED INFORMATION

[Implementing the PID Algorithm with the PID VIs](#) on page 1710[Advanced PID Algorithm](#) on page 1717[Autotuning PID Controllers \(PID and Fuzzy Logic\)](#) on page 1721[Lead/Lag](#) on page 1714[Fault Protection](#) on page 1680[Multi-Loop Control](#) on page 1675[Setpoint Profile Generation](#) on page 1698[Gain Scheduling](#) on page 1688[Control Output Rate Limiting](#) on page 1736[Lead/Lag](#) on page 1714[Converting Between Percentage of Full Scale and Engineering Units](#) on page 1736[Filtering Control Inputs](#) on page 1681

## Implementing the PID Algorithm with the PID VIs

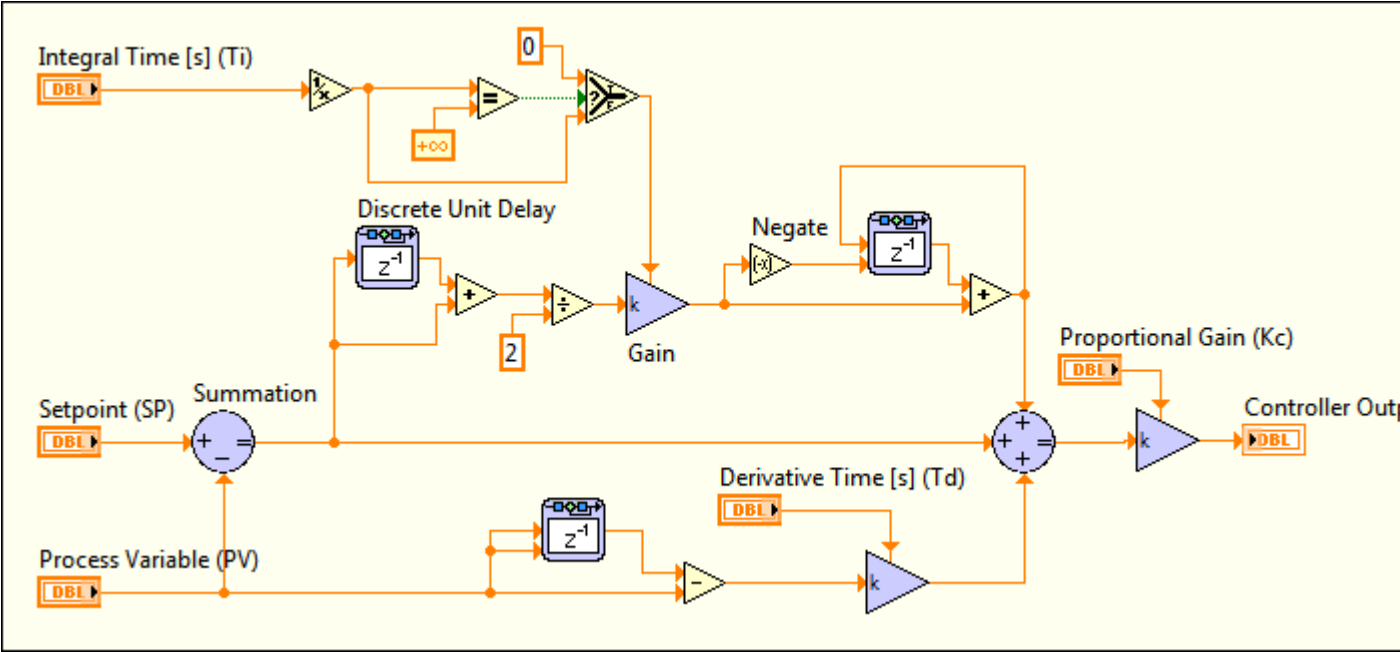
The following sections describe how the basic PID VIs (not including PID Advanced, PID Advanced Autotuning, and other VIs with advanced options) implement the [PID algorithm](#), and the assumptions and transformations necessary to implement the PID controller. To implement a PID controller, LabVIEW requires the algorithm to sample the input signals and discretize the integral and derivative action.



## NOTE

The following formulas apply to most VIs on the PID palette. The PID Advanced and PID Advanced Autotuning VIs use [extended formulas](#) with more advanced optional features.

The following simulation diagram represents the PID implementation provided by the basic PID VIs:



**NOTE**

Constructing a simulation diagram like the one in the previous image requires the LabVIEW Control Design and Simulation Module. However, the PID VIs implement PID controllers for you. You do not need the Control Design and Simulation Module to build PID controllers. Refer to the National Instruments website at [ni.com](http://ni.com) for more information about the Control Design and Simulation Module.

**Error Calculation**

The following formula represents the current error used in calculating proportional, integral, and derivative action:

$$e(k) = (SP(k) - PV(k))$$

|                            |                                                    |
|----------------------------|----------------------------------------------------|
| where                      | k is the index of the sampled signal at time k * t |
| e(k) is the current error  |                                                    |
| SP is the setpoint         |                                                    |
| PV is the process variable |                                                    |

**Proportional Action**

Proportional action is the controller gain times the error, as shown in the following formula:

$$u_p(k) = (K_c * e(k))$$

|                             |                              |
|-----------------------------|------------------------------|
| where                       | $K_c$ is the controller gain |
| $e(k)$ is the current error |                              |

The PID VIs express the proportional component in terms of controller gain. The relationship between controller gain  $K_c$  and proportional band PB is  $K_c = 100 / PB$ .

Integral Action (Trapezoidal Integration)

There are several options for discretizing integral action, such as forward difference, backward difference, and trapezoidal approximation, which is also known as Tustin or Bilinear transformation. The PID VIs use trapezoidal integration to avoid sharp changes in integral action when there is a sudden change in PV or SP, as represented by the following formula:

$$u_I(k) = u_I(k-1) + \frac{K_c}{T_i} \left( \frac{e(k) + e(k-1)}{2} \right) \Delta T$$

|       |                                                   |
|-------|---------------------------------------------------|
| where | $\Delta T$ is the sampling time of the controller |
|-------|---------------------------------------------------|

Derivative Action

Abrupt changes in SP can generate bumps to the output of the controller as a result of applying derivative action to the error e. These bumps are referred to as derivative kick. To avoid derivative kick, you can apply derivative action to the PV only, and not to the error e. The following formula represents the derivative action implemented by the PID VIs, which avoids derivative kick:

$$u_D(k) = -K_c \frac{T_d}{\Delta t} (PV(k) - PV(k-1))$$

Controller Output

Controller output is the summation of the proportional, integral, and derivative action, as shown in the following formula:

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$

Output Limiting and the Anti-Windup Algorithm

The actual controller output is limited to the range specified for control output.

|    |                     |      |                  |
|----|---------------------|------|------------------|
| If | $u(k) \geq u_{max}$ | then | $u(k) = u_{max}$ |
|----|---------------------|------|------------------|

and

|    |                     |      |                  |
|----|---------------------|------|------------------|
| if | $u(k) \leq u_{min}$ | then | $u(k) = u_{min}$ |
|----|---------------------|------|------------------|

The PID VIs use an integral sum correction algorithm that facilitates anti-windup. Windup occurs at the upper limit of the controller output. When the error  $e$  decreases, the controller output decreases, moving out of the windup area. The integral sum correction algorithm prevents abrupt controller output changes when you switch from manual to automatic mode or change any other parameters. The integral sum correction works as follows:

|    |                               |      |                           |
|----|-------------------------------|------|---------------------------|
| If | $ u_P(k) + u_I(k)  >  limit $ | then | $u_I(k) = limit - u_P(k)$ |
|----|-------------------------------|------|---------------------------|

The previous statement demonstrates that the integral sum correction does not take into account the derivative action on value changes.

### Default PID Values and Gain Changes

The default ranges for the parameters SP, PV, and output range correspond to percentage values. However, you can use actual engineering units. Adjust corresponding ranges accordingly. The parameters  $T_i$  and  $T_d$  are specified in minutes.

You can call the PID VIs from inside a While Loop with a fixed cycle time. All the PID VIs are reentrant. Multiple calls from high-level VIs use separate and distinct data. Also, the PID VI has a multi-channel mode where you provide an array that represents an input for each channel.

PID gains can change at any time. When a change is detected, the PID algorithm tries to perform a bumpless transition by adjusting the integral action to keep the output constant with the new parameters. This avoids output changes when you try to modify the PID parameters, and is useful in gain scheduling.

### Gain Scheduling

**Gain scheduling** refers to a system where you change controller parameters based on measured operating conditions. For example, the scheduling variable can be the setpoint, the process variable, a controller output, or an external signal. For historical reasons, the term gain scheduling is used even if other parameters such as derivative time or integral time change. Gain scheduling effectively controls a system with dynamics that change with operating conditions.

With the PID Gain Schedule VI, you can define unlimited sets of PID parameters for gain scheduling. For each schedule, you can use **autotuning** to update the PID parameters.

# Internal Model Control Autotuning Method

Internal model control (IMC) is a control technique that uses an internal model to try to estimate the system response. PID and Fuzzy Logic VIs use IMC when it cannot successfully implement the **Ziegler-Nichols method**, such as when a model has no delay. You also can use the PID Autotuning Design VI to directly implement this autotuning method.

This method assumes a first-order model with delay. If your system is not a first-order model, these techniques try to approximate one during tuning.

The following table shows the tuning formula for the internal model control technique, where K is the model gain,  $\tau$  is the plant time constant,  $\tau_c$  is the desired time constant, and  $\theta$  is the plant delay.

| Controller              | Model                                | $K_c$                                                                   | $T_i$                     | $T_d$                                |
|-------------------------|--------------------------------------|-------------------------------------------------------------------------|---------------------------|--------------------------------------|
| PI (delay-free)         | $\frac{K}{\tau s + 1}$               | $\frac{1}{K} \frac{\tau}{\tau_c}$                                       | $\tau$                    | —                                    |
| PI + Plant (with delay) | $\frac{K_e^{-\theta s}}{\tau s + 1}$ | $\frac{1}{K} \frac{\tau}{\tau_c + \theta}$                              | $\tau$                    | —                                    |
| PID                     | $\frac{K_e^{-\theta s}}{\tau s + 1}$ | $\frac{1}{K} \frac{\tau + \frac{\theta}{2}}{\tau_c + \frac{\theta}{2}}$ | $\tau + \frac{\theta}{2}$ | $\frac{\tau \theta}{2\tau + \theta}$ |

This method is based on algorithms and techniques described in reference material by Seborg, Dale E., Thomas F. Edgar, and Duncan A. Mellichamp. For more information about this method, refer to the reference material by those authors.

## Lead/Lag

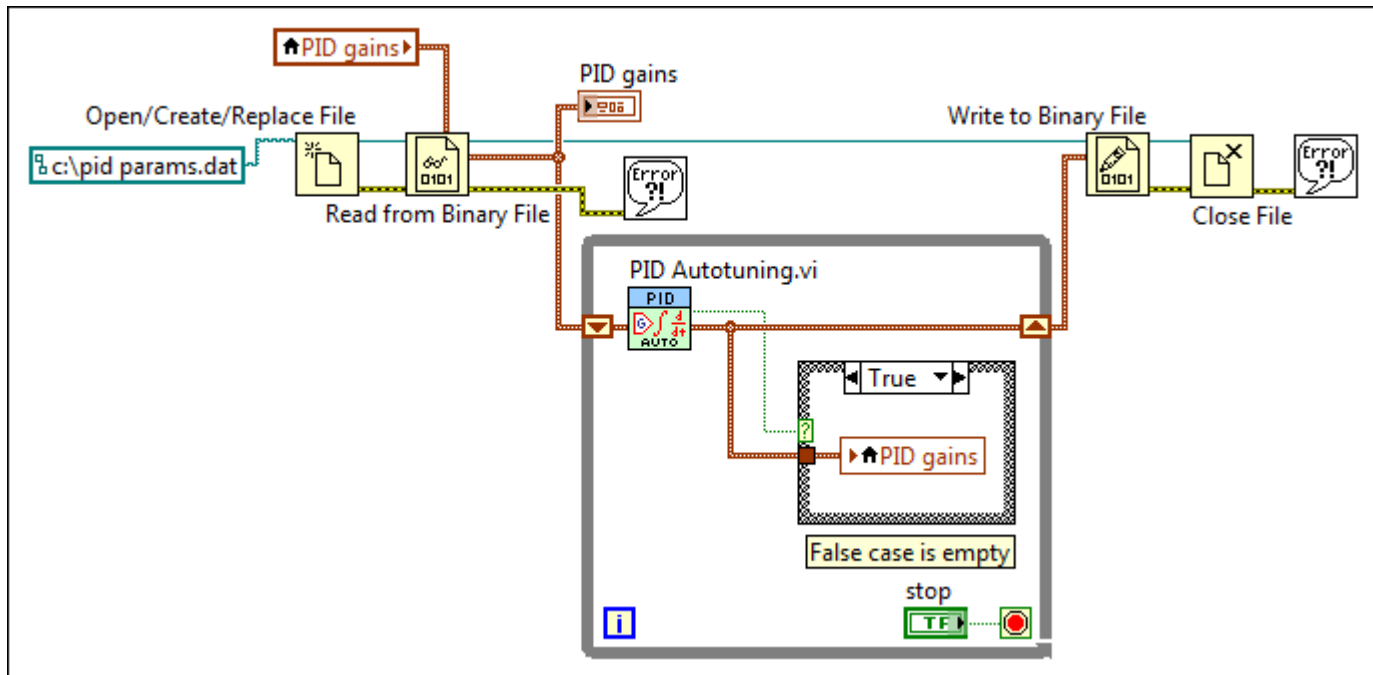
You can use the PID Lead-Lag VI to implement a PID controller with a lead/lag function. The VI uses a positional algorithm that approximates a true exponential lead/lag. Feedforward control schemes often use this kind of algorithm as a dynamic compensator.

The output value on the first call to the VI is the same as the input value.

## Logging Parameter Values After Autotuning

The PID autotuning VIs, including PID Autotuning, PID Advanced Autotuning, and PID Online Autotuning, update the parameters of a PID controller and return updated PID gains values for the next iteration of the control application. To ensure that the correct PID gains values are available each time you run your control application, you must either save the VI you are using to implement the application each time it runs, or use a datalog file to save the PID gains.

The following block diagram shows a VI that uses the File I/O VIs and functions to create a datalog file and save the latest value of the PID gains out output of the PID Autotuning VI to the file.



Before the control loop begins, the Open/Create/Replace File function opens a datalog file, and the Read from Binary File function reads the file to obtain the PID gains parameters. When the autotuning procedure runs, a local variable updates the PID gains control. After the control loop is complete, the Write to Binary File function writes the current PID gains cluster to the datalog file and saves the file. This ensures that each time the control application runs, the PID Autotuning VI uses updated parameters.



#### RELATED INFORMATION

[Autotuning PID Controllers \(PID and Fuzzy Logic\)](#) on page 1721

[Updating PID Parameters After Autotuning](#) on page 1725

## Online Autotuning Techniques

To tune PID parameters of a system, some PID VIs allow you to select the technique to apply to the system. You can choose from two techniques that implement a step response-based measurement and two techniques that implement a frequency response-based measurement.

On the PID Advanced Autotuning VI, the autotuning parameters input allows you to configure the technique to use. With the PID Online Autotuning polymorphic VI, select the polymorphic instance that corresponds to a tuning technique you want to use. Other PID VIs control the tuning technique for you.

## Step Response Techniques

The following techniques are based on step response measurement:

- **Step Open Loop** —Performs an open-loop step test. This technique applies a step to the input and waits until it reaches steady-state, when the process variable does not change. This technique assumes that you can model any process as a first-order lag and a pure deadtime system.
- **Step Closed Loop** —Performs a closed-loop step test similar to the Step Open Loop technique.

If you do not know the characteristics of the model, such as initial PID gains, you can first perform Step Open Loop tuning on the model to obtain the parameters. Wait until the input reaches steady-state. This process is also known as commissioning. Next, perform Step Closed Loop tuning on the model using the parameters computed by the Step Open Loop tuning process as initial parameters. Both techniques assume a first-order model with delay. If your system is not a first-order model, these techniques try to approximate one during tuning.

## Frequency Response Techniques

The following techniques are based on frequency response measurement

- **PID Relay** —Uses the setpoint relay experiment to determine the information needed to tune the controller. The Autotuning Wizard uses this technique.
- **Relay Feedback** —Uses an on-off relay in a feedback loop to determine ultimate gain and ultimate frequency. This technique is a variation on a closed-loop step test, but is more efficient for systems with a large time constant. This method also incorporates hysteresis to avoid noise in the system.

## Related Information

Step Open Loop Autotuning Technique

Step Closed Loop Autotuning Technique

PID Relay Autotuning Technique



## Relay Feedback Autotuning Technique



### RELATED INFORMATION

[Autotuning PID Controllers \(PID and Fuzzy Logic\)](#) on page 1721

[Step Open Loop Autotuning Technique](#) on page 1724

[Step Closed Loop Autotuning Technique](#) on page 1724

[PID Relay Autotuning Technique](#) on page 1695

[Implementing Autotuning with the Autotuning Wizard](#) on page 1707

[Relay Feedback Autotuning Technique](#) on page 1722

[Step Open Loop Autotuning Technique](#) on page 1724

[Step Closed Loop Autotuning Technique](#) on page 1724

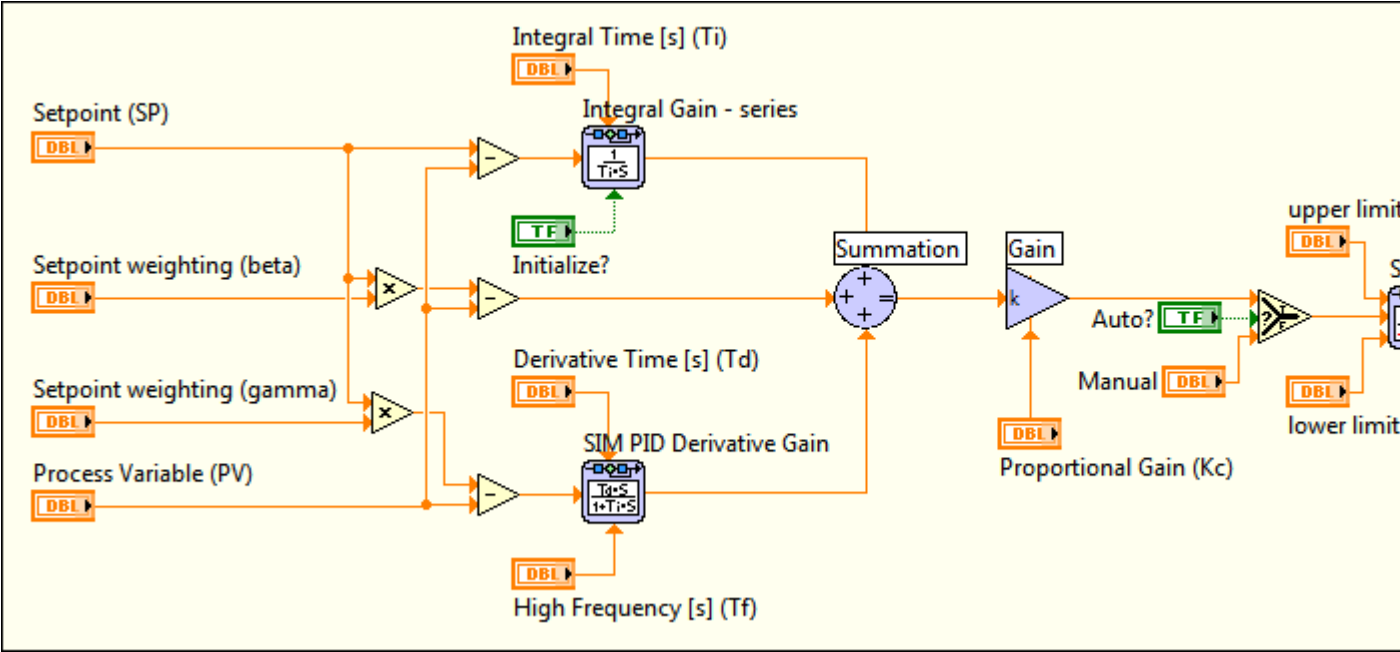
[PID Relay Autotuning Technique](#) on page 1695

[Relay Feedback Autotuning Technique](#) on page 1722

## Advanced PID Algorithm

The PID Advanced VI uses the same algorithm and assumptions as the [PID VIs algorithm](#), but adds the functionality described in the following sections.

The following simulation diagram represents the PID implementation provided by the PID Advanced VI:



**NOTE**  
Constructing a simulation diagram like the one in the previous image requires the LabVIEW Control Design and Simulation Module. However, the PID VIs implement PID controllers for you. You do not need the Control Design and Simulation Module to build PID controllers. Refer to the National Instruments website at [ni.com](http://ni.com) for more information about the Control Design and Simulation Module.

Manual/Automatic Operation

In some situations, you might need to switch off the PID controller and operate the system in open-loop, or manual, mode. The PID Advanced VI uses a method similar to the **anti-windup algorithm** to ensure the transition between manual and automatic modes does not cause any bump to the output of the controller.

Setpoint Weighting for Proportional Action and Derivative Action

Setpoint weighting refers to corrections applied to the error value of the controller. The beta input (B), which can range between 0 and 1, is applied to the proportional action as:

$$eb(k) = (\beta SP(k) - PV(k))$$
$$u_P(k) = K_C * eb(k)$$

|                                   |                |
|-----------------------------------|----------------|
| where                             | e is the error |
| SP is the setpoint                |                |
| PV is the process variable        |                |
| K <sub>C</sub> is controller gain |                |

You can use the beta input to reduce the overshoot of setpoint change and to allow increasing the gain of the controller to improve the disturbance rejection of the controller.

The gamma (γ) input, which also can range between 0 and 1, is applied to the error and to the derivative as:

$$ed(k) = (\gamma SP(k) - PV(k))$$

$$u_D(k) = K_c \frac{T_d}{\Delta T} (ed(k) - ed(k-1))$$

### Filter for Derivative Action

### Error-Squared PID Implemented by the Linearity Factor (L)

The **derivative action** implemented by the PID Advanced VI is based on the backward method for derivative action. However, depending on the values of the derivative action, proportional gain, and sampling time, the controller could generate responses with fast oscillation, especially when performing autotuning with PID. To prevent this issue, this VI provides an alpha parameter that implements an additional filter for the derivative action D. This filter reduces the bandwidth content as defined by:

$$u_D(s) = -K_c \frac{T_d s}{1 + \alpha T_d s}$$

where  $T_d$  is the derivative time in minutes, also called the rate time.

The linearity input introduces linearity change on the gain, as shown in the following equations:

$$e(k) = (SP - PV_f) \left( L + (1 - L) * \frac{|SP - PV_f|}{SP_{range}} \right)$$

The error for calculating proportional action with two degree-of-freedom control is shown in the following formula:

$$eb(k) = (\beta * SP - PV_f) \left( L + (1 - L) * \frac{|\beta SP - PV_f|}{SP_{range}} \right)$$

|                                                                                                                                        |                                           |
|----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| where                                                                                                                                  | $SP_{range}$ is the range of the setpoint |
| $\beta$ is the setpoint factor for the two degree-of-freedom PID algorithm described by the proportional action formula                |                                           |
| $L$ is the linearity factor that produces a nonlinear gain term in which the controller gain increases with the magnitude of the error |                                           |

If L is 1, the controller is linear. A value of 0.1 makes the minimum gain of the controller 10%  $K_c$ . Use of a nonlinear gain term is referred to as an error-squared PID algorithm.

Error-Squared PID Implemented by the Linearity Factor (L)

The linearity input introduces linearity change on the gain, as shown in the following equations:

$$e(k) = (SP - PV_f) \left( L + (1 - L) * \frac{|SP - PV_f|}{SP_{range}} \right)$$

The error for calculating proportional action with two degree-of-freedom control is shown in the following formula:

$$eb(k) = (\beta * SP - PV_f) \left( L + (1 - L) * \frac{|\beta SP - PV_f|}{SP_{range}} \right)$$

|                                                                                                                                      |                                           |
|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| where                                                                                                                                | $SP_{range}$ is the range of the setpoint |
| $\beta$ is the setpoint factor for the two degree-of-freedom PID algorithm described by the proportional action formula              |                                           |
| L is the linearity factor that produces a nonlinear gain term in which the controller gain increases with the magnitude of the error |                                           |

If L is 1, the controller is linear. A value of 0.1 makes the minimum gain of the controller 10%  $K_c$ . Use of a nonlinear gain term is referred to as an error-squared PID algorithm.

Autotuning Methods for Testing PID Parameters

Autotuning automatically tests different parameter values for a PID controller to determine the best response. A variety of methods exist for testing these parameters. PID and Fuzzy Logic VIs support the following methods for PID autotuning:

- Ziegler-Nichols
- Internal Model Control
- Cohen-Coon
- Chien-Hrones-Reswick

The Autotuning Wizard and most of the autotuning PID VIs use the Ziegler-Nichols method or, if Ziegler-Nichols is not possible, the Internal Model Control method. Only the PID

Autotuning Design VI can implement the Cohen-Coon and Chien-Hrones-Reswick tuning methods.



#### RELATED INFORMATION

[Ziegler-Nichols Autotuning Method](#) on page 1726

[Internal Model Control Autotuning Method](#) on page 1714

[Cohen-Coon Autotuning Method](#) on page 1733

[Chien-Hrones-Reswick Autotuning Method](#) on page 1732

## Autotuning PID Controllers (PID and Fuzzy Logic)

Often, many PID controllers are poorly tuned. As a result, some controllers are too aggressive and some controllers are too sluggish. Tuning a controller involves adjusting the parameters of the system to get the desired, or optimal, response. However, manually tuning PID controllers can be difficult when you do not know the process dynamics or disturbances. In this case, you can use autotuning to automatically test different parameter values and determine the best response.

Choose from the following VIs that integrate autotuning based on your specific application needs and your knowledge of tuning theory.

| Use Case                                                                                                                                                                                                                                                                                                                                                                        | VI Name                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| <ul style="list-style-type: none"> <li>Autotuning that runs with minimal user input or knowledge of tuning algorithms.</li> <li>Single VI implements a controller for a system and allows tuning.</li> </ul>                                                                                                                                                                    | PID Advanced Autotuning VI      |
| <ul style="list-style-type: none"> <li>Autotuning that runs with minimal user input or knowledge of tuning algorithms.</li> <li>Single VI implements a controller for a system with significant deadtime and allows tuning. How-To: Programmatically Implementing and Autotuning a PID Controller</li> </ul>                                                                    | PID Autotuning (Temperature) VI |
| <ul style="list-style-type: none"> <li>Controlling the autotuning technique used to identify the system. Other PID VIs automatically determine the technique.</li> <li>Tuning system parameters, then passing the output parameters to the PID Advanced VI to implement that controller. How-To: Programmatically Autotuning a Previously Implemented PID Controller</li> </ul> | PID Online Autotuning VI        |

| Use Case                                                                                                                                                                                                                                                                                                                                                                                                    | VI Name                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| <ul style="list-style-type: none"> <li>Generating PID parameters for a system given an input/output signal. This technique is useful when you have insufficient information about the system and you want to develop initial parameters based on system response.</li> <li>Provides access to multiple methods for determining PID parameters. Other PID VIs automatically determine the method.</li> </ul> | PID Autotuning Design VI |
| <ul style="list-style-type: none"> <li>Performing interactive autotuning in a dialog box that you use to configure the process.</li> <li>Not supported on RT or FPGA hardware targets. How-To: Implementing Autotuning with the Autotuning Wizard</li> </ul>                                                                                                                                                | PID Autotuning VI        |
| <ul style="list-style-type: none"> <li>PID applications with high-speed control and/or high channel count on an FPGA target.</li> <li>Implements a fixed-point PID algorithm.</li> </ul>                                                                                                                                                                                                                    | PID (FPGA) VI            |



## RELATED INFORMATION

[Online Autotuning Techniques](#) on page 1715

[Programmatically Autotuning a Previously Implemented PID Controller](#) on page 1696

[Implementing Autotuning with the Autotuning Wizard](#) on page 1707

[Updating PID Parameters After Autotuning](#) on page 1725

[Logging Parameter Values After Autotuning](#) on page 1714

[Online Autotuning Techniques](#) on page 1715

[Autotuning Methods for Testing PID Parameters](#) on page 1720

## Relay Feedback Autotuning Technique

The Relay Feedback autotuning technique, which is used by the Relay Feedback instance of the PID Online Autotuning VI, uses an on-off relay in a feedback loop to determine ultimate gain and ultimate frequency. This technique is a variation on a closed-loop (ultimate gain) test, but is more efficient for systems with a large time constant. This method also incorporates hysteresis to avoid noise in the system.

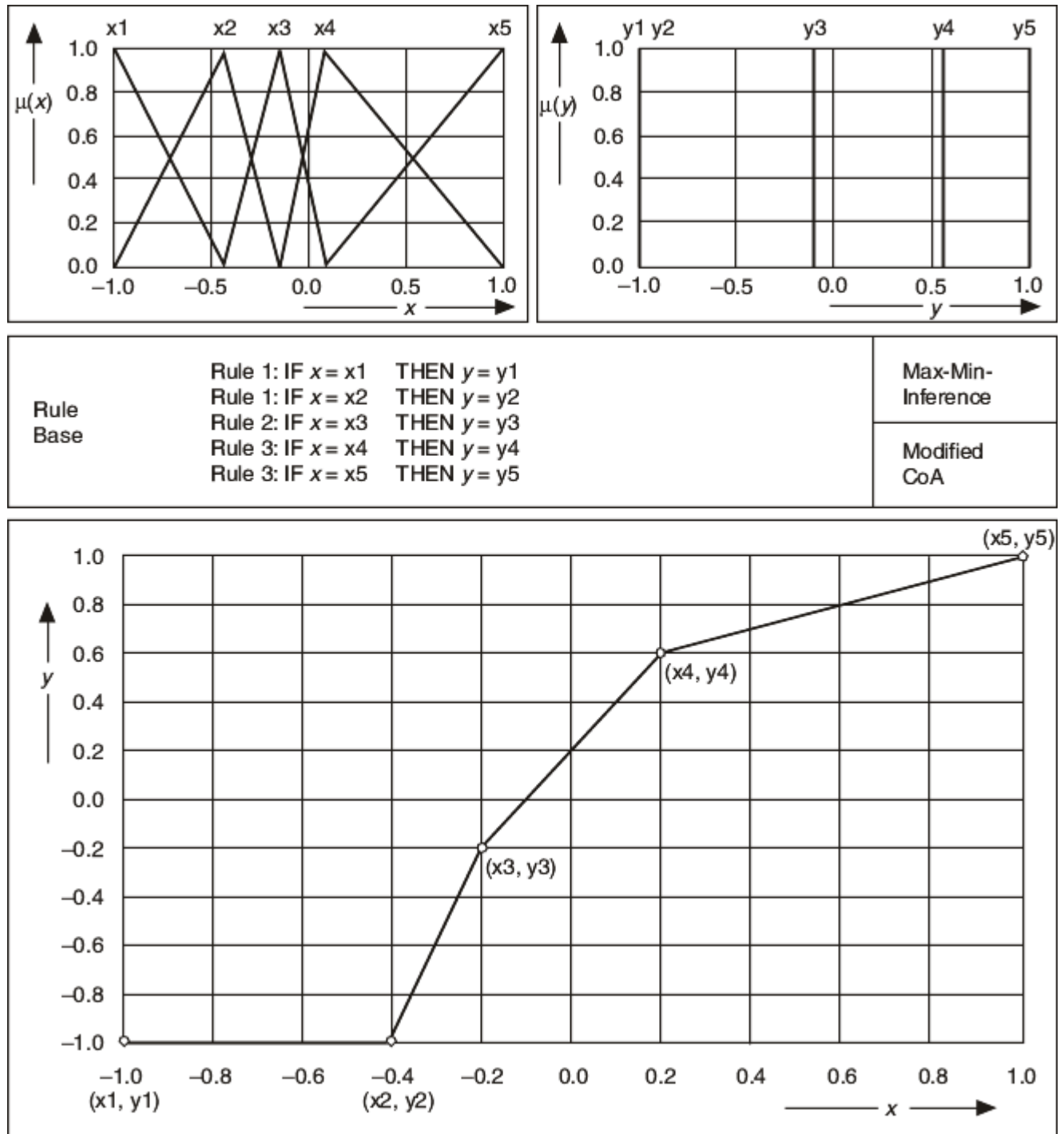


## RELATED INFORMATION

[Online Autotuning Techniques](#) on page 1715

## Reproducing a Given I/O Characteristic

Consider the stepped linear characteristic curve shown in the following image. You can describe the four linear sections with the five circled base points  $(x_i, y_i)$ .



To use a single-input fuzzy controller to reproduce the given characteristic, use five linguistic terms each for the input and output quantities, naming them  $x_1, x_2, \dots, x_5$  and  $y_1, y_2, \dots, y_5$ , respectively. To obtain the stepped linear sections between the base points, you always must have exactly two available active **rules**. To implement this situation, entirely

overlap the triangular **membership functions** for the input variable, giving each a typical value that corresponds to a certain base point component,  $x_i$ .

To obtain characteristic sections that are exactly linear, you must model the output variable with **singleton** membership functions, each of which has a typical value that corresponds to a certain base point component,  $y_i$ . The rule base is then a linguistic enumeration of the five base points.

## Step Closed Loop Autotuning Technique

The Step Closed Loop autotuning technique, which is used by the Step Closed Loop instance of the PID Online Autotuning VI, performs a closed-loop step test. This technique is very accurate and reaches steady state faster than the Step Open Loop technique.

In closed-loop tuning, the PID Online Autotuning VI applies the Step Open Loop response with the controller in a closed loop, and then back-calculates the dynamics of the system by removing the closed loop gain.

The Step Closed Loop autotuning technique measures the deadtime  $T_d$ , time constant  $T$ , and process gain  $K$  values of a controller and multiplies them according to the formulas for the heuristic method in place. For example, most PID VIs use the Ziegler-Nichols method.



### RELATED INFORMATION

[Online Autotuning Techniques](#) on page 1715

[Step Open Loop Autotuning Technique](#) on page 1724

[Step Open Loop Autotuning Technique](#) on page 1724

[Autotuning Methods for Testing PID Parameters](#) on page 1720

[Ziegler-Nichols Autotuning Method](#) on page 1726

## Step Open Loop Autotuning Technique

The Step Open Loop autotuning technique, which is used by the Step Open Loop instance of the PID Online Autotuning VI, performs an open-loop step test. This technique assumes that you can model any process as a first-order lag and a pure deadtime.

The Step Open Loop autotuning technique measures the deadtime  $T_d$ , time constant  $T$ , and process gain  $K$  values of a controller and multiplies them according to the formulas for the heuristic method in place. For example, most PID VIs use the Ziegler-Nichols method.



### RELATED INFORMATION

[Online Autotuning Techniques](#) on page 1715

[Autotuning Methods for Testing PID Parameters](#) on page 1720

[Ziegler-Nichols Autotuning Method](#) on page 1726



## Updating PID Parameters After Autotuning

The PID **autotuning** VIs, including PID Autotuning, PID Advanced Autotuning, and PID Online Autotuning, calculate new parameters for a PID controller that result in the best response. When autotuning, you must write your code to apply the new PID controller parameters (the PID gains) when the autotuning is complete.

You can use the following methods to update PID parameters after autotuning.

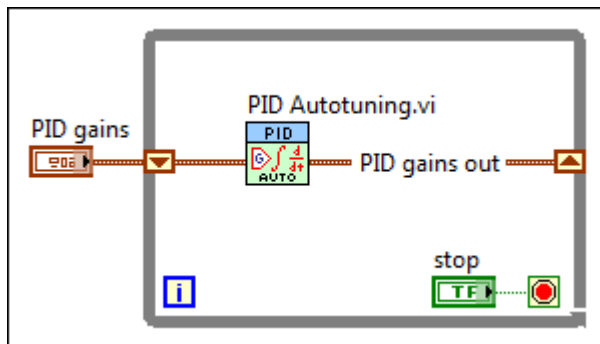


### NOTE

In both of the following methods, you must manually save the PID gains so that you can use the PID gains out values the next time you run the control application. Ensure that the PID gains control shows the current updated parameters, select Operate»Make Current Values Default, and save the VI. If you do not want to manually save the VI after each run, you can [use a datalog file to save PID gains values](#).

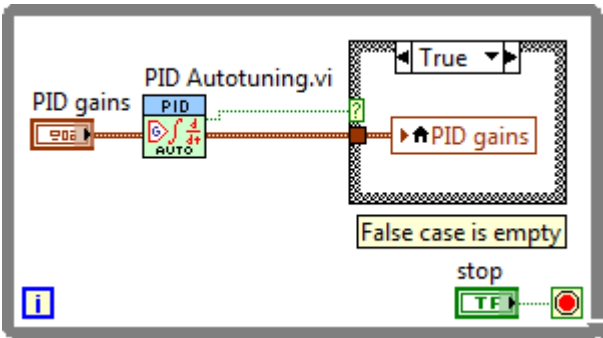
### Using Shift Registers to Update PID Parameters

The following block diagram shows a simple implementation of using shift registers to update PID parameters in the PID Autotuning VI. The shift register on the left stores the initial value of the PID gains. PID gains out then passes an updated value to the right-hand shift register terminal when each control loop iteration completes. This method is simple, but limited in that the user cannot change PID gains manually while the control loop runs.



### Using Local Variables to Update PID Parameters

In place of shift registers, you can use a local variable to store updated PID gains values. In the following block diagram, PID Autotuning VI reads the PID gains control on each iteration of the While Loop, and a local variable updates the control only when tuning complete? is TRUE. This method allows for manual control of the PID gains while the control loop executes.



## Ziegler-Nichols Autotuning Method

Ziegler and Nichols developed heuristic methods for determining the parameters of a PID controller. The Autotuning Wizard and PID VIs that use the **PID Relay autotuning technique** use this method.

When using the Ziegler-Nichols autotuning method, select one of the following three types of desired loop response performance: fast (1/4 damping ratio), normal (some overshoot), and slow (little overshoot). The following tables show the tuning formulas for each type of loop performance.



### 노트

During tuning, the process remains under closed-loop PID control. You do not need to switch off the existing controller and perform the experiment under open-loop conditions. In the setpoint relay experiment, the setpoint signal mirrors the setpoint for the PID controller.

## Ziegler-Nichols Closed Loop/Frequency Response Method

The following tables show the tuning formulas for different types of loop performance under proportional-only control, where  $K_u$  is the controller gain and  $T_u$  is the period of oscillation.

| Fast Performance |           |           |            |
|------------------|-----------|-----------|------------|
| Controller       | $K_c$     | $T_i$     | $T_d$      |
| P                | $0.5 K_u$ | —         | —          |
| PI               | $0.4 K_u$ | $0.8 T_u$ | —          |
| PID              | $0.6 K_u$ | $0.5 T_u$ | $0.12 T_u$ |

| Normal Performance |           |       |       |
|--------------------|-----------|-------|-------|
| Controller         | $K_c$     | $T_i$ | $T_d$ |
| P                  | $0.2 K_u$ | —     | —     |

| Normal Performance |            |           |            |
|--------------------|------------|-----------|------------|
| PI                 | $0.18 K_u$ | $0.8 T_u$ | —          |
| PID                | $0.25 K_u$ | $0.5 T_u$ | $0.12 T_u$ |

| Slow Performance |            |           |            |
|------------------|------------|-----------|------------|
| Controller       | $K_c$      | $T_i$     | $T_d$      |
| P                | $0.13 K_u$ | —         | —          |
| PI               | $0.13 K_u$ | $0.8 T_u$ | —          |
| PID              | $0.15 K_u$ | $0.5 T_u$ | $0.12 T_u$ |

### Ziegler-Nichols Open Loop/Step Response Method

The following tables show the tuning formulas for different types of loop performance under PI or PID control, where  $T_P$  is the time constant and  $\tau$  is the dead time.

| Fast Performance |                  |             |            |
|------------------|------------------|-------------|------------|
| Controller       | $K_c$            | $T_i$       | $T_d$      |
| P                | $T_P / \tau$     | —           | —          |
| PI               | $0.9 T_P / \tau$ | $3.33 \tau$ | —          |
| PID              | $1.1 T_P / \tau$ | $2.0 \tau$  | $0.5 \tau$ |

| Normal Performance |                   |             |            |
|--------------------|-------------------|-------------|------------|
| Controller         | $K_c$             | $T_i$       | $T_d$      |
| P                  | $0.44 T_P / \tau$ | —           | —          |
| PI                 | $0.4 T_P / \tau$  | $5.33 \tau$ | —          |
| PID                | $0.53 T_P / \tau$ | $4.0 \tau$  | $0.8 \tau$ |

| Slow Performance |                   |             |       |
|------------------|-------------------|-------------|-------|
| Controller       | $K_c$             | $T_i$       | $T_d$ |
| P                | $0.26 T_P / \tau$ | —           | —     |
| PI               | $0.24 T_P / \tau$ | $5.33 \tau$ | —     |

| Slow Performance |                   |            |            |
|------------------|-------------------|------------|------------|
| PID              | $0.32 T_P / \tau$ | $4.0 \tau$ | $0.8 \tau$ |



**노트**  
The previous tables provides proportional gain  $K_c$  values. The following equation describes the relationship between proportional gain and proportional band PB:  
 $K_c = 100 / PB$

## 역퍼지화

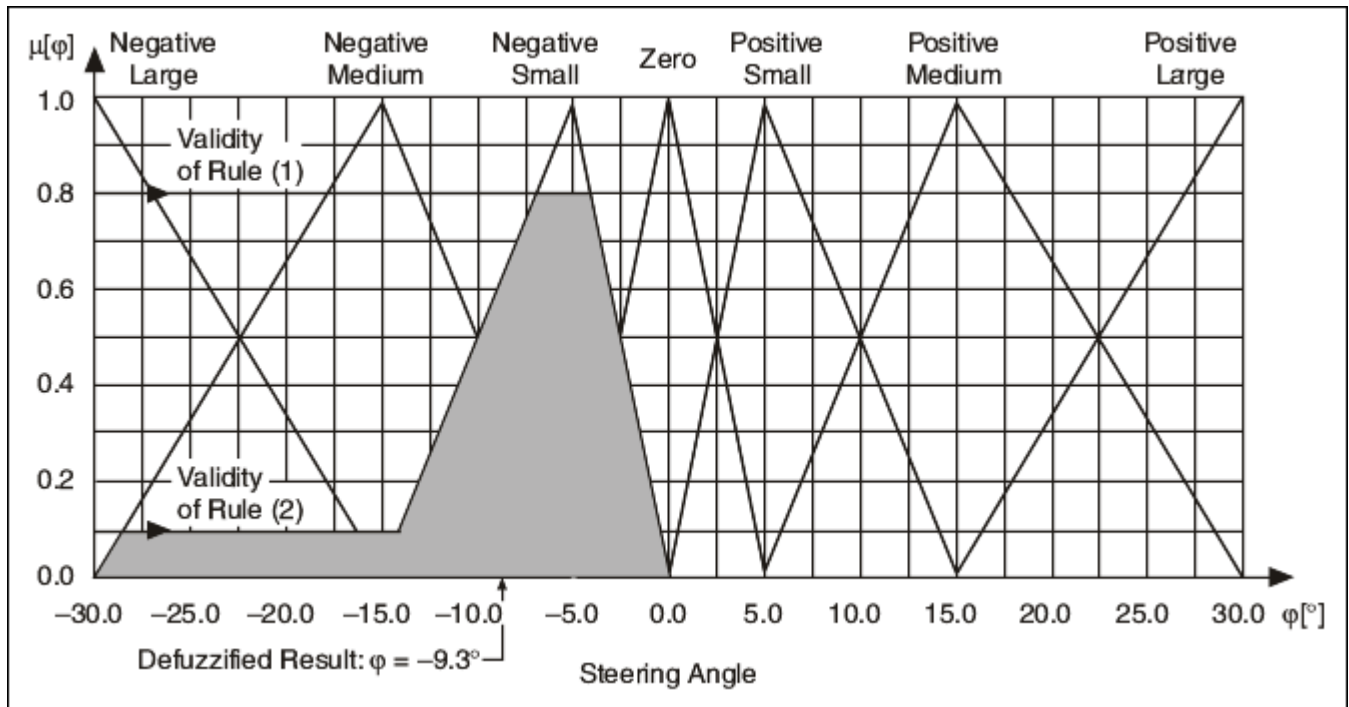
### Center of Area (CoA)

In the Center of Area (CoA) **defuzzification method**, also called the Center of Gravity (CoG) method, the fuzzy controller first calculates the area under the scaled **membership functions** and within the range of the output variable. The fuzzy logic controller then uses the following equation to calculate the geometric center of this area.

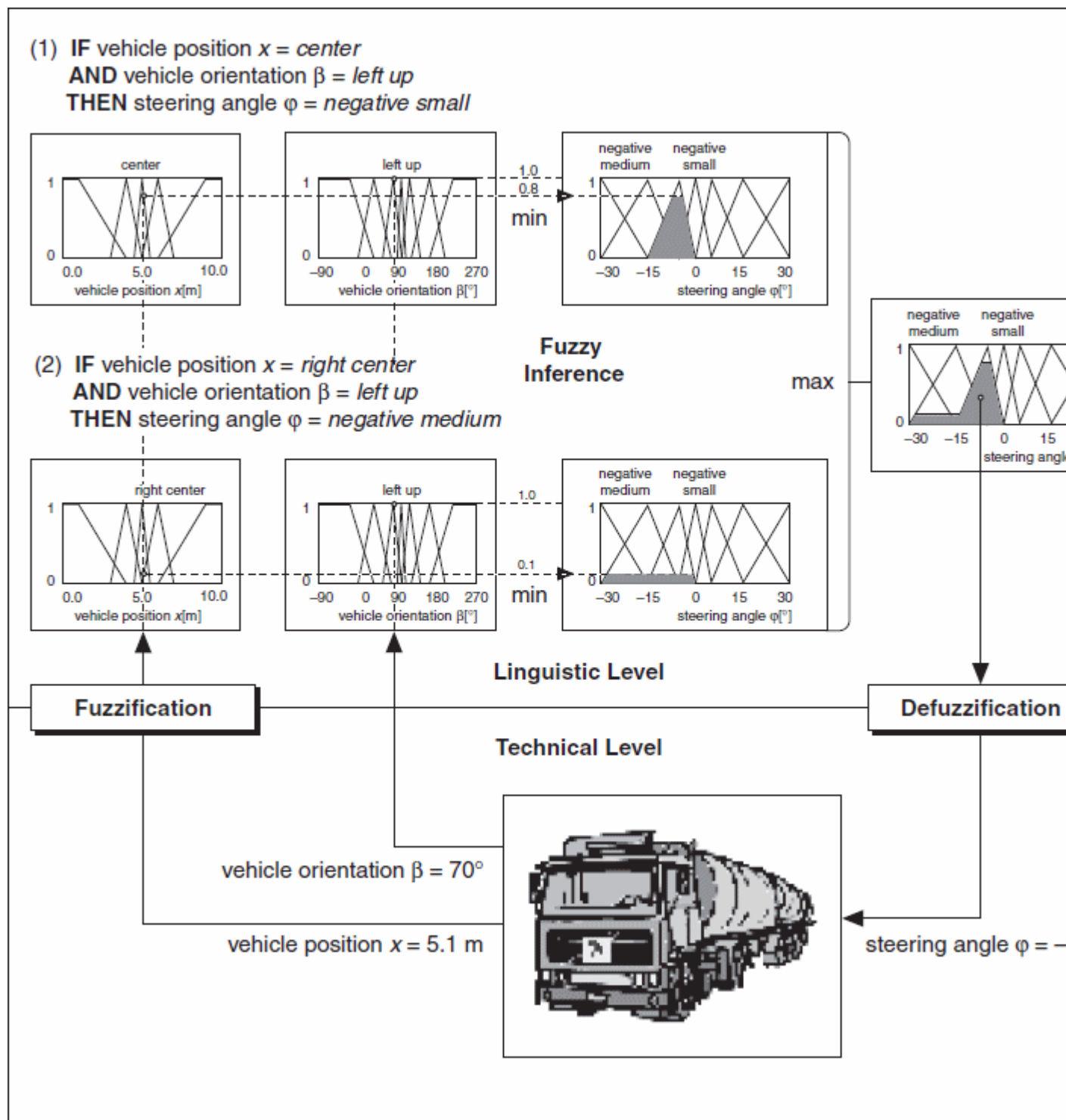
$$CoA = \frac{\int_{x_{min}}^{x_{max}} f(x) * x \, dx}{\int_{x_{min}}^{x_{max}} f(x) \, dx}$$

where CoA is the center of area, x is the value of the linguistic variable, and  $x_{min}$  and  $x_{max}$  represent the range of the **linguistic variable**. The Center of Area defuzzification method effectively calculates the best compromise between multiple output linguistic terms.

The following image shows the Center of Area defuzzification method for the Steering Angle  $\phi$  output linguistic variable of a vehicle maneuvering fuzzy system, assuming the **Minimum** implication method. The shaded portion of the graph represents the area under the scaled membership functions.



The following image summarizes the process of a fuzzy controller for the vehicle maneuvering example using the Center of Area defuzzification method.



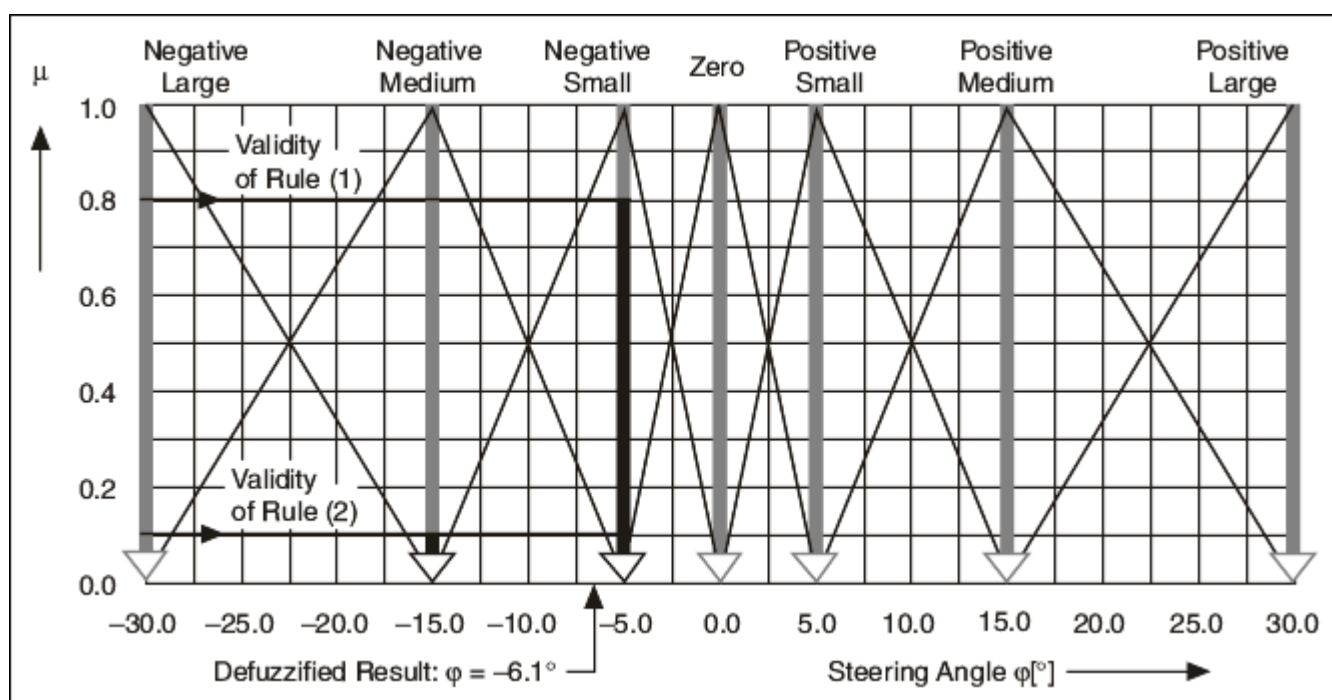
## Center of Maximum (CoM)

In the Center of Maximum (CoM) **defuzzification method**, the fuzzy logic controller first determines the typical numerical value for each scaled **membership function**. The typical numerical value is the mean of the numerical values corresponding to the degree of membership at which the membership function was scaled.

$$x_{final} = \frac{(x_1\mu_1 + x_2\mu_2 + \dots + x_n\mu_n)}{(\mu_1 + \mu_2 + \dots + \mu_n)}$$

where  $x_n$  is the typical numerical value for the scaled membership function  $n$ , and  $\mu_n$  is the degree of membership at which the membership function  $n$  was scaled.

The following image illustrates how to use the CoM defuzzification method with the vehicle maneuvering example.



The values  $-15^\circ$  and  $-5^\circ$  are the typical values of the **linguistic terms** Negative Medium and Negative Small. The degrees of truth for these linguistic terms are 0.1 and 0.8, respectively. Therefore, the defuzzified crisp output value  $\phi_{final}$  is calculated by the following equation:

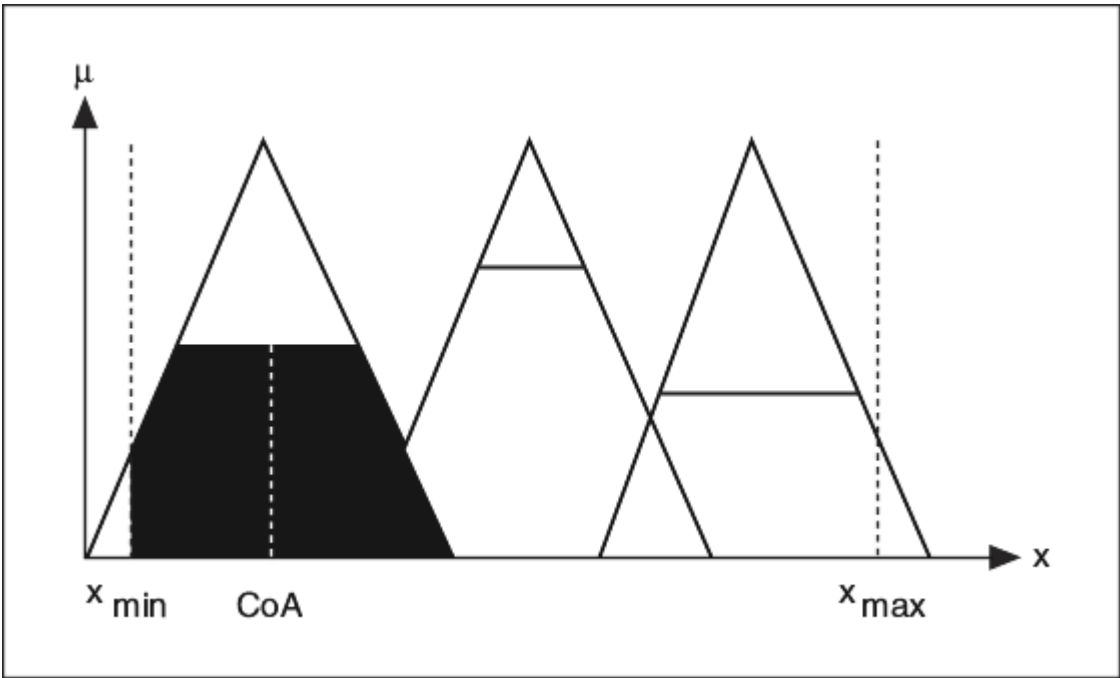
$$\phi_{final} = \frac{((-15^\circ)(0.1) + (-5^\circ)(0.8))}{(0.1 + 0.8)} = -6.1^\circ$$

The CoM defuzzification method is identical to the Center of Area (CoA) method with singleton membership functions.

The CoM and CoA defuzzification methods usually apply to closed-loop control applications of fuzzy logic. These methods usually result in continuous output signals because a small change in input values does not change the best compromise value for the output.

## Center of Sums (CoS)

In the Center of Sums (CoS) **defuzzification method**, the fuzzy logic controller first calculates the geometric center of area for each **membership function**, as shown in the following image.



The fuzzy controller then uses the following equation to calculate a weighted average of the geometric center of area for all membership functions.

$$x_{final} = \frac{(CoA_1 area_1 + CoA_2 area_2 + ... + CoA_n area_n)}{(area_1 + area_2 + ... + area_n)}$$

where  $CoA_n$  is the geometric center of area of the scaled membership function  $n$ , and  $area_n$  is the area of the scaled membership function  $n$ .

## Chien-Hrones-Reswick Autotuning Method

The Chien-Hrones-Reswick autotuning method focuses on setpoint response and disturbance response. This method provides formulas for 0% and 20% overshoot.



### 노트

You must use the PID Autotuning Design VI to implement this autotuning method. Select the Chien-Hrones-Reswick polymorphic instance and use the Tuning specifications input to configure which formula you want to use.

The following tables shows the Chien-Hrones-Reswick recommendations for each tuning formula:

| Regulator – 0% overshoot |                  |          |       |
|--------------------------|------------------|----------|-------|
| Controller               | $K_c$            | $T_i$    | $T_d$ |
| P                        | $0.3 T_P / \tau$ | —        | —     |
| PI                       | $0.6 T_P / \tau$ | $4 \tau$ | —     |



**Regulator – 0% overshoot**

|     |                   |            |             |
|-----|-------------------|------------|-------------|
| PID | $0.95 T_P / \tau$ | $2.4 \tau$ | $0.42 \tau$ |
|-----|-------------------|------------|-------------|

where  $T_P$  is the time constant and  $\tau$  is the deadtime.

**Regulator – 20% overshoot**

| Controller | $K_C$            | $T_i$      | $T_d$       |
|------------|------------------|------------|-------------|
| P          | $0.7 T_P / \tau$ | —          | —           |
| PI         | $0.7 T_P / \tau$ | $2.3 \tau$ | —           |
| PID        | $1.2 T_P / \tau$ | $2 \tau$   | $0.42 \tau$ |

where  $T_P$  is the time constant and  $\tau$  is the deadtime.

**Servo – 0% overshoot**

| Controller | $K_C$             | $T_i$      | $T_d$      |
|------------|-------------------|------------|------------|
| P          | $0.3 T_P / \tau$  | —          | —          |
| PI         | $0.35 T_P / \tau$ | $1.2 \tau$ | —          |
| PID        | $0.6 T_P / \tau$  | $\tau$     | $0.5 \tau$ |

where  $T_P$  is the time constant and  $\tau$  is the deadtime.

**Servo – 20% overshoot**

| Controller | $K_C$             | $T_i$      | $T_d$       |
|------------|-------------------|------------|-------------|
| P          | $0.7 T_P / \tau$  | —          | —           |
| PI         | $0.6 T_P / \tau$  | $\tau$     | —           |
| PID        | $0.95 T_P / \tau$ | $1.4 \tau$ | $0.47 \tau$ |

where  $T_P$  is the time constant and  $\tau$  is the deadtime.

## Cohen-Coon Autotuning Method

The Cohen-Coon autotuning method is an offline method for tuning PID controllers. This method uses PID parameters obtained from an open-loop transfer function experiment.

This method is similar to the [Ziegler-Nichols method](#), but provides better results when the controller has a large  $T_d$  relative to the time constant  $T$ .



**NOTE**  
You must use the PID Autotuning Design VI to implement this autotuning method.

The following table shows the Cohen-Coon recommendations for tuning parameters under different tuning specifications:

| Controller | $K_c$                                                             | $T_i$                                                                                                | $T_d$                                              |
|------------|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| P          | $\frac{T}{K_c * T_d} \left[ 1 + \frac{T_d}{3T} \right]$           | —                                                                                                    | —                                                  |
| PI         | $\frac{T}{K_c * T_d} \left[ 0.9 + \frac{T_d}{12T} \right]$        | $\frac{T_d \left[ 30 + 3 \left( \frac{T_d}{T} \right) \right]}{9 + 20 \left( \frac{T_d}{T} \right)}$ | —                                                  |
| PID        | $\frac{T}{K_c * T_d} \left[ \frac{4}{3} + \frac{T_d}{4T} \right]$ | $\frac{T_d \left[ 32 + 6 \left( \frac{T_d}{T} \right) \right]}{13 + 8 \left( \frac{T_d}{T} \right)}$ | $\frac{4T_d}{11 + 2 \left( \frac{T_d}{T} \right)}$ |

Where  $T$  is the time constant

$K_c$  is the proportional gain

$T_d$  is the deadtime

## Complete Rule Bases

A **rule base** with at least one active rule for each possible combination of input linguistic variables and linguistic terms is a complete rule base. If you define an incomplete rule base, you must specify a default linguistic term for each output linguistic variable so the fuzzy controller can handle situations in which no rules are active.

The Vehicle Position  $x$  input linguistic variable has five linguistic terms, and the Vehicle Orientation  $\beta$  linguistic variable has seven linguistic terms. Therefore, the rule base of the vehicle maneuvering example consists of  $N = 5 * 7 = 35$  rules. You can document the complete rule base in matrix form, as shown in the following image for the vehicle maneuvering fuzzy system.

| AND                             |            | Vehicle Position $x$ [m] |                 |                 |                 |                 |
|---------------------------------|------------|--------------------------|-----------------|-----------------|-----------------|-----------------|
|                                 |            | Left                     | Left Center     | Center          | Right Center    | Right           |
| Vehicle Orientation $\beta$ [°] | Left Down  | Negative Small           | Negative Medium | Negative Medium | Negative Large  | Negative Large  |
|                                 | Left       | Positive Small           | Negative Small  | Negative Medium | Negative Large  | Negative Large  |
|                                 | Left Up    | Positive Medium          | Positive Small  | Negative Small  | Negative Medium | Negative Large  |
|                                 | Up         | Positive Medium          | Positive Medium | Zero            | Negative Medium | Negative Medium |
|                                 | Right Up   | Positive Large           | Positive Medium | Positive Small  | Negative Small  | Negative Medium |
|                                 | Right      | Positive Large           | Positive Large  | Positive Medium | Positive Small  | Negative Small  |
|                                 | Right Down | Positive Large           | Positive Large  | Positive Medium | Positive Medium | Negative Small  |

In the previous image, each column or row represents an antecedent of a rule. The term at the intersection of a column and a row is the consequent of the rule corresponding to the aggregated rule antecedent. For example, the following rule is highlighted:

---

IF Vehicle Position  $x$  is Left Center AND Vehicle Orientation  $\beta$  is Left, THEN Steering Angle  $\phi$  is Negative Small.

---

### Cascading Fuzzy Systems

Plotting a rule base in matrix form, as in the previous image, is helpful for detecting inconsistencies, such as contradictory rules. However, plotting a rule base in matrix form is efficient only for small rule bases. Detecting inconsistencies in large rule bases is difficult. For fuzzy systems with numerous controller inputs, you can use cascading fuzzy systems to avoid large rule bases. In cascading fuzzy systems, the outputs of the first fuzzy system serve as the inputs to the next fuzzy system, and so on.

## Control Output Rate Limiting

Sudden changes in control output are often undesirable or even dangerous for many control applications. For example, a sudden large change in setpoint can cause a very large change in controller output. Although in theory this large change in controller output results in fast response of the system, it also can cause unnecessary wear on actuators or sudden large power demands. In addition, the PID controller can amplify noise in the system and result in a constantly changing controller output.

You can use the PID Output Rate Limiter VI to avoid the problem of sudden changes in controller output. Wire the output value of the PID VI to the input (controller output) input of the PID Output Rate Limiter VI to limit the slew, or rate of change, of the output to the value of the output rate (EGU/min).

Assign a value to initial output to specify the output value on the first call to the VI. Use the dt (s) input to specify the same control-loop cycle time you use for the PID VI.

## Converting Between Percentage of Full Scale and Engineering Units

The default setpoint, process variable, and output ranges for the PID VIs correspond to percentage of full scale. In other words, proportional gain  $K_c$  relates percentage of full scale output to percentage of full scale input. This is the default behavior of many PID controllers used for process control applications. To implement PID in this way, you must scale all inputs to percentage of full scale and all controller outputs to actual engineering units, for example, volts for analog output.

You can use the PID EGU to Percentage VI to convert any input from real engineering units to percentage of full scale, and you can use the PID Percentage to EGU VI to convert the controller output from percentage to real engineering units.



### 노트

The PID VIs do not use the setpoint range and output range information to convert values to percentages in the PID algorithm. The controller gain relates the output in engineering units to the input in engineering units. For example, a gain value of 1 produces an output of 10 for a difference between setpoint and process variable of 10, regardless of the output range and setpoint range.

## Creating a Rule Base

After **create linguistic variables**, **defining linguistic terms**, and **creating membership functions**, the last step of **fuzzy system design** is creating a rule base. Rules describe, in words, the relationships between input and output linguistic variables based on their linguistic terms. A rule base is the set of rules for a fuzzy system.

To create a rule, you must specify the antecedents, or IF portions, and consequents, or THEN portions, of the rule. For example, consider the following rule for a vehicle maneuvering fuzzy system:

---

IF Vehicle Position  $x$  is Left Center AND Vehicle Orientation  $\beta$  is Left Up, THEN Steering Angle  $\phi$  is Positive Small.

---

The clauses " Vehicle Position  $x$  is Left Center " and " Vehicle Orientation  $\beta$  is Left Up " are the antecedents of this rule. The clause " Steering Angle  $\phi$  is Positive Small " is the consequent of this rule.

Associate an input linguistic variable with a corresponding linguistic term to form an antecedent. Associate an output linguistic variable with a corresponding linguistic term to form a consequent. The consequent of a rule represents the action you want the fuzzy controller to take if the linguistic terms of the input linguistic variables in the rule are met.

When constructing a rule base, avoid contradictory rules, or rules with the same IF portion but different THEN portions. A consistent rule base is a rule base that has no contradictory rules.



#### 노트

The rule base itself is the biggest influence on the [I/O characteristics](#) of a fuzzy controller. The rule base determines the principal functionality of the controller. Refer to the [Using the Mean of Maximum \(MoM\) Defuzzification Method](#) example to see how changing the rule base can affect the output characteristic of a fuzzy controller.

### Total Possible Rules

The total number  $N$  of possible rules for a fuzzy system is defined by the following equation:

$$N = p_1 * p_2 * \dots * p_n$$

where  $p_n$  is the number of linguistic terms for the input linguistic variable  $n$ .

If each input linguistic variable has the same number of linguistic terms, the total number  $N$  of possible rules is defined by the following equation:

$$N = p^m$$

where  $p$  is the number of linguistic terms for each input linguistic variable and  $m$  is the number of input linguistic variables. For example, for three input linguistic variables with five linguistic terms each, the total number of possible rules is  $N = 5^3 = 125$ .

### Additional Information About Rule Bases

The following topics contain additional important information about creating rule bases:

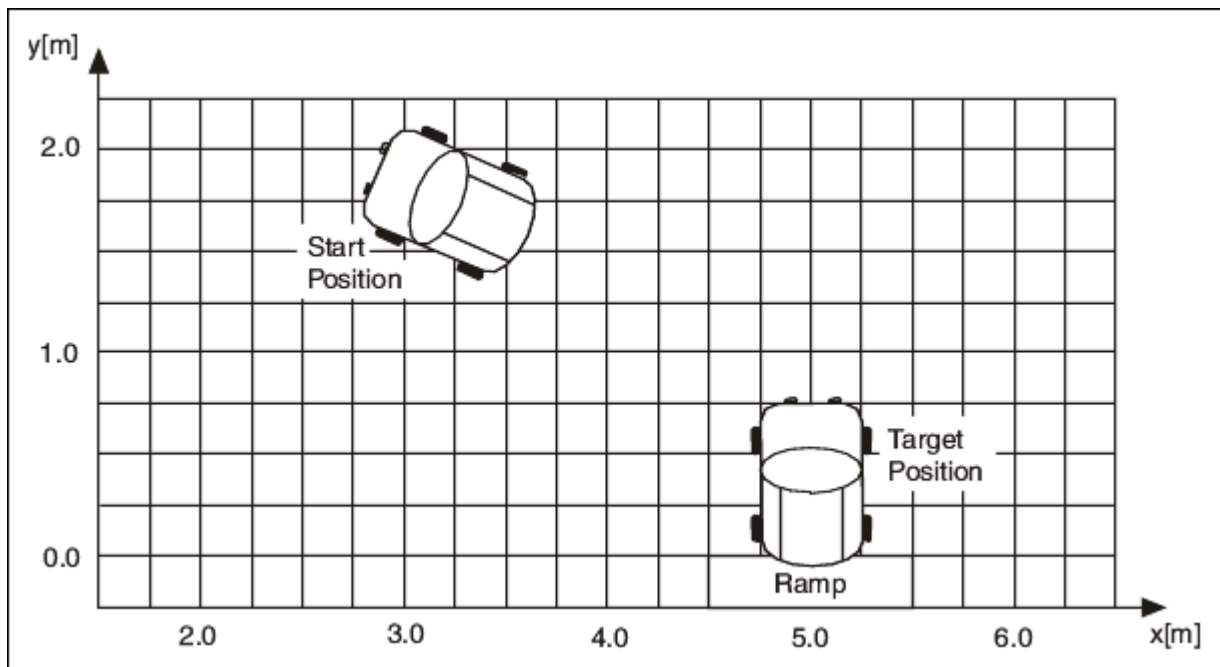
- [Complete Rule Bases](#)
- [Specifying an Antecedent Connective](#)
- [Specifying a Consequent Implication Method](#)

## Creating Linguistic Variables

When you **design a fuzzy system**, linguistic variables represent, in words, the input variables and output variables of the system you want to control.

When you create a linguistic variable to represent an input or output variable, decide how many linguistic terms, or categories of values of the linguistic variable, you want to create. Linguistic variables usually have an odd number of linguistic terms, with a middle linguistic term and symmetric linguistic terms at each extreme. In most applications, three to seven linguistic terms are sufficient for categorizing the values of a linguistic variable.

Consider an example in which you want to automate a vehicle to park itself from an arbitrary starting position. A driver can control the vehicle by constantly evaluating the current status of the vehicle, such as the distance from the target position and the orientation of the vehicle, to derive the correct steering angle. The following image illustrates this example.



You can define two input linguistic variables for this example. Vehicle Position  $x$  represents the vehicle position in relation to the destination. Vehicle Orientation  $\beta$  represents the orientation of the vehicle. You also can define an output linguistic variable, Steering Angle  $\phi$ , to represent the steering angle of the vehicle that you want to control.

### Defining Linguistic Terms

You can define linguistic terms of Left, Left Center, Center, Right Center, and Right for the Vehicle Position  $x$  input linguistic variable to describe the possible positions of the vehicle in relation to the destination. You can define linguistic terms of Left Down, Left, Left Up, Up, Right Up, Right, and Right Down for the Vehicle Orientation  $\beta$  input linguistic variable to describe the possible orientations of the vehicle. The linguistic terms of the Steering Angle  $\phi$  output linguistic variable must represent both the direction and magnitude that the steering angle changes. Therefore, you can use the linguistic terms Negative Large,

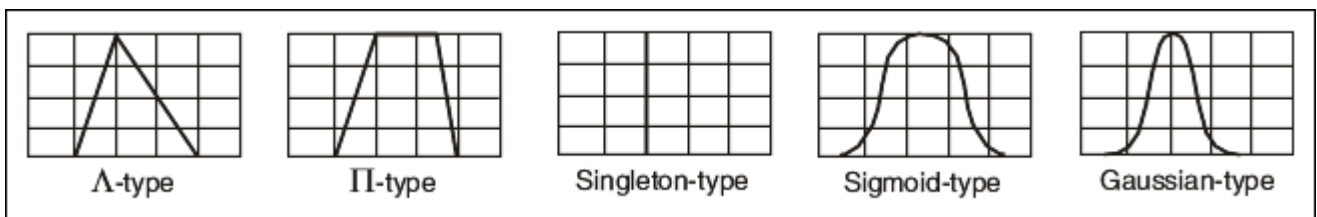
Negative Medium, Negative Small, Zero, Positive Small, Positive Medium, and Positive Large for this output linguistic variable.

The next step in designing a fuzzy system is **creating membership** functions that correspond to the linguistic terms you define.

## Creating Membership Functions

When you **design a fuzzy system**, you first **create linguistic variables** for which you **define linguistic terms**. After you define the linguistic terms, you must apply membership functions to those terms. Membership functions are simply numerical functions that correspond to linguistic terms. A membership function represents the degree of membership of linguistic variables within their linguistic terms.

You can apply the normalized standard membership functions shown in the following image to most technical processes. These standard functions include  $\Lambda$ -type (triangular shape),  $\Pi$ -type (trapezoidal shape), singleton-type (vertical line shape), Sigmoid-type (wave shape), and Gaussian-type (bell shape) membership functions.

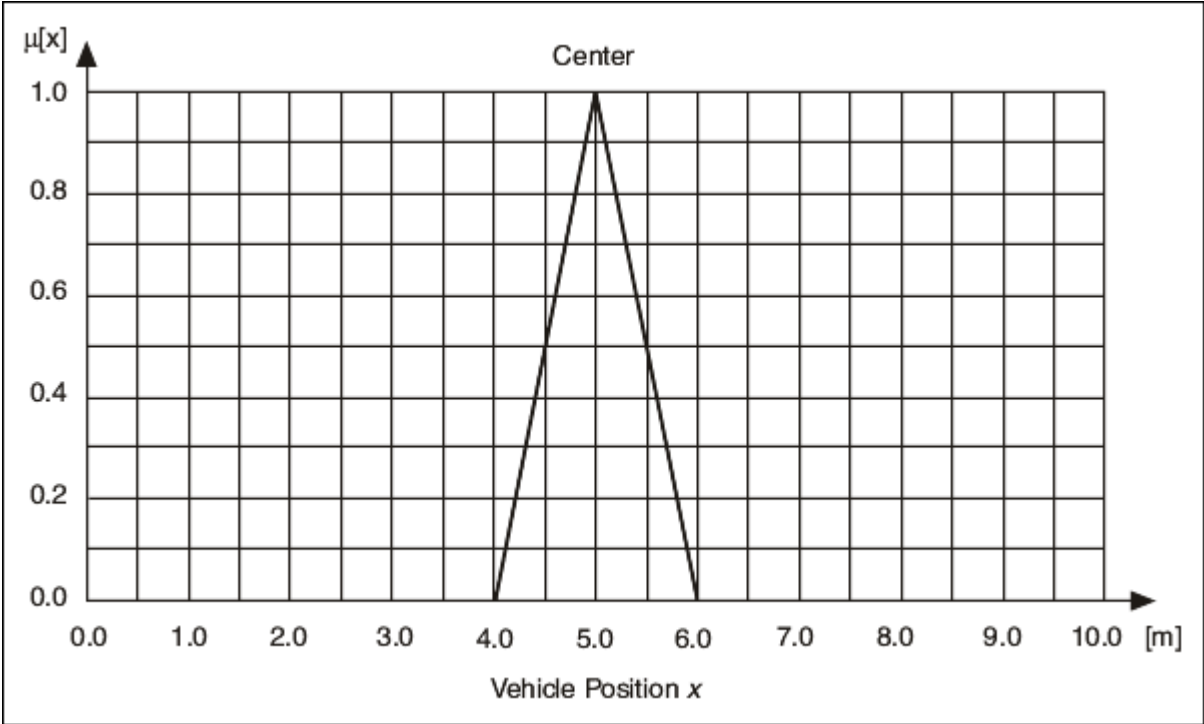


Example Triangle Membership Function for the Linguistic Term

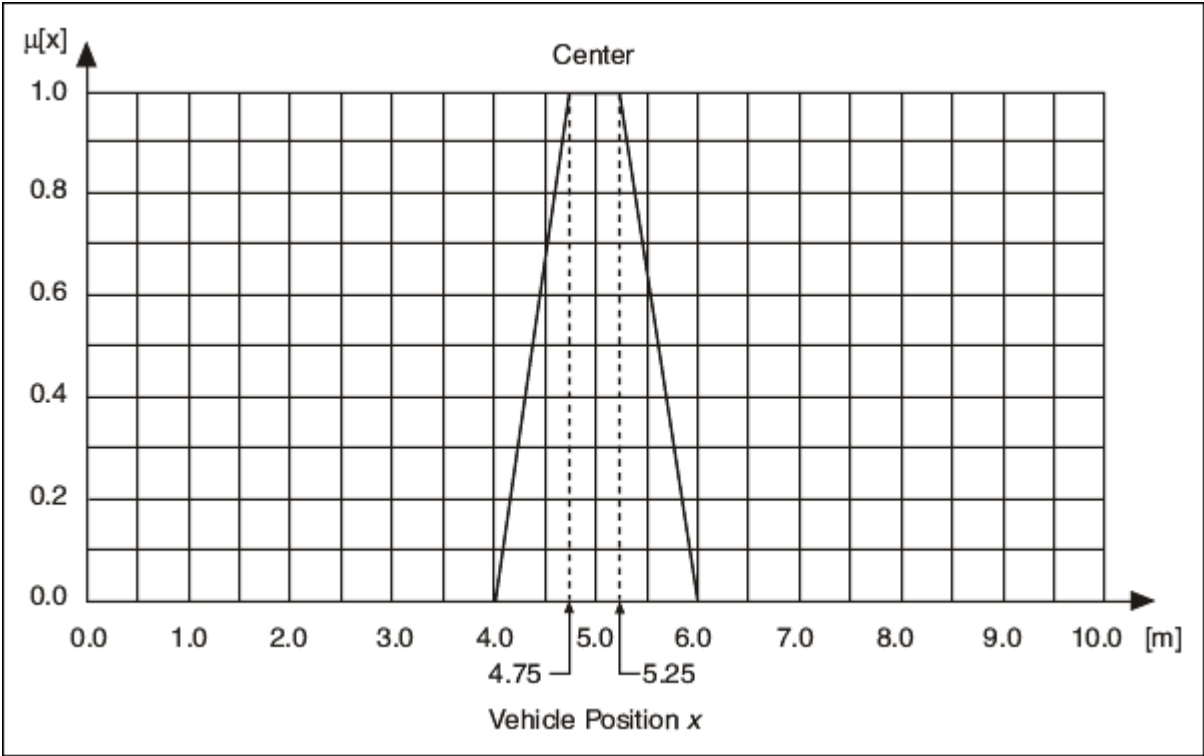
Example Trapezoidal Membership Function for the Linguistic Term Center

Membership Functions for a Vehicle Maneuvering Fuzzy System

For example, the linguistic variable Vehicle Position  $x$  might have full membership (1) within the linguistic term Center at 5 meters, no membership (0) within that term at 4 meters or less and 6 meters or greater, and partial membership at all distances between 4 and 6 meters. If you plot the degree of membership against the value of Vehicle Position  $x$ , you can see that the resulting membership function is a triangle function.

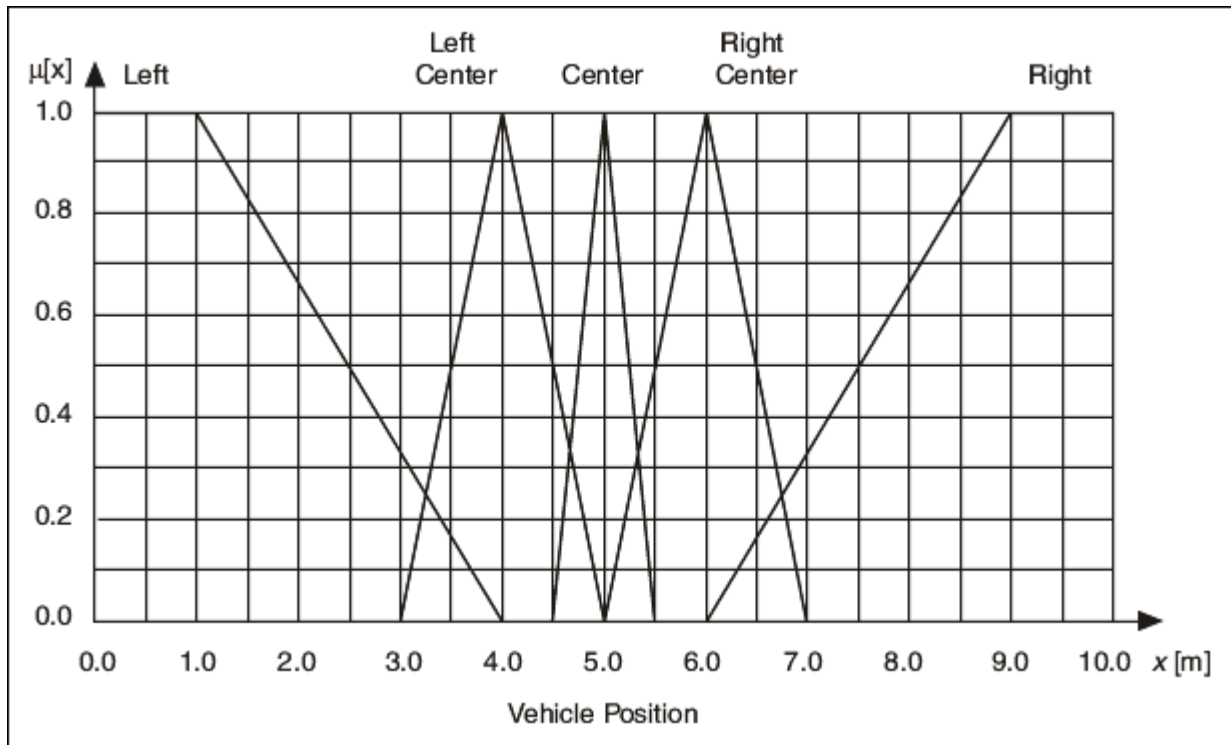


Sometimes a linguistic variable has full membership within a linguistic term at a range of values rather than at a point value. If, for example, the linguistic variable Vehicle Position  $x$  has full membership within the linguistic term Center at values  $x = 5 \pm 0.25$  m, a trapezoidal membership function applies, as shown in the following image.

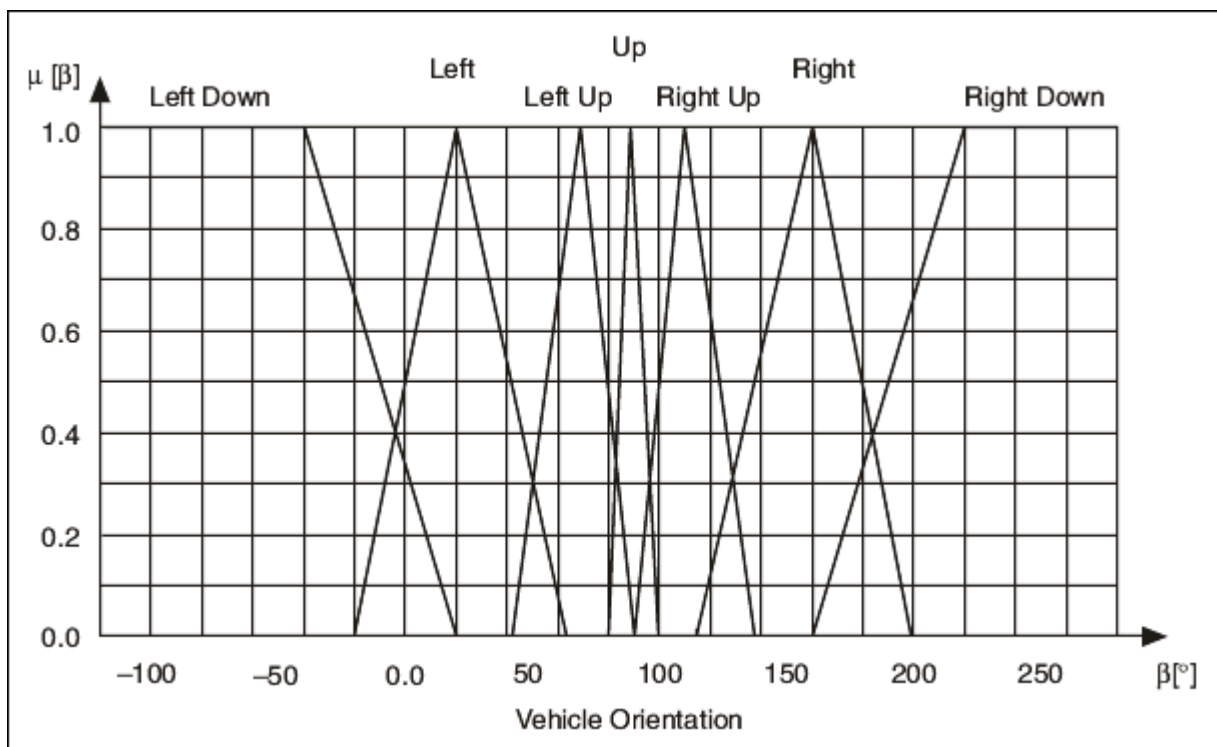


The following image shows the membership functions for Vehicle Position  $x$ .

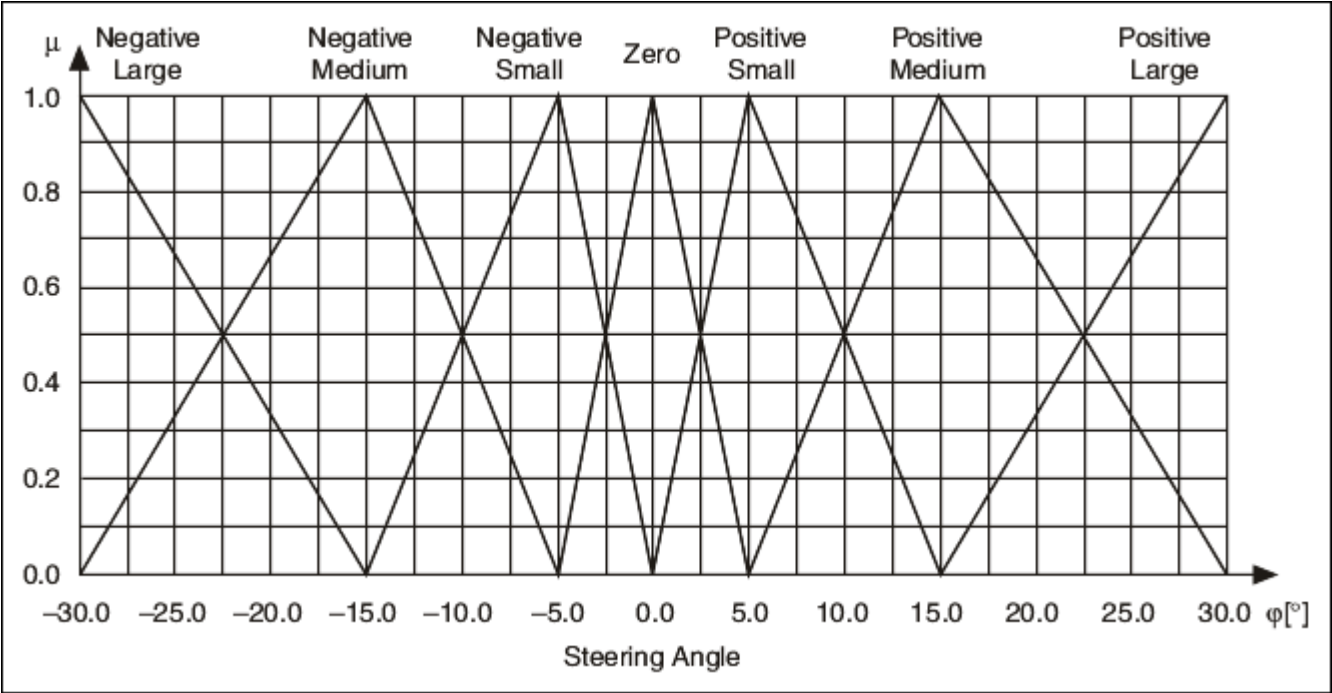




The following image shows the membership functions for Vehicle Orientation  $\beta$ .



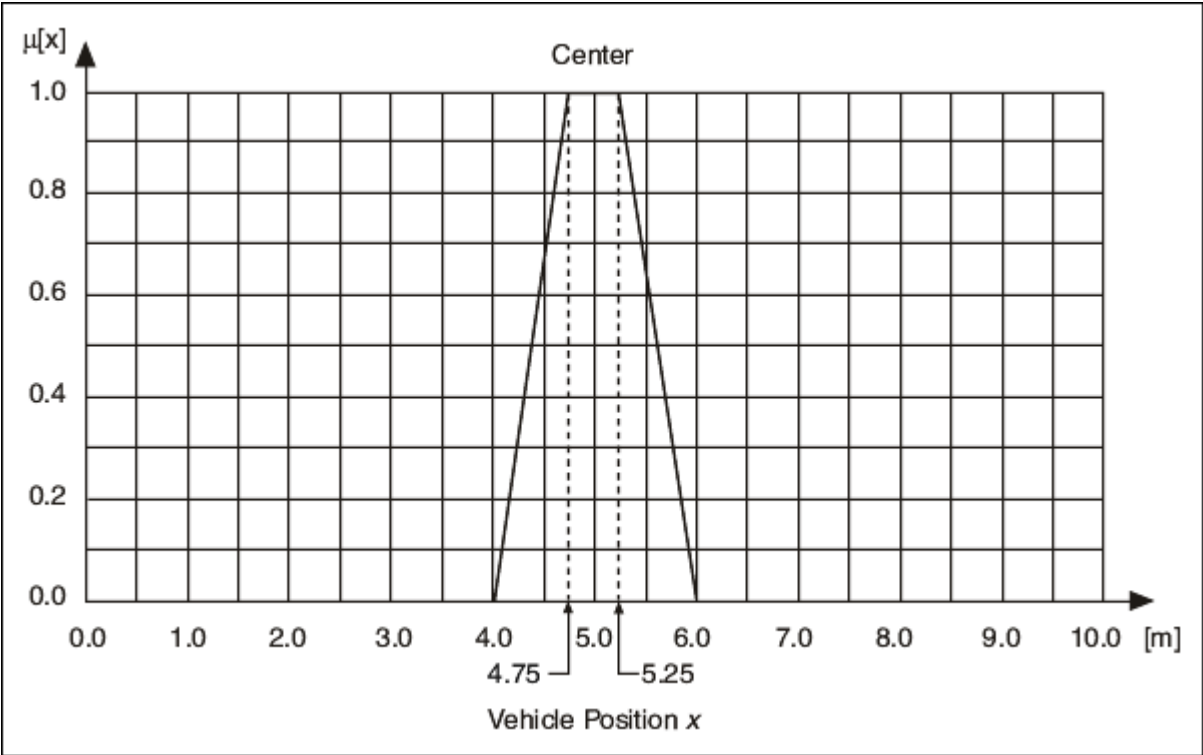
The following image shows the membership functions for Steering Angle  $\phi$ .



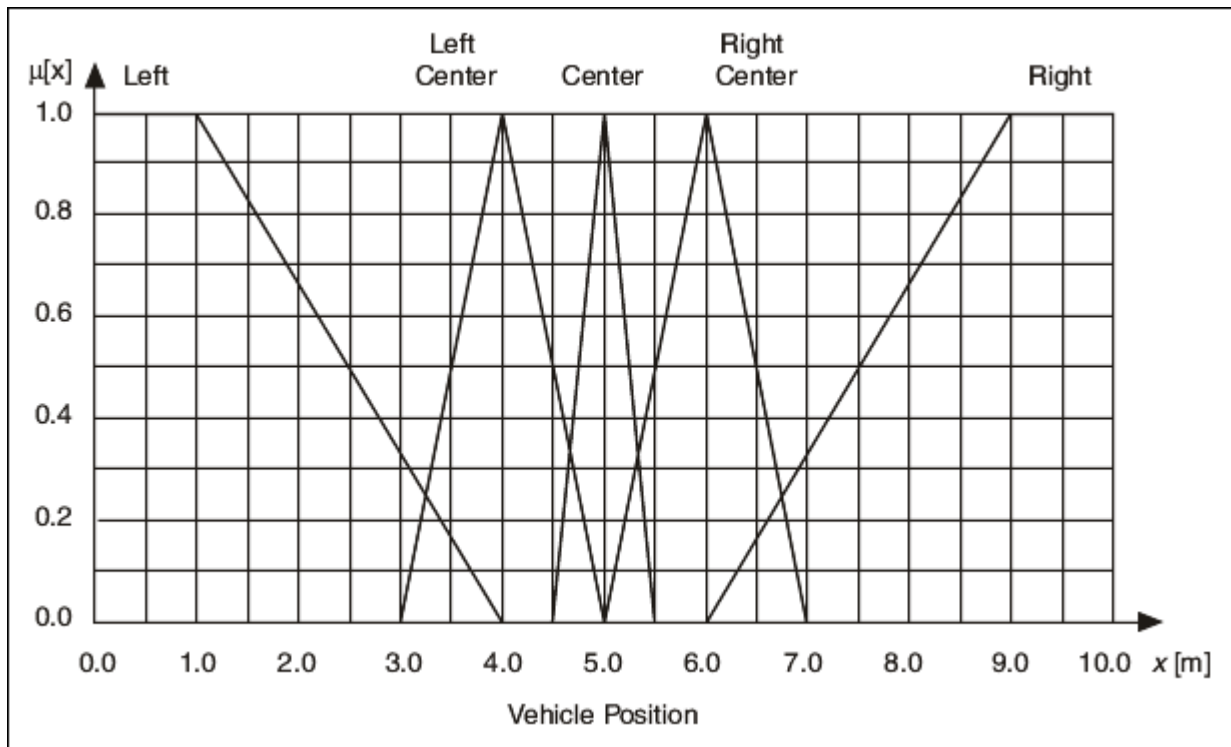
Example Trapezoidal Membership Function for the Linguistic Term

Membership Functions for a Vehicle Maneuvering Fuzzy System

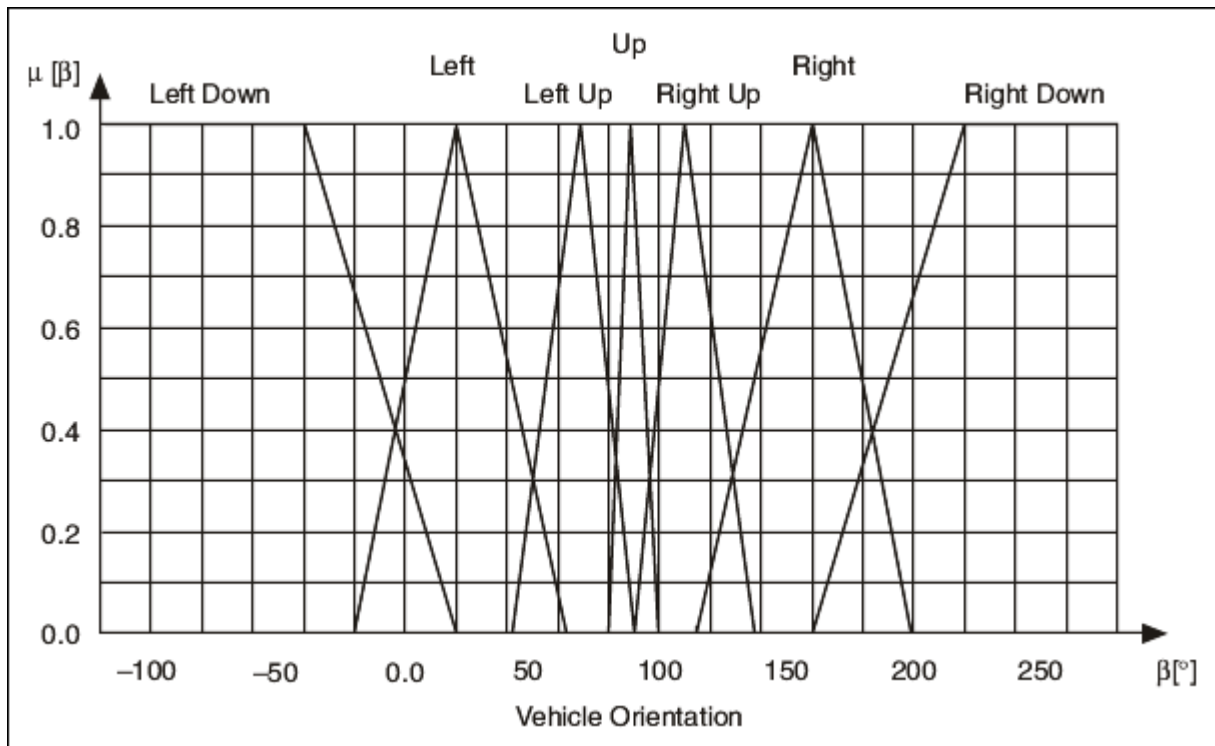
Sometimes a linguistic variable has full membership within a linguistic term at a range of values rather than at a point value. If, for example, the linguistic variable Vehicle Position  $x$  has full membership within the linguistic term Center at values  $x = 5 \pm 0.25$  m, a trapezoidal membership function applies, as shown in the following image.



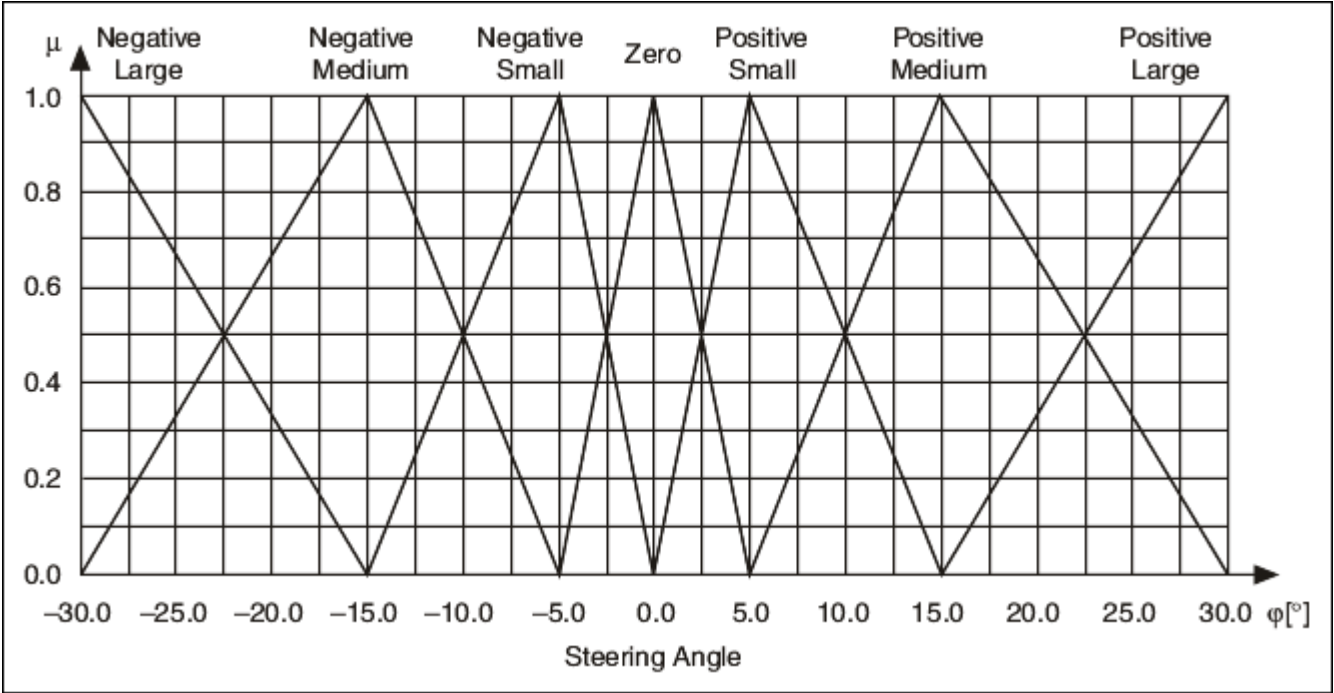
The following image shows the membership functions for Vehicle Position  $x$ .



The following image shows the membership functions for Vehicle Orientation  $\beta$ .

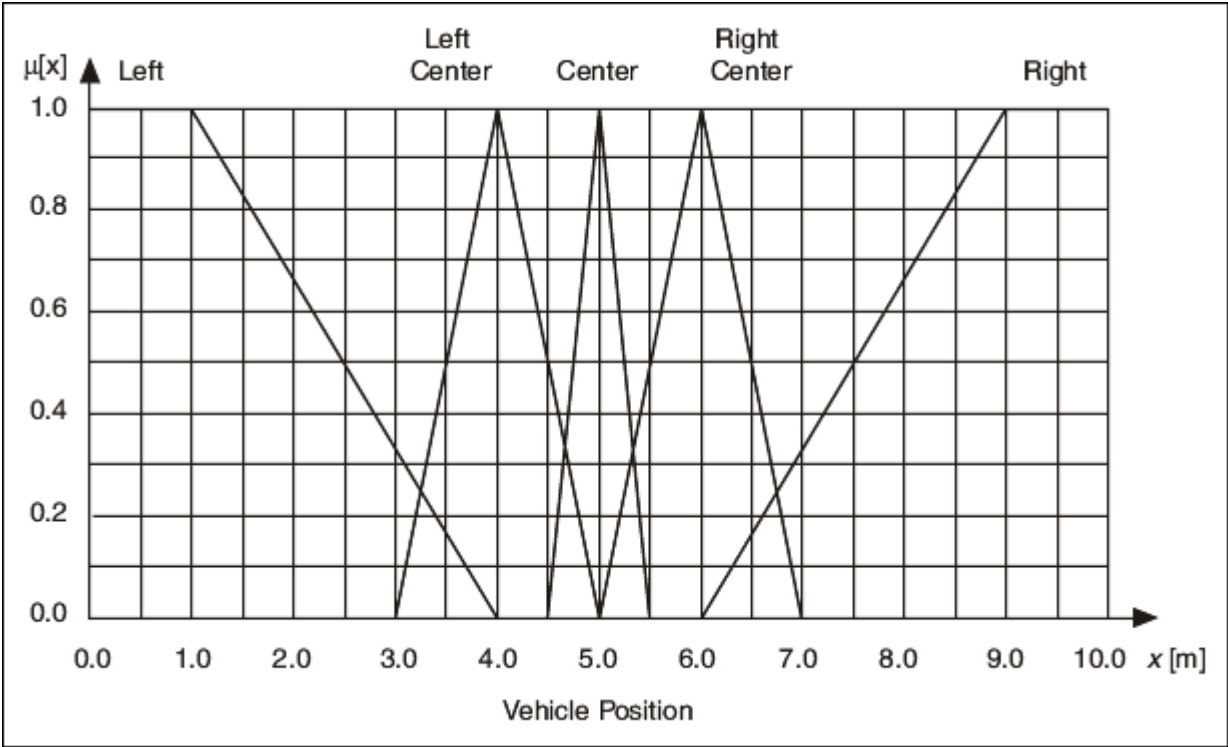


The following image shows the membership functions for Steering Angle  $\phi$ .

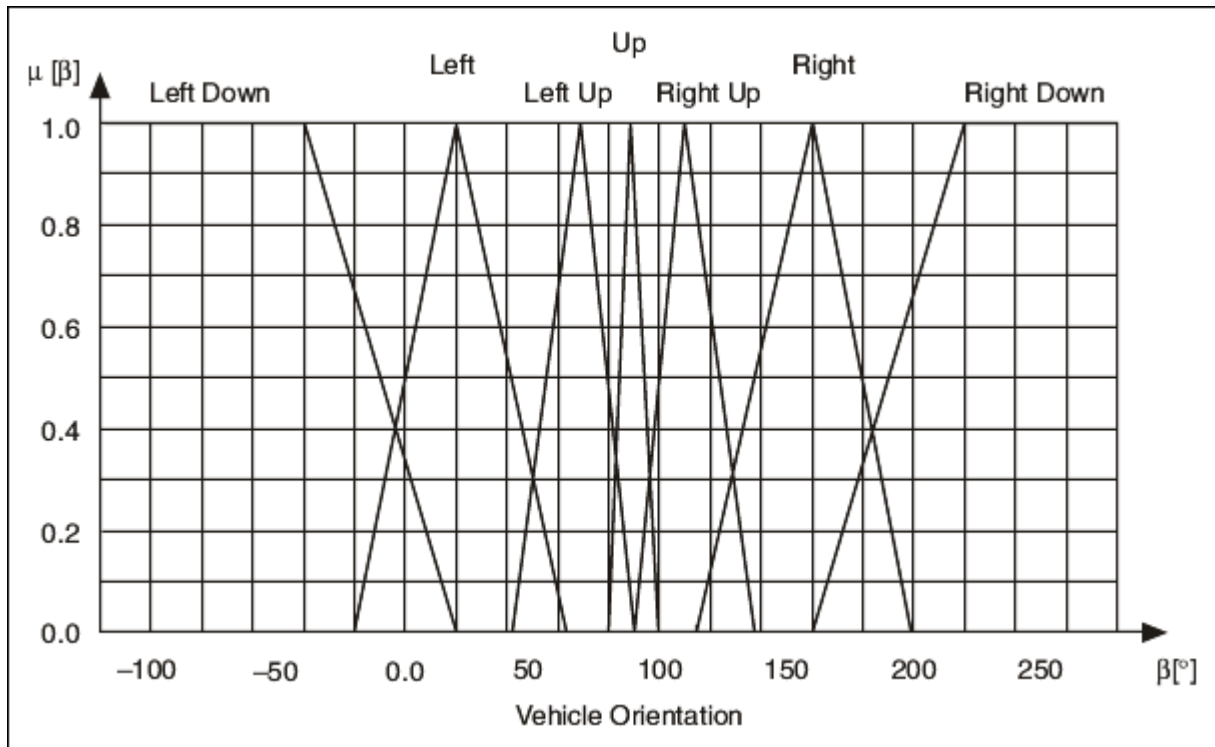


Membership Functions for a Vehicle Maneuvering Fuzzy System

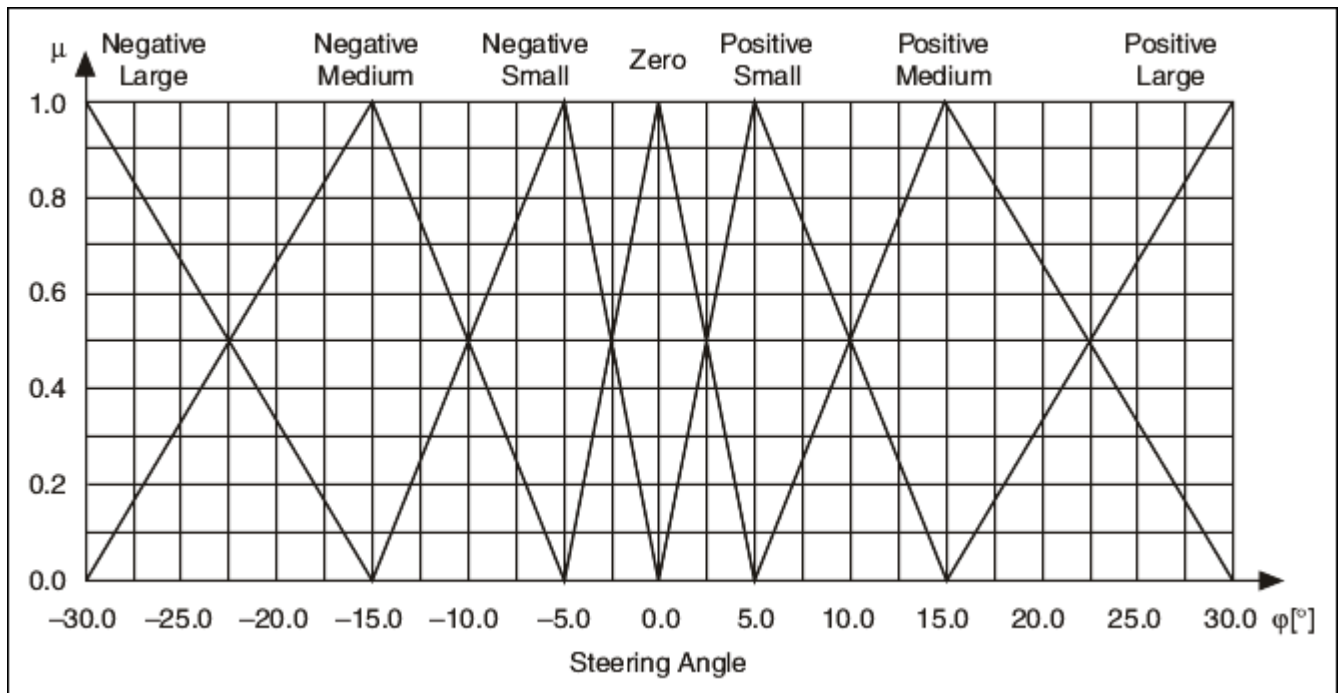
The following image shows the membership functions for Vehicle Position  $x$ .



The following image shows the membership functions for Vehicle Orientation  $\beta$ .



The following image shows the membership functions for Steering Angle  $\phi$ .



## Defuzzification Methods

Defuzzification is the process of converting the degrees of membership of output linguistic variables within their linguistic terms into crisp numerical values. Consider the following rules:

|                                               |                                                                      |               |                                                                           |
|-----------------------------------------------|----------------------------------------------------------------------|---------------|---------------------------------------------------------------------------|
| (1)                                           | IF Vehicle Position $x$ is Center (degree of membership = 0.8)       | AND (Minimum) | Vehicle Orientation $\beta$ is Left Up (degree of membership = 1.0) = 0.8 |
| THEN Steering Angle $\phi$ is Negative Small  |                                                                      |               |                                                                           |
| (2)                                           | IF Vehicle Position $x$ is Right Center (degree of membership = 0.1) | AND (Minimum) | Vehicle Orientation $\beta$ is Left Up (degree of membership = 1.0) = 0.1 |
| THEN Steering Angle $\phi$ is Negative Medium |                                                                      |               |                                                                           |

These two rules specify two non-zero values for the Steering Angle  $\phi$  output linguistic variable:

|                                |                               |         |
|--------------------------------|-------------------------------|---------|
| Negative Medium Negative Small | to a degree of to a degree of | 0.1 0.8 |
|--------------------------------|-------------------------------|---------|



**노트**

Fuzzy controllers use an implication method to scale the membership functions of output linguistic variables before performing defuzzification.

A fuzzy controller can use one of several mathematical methods to perform defuzzification: Center of Area (CoA), modified Center of Area (mCoA), Center of Sums (CoS), Center of Maximum (CoM), or Mean of Maximum (MoM). Selecting a defuzzification method depends on the context of the design you want to calculate with the fuzzy controller.

Related Information

Center of Area (CoA)

Modified Center of Area (mCoA)

Center of Maximum (CoM)

Mean of Maximum (MoM)

Center of Sums (CoS)

## Selecting a Defuzzification Method



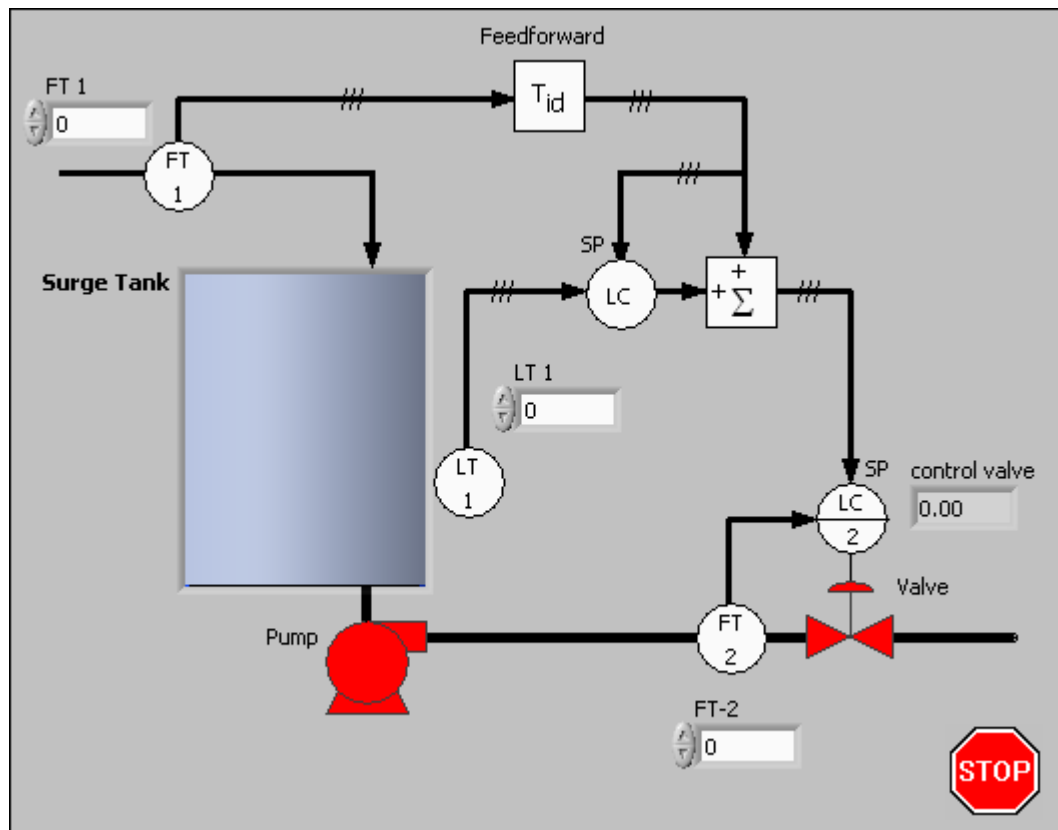
### 관련 정보

- [Creating Linguistic Variables](#) ( 1738 페이지 참조)
- [Fuzzy Controllers](#) ( 1683 페이지 참조)
- [Specifying a Consequent Implication Method](#) ( 1701 페이지 참조)
- [Creating Membership Functions](#) ( 1739 페이지 참조)
- [Center of Area \(CoA\)](#) ( 1728 페이지 참조)
- [Modified Center of Area \(mCoA\)](#) ( 1752 페이지 참조)
- [Center of Sums \(CoS\)](#) ( 1731 페이지 참조)
- [Center of Maximum \(CoM\)](#) ( 1730 페이지 참조)
- [Mean of Maximum \(MoM\)](#) ( 1751 페이지 참조)
- [Selecting a Defuzzification Method](#) ( 1753 페이지 참조)
- [Center of Area \(CoA\)](#) ( 1728 페이지 참조)
- [Modified Center of Area \(mCoA\)](#) ( 1752 페이지 참조)
- [Center of Maximum \(CoM\)](#) ( 1730 페이지 참조)
- [Mean of Maximum \(MoM\)](#) ( 1751 페이지 참조)
- [Center of Sums \(CoS\)](#) ( 1731 페이지 참조)
- [Selecting a Defuzzification Method](#) ( 1753 페이지 참조)

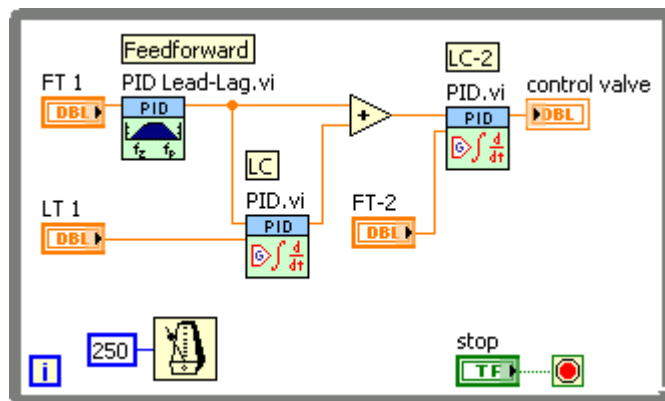
## Designing a Control Strategy

You can design a control strategy by sketching a flowchart that includes the physical process and control elements such as valves and measurements. Also include feedback from the process and any required computations. You then can use the PID VIs, combined with the math and logic VIs and functions in LabVIEW, to translate the flowchart into a LabVIEW block diagram.

The following image shows a sample control flowchart:



The following image shows the equivalent LabVIEW block diagram:



In the previous images, FT is the flow transmitter, LT is the level transmitter, LC is the level of control, and SP is the setpoint. The only elements missing from this simplified VI are the loop-tuning parameters and the automatic-to-manual switching.

You can handle the inputs and outputs through DAQ devices, FieldPoint I/O modules, GPIB instruments, or serial I/O ports. You can adjust polling rates in real time. Potential polling rates are limited only by your hardware and by the number and graphical complexity of your VIs.



## Setting Timing

The PID VI and the PID Lead-Lag VI are time-dependent. A VI can acquire timing information either from a value you supply to the cycle time control,  $dt$ , or from a time keeper such as those built into the PID VIs. If  $dt$  is less than or equal to zero, the VI calculates new timing information each time LabVIEW calls it. At each call, the VI measures the time since the last call and uses that difference in its calculations. If you call a VI from a While Loop that uses one of the LabVIEW Timing VIs and functions, you can achieve fairly regular timing, and the internal time keeper compensates for variations. However, the resolution of the Tick Count (ms) function is limited to 1 ms.

If  $dt$  is a positive value in seconds, the VI uses that value in the calculations, regardless of the elapsed time. National Instruments recommends you use this method for fast loops, such as when you use acquisition hardware to time the controller input or real-time applications. Refer to the example VIs located in the `labview\examples\control\PID` directory for examples of using timing with the PID VIs.



Find related examples

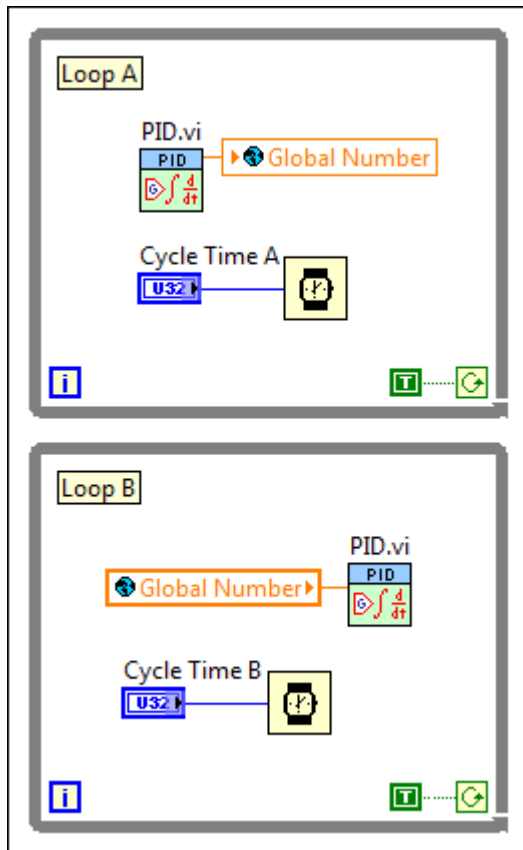


### 노트

If you installed NI-DAQmx, you also can view relevant examples in the `labview\examples\DAQmx\Control\Control.11b` directory.

According to control theory, a control system must sample a physical process at a rate about 10 times faster than the fastest time constant in the physical process. For example, a time constant of 60 seconds is typical for a temperature control loop in a small system. In this case, a cycle time of about 6 seconds is sufficient. Faster cycling does not improve performance. In fact, running all your control VIs too fast can degrade the response time of your LabVIEW application.

All VIs within a loop execute once per iteration at the same cycle time. To run several control VIs at different cycle times and still share data between them, as for example in a cascade, you must separate the VIs into independently timed While Loops. The following block diagram shows a cascade with two independently timed While Loops:

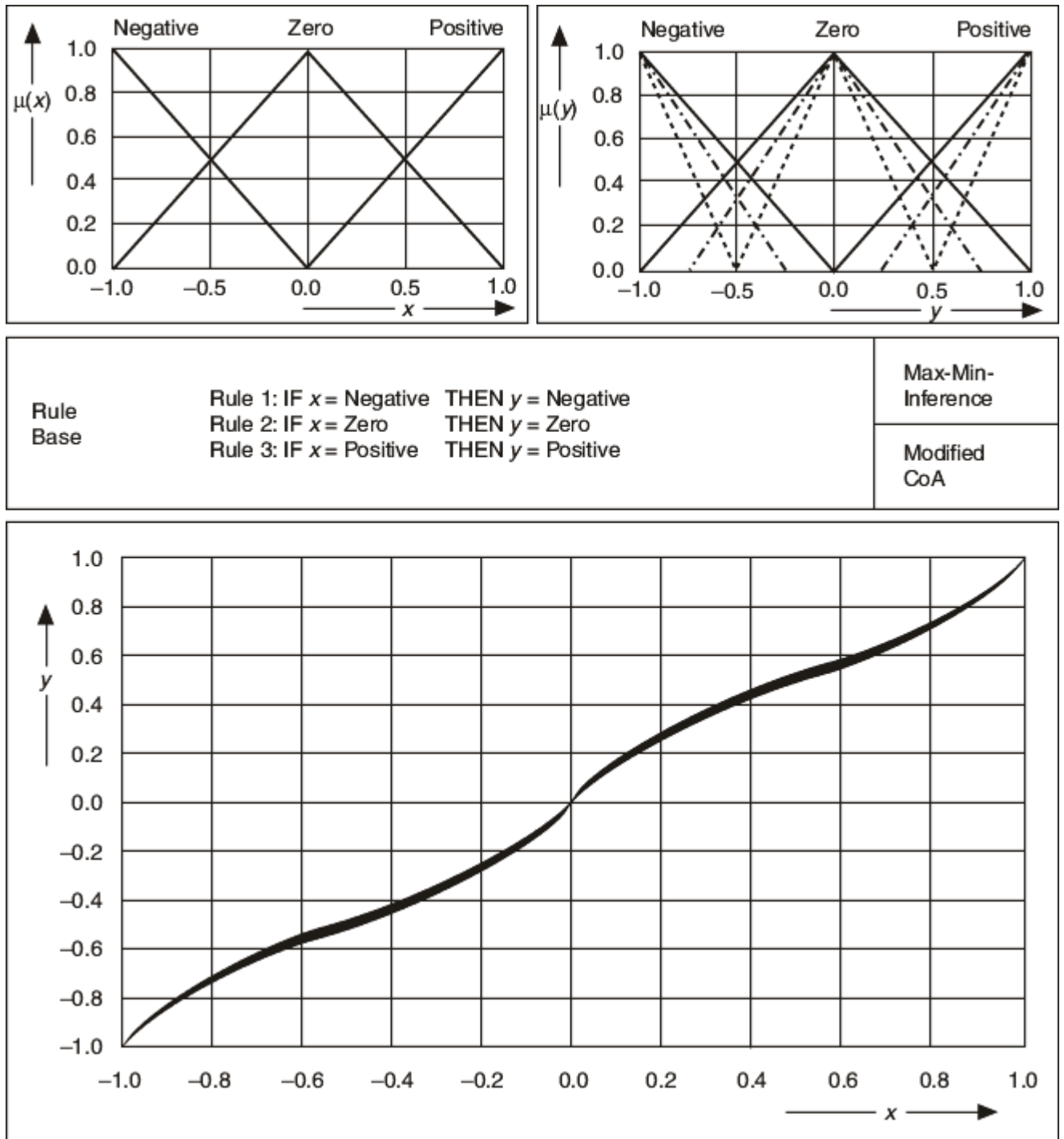


A global variable passes the output of the PID VI in Loop A to the process variable input of the PID VI in Loop B. You can place both While Loops on the same block diagram or in separate VIs. Use additional global or local variables to pass any other necessary data between the two While Loops.

If the front panel of your VI does not contain graphics that LabVIEW must update frequently, the PID VIs can execute at kilohertz (kHz) rates. However, actions such as mouse activity and window scrolling interfere with these rates.

## Different Overlapping Degrees of Membership Functions for Output Terms

The following image shows that if all the consequent terms are equal in width, the overlapping degree of the membership functions for the consequent terms has no significant influence on the controller characteristic.



#### 관련 정보

[Creating Membership Functions](#) ( 1739 페이지 참조)

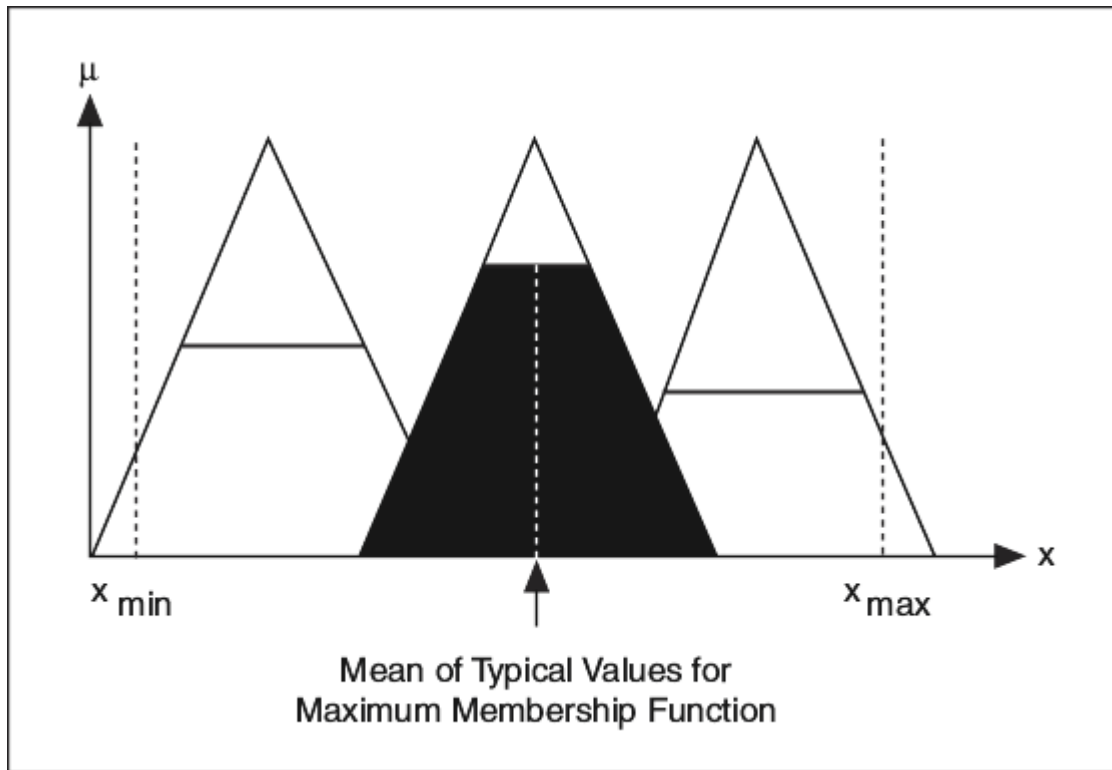
[I/O Characteristics of Fuzzy Controllers](#) ( 1690 페이지 참조)

## Mean of Maximum (MoM)

Use the Mean of Maximum (MoM) [defuzzification method](#) for pattern recognition applications. This defuzzification method calculates the most plausible result. Rather than

averaging the degrees of membership of the output **linguistic terms**, the MoM defuzzification method selects the typical value of the most valid output linguistic term.

The following image illustrates the MoM defuzzification method.



## Modified Center of Area (mCoA)

Because the **Center of Area (CoA)** defuzzification method evaluates the area under the scaled **membership functions** only within the range of the output **linguistic variable**, the resulting crisp output values cannot span the full range. To solve this problem, use the modified Center of Area defuzzification method.

The modified Center of Area defuzzification method is similar to the Center of Area defuzzification method. However, the fuzzy logic controller considers the full area under the scaled membership functions, even if this area extends beyond the range of the output variable. The fuzzy logic controller uses the following equation to calculate the geometric center of the full area under the scaled membership functions.

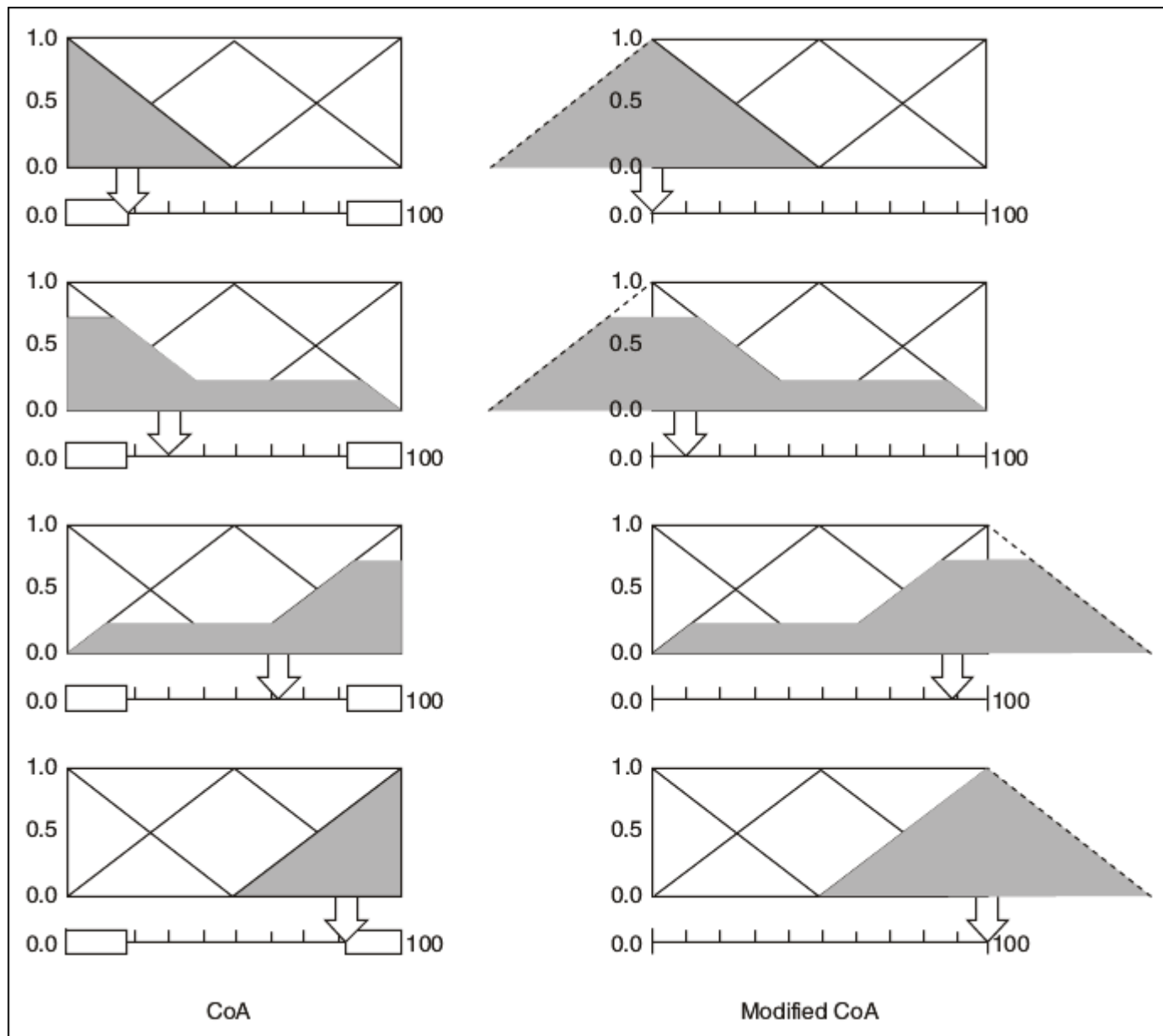
$$mCoA = \frac{\int f(x) * x \, dx}{\int f(x) \, dx}$$

where mCoA is the modified center of area.

The interval of integration is between the minimum membership function value and the maximum membership function value. Note that this interval might extend beyond the range of the output variable.

## Comparing CoA and Modified CoA Defuzzification Methods

The following image illustrates the difference between the Center of Area and modified Center of Area defuzzification methods.



## Selecting a Defuzzification Method

In decision support systems, the choice of the **defuzzification method** depends on the context of the decision you want to calculate with the fuzzy controller. For quantitative decisions like project prioritization, apply the **Center of Maximum (CoM)** method. For qualitative decisions, such as an evaluation of credit worthiness, **Mean of Maximum (MoM)** is the correct method.

An important aspect of a defuzzification method is the continuity of the output signal. Consider a fuzzy system with a complete rule base and overlapping membership functions.

A defuzzification method is continuous if an arbitrary small change of an input value never causes an abrupt change in the output signal.

In this respect, the defuzzification methods CoM and **Center of Area (CoA)** are continuous because, assuming overlapping output membership functions, the best compromise does not jump to a different value with a small change to the inputs. The defuzzification method MoM, however, is discontinuous because an arbitrary small change in the input values of the fuzzy system can cause the output value to switch to another, more plausible result.

Using CoA or CoM as the defuzzification method results in continuous **controller characteristic** functions, especially within those intervals of the input values in which two or more rules are active simultaneously. This behavior results from the averaging character of the defuzzification methods.

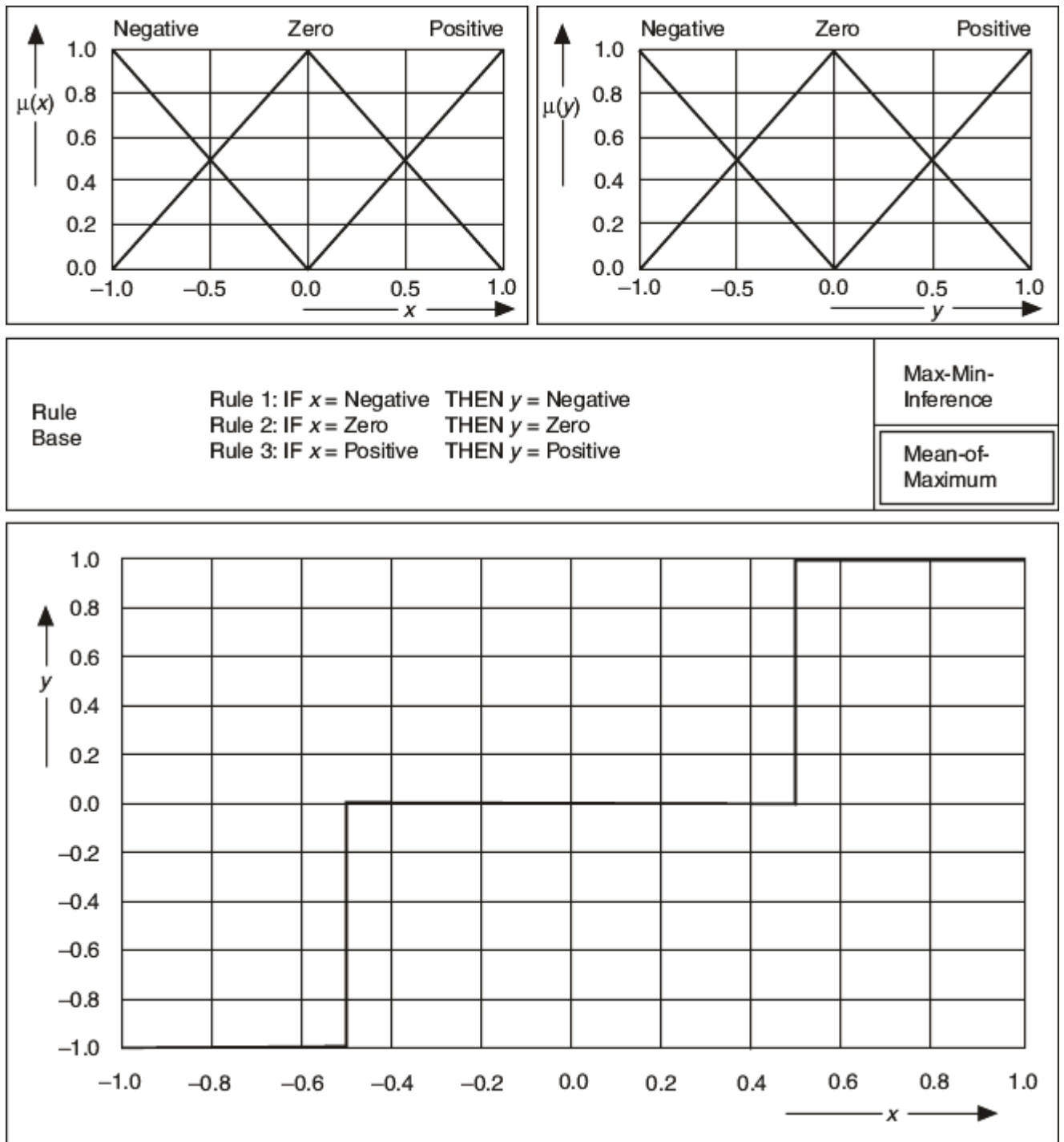
The following table compares the different defuzzification methods based on various assessment criteria.

| Assessment Criteria       | Method                                                                                          |                                                                                                 |                                                      |                                                      |
|---------------------------|-------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|------------------------------------------------------|------------------------------------------------------|
| Center of Sums (CoS)      | Center of Maximum (CoM)                                                                         | Mean of Maximum (MoM)                                                                           |                                                      |                                                      |
| Linguistic Characteristic | Best Compromise                                                                                 | Best Compromise                                                                                 | Best Compromise                                      | Most Plausible Result                                |
| Fit with Intuition        | Implausible with varying membership function shapes and strong overlapping membership functions | Implausible with varying membership function shapes and strong overlapping membership functions | Good                                                 | Good                                                 |
| Continuity                | Yes                                                                                             | Yes                                                                                             | Yes                                                  | No                                                   |
| Computational Effort      | Very High                                                                                       | Medium                                                                                          | Low                                                  | Very Low                                             |
| Application Field         | Closed-Loop Control, Decision Support, Data Analysis                                            | Closed-Loop Control, Decision Support, Data Analysis                                            | Closed-Loop Control, Decision Support, Data Analysis | Pattern Recognition, Decision Support, Data Analysis |

## Using the Mean of Maximum (MoM) Defuzzification Method

When you use the **Mean of Maximum (MoM) defuzzification method**, you calculate the most plausible result. In other words, the fuzzy controller uses the typical value of the

consequent term of the most valid rule as the crisp output value. This behavior results in stepped output characteristics, as shown in the following image.



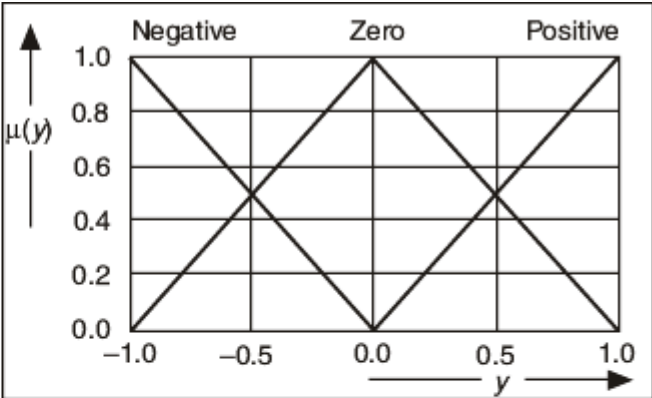
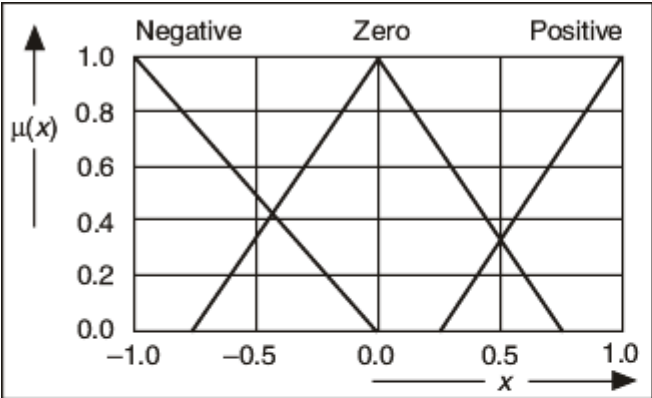
The previous image shows entirely overlapping **membership functions** for input and output terms.

### Changing the Rule Base

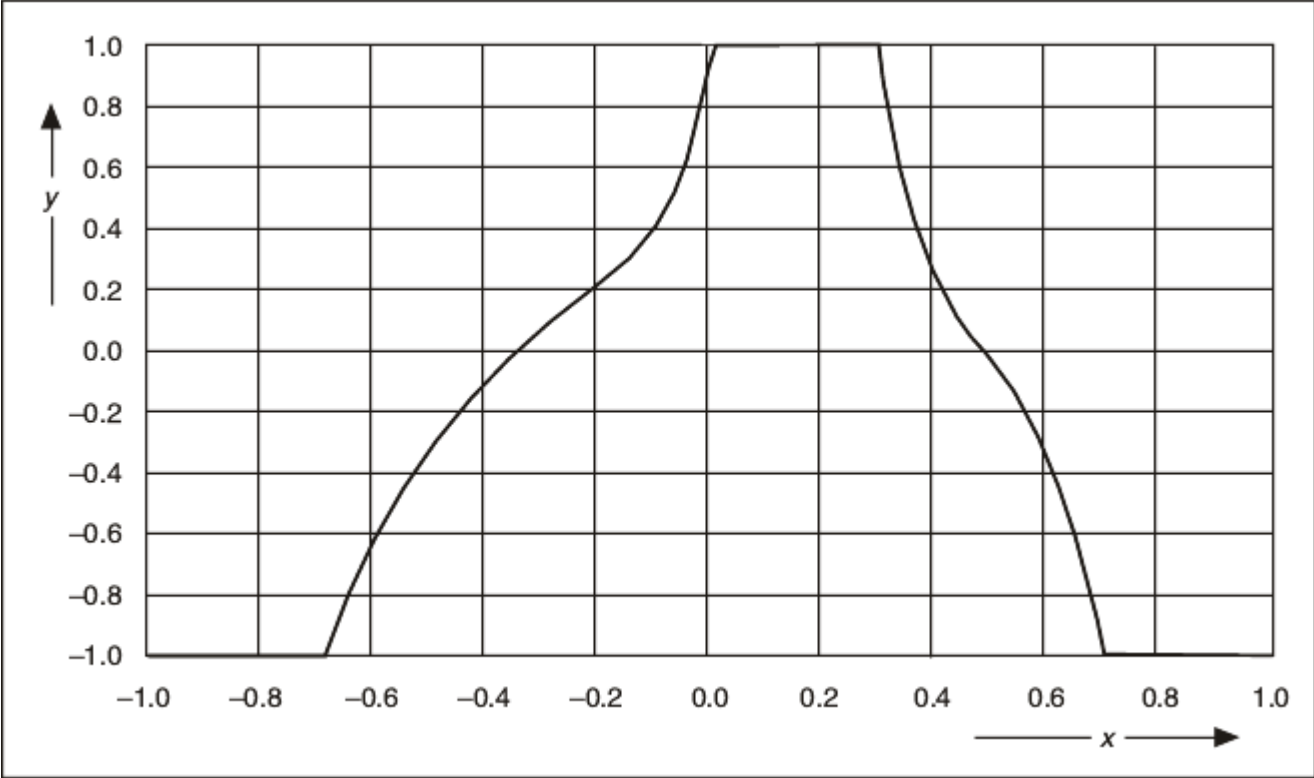
The **rule base** itself has the biggest influence on the controller characteristic. The rule base determines the principal functionality of the controller.

The following image illustrates how the controller characteristic changes if you change the rule base of the previous example to include the following rules:

|         |    |              |      |              |
|---------|----|--------------|------|--------------|
| Rule 1: | IF | x = negative | THEN | y = negative |
| Rule 2: | IF | x = zero     | THEN | y = positive |
| Rule 3: | IF | x = positive | THEN | y = negative |



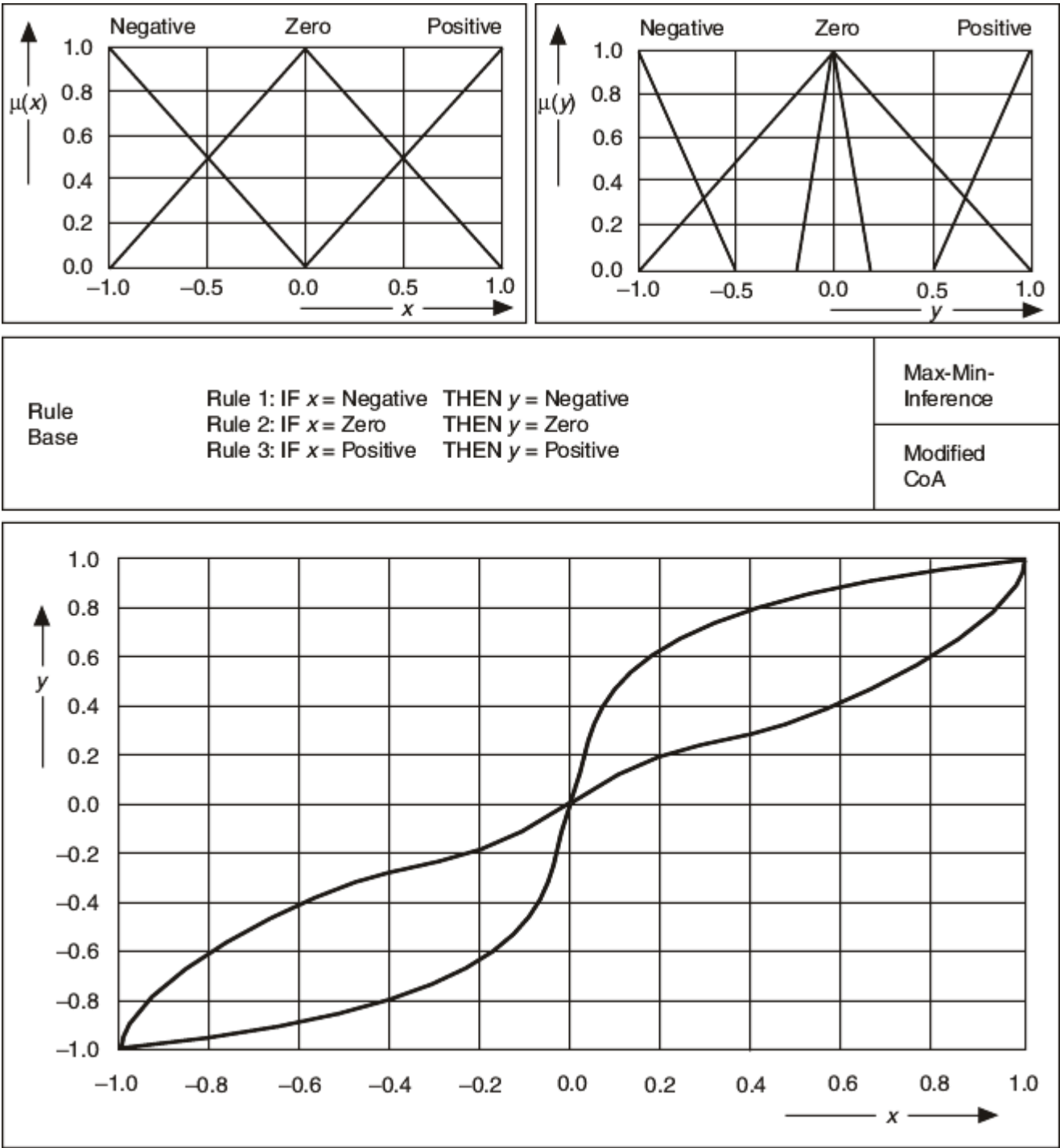
|           |                                                                                                                                 |                   |
|-----------|---------------------------------------------------------------------------------------------------------------------------------|-------------------|
| Rule Base | Rule 1: IF x = Negative THEN y = Negative<br>Rule 2: IF x = Zero THEN y = Positive<br>Rule 3: IF x = Positive THEN y = Negative | Max-Min-Inference |
|           |                                                                                                                                 | Modified CoA      |





## Wide and Small Membership Functions for Output Terms

If you want to significantly influence the controller characteristic, use output terms that membership functions model with equally distributed typical values but different scopes of influence instead. The different terms have different areas and thus different weights with respect to the defuzzification process. A wide output term has more influence on the inference result than a small neighboring output term. The following image demonstrates this effect.



RELATED INFORMATION

- [I/O Characteristics of Fuzzy Controllers](#) on page 1690
- [Creating Membership Functions](#) on page 1739
- [Defuzzification Methods](#) on page 1745

# 배포된 시스템 관리자

## 알람과 이벤트

### Acknowledging Alarms and Events

You can acknowledge alarms and events that the **Alarms and Events** view displays by using the **Acknowledge Selected Alarms and Events** or **Acknowledge All Alarms and Events** dialog box.



#### NOTE

When you acknowledge an alarm, the alarm appears in purple as acknowledged in the Alarms and Events view. When the alarm is no longer in the alarm state, the alarm disappears from the Alarms and Events view. When you acknowledge an event, the event disappears from the Alarms and Events view.

Complete the following steps to acknowledge alarms or events by using the Alarms and Events view.

1. Select the alarms and events that you want to acknowledge. You can press the <Ctrl> key and select multiple alarms and events.
2. Right-click a selected alarm or event and select Acknowledge from the shortcut menu to display the Acknowledge Selected Alarms and Events dialog box.
3. (Optional) Enter comments in the Comments text box.



#### NOTE

Comments apply to alarms only.

4. Click the OK button to acknowledge the selected items and close the dialog box.



#### NOTE

To acknowledge all the alarms and events at one time, you can right-click in this view and select Acknowledge All from the shortcut menu to display the Acknowledge All Alarms and Events dialog box. Then, complete steps 3 and 4 to acknowledge all alarms and events in this view.

### Actions Menu

The Actions menu contains items for starting and stopping the Shared Variable Engine, refreshing your workspace, and logging into a system.

- Start Local Shared Variable Engine —Restarts the **Shared Variable Engine**.
- 
**노트**  
 You must use an alternate method to start the Shared Variable Engine on Windows 7/Vista.
- Stop Local Shared Variable Engine —Shuts down the **Shared Variable Engine**.
- 
**노트**  
 You must use an alternate method to stop the Shared Variable Engine on Windows 7/Vista.
- Register Computer —Displays the Register Computer dialog box, which you can use to register remote computers.
- Alias Manager —Displays the Alias Manager dialog box, which you can use to configure aliases for local computers and IP addresses.
  - (DSC Module) The Shared Variable Engine uses aliases when running custom I/O servers that refer to other variables in the system.
- Log In —Displays the NI Security Login dialog box.
- Log Out —Logs out of the session configured in the NI Security Login dialog box.
- Configure Security —Displays the NI ACL Configuration dialog box, which you can use to update security settings.
- Add System to My Systems —Displays the Add Systems to My Systems dialog box that allows you to add a system to My Systems on the system tree.

## Alarm Probe View

Requires: DSC Module

In the system tree pane, right-click an alarm and select Probe to display this view. This view shows the status and properties of the selected alarm.

This view includes the following components:

- Location —Displays the location of the alarm.
- Status —Displays the status of the alarm.
- Priority —Displays the priority of the alarm.
- Level —Displays the level of the alarm.
- Description —Displays the description of the alarm.

## Alarms and Events View

Requires: NI Distributed System Manager (Windows)

Select View»Alarms and Events to display or hide the Alarms and Events view. The NI Distributed System Manager supports only one Alarms and Events view.

Use the Alarms and Events view to **monitor** and **acknowledge** alarms and events. You can view alarms and events of other processes by using the **Select Processes** dialog box. By default, the Alarms and Events view displays alarms and events of systems in My Systems. You can **add a system** to My Systems to access systems that do not appear in the system tree pane.



#### NOTE

You must have the LabVIEW Datalogging and Supervisory Control (DSC) Module installed to monitor and acknowledge alarms.

This view includes the following components:

- **Select Processes** —Displays the **Select Processes** dialog box, which you use to select multiple processes from the system tree pane. The Alarms and Events view then displays alarms and events that these processes generate.
- **Show Alarms** —Specifies whether to display alarms in the Alarms and Events view. By default, this checkbox contains a checkmark.
- **Show Events** —Specifies whether to display events in the Alarms and Events view. By default, this checkbox contains a checkmark.
- **Acknowledge** —Displays the **Acknowledge Selected Alarms and Events** dialog box, which you use to acknowledge alarms and events you select in the Alarms and Events view.
- **Acknowledge All** —Displays the **Acknowledge All Alarms and Events** dialog box, which you use to acknowledge all alarms and events in the Alarms and Events view.
- **Filter** —Displays the **Filter Options** dialog box, which you use to display alarms and events that meet certain criteria.
- **Columns** —Displays the Show Columns dialog box, which you use to show or hide columns in the Alarms and Events view.

## Alerts Probe View

Requires: Real-Time Module

Select a real-time target to display a target probe in the Auto View. You also can right-click the target and select Probe to display the probe in a separate window. Click the Alerts tab of a real-time target probe to display this view. Use this view to configure and monitor alerts on the real-time target.

This view includes the following components:

- **Log Alert when memory usage is above** —Enables alert logging when memory usage on the real-time target exceeds the value you specify.
- **Log Alert when CPU usage is above** —Enables alert logging when the real-time target CPU usage exceeds the value you specify.
- **Log Alert when VI changes state** —Enables alert logging when a VI listed in the **VI States** view changes state.
- **Recent Alerts** —Lists recently-logged alerts.

- Clear —Removes all entries from the Recent Alerts list.
- Save —Saves the Recent Alerts list as a text file on the local computer.

## CPU/Memory Probe View

Requires: NI Scan Engine

Select a real-time target to display a target probe in the Auto View. You also can right-click the target and select Probe to display the probe in a separate window. Click the CPU/Memory tab of a real-time target probe to display this view. Use this view to monitor CPU and memory usage on the target.

This view includes the following components:

- CPU N Usage Chart (%) —Tracks utilization of CPU N over time, as a percentage of capacity.
- Memory Usage Chart (MB) —Tracks target memory usage over time.
- CPU N Statistics —Includes current CPU usage statistics including total CPU usage, CPU usage by priority level, and CPU usage dedicated to Timed Structures and interrupt service routines (ISRs). Each value represents a percentage of total capacity.
- Memory Statistics —Displays current memory usage data.
  - Allocated —The amount of memory, in MB, currently allocated on the target.
  - Available —The amount of memory, in MB, currently available on the target.
  - Contiguous —The largest contiguous block of available memory on the target, in MB.
  - Total —The total amount of memory available to the operating system, in MB.
  - Usage —The percentage of total memory currently allocated on the target.

## Date Time Probe View

Requires: NI Distributed System Manager

Select a system from the system tree to display the system in the Auto View. You also can right-click the system and select Probe to display the Probe view in a separate window. Click the Date Time tab of a system probe to display this view. Use this view to set the date and time on the system. Date Time only appears for systems that you have sufficient privileges to modify the date and time settings.

This view includes the following components:

- Local System Time (UTC) —Displays the time on the current system running NI Distributed System Manager.

- Target System Time (UTC) —Displays the time on the selected target system displayed in the Probe view or Auto View.
- Set to local —Assigns the value of the Target System Time (UTC) to the value of the Local System Time (UTC). Use this button to sync times between the system running NI Distributed System Manager and the system displayed in the Probe view or Auto View.
- Refresh —Refreshes the values that appear in Local System Time (UTC) and Target System Time (UTC).

## I/O Variable Probe View

Requires: NI Scan Engine

Click an I/O variable in the system tree to display this view in the Auto View. You also can right-click an I/O variable and select Probe to display this view in a separate window.

This view displays the location of an I/O variable and allows you to view or customize the following components:

- Location —Specifies the location of the item you want to probe. To probe a variable, for example, enter the information in the following format: `//SystemName/Process/Variable`.
- Current Value —Displays the current value of the selected variable.
- New Value —Allows you to enter a new value to assign to the variable.
- Force — Forces the variable to assume New Value until you unforce the variable, reboot the target, or force the variable to assume another New Value.
- Unforce —Stops forcing the variable and returns control of the I/O value to the scan engine.
- Set —Sets the value in the New Value field as the current value.
- Show Trend —Displays the [trend view](#) graph of the selected variable.
- Data Type —Displays the data type of the selected variable.
- Timestamp —Displays the timestamp of the selected variable.
- Quality —Displays the quality of the selected variable. The quality indicates errors or other conditions that apply to the variable.
- Access Type —Displays the supported access mode(s) of the variable. This includes Read-only, Write-only, or Read/Write. LabVIEW modules and toolkits you have installed might provide additional access types.

## OpenCSRs Probe View

Requires: NI Distributed System Manager (Windows)

Right-click OpenCSRs and select Probe from the shortcut menu to display this view. This view displays [open certificate signing requests](#) available on the Web Server. You can

create new CSRs, display or delete existing CSRs, and install digitally signed certificates that correspond to an open CSR.

This view includes the following components:

- Open Certificate Signing Requests —Displays open CSRs on the Web Server.
- Create —Launches the **Create a Certificate Signing Request** dialog box.
- Delete —Deletes the selected CSR.
- Show —Displays the value of the selected CSR.
- Install —Displays the text field to paste the **signed certificate returned by the certificate authority (CA)**.

Refer to the LabVIEW Help for information about enabling SSL on the LabVIEW Web Server.

## Process Probe View

Requires: NI Distributed System Manager (Windows)

Right-click a process and select Probe from the shortcut menu to display this view. This view shows the status of the selected **process** and allows you to start or stop that process.

This view includes the following components:

- Location —Displays the location of the selected process.
- State —Displays whether the process is online or offline, or displays if you do not have sufficient access privileges to modify the state.
- Start Process —Starts the selected process. You must have sufficient access privileges to start a process.
- Stop Process —Stops the selected process. You must have sufficient access privileges to stop a process.

## Scan Engine Probe View

Requires: NI Scan Engine

Select a target with the **NI Scan Engine** installed to display this view in the Auto View. You also can right-click the target and select Probe to display this view in a separate window. Use this view to monitor and manage the scan engine running on the target.

This view includes the following components:

- Forcing —Contains information and options related to **I/O variable forcing**.



노트

Forcing does not persist after a reboot of the target.



- Forcing Enabled —Indicates whether forcing is enabled on the target. Forcing must be enabled on the target for individual I/O variable or I/O alias forces to take effect. This setting serves as a master switch for all I/O variable and I/O alias forcing on the target.
- Enable Forcing —Enables forcing on the target, switching the state of the Forcing Enabled setting to TRUE. This option is available only when I/O variable forcing is disabled.
- Disable Forcing —Disables forcing on the target, switching the state of the Forcing Enabled setting to FALSE. This option is available only when I/O variable forcing is enabled.



#### 노트

Unless you clear forcing, the previous forced values of all forced I/O variables and I/O aliases return when you re-enable forcing on the target.

- Clear Forcing —Unforces all currently forced I/O variables and I/O aliases on the target and returns control of the I/O values to the NI Scan Engine.



#### 노트

This button does not affect the master Forcing Enabled setting on the target.

- Faults —Contains information and options related to **scan engine faults**.
  - Fault Present —Indicates the presence of **faults** on the target. The LED appears black when no fault is present, orange when only minor faults are present, and red when major or unrecoverable faults are present.
  - Clear All Faults —Clears all **faults** on the target. If the target is in Fault mode, clearing all faults returns the target to the previous mode in effect before entering **Fault Mode**.
  - Clear Selected Fault —Clears the fault currently selected in the Fault List. If no fault is currently selected, no action occurs.
  - Fault List —Lists the faults active on the target.
    - Error Code —The error code associated with the fault.
    - Highest Level —The highest level of the fault since last cleared.
    - First Occurrence —The time of the first occurrence of the fault since last cleared.
    - Num of Occurrences —The number of occurrences of the fault since last cleared.
    - Description —The fault description.
- Scan Engine Mode —Contains information and options related to the **mode** of the NI Scan Engine on the target.



#### 노트

This section is available only with expansion I/O hardware. This section is not available if only local I/O drivers are installed on the target. Refer to the I/O hardware documentation for information about hardware-specific mode behavior.

- Configuration —Indicates whether the scan engine is in Configuration mode.

- Active —Indicates whether the scan engine is in Active mode.
- Fault —Indicates whether the scan engine is in Fault mode.
- I/O Drivers —Lists the I/O drivers installed on the target that interface with the NI Scan Engine.
- Change to Configuration —Switches the scan engine to Configuration mode. If the target was previously in Fault mode, switching to Configuration mode automatically changes the level of all active major faults to minor.
- Change to Active —Switches the scan engine to Active mode. You cannot switch directly to Active mode from Fault mode. You must clear all major faults to leave Fault mode before you can switch to Active mode.
- Refresh Local Modules —Discovers new I/O modules on the target. If a local I/O module is detected in a slot that was not previously deployed from a LabVIEW project, this operation creates new I/O variables for the I/O module. These automatically created I/O variables use the default I/O variable settings. This operation does not add the new I/O variables to a LabVIEW project, and the new I/O variables do not persist when the target restarts. If you physically change an I/O module deployed from a LabVIEW project, this operation cannot overwrite the deployed configuration until you undeploy the I/O module in the LabVIEW project. However, if you physically change an I/O module that was automatically detected by this operation or the corresponding Refresh Local Modules VI, this operation adds and deletes automatically created I/O variables to match the new physical configuration.

## ServerCertificates Probe View

Requires: NI Distributed System Manager (Windows)

Right-click ServerCertificates and select Probe from the shortcut menu to display this view. This view displays **self-signed certificates** available on the Web Server, including the default LabVIEW self-signed certificate. You can create a new self-signed certificate and display or delete existing self-signed certificates.

This view includes the following components:

- Server Certificates —Displays self-signed certificates on the Web Server.
- Create —Launches the **Create a Self-Signed Certificate** dialog box.
- Delete —Deletes the selected self-signed certificate.
- Show —Displays the value of the selected self-signed certificate.

Refer to the LabVIEW Help for information about enabling SSL on the LabVIEW Web Server and the default LabVIEW self-signed certificate.

## Variable Probe View

Requires: NI Distributed System Manager (Windows)

Right-click a variable and select Probe from the shortcut menu to display this view. This view displays the location of the selected **shared or network variable** and allows you to view or customize aspects of that variable.

This view includes the following components:

- Location —Displays the location of the selected variable.
- Current Value —Displays the current value of the selected variable.
- New Value —Allows you to enter a new value to assign to the variable.
- Set —Sets the value in the New Value field as the current value.
- Show Trend —Displays the **trend view** graph of the selected variable.
- Data Type —Displays the data type of the selected variable.
- Timestamp —Displays the timestamp of the selected variable.
- Quality —Displays the quality of the selected variable. The quality indicates errors or other conditions that apply to the variable.
- Access Type —Displays the supported access mode(s) of the variable. This includes Read-only, Write-only, or Read/Write. LabVIEW modules and toolkits you have installed might provide additional access types.

## Web Server Probe View

Requires: NI Distributed System Manager (Windows)

Right-click a Web Server and select Probe from the shortcut menu to display this view. This view displays Web services available on the Web Server. You can pause, resume, restart, delete, or reset statistics for one or all of the available Web services. Refer to the LabVIEW Help for information about LabVIEW Web services and related terminology.

This view includes the following components:

- Location —Displays the location of the selected Web Server.
- Deployed Web Services —Displays a listbox of all Web services available on the Web Server.
  - Operations on selected —Displays components that perform operations on the selected Web service.
    - Pause —Pauses the selected Web service.
    - Resume —Resumes the selected Web service.
    - Restart —Restarts the selected Web service.
    - Reset Statistics —Resets all statistical variables for the selected Web service.
    - Delete —Removes the selected Web service from the Web Server.
- Operations on all —Displays components that perform operations on all Web services on the Web Server.
  - Pause All —Pauses all Web services on the Web Server.

- Resume All —Resumes all Web services on the Web Server.
- Restart All —Restarts all Web services on the Web Server.
- Reset All Statistics —Resets statistical variables for all Web services on the Web Server.

## Web Server URL Mapping View

Requires: NI Distributed System Manager (Windows)

Select a URL mapping and click the Run Selected button on the [Web Server Probe](#) view to display this view. This view allows you to assign values to connector pane inputs of the Web method VI specified by the selected URL mapping. You then can send an HTTP request with these values to the Web method VI. Refer to the LabVIEW Help for information about LabVIEW Web services and related terminology.

This view includes the following components:

- Parameters —Displays available connector pane inputs for the Web method VI associated with the selected URL mapping. You must assign connector pane inputs before you send an HTTP request to the Web method VI.
- Values —Specifies values for the connector pane inputs of the Web method VI associated with the selected URL mapping.
- Results —Displays the response from the Web service after you send an HTTP request.
- Host —Displays the computer name of the Web Server running the selected Web service.
- Port —Displays the port of the Web Server interface running the selected Web service.
- Request —Displays the complete URL mapping for the HTTP request.
- Send request —Sends an HTTP request to the Web method VI associated with the selected URL mapping. You must specify Values for the connector pane inputs of the Web method VI before you send an HTTP request.

## Web Service Probe View

Requires: NI Distributed System Manager (Windows)

Right-click a Web service and select Probe from the shortcut menu to display this view. This view displays the state of the selected Web service and allows you to pause, resume, restart, or reset statistics for the Web service. You also can send HTTP requests to the Web service using available URL mappings. Refer to the LabVIEW Help for information about LabVIEW Web services and related terminology.

This view includes the following components:

- Location —Displays the location of the selected Web Service.

- State —Displays whether the Web service is running or paused.
- Pause —Pauses the selected Web service.
- Resume —Resumes the selected Web service.
- Restart —Restarts the selected Web service.
- Reset Statistics —Resets all statistical variables for the selected Web service.
- URL Mappings —Displays a listbox of available URL Mappings for the selected Web service.
  - Run Selected —Displays the **Web Server URL Mapping** view for the selected URL mapping.

## Configure Web Services Security View

Requires: NI Distributed System Manager (Windows)

Right-click a Web Server in the system tree and select Web Services Security from the shortcut menu to display this dialog box. Use this dialog box to configure security for LabVIEW Web services. Refer to the LabVIEW Help for information about LabVIEW Web services security and related terminology.

This view includes the following components:

- Web Services security key —Configures the API key used for Web services security.
  - Access ID —Specifies the public ID component of the API key. The Access ID works similarly to a username.
  - Secret ID —Specifies the private ID component of the API key. The Secret ID works similarly to a password and only authorized clients should possess this string.
  - Paste from Clipboard —Pastes an API key located on the clipboard into the Access ID and Secret ID fields. If the content of the clipboard does not conform to the correct format of an API key, the content of the clipboard does not import and the current Access ID and Secret ID remain.
  - Reset —Resets the Access ID and Secret ID.

## Configuring Alarms for Shared Variables (DSC Module)

Complete the following steps to configure alarms for shared variables.

1. In the system tree pane, locate the alarm you want to configure and expand this alarm. The System Manager displays the attributes of the alarm.
2. Right-click the attribute you want to configure and select Probe from the shortcut menu. A Probe view of the attribute appears.

- Specify a new value of the attribute in the New Value text box or option buttons to configure this attribute.



#### 노트

You cannot configure the alarm statuses, such as Ack, Description, and Set.

- Click the Set button to set the new value of the attribute.



#### 노트

You also can configure alarms by using the Auto View.

## Monitoring Alarms and Events

Use the **Alarms and Events** view to monitor alarms and events. Complete the following steps to monitor alarms and events by using the Alarms and Events view.

- Locate the Alarms and Events view. If you do not see this view, select View»Alarms and Events to display this view.
- (Optional) Configure the filter options to display alarms, events, or both alarms and events in this view.
- Click the Select Processes button to display the **Select Processes** dialog box. You also can display this dialog box by right-clicking in the Alarms and Events view and selecting Select Processes from the shortcut menu.
- In the Available Processes list, select the processes you want to monitor. Press and hold the <Shift-Ctrl> keys to select multiple processes from the list.
- Click the Add button to add this process to the Selected Processes list.



#### NOTE

You can select a process from the Selected Processes list and click the Remove button to remove the process from this list. You also can click the Clear button to remove all processes from this list.

- Repeat steps 4 and 5 to add all the processes that you want to monitor to the Selected Processes list.
- Click the OK button to close the Select Processes dialog box.

The Alarms and Events view displays alarms and events that the selected processes generate. You can **acknowledge** these alarms and events by using the Alarms and Events view.

## Monitoring and Editing Data Item Ranges

Complete the following steps to monitor and edit data items by using the Probe view or Auto View.



#### NOTE

You cannot monitor or edit extended data items or array data items in the NI System Manager.

1. In the **system tree**, right-click a data item range and select Probe from the shortcut menu to create a Probe view. The Probe view displays detailed information about the data item range.
2. In the Change item text box, enter the data item that you want to monitor or edit.
3. Click the Apply button to display the value of the data item.
4. Enter the new value of the data item in the New Value section.
5. Click the Set button to apply the new value to the data item.

**NOTE**

You also can click the Auto View to display the detailed information.

## Monitoring Data Items in Data Item Ranges

Complete the following steps to create a new Watch List view for data item ranges or add a data item range to an existing Watch List view.

1. In the **system tree**, right-click a data item range and select Watch List from the shortcut menu. The System Manager displays the Add Item to Watch List dialog box. You also can drag a data item to an existing Watch List view to display this dialog box.
2. In the Items from range text box, enter the data item to monitor.
3. Click the OK button to create a watch list for the data item.

**NOTE**

You can create a Watch List view for multiple data item ranges or multiple system tree objects that include data item ranges. You also can add these system tree objects to an existing Watch List view. When you create a Watch List view for these system tree objects or add these system tree objects to an existing Watch List view, the Add Item to Watch List dialog box does not appear. The System Manager adds the first data items of the data item ranges by default.

## Monitoring Variables with the Watch List View

The Watch List view displays information about a single **variable** or multiple variables contained within a system or process. The Watch List view displays the variable location, current value, data type, timestamp, access (read/write) status, and quality. The quality indicates errors or other conditions that apply to the variable.

**NOTE**

Variables that use the array and cluster data type do not display the values of individual components.

Complete the following steps to monitor variables using the Watch List view.

1. Right-click a **system tree object**, which includes all **systems**, **processes**, and variables available in the system tree.
2. Select Watch List from the shortcut menu to open a new Watch List view.
3. To customize the Watch List view, right-click anywhere inside the Watch List view.
4. Select Show Columns from the shortcut menu to show and hide any of the information columns available in the Watch List view. You also can reorder the columns by dragging the column headings.

## 인증서 및 프로세스 생성하기

### Creating a Certificate from a CSR

After you **create a certificate signing request (CSR)**, you must have the CSR digitally signed by a trusted certificate authority (CA) in order to create a certificate.

Complete the following steps to create a certificate from a CSR.

1. Create a CSR.
2. In System Manager, expand the tree item for the system, such as localhost.
3. Expand the item NI\_SSLConfiguration.
4. Expand the item OpenCSRs.
5. Select a CSR from the tree, such as Request Handle - 100.
6. Copy the text from the Certificate Signing Request field of the **OpenCSRs Probe** view for the item. If the OpenCSRs Probe view does not appear, right-click the CSR and select Probe from the shortcut menu.
7. Send the text to a trusted CA that can digitally sign the CSR. If you do not have a trusted CA to sign the request, you also can **create a self-signed certificate** rather than using a CSR and a CA.
8. When a CA returns a signed certificate, right-click the CSR from the tree and select Install from the shortcut menu.
9. Paste the character string provided by the CA into the text box.
10. Click the Install button.

If the installation is successful, a new certificate with the same name as the CSR appears under the ServerCertificates item in the tree. The System Manager automatically deletes the CSR from both the tree and the file system.

### Creating a Certificate Signing Request

You can create a certificate signing request (CSR) using the System Manager.

Complete the following steps to create a CSR in the System Manager.



1. Enable SSL on an instance of the LabVIEW Web Server. Refer to the LabVIEW Help for information about enabling SSL on the LabVIEW Web Server.
2. Expand the tree item for a system, such as localhost.
3. Expand the item NI\_SSLConfiguration.
4. Right-click OpenCSRs from the tree and select Probe from the shortcut menu to display the **OpenCSRs Probe** view.
5. In the OpenCSRs Probe view, click the Create button to display the **Create a Certificate Signing Request** dialog box.
6. In the Create a Certificate Signing Request dialog box, enter the appropriate identifying information into the Values column.
7. Click the Submit button. If you experience errors, confirm that the identifying information uses the **correct syntax**.

The dialog box displays the name and character string for the new CSR. You must send this character string to a trusted CA to **create a certificate from the CSR**.

## Creating a Self-Signed Certificate

You can create a custom self-signed certificate using the System Manager. You can use this self-signed certificate in place of the default LabVIEW self-signed certificate.

Complete the following steps to create a custom self-signed certificate in the System Manager.

1. Enable SSL on an instance of the LabVIEW Web Server. Refer to the LabVIEW Help for information about enabling SSL on the LabVIEW Web Server.
2. Expand the tree item for a system, such as localhost.
3. Expand the item NI\_SSLConfiguration.
4. Right-click ServerCertificates from the tree and select Probe from the shortcut menu to display the **ServerCertificates Probe** view.
5. In the ServerCertificates Probe view, click the Create button to display the **Create a Self-Signed Certificate** dialog box.



### 노트

The default LabVIEW self-signed certificate appears as Certificate Handle-0 in the tree. All custom self-signed certificates that you create are assigned a subsequent number starting at 100, such as Certificate Handle-100.

6. In the Create a Self-Signed Certificate dialog box, enter the **appropriate identifying information** into the Values column.
7. Select the duration of the certificate in the Valid for how many days? option.
8. Click the Submit button. If you experience errors, confirm that the identifying information uses the correct syntax.

The dialog box displays the name and public key for the new certificate. The certificate also appears in the tree under the ServerCertificates item.

## Creating Processes

You can create processes on a local or LabVIEW Real-Time target.

Complete the following steps to create a process with the System Manager.

1. Right-click a system in the system tree.
2. Select Add Process from the shortcut menu to display the Add Process dialog box.
3. Configure the process in the Add Process dialog box. You can use this dialog box to configure the process name and whether the process will run after you create it.

After you create a process, you can add shared or network variables or add I/O servers to that process.



### 관련 정보

[Processes](#) ( 1778 페이지 참조)

[Systems](#) ( 1787 페이지 참조)

[Creating, Modifying, or Deleting Shared or Network Variables](#) ( 1774 페이지 참조)

[I/O Servers](#) ( 1783 페이지 참조)

## Creating, Modifying, or Deleting Shared or Network Variables

You can create [shared or network variables](#) within [processes](#) on a local or real-time target.

Complete the following steps to create a variable with the System Manager.

1. Select an existing process in the system tree or [create a new process](#).
2. Right-click the process and select Add Variable from the shortcut menu to display the [Shared Variable Properties](#) dialog box.
3. Configure the variable in the Shared Variable Properties dialog box. You can use this dialog box to configure settings such as the variable name, data type, and buffering.

You now can use the variable in an application. When the process is running, the [Watch List](#) view updates to include information about the variable such as current value, timestamp, and quality messages. Quality messages list errors and other conditions that apply to the variable.

## Modifying a Shared or Network Variable

Right-click the variable in the system tree and select **Edit Variable** from the shortcut menu to display the Shared Variable Properties dialog box.

## Deleting a Shared or Network Variable

Right-click the variable in the system tree and select **Remove Variable** from the shortcut menu.

# Custom VI-based Server - On Input Change Wizard

Requires: DSC Module

Click the **New** button on the **Configure Custom VI - On Input Change I/O Server** dialog box to display this wizard. You also can select **Tools»DSC Module»VI Based Server Wizard»On Input Change** in LabVIEW to display this wizard.

This wizard includes the following pages:

- **Select VI** —Use this page to specify a VI you want to convert into an I/O server. Contains the following options:
  - **Path to VI** —Specifies the path to the VI that you want to convert to a custom I/O server.
  - **Run VI Analyzer on VI** —Displays the LabVIEW VI Analyzer Toolkit, which runs tests that can identify improvements you can make to the VI. You must have the LabVIEW VI Analyzer Toolkit installed to use this option.
- **Configure Server Distribution Component** —Specifies build options for the I/O server. Contains the following options:
  - **Configuration name** —Specifies the name of the configuration for the custom I/O server. You can have multiple configurations for each custom I/O server. The default name is the name of the VI converted to a custom I/O server.
  - **Support files** —Lists additional VIs you want to include in the LabVIEW project library to deploy. The wizard automatically includes any subVIs or other files that are in the VI hierarchy. You only need to add dynamically called VIs or files.
  - **Select Support File** —Specifies the path to the support files needed to run the VI-based server. Click the **Browse** button to navigate to and select the support files. You must select a file before you can click the **Add** button to add the file to the Support files list.
  - **Add** —Adds the VI or file to the Support files list.
  - **Remove** —Removes the VI or file you select in the Support files list.
- **Server Distribution Component** —Displays a summary of the server the Custom VI-based Server - On Input Change Wizard builds and allows you to build the server. Contains the following options:

- Server configuration file —Specifies the path to the VI that the wizard converts to a custom I/O server.
- Server support files directory —Specifies the path to the files created when the wizard converts the VI to a custom I/O server.

Click the Build button to build the on input change I/O server. When the Custom VI Server - On Input Change Wizard generates the files, you can open the Configure Custom VI - On Input Change I/O Server dialog box to configure the I/O server.

## Custom VI-based Server - Periodic Wizard

Requires: DSC Module

Click the New button on the [Configure Custom VI - Periodic I/O Server](#) dialog box to launch the Custom VI Server - Periodic Wizard. You also can select Tools»DSC Module»VI Based Server Wizard»Periodic to launch this dialog box. Use this wizard to create a custom VI-based periodic I/O server.

This wizard includes the following pages:

- Select VI —Use this page to specify a VI you want to convert into an I/O server. Contains the following options:
  - Path to VI —Specifies the path to the VI that you want to convert to a custom I/O server.
  - Run VI Analyzer on VI —Displays the LabVIEW VI Analyzer Toolkit, which runs tests that can identify improvements you can make to the VI. You must have the LabVIEW VI Analyzer Toolkit installed to use this option.
- Select Controls and Indicators To Publish —Specifies the controls and indicators of the VI for which you want to publish values. Contains the following options:
  - Controls —Lists the controls on the VI you can select to convert to a custom I/O server.
  - Select All —Selects all of the controls or indicators on the VI.
  - Deselect All —Deselects all of the controls or indicators on the VI.
  - Indicators —Lists the indicators on the VI you can select to convert to a custom I/O server.
  - Select All —Selects all of the controls or indicators on the VI.
  - Deselect All —Deselects all of the controls or indicators on the VI.
- Select Method To Stop The Server —Specifies a mechanism to stop the periodic I/O server. Contains the following options:
  - Abort VI —Configures the custom I/O server to abort the VI.
  - Stop the following While Loops —Configures the custom I/O server to use the While Loop you select to stop running the VI. Contains the following options:
    - While Loops —Lists the While Loops available to stop the VI.

- View on block diagram —Opens the block diagram from which you can select a While Loop to stop the VI.
- Timeout —Specifies the amount of time, in milliseconds, you want the VI to wait before timing out.
- Configure Server Distribution Component —Specifies build options for the I/O server. Contains the following options:
  - Configuration name —Specifies the name of the configuration for the custom I/O server. You can have multiple configurations for each custom I/O server. The default name is the name of the VI converted to a custom I/O server.
  - Support files —Lists additional VIs you want to include in the LabVIEW project library to deploy. The wizard automatically includes any subVIs or other files that are in the VI hierarchy. You only need to add dynamically called VIs or files.
  - Select Support File —Specifies the path to the support files needed to run the VI-based server. Click the Browse button to navigate to and select the support files. You must select a file before you can click the Add button to add the file to the Support files list.
  - Add —Adds the VI or file to the Support files list.
  - Remove —Removes the VI or file you select in the Support files list.
- Server Distribution Component —Displays a summary of the files the Custom VI Server - Periodic Assistant builds. Contains the following options:
  - Server configuration file —Specifies the path to the VI that the wizard converts to a custom I/O server.
  - Server template file —Specifies the path to the template file created when the wizard converts the VI to a custom I/O server.
  - Server support files directory —Specifies the path to the files created when the wizard converts the VI to a custom I/O server.

Click the Build button to build the periodic I/O server. When LabVIEW generates the files, you can edit the configuration and set initial values for the I/O server in the Configure Custom VI - Periodic I/O Server dialog box.

## Customizing the System Manager Workspace

The System Manager allows you to customize the workspace. Most System Manager views appear as a window by default. You can dock these views on the workspace to build a custom work environment.

Complete the following steps to customize the System Manager workspace.

1. Open a new view, such as a [Watch List](#) or Event Window view.
2. Click and hold the top bar of the desired view window.

3. Drag the window to the top, bottom, or sides of the System Manager workspace to dock the window.



#### 노트

If you dock multiple views in an area of the workspace, the System Manager automatically adds tabs to allow navigation between the views.

4. To undock a view to a separate window, select the Restore Down button  in the top-right of the view.



#### 노트

Select View»Auto View to display the Variable Details view for the selected **system tree object**. The Variable Details view appears docked by default when you select Auto View.

5. Select File»Save Workspace to save the workspace as a .smw file.
6. Select File»Open Workspace and navigate to a .smw file to restore a previously customized workspace.

## Processes

Processes are hierarchies of **shared or network variables** and **I/O servers**. Processes provide a logical means of organizing data. For example, if multiple, independent applications run on one server, you can create a process for each application. Similarly, if you have a very large application, you might divide the application into discrete subsystems and create a process for each subsystem. In addition to organizing data, you can start and stop each process independently, which allows you to take one subsystem offline while the rest of the application continues to run. When you stop a process, the values of variables in that process become unavailable and do not update.

Use the System Manager to **create processes** and start and stop existing processes.

### Shortcut Menu

Right-click a process on the **system tree** to display the following options.

- Watch List —Creates a new **Watch List** view that displays the information for all variables contained within the process you select.
- Probe —Displays the **Process Probe** view for the selected process.
- Add Variable —Displays the **Shared Variable Properties** dialog box, which you can use to **create a new variable** within the selected process.
- Add I/O Server —Displays the **Create New I/O Server** dialog box to create an I/O server and add it to the **Shared Variable Engine**.
- Stop Process —Stops the selected process in the system tree. When a process stops, the values of variables in that process become unavailable and do not update. You must have sufficient access privileges to stop a process.
- Start Process —Starts the selected process in the system tree. Processes automatically start when you create them. If you select the Stop Process item to

stop a process, you can use Start Process to restart the process. You must have sufficient access privileges to start a process.

- Remove Process —Removes the selected process in the system tree. You must have sufficient access privileges to remove a process.

## 시스템 관리

### System Manager Overview

Use the System Manager to complete the following tasks:

- Create shared or network variables.
- Monitor shared or network variables.
- Create and monitor processes.
- Create and monitor I/O servers.
- Manage security.
- Manage aliases.
- Start and stop the Shared Variable Engine.
- Force I/O variable values.
- View and clear scan engine faults.
- View and set scan engine modes.
- Monitor real-time target resources.
- Monitor alarms and events.
- Acknowledge alarms and events.
- Configure alarms for shared variables.
- Monitor historical trends.
- Log historical trends.
- Synchronize time for networked computers.



#### NOTE

You must use an alternate method to start or stop the Shared Variable Engine on Windows 7/Vista.

Select Start»All Programs»National Instruments»Distributed System Manager to launch the System Manager. In LabVIEW, you also can launch the System Manager by selecting Tools»Distributed System Manager.

### System Manager Components

The System Manager includes the following components:

- System tree —Displays all available objects on the network. The system tree appears as the primary navigation pane in the System Manager workspace.
- System tree object —Refers to all types of objects that might appear in the system tree. System tree object can refer to a system, process, variable, or additional object types installed with modules and toolkits.
- Watch list — Monitors a single variable or multiple variables contained within a system or process.
- Auto View —Displays the Probe view for any system tree object that you select. If you select multiple objects, this view displays a Watch List view that contains all of the selected objects.
- Probe views —Display information and configuration options for the following components:
  - CPU/Memory
  - I/O Variable
  - Process
  - Scan Engine
  - Variable
  - Web Server



- Web Service



#### RELATED INFORMATION

[Creating, Modifying, or Deleting Shared or Network Variables](#) on page 1774

[Monitoring Variables with the Watch List View](#) on page 1771

[Processes](#) on page 1778

[I/O Servers](#) on page 1783

[Shared Variable Engine](#) on page 1785

[Forcing I/O Variable Values \(NI Scan Engine\)](#) on page 1782

[Viewing and Clearing Faults \(NI Scan Engine\)](#) on page 1788

[Viewing and Setting the Scan Engine Mode \(NI Scan Engine\)](#) on page 1789

[Monitoring Alarms and Events](#) on page 1770

[Acknowledging Alarms and Events](#) on page 1759

[Configuring Alarms for Shared Variables \(DSC Module\)](#) on page 1769

[Synchronizing Time for Networked Computers](#) on page 1786

[Systems](#) on page 1787

[Processes](#) on page 1778

[Shared or Network Variables](#) on page 1785

[Monitoring Variables with the Watch List View](#) on page 1771

[CPU/Memory Probe View](#) on page 1762

[I/O Variable Probe View](#) on page 1763

[Process Probe View](#) on page 1764

[Scan Engine Probe View](#) on page 1764

[Variable Probe View](#) on page 1766

[Web Server Probe View](#) on page 1767

[Web Service Probe View](#) on page 1768

## File Menu

The File menu contains items for managing System Manager workspaces. Workspaces display information about all [shared or network variables](#) that are currently deployed, including the URL, current value, data type, timestamp information, and any error or quality messages. Quality messages list errors and other conditions that apply to the variable.

- **New Workspace** —Creates a new workspace. When you select this menu item, the System Manager removes all variables from the Watch List view and resets the workspace layout.

- Open Workspace —Opens a saved workspace.
- Save Workspace —Saves the current workspace with the .smw extension.
- Settings —Displays the **Settings** dialog box where you can customize workspace layout, set the data update rate, and specify the System Manager to save its workspace between sessions.

## Forcing I/O Variable Values (NI Scan Engine)

You can use an **I/O variable probe** to force I/O variable values. Forcing an I/O variable causes the input or output and all associated aliases of the variable to assume the forced value until you unforce the value. If an input from hardware returns an error, and the associated I/O variable is forced, System Manager does not display the error. However, if the error results in a **fault**, LabVIEW registers the fault. If an output returns an error, System Manager displays the error.



### NOTE

If you link several I/O variables in a chain of aliases and force one of the I/O variables in the chain, the forced value applies to all I/O variables in the chain.

### Enabling I/O Variable Forcing

Complete the following steps to enable and disable I/O variable forcing on a target with the **NI Scan Engine** installed.

1. Click a scan-enabled target in the system tree to display a target probe in the Auto View. You also can right-click a target and select Probe to display a target probe in a separate window.
2. The **Scan Engine** page of a real-time target probe displays the Forcing Enabled LED indicator, which indicates whether I/O variable forcing is enabled on the controller.
3. Use the Enable Forcing button to enable forcing. Use the Disable Forcing button to disable forcing and unforce all forced I/O variable values. Use the Clear Forcing button to clear forcing on all I/O variables.



### NOTE

If you disable forcing and then enable forcing again, all previously-forced I/O variables return to their last forced value, unless you have cleared those values using the Clear Forcing button.

### Forcing I/O Variable Values

Complete the following steps to force and unforce I/O variable values.

1. Click an I/O variable in the system tree to display an I/O variable probe in the Auto View or right-click an I/O variable and select Probe to display an I/O variable probe in a separate window.

2. In the New Value section of the I/O variable probe, enter or select the new value for the I/O variable.
3. Click the Force button to force the I/O variable to the value you specified.
4. Click the Unforce button to unforce the I/O variable value.

**NOTE**

Individual I/O variable forcing applies only if forcing is enabled on the target. Use the Enable Forcing and Disable Forcing buttons to enable and disable forcing on the target.

## Batch Forcing I/O Variable Values

Complete the following steps to perform batch forcing on a set of I/O variables.

1. Disable forcing on the target.
2. Force each I/O variable to the desired value.
3. Enable forcing on the target.

## Help Menu

The Help menu contains items to explain and define System Manager features and provide System Manager documentation.

- System Manager Overview —Displays the NI Distributed System Manager Help. Use this help file for information about creating and managing shared or network variables and other **system tree objects**.

**NOTE**

Refer to the LabVIEW Help, NI Measurement Studio Help, or LabWindows/CVI Help for more information about using variables created with the System Manager in your applications.

- About System Manager —Displays general information about the current installation, such as the version number.

## I/O Servers

An I/O server is an application that communicates with and manages input and output devices such as programmable logic controllers (PLCs), remote input/output devices, remote Shared Variable Engines, and data acquisition (DAQ) plug-in devices. These I/O servers read selected input items and write to the selected output items on demand.

### Shortcut Menu

Right-click an I/O server on the system tree to display the following options.

- Watch List —Creates a new **Watch List** view that contains the single **variable** or multiple variables contained within the selected **system** or **process**.
- Probe —Displays the I/O Server Probe view for the selected system.
- Edit I/O Server —Configures the selected I/O server.
- Remove I/O Server —Removes the selected I/O server.

## Observing Trend View of Variables

Use the **Variable Probe** view in the System Manager to monitor the value of a **variable** over time using the trend view graph.

Complete the following steps to monitor the value of a variable over time using the trend view graph.

1. Right-click a variable in the **system tree**.
2. Select Probe from the shortcut menu to open a new Variable Probe view for the selected variable.
3. Place a checkmark in the Show Trend checkbox to display the trend view graph.

## Programmatically Accessing Published Values

You can access **system tree objects** such as **shared or network variables** from outside the System Manager environment. Use the network path for a system tree object to access that object from another environment.

The network path to shared or network variables consists of the system name, the name of the process in which the shared variable resides, and the shared variable name: `\system\process\variable`. For example, the network path `\system1\myprocess\myvariable` identifies a shared variable named `myvariable` in the process named `myprocess` on a system named `system1`.

You can determine the network path using the **Probe** view for any system tree object. The Location field displays the network path for the selected object.



### NOTE

Refer to the LabVIEW Help, NI Measurement Studio Help, or LabWindows/CVI Help for more information about accessing shared or network variables created or modified with the System Manager from other applications.

## Saving and Restoring a System Manager Workspace

When you **customize the System Manager workspace**, you can save the customized workspace and restore it in a later System Manager session.

Complete the following steps to save and restore a customized System Manager workspace.

1. Customize the System Manager workspace.
2. Select File»Save Workspace to save your workspace as a .smw file.
3. To restore a workspace in a later session, select File»Open Workspace and navigate to the .smw file you want to restore.

## Shared or Network Variables

LabVIEW uses the term shared variable and Measurement Studio and LabWindows/CVI use the term network variable to describe software items that exist on a network and can communicate between programs, applications, remote computers, and hardware through the **Shared Variable Engine (SVE)**. Variables can represent a value or an I/O point. The SVE uses the NI Publish-Subscribe Protocol (NI-PSP) data transfer protocol to write data and allow users to read live data. NI-PSP is a proprietary technology that provides fast and reliable data transmission for large and small applications and is installed as a service on the computer when you install LabVIEW, Measurement Studio, or LabWindows/CVI.

Use the System Manager to **create, modify, or delete shared or network variables**. You also can **monitor shared or network variables** using the Watch List view.

### Shortcut Menu

Right-click a variable on the **system tree** to display the following options.

- Watch List —Creates a new **Watch List** view that contains the selected variable.
- Probe —Displays the **Variable Probe** view for the selected system.
- Edit Variable —Displays the **Shared Variable Properties** dialog box, which you can use to configure the selected variable.
- Remove Variable —Removes the selected variable from the owning process.

## Shared Variable Engine

The Shared Variable Engine (SVE) uses the NI Publish-Subscribe Protocol (NI-PSP) to enable the data transfer of **shared or network variables** through a network. The SVE manages the use and connectivity of shared or network variables on single or multiple **systems**.

### Stopping Shared Variable Engine

Select Actions»Stop Local Shared Variable Engine to shut down the SVE. You can **create, configure, and remove variables** in the System Manager after you shut down the SVE.

While the SVE is shut down, variable values do not update, and the SVE does not log alarms for variables.



#### 노트

You must use an alternate method to stop the Shared Variable Engine on Windows 7/Vista.

### Starting Shared Variable Engine

Select Actions»Start Local Shared Variable Engine to restart the SVE. The SVE is always running after you install LabVIEW, LabWindows/CVI, or Measurement Studio unless you select Actions»Stop Local Shared Variable Engine to manually shut down the SVE.



#### 노트

You must use an alternate method to start the Shared Variable Engine on Windows 7/Vista.

## Synchronizing Time for Networked Computers

Complete the following steps to set time synchronization.

1. Select View»Configure Time Servers to display the **Shared Variable Engine Time Server Configuration** dialog box.
2. Notice the Time server search order list. Any computer that is running the time synchronization service can serve as a time server or a time client. The primary time server is the first computer listed in the Time Server Search Order list. If no computer is set as a primary time server, the computer synchronizes to itself.
3. To add a computer to the Time server search order list, click the Add button to display the **Select Computer** dialog box.
4. If you know the name of the computer you want to add, you can type the name into the Host text box. If you do not know the exact name of the computer, you can browse for the computer using the network tree in the Machines list. To remove a computer from the Time server search order list, select the computer name and click the Remove button.
5. To change the order in which computers search for a time synchronization server, select the computer name and click the Up or Down button.
6. Use Sleep time (seconds) to set how long each computer waits between each synchronization. Set the primary time synchronization server sleep time to 60 seconds. If the primary server is off-line for some reason, a computer scheduled to synchronize automatically seeks out the second computer on the synchronization server list. At the time of the next synchronization, the computer first looks for the primary server before seeking a secondary synchronization server.
7. Click the OK button.
8. Repeat steps 4 through 7 for all computers on the network that you want to synchronize to make sure that the order of search for time servers is the same for all the computers, including the primary time synchronization server.

## Systems

Systems are local computers or LabVIEW Real-Time targets available through the System Manager. Systems available on your local subnetwork appear by default. Select Actions»Add System to My Systems to access systems outside your local subnetwork that do not appear in the [system tree](#).

### Shortcut Menu

Right-click a system on the system tree to display the following options.

- Add to My Systems —Moves a selected system listed in Network Items to My Systems on the system tree.
- Remove From My Systems —Removes a selected system listed in My Systems.
- Watch List —Creates a new [Watch List](#) view that contains the single [variable](#) or multiple variables contained within the system or [process](#) that you select.
- Probe —Displays the Systems Probe view, which displays the location address for the selected system.
- Add Process —Displays the Add Process dialog box.
- Register Computer — Registers the selected system.
- Unregister Computer — Unregisters the selected system.

## VI States

Requires: Real-Time Module

Select a real-time target to display a target probe in the Auto View. You also can right-click the target and select Probe to display the probe in a separate window. Click the VI States tab of a real-time target probe to display this view. Use this view to start, stop, and monitor VIs on the real-time target.

This view includes the following components:

- Port —Specifies the TCP/IP port at which the VI server listens for requests on the real-time target. Right-click the real-time target in the [Project Explorer](#) window and select Properties from the shortcut menu to display the RT Target Properties dialog box. Select VI Server from the Category list to display the VI Server page, where you can configure the TCP/IP port for the VI Server.
- Update Interval —Sets the minimum time that the server waits between updates. The actual update interval might exceed the time you specify if higher priority tasks are running on the real-time target.
- Show Top Level VIs Only —Filters all subVIs from the list of real-time target VIs.
- Start Monitoring —Starts monitoring VIs on the real-time target.
- Stop Monitoring —Stops monitoring VIs on the real-time target.

- **Start VI** —Starts an idle VI that you select from the list of real-time target VIs.
- **Stop VI** —Stops a VI that you select from the list of real-time target VIs.

## View Menu

The View menu contains the options to display certain windows in the workspace and access a system tree object that does not appear in the system tree.

- **Auto View** —Displays the Probe view for any system tree object that you select. If you select multiple objects, this view displays a Watch List that contains all of the selected objects.
- **Alarms and Events** —Displays the Alarms and Events view for any systems located within My Systems on the system tree.
- **Probe Location** —Displays the Probe Location dialog box for a system, process, or variable. You can enter the location of objects that appear in the system tree, as well as those that do not.
- **Configure Time Servers** — (DSC Module) Displays the Shared Variable Engine Time Server Configuration dialog box. You can use the dialog box to synchronize the time of machines on the network with a central time server.
- **New Watch List** —Creates a blank Watch List view. Drag a system tree object into this view to add it to the Watch List view.
- **Refresh** —Updates the contents of the system tree and Watch List views.
- **New Historical Trend** — (DSC Module) Creates a blank Historical Trend view. Drag a system tree pane object into this view to monitor the historical trend of this object.



### RELATED INFORMATION

[System Manager Overview](#) on page 1779

[System Manager Overview](#) on page 1779

[System Manager Overview](#) on page 1779

[Monitoring Variables with the Watch List View](#) on page 1771

[Alarms and Events View](#) on page 1760

[Systems](#) on page 1787

[Processes](#) on page 1778

[Shared or Network Variables](#) on page 1785

## Viewing and Clearing Faults (NI Scan Engine)

You can view and clear **faults** on a target with the **NI Scan Engine** installed. Complete the following steps to view and clear faults.



1. Click a scan-enabled target in the system tree to display a target probe in the Auto View. You also can right-click a target and select Probe to display a target probe in a separate window. The Scan Engine page of the target probe displays the faults active on the target in a table that includes the fault code and description. The Fault Present LED indicator appears black when no fault is present, orange when only **minor** faults are present, and red when **major** or **unrecoverable** faults are present.
2. Click the Clear Faults button to clear all faults on the target.
3. To ensure that you successfully cleared all faults on the target, check the Fault Present LED indicator.

## Viewing and Setting the Scan Engine Mode (NI Scan Engine)

You can use the **Scan Engine** page of a target probe to view and set the **scan engine mode** of a target with the **NI Scan Engine** installed.

Complete the following steps to view and set the scan engine mode.

1. Click a scan-enabled target in the system tree to display a target probe in the Auto View. You also can right-click a target and select Probe to display a target probe in a separate window.
2. The Scan Engine page of a target probe displays scan engine modes under the Scan Engine Enabled LED indicator and highlights the current scan engine mode with a green LED indicator.
3. Use the Change to Config button to set the scan engine mode to Config Mode or the Change to Active button to set the scan engine mode to Active Mode. You cannot enter Active Mode directly from Fault Mode. You must first **clear** all major **faults** on the target.



### NOTE

Entering Config Mode from Fault Mode automatically converts all major faults to minor faults. You cannot enter fault mode manually. The target enters fault mode only when a **major** or **unrecoverable** fault occurs.

