LabVIEW™ Connectivity 교육과정 매뉴얼

교육과정 소프트웨어 버전 2011
2011 년 12 월판
325627A-0129

전세계 기술 지원 및 제품 정보

ni.com/korea

전 세계 현지 사무소

ni.com/niglobal 을 방문하여 최신 연락 정보 , 지원 전화번호 , E- 메일주소 및 이벤트 정보를 제공하는 각 사무소의 웹 사이트에 접속할 수 있습니다 .

National Instruments 본사

11500 North Mopac Expressway Austin, Texas 78759-3504 USA 전화 : 512 683 0100

추가적인 지원 정보는 , *추가적인 정보와 리소스* 부록을 참조하십시오 . National Instruments 문서에 대한 문의사항은 National Instruments 웹 사이트의 ni.com/info 에서 정보 코드 feedback 을 입력하십시오 .

목차

5 과
UDP 를 사용하여 데이터 브로드캐스트하기 및 TCP 를 사용하여 클라이언트에 데이터 제공하기

6 과
LabVIEW 웹 서비스 사용하기

부록 A
LabVIEW Connectivity 옵션

부록 B
추가적인 정보와 리소스

1

LabVIEW 에서 공유 라이브러리 호출 하기

다른 언어로 작성한 코드를 LabVIEW 에서 다음과 같은 방법으로 호출할 수 있습니다 .

- 플랫폼 특정 프로토콜 사용하기
- 라이브러리 함수 호출 노드를 사용하여 다음 타입의 공유 라이브러리 호출 하기
  - Windows 에서 다이나믹 링크 라이브러리 (DLL)
  - Mac OS 에서 Frameworks
  - Linux 에서 Shared Libraries

이 코스에서는 라이브러리 함수 호출 노드를 사용하여 Windows 의 다이나믹 링크 라이브러리 (DLL, 공유 라이브러리) 를 호출하는 방법을 다룹니다 . LabVIEW 를 사용하여 다른 언어로 작성된 코드를 호출하는 다른 방법에 대한 자세한 내용은 *LabVIEW 도움말*을 참조하십시오 .

토픽

A. 공유 라이브러리 개요
B. 공유 라이브러리 호출
C. 공유 라이브러리 반입 마법사 사용

A. 공유 라이브러리 개요

Windows 에서는 공유 라이브러리를 DLL 이라고 부릅니다 . 공유 라이브러리는 Windows 어플리케이션에 사용할 수 있는 실행 함수 또는 데이터의 라이브러리입니다 . 공유 라이브러리는 프로그램에서 공유 라이브러리에 연결되는 정적 또는 다이나믹 링크를 만들어 접근할 수 있는 하나 이상의 특정 함수를 제공합니다 . 정적 링크는 프로그램이 실행되는 동안 일정하게 유지되며 다이나믹 링크는 프로그램에서 필요에 따라 만듭니다 . 공유 라이브러리는 2 진 파일로 저장됩니다 .

LabVIEW 가 지원하는 호출 형식인 `stdcall` 또는 `C` 중 하나를 사용하여 공유 라이브러리를 호출할 수 있다면 , 어떤 언어로 공유 라이브러리를 작성해도 무방합니다 . 예제와 문제해결 정보는 공유 라이브러리를 만들어 사용하고 LabVIEW 에서 라이브러리 함수 호출 노드를 성공적으로 구성할 수 있도록 도와줍니다 . 여기서 DLL 에 대해 설명한 일반적인 방법은 다른 타입의 공유 라이브러리에도 적용됩니다 .

LabVIEW 는 공유 라이브러리를 하나의 유일한 어플리케이션 인스턴스로 로드합니다 . 이렇듯 유일한 어플리케이션 인스턴스에서 공유 라이브러리를 열게 되면 , 공유 라이브러리 내의 VI 와 공유 라이브러리 밖에서의 VI 가 이름이 같아 충돌하게 되는 경우를 방지할 수 있습니다 .

공유 라이브러리 사용 예제는 `labview\examples\dll` 디렉토리를 참조하십시오 .

B. 공유 라이브러리 호출

이 섹션에서는 LabVIEW 에서 공유 라이브러리를 사용하는 과정을 설명합니다 .

공유 라이브러리 설명 및 정의

라이브러리 함수 호출 노드를 사용하면 대부분의 표준 공유 라이브러리를 호출할 수 있습니다 . 이 공유 라이브러리는 Windows 에서는 DLL, Mac OS 에서는 Frameworks, Linux 에서는 Shared Libraries 입니다 . 라이브러리 함수 호출 노드는 여러가지 데이터 타입과 호출 형식을 지원합니다 . 라이브러리 함수 호출 노드를 사용하여 대부분의 표준 및 사용자 정의 라이브러리에서 함수를 호출할 수 있습니다 .

DLL 의 목적 , 장점 및 제한사항

오늘날 대부분의 개발 환경은 DLL 생성을 지원합니다 .

때로는 특정한 실행에서 추가적으로 태스크를 실행해야 할 경우가 있습니다 . 이러한 경우를 위해 , 라이브러리 함수 호출 노드는 예약 , 예약해제 , 강제 종료라는 세 가지 엔트리 포인트를 제공합니다 . 예를 들어 , 라이브러리 함수 호출 노드를 예약할 때 데이터 구조를 초기화하거나 , 라이브러리 함수 호출 노드를 예약해제하거나 강제 종료할 때 프라이빗 데이터를 해제시킬 수 있습니다 . 이런 경우 , DLL 에서 미리 정의해 둔 시간에서 LabVIEW 가 호출하는 루틴을 작성하고 반출할 수 있습니다 . 엔트리 포인트는 **라이브러리 함수 호출** 대화 상자의 **콜백** 페이지에서 설정합니다 .

라이브러리 함수 호출 노드는 호출하고자 하는 기존 코드가 있거나 , 표준 공유 라이브러리를 만드는 과정을 잘 알고 있는 경우 사용하기에 가장 좋습니다 . 라이브러리는 다양한 개발 환경에서 표준이 되는 포맷을 사용하기 때문에 , 거의

모든 개발 환경을 사용하여 LabVIEW 에서 호출가능한 라이브러리를 생성할 수 있습니다 . 표준 공유 라이브러리를 생성할 수 있는지의 여부를 결정하려면 컴파일러 문서를 참조하십시오 .

LabVIEW 컴파일러를 사용하면 대부분의 프로그래밍 태스크에 충분히 빠른 코드를 생성할 수 있습니다 . LabVIEW 에서 공유 라이브러리를 호출하여 다른 언어를 사용하는 태스크를 완성할 수 있습니다 .

공유 라이브러리는 동기화되어 실행되기 때문에 해당 객체에 사용된 실행 스레드를 LabVIEW 에서 다른 태스크에 사용할 수 없습니다 .

LabVIEW 는 실행 중인 객체 코드를 차단할 수 없으므로 실행 중인 VI 가 완료될 때까지 공유 라이브러리를 리셋할 수 없습니다 . 긴 태스크를 수행하는 공유 라이브러리를 작성하려면 , 객체가 실행되는 동안 LabVIEW 에서 동일한 스레드의 다른 태스크를 수행할 수 없다는 점에 주의해야 합니다 .

공유 라이브러리 호출 방법

라이브러리 함수 호출 노드를 사용하여 Windows DLL, Mac OS Framework 또는 Linux Shared Library 함수를 직접 호출할 수 있습니다 . 이 노드를 사용하여 , LabVIEW 에서 인터페이스를 생성하고 LabVIEW 용으로 특별히 작성된 기존 라이브러리 또는 새 라이브러리를 호출할 수 있습니다 . National Instruments 는 라이브러리 함수 호출 노드를 사용하여 외부코드에 대한 인터페이스를 생성할 것을 권장합니다 .

노트 라이브러리 함수 호출 노드를 사용하거나 라이브러리 함수 호출 노드에서 호출하는 코드를 작성할 경우에는 Windows 메시지 WM_USER 에서 WM_USER+99 까지가 LabVIEW 내부에서만 사용하도록 예약되어 있다는 점에 주의해야 합니다 .

(라이브러리 함수 호출 노드) 에서 마우스 오른쪽 버튼을 클릭한 후 바로 가기 메뉴에서 **설정**을 선택하여 **라이브러리 함수 호출** 대화 상자를 디스플레이합니다 . **라이브러리 함수 호출** 대화 상자를 사용하여 라이브러리 , 함수 , 파라미터 , 반환값 및 Windows 에서의 호출 형식과 함수 콜백을 지정합니다 . **라이브러리 함수 호출** 대화 상자에서 **확인** 버튼을 클릭하면 , 사용자의 설정에 따라 라이브러리 함수 호출 노드가 업데이트되고 , 정확한 터미널 수가 표시되며 , 터미널이 정확한 데이터 타입으로 설정됩니다 .

노트 같은 컴퓨터에서 다른 LabVIEW 버전에서 생성한 어플리케이션이나 공유 라이브러리를 실행하고자 하는 경우 , 컴퓨터에 어플리케이션 또는 공유 라이브러리를 생성하는데 사용된 각 LabVIEW 의 버전과 호환되는 LabVIEW 런타임 엔진 버전이 설치되어야 합니다 .

호출 형식 설정

호출 형식은 코드에서 함수로의 정보 전달 방식을 정의합니다 . **라이브러리 함수 호출** 대화 상자에서 **함수** 탭의 **호출 형식** 컨트롤을 사용하여 함수의 호출 형식을 선택합니다 . 기본 호출 형식은 c 입니다 . c 호출 형식을 사용하면 , 가변적인 파라미터 리스트를 만들 수 있고 파라미터가 역순으로 스택에 전달됩니다 . 이때 속도가 약간 느려질 수 있습니다 .

(Windows) 표준 Windows 호출 형식인 `stdcall` 을 사용할 수도 있습니다 . `stdcall` 을 사용하면 , 함수는 파라미터를 함수 선언에 나타나는 순서대로 스택에 전달합니다 . 함수에 전달되는 파라미터의 개수는 고정되어 있습니다 .

적절한 호출 형식에 대해서는 호출할 DLL 문서를 참조하십시오 .

주의 적절하지 않은 호출 형식을 사용하면 LabVIEW 가 예상치 않게 종료될 수 있습니다 .

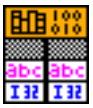
파라미터 설정

라이브러리 함수 호출 노드의 파라미터를 설정하려면 , **라이브러리 함수 호출** 대화 상자의 **파라미터** 탭으로 이동합니다 . 시작할 때에는 라이브러리 함수 호출 노드에 파라미터가 없으며 반환 타입도 **Void** 입니다 .

파라미터를 설정할 때 **함수 원형** 텍스트 박스에는 현재 만들고 있는 함수의 C 원형이 표시됩니다 . 이 텍스트 박스는 읽기 전용 디스플레이입니다 .

노트 타입 라이브러리를 찾으면 , 파라미터가 선택한 함수의 타입 라이브러리에서 찾은 파라미터와 일치하도록 업데이트됩니다 . 파라미터의 순서는 라이브러리에서 찾은 함수의 원형 (prototype) 과 일치해야 합니다 .

라이브러리 함수 호출 노드의 반환 타입은 터미널의 오른쪽 상단에서 반환됩니다 . 반환 타입이 **Void** 인 경우 , 이 상단 터미널은 사용되지 않습니다 . 각각의 추가적인 터미널 쌍은 라이브러리 함수 호출 노드의 **파라미터** 리스트에 있는 파라미터에 해당합니다 . 터미널 쌍의 왼쪽 터미널에 와이어를 연결하여 라이브러리 함수 호출 노드에 값을 전달합니다 . 터미널 쌍의 오른쪽 터미널에 와이어를 연결하여 라이브러리 함수 호출 노드를 호출한 후 파라미터의 값을 읽습니다 . 다음 그림은 **Void**, 문자열 파라미터 및 숫자형 파라미터 반환 타입의 라이브러리 함수 호출 노드를 나타냅니다 .



반환 타입

반환 타입을 지정하려면 **타입**을 **Void**, 숫자형 또는 문자열로 설정합니다 . **Void** 는 반환 타입에만 사용할 수 있으며 다른 파라미터에는 사용할 수 없습니다 . 값을 반환하지 않는 함수의 경우 **Void** 를 반환 타입으로 사용합니다 .

호출한 함수가 값을 반환하더라도 반환 타입으로 **Void** 를 사용할 수 있습니다 . 함수가 값을 반환할 때 반환 타입으로 **Void** 를 선택하면 함수에서 반환되는 값이 무시됩니다 .

노트 호출하는 함수가 C 문자열 포인터를 반환할 수 있습니다 . LabVIEW 에서 포인터는 자동으로 할당해제되지 않으므로 , 반드시 외부에서 포인터를 할당해제해야 합니다 .

팁 호출하는 함수가 리스트에 없는 데이터 타입을 반환할 경우에는 , 함수에 의해 반환된 것과 데이터 크기가 같은 반환 데이터 타입을 선택해야 합니다 . 예를 들어 , 함수가 문자 데이터 타입을 반환할 경우 8 비트 부호없는 정수를 사용해야 합니다 . LabVIEW 에는 포인터 타입이 없기 때문에 DLL 의 함수 호출은 포인터를 반환하지 않습니다 . 그러나 , 포인터와 같은 크기의 정수로 반환 타입을 지정할 수 있습니다 . 그

러면 LabVIEW 에서는 해당 주소를 단순한 정수로 처리하여 , 추후 이 주소를 DLL 호출에 전달할 수 있습니다 .

파라미터 추가 및 삭제

라이브러리 함수 호출 노드에 파라미터를 추가하려면 , **라이브러리 함수 호출** 대화 상자의 **파라미터** 탭으로 이동합니다 . **파라미터 추가** 버튼을 클릭합니다 . 파라미터를 제거하려면 , **선택한 파라미터 삭제** 버튼을 클릭합니다 . 파라미터의 순서를 변경하려면 , 파라미터 리스트의 오른쪽에 있는 **선택한 파라미터의 하나 위로 이동** 및 **선택한 파라미터의 하나 아래로 이동** 버튼을 사용합니다 .

파라미터 편집

데이터 타입이나 파라미터 이름을 편집하려면 **파라미터** 리스트에서 해당 파라미터를 선택합니다 . 선택된 파라미터의 이름을 추가적인 설명이 포함된 이름으로 편집하여 파라미터가 서로 쉽게 구별되도록 할 수 있습니다 . 파라미터 이름은 호출에 영향을 주지는 않지만 출력에 전달됩니다 . 선택된 파라미터의 **현재 파라미터**에 있는 모든 필드도 편집할 수 있습니다 .

파라미터 타입 선택

타입 풀다운 메뉴를 사용하여 각 파라미터의 데이터 타입을 나타냅니다 . 다음 파라미터 타입 중에서 선택할 수 있습니다 :

- 숫자형
- 배열
- 문자열
- 웨이브폼
- 디지털 웨이브폼
- 디지털 데이터
- ActiveX
- 타입에 적용
- 인스턴스 데이터 포인터

타입 풀다운 메뉴에서 아이템을 선택하면 파라미터 및 데이터를 라이브러리 함수에 전달하는 방법에 대한 자세한 설명을 나타내는 더 많은 아이템이 나타납니다 . 라이브러리마다 필요한 데이터 타입이 매우 다양하기 때문에 라이브러리 함수 호출 노드는 다양한 파라미터 타입용 아이템을 지원합니다 . 사용할 파라미터 타입을 결정하려면 호출한 라이브러리의 문서를 참조하십시오 .

다음 섹션에서는 **타입** 풀다운 메뉴에서 사용할 수 있는 여러 파라미터 타입을 설명합니다 .

(Windows) 공유 라이브러리의 데이터 타입 예제는 `labview\examples\dll\data passing\Call Native Code.llb` 를 참조하십시오 .

숫자형

숫자형 데이터 타입의 경우 , **데이터 타입** 풀다운 메뉴를 사용하여 정확한 숫자형 타입을 나타내야 합니다 . 숫자형 타입은 다음 데이터 타입 중에서 선택할 수 있습니다 :

- 8 비트 , 16 비트 , 32 비트 , 64 비트 및 포인터 크기의 부호있는 정수 및 부호없는 정수
- 4 바이트 단정도 숫자
- 8 바이트 배정도 숫자

포인터 크기 정수를 사용하면 , 라이브러리 함수 호출 노드는 실행되는 LabVIEW 의 버전에 맞춰 적합한 크기의 데이터를 라이브러리 함수로 전달합니다 . LabVIEW 는 64 비트와 32 비트 플랫폼에서 데이터를 표현하며 , 숫자형 데이터 타입을 32 비트 정수 타입으로 변환합니다 .

노트 확장형 정밀도 수와 복소수는 **타입** 풀다운 메뉴에서 **타입에 적용**을 선택하여 전달할 수 있습니다 . 그러나 , 일반적으로 표준 라이브러리에는 확장형 수와 복소수가 사용되지 않습니다 .

전달 방법 풀다운 메뉴를 사용하여 값을 전달할지 또는 값의 포인터를 전달할지를 나타냅니다 .

배열

데이터 타입 풀다운 메뉴를 사용하여 배열의 데이터 타입을 나타냅니다 . 숫자형 파라미터에 사용할 수 있는 타입과 동일한 데이터 타입 중에서 선택할 수 있습니다 .

차원에서 배열의 차원을 지정합니다 .

배열 포맷 풀다운 메뉴에서 다음 항목 중 하나를 선택합니다 .

- **배열 데이터 포인터** — 배열 데이터 포인터를 전달하여 , 호출된 라이브러리가 배열 데이터의 특정한 데이터 타입대로 배열 데이터에 접근할 수 있도록 합니다 .
- **배열 핸들** — 각 차원에 대한 4 바이트 값을 가리키는 포인터에 포인터를 전달한 다음 데이터를 전달합니다 .
- **배열 핸들 포인터** — 배열 핸들에 포인터를 전달합니다 .

최소 크기 컨트롤을 사용하여 LabVIEW 실행시 배열 데이터 포인터에 할당된 메모리가 적어도 **최소 크기**인지 확인하도록 합니다 . 1D 배열의 **최소 크기**를 나타내기 위해 숫자형 값을 입력할 수 있습니다 . 또는 **파라미터** 리스트에서 정수 파라미터를 설정한 경우 , 풀다운 메뉴에서 이 파라미터를 선택할 수 있습니다 . 이 옵션은 1 차원 데이터 포인터에서만 사용 가능합니다 .

노트 **최소 크기**보다 작은 배열로 전달하는 경우 , LabVIEW 는 배열의 크기를 최소 크기로 확대합니다 . 그러나 최소 크기보다 큰 배열로 전달하는 경우 , 배열은 큰 사이즈를 그대로 유지합니다 .

문자열

문자열 포맷 풀다운 메뉴를 사용하여 문자열 포맷을 나타냅니다 . 다음 문자열 포맷 중에서 선택할 수 있습니다 :

- **C 문자열 포인터 —** 다음에 널 문자가 오는 문자열 .
- **파스칼 문자열 포인터 —** 길이 바이트 뒤에 오는 문자열 .
- **문자열 핸들 —** 길이 정보를 위한 4 바이트 값을 가리키는 포인터 . 이후 문자열 데이터가 옵니다 .
- **문자열 핸들 포인터 —** 문자열 핸들의 배열 포인터 .

라이브러리 함수에서 예상되는 문자열 형식을 선택해야 합니다 . 대부분의 표준 라이브러리에서는 C 문자열 또는 파스칼 문자열이 예상됩니다 . 호출하는 라이브러리 함수가 LabVIEW 용으로 작성된 경우에는 **문자열 핸들** 형식을 사용하는 것이 좋습니다 . 파스칼 문자열 포인터를 설정할 때 , 블록다이어그램에서 값을 문자열 입력에 연결해야 합니다 . 이 값은 파스칼 문자열에 작성될 새로운 문자열을 유지할 정도의 문자로 초기화되어야 합니다 . C 문자열 포인터를 설정할 때에는 다음의 두 가지 옵션이 가능합니다 :

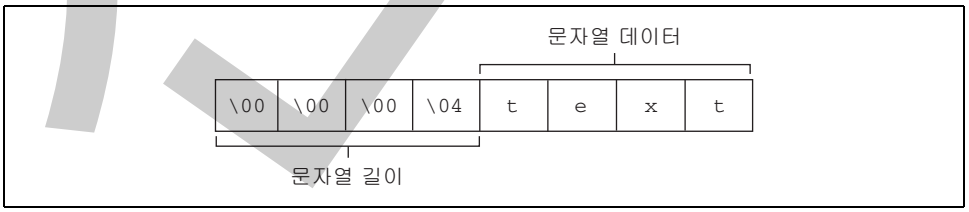
- 값을 문자열 입력에 연결합니다 . 이 때 문자열 입력은 쓰게 되는 새 문자열의 크기에 맞게 초기화됩니다 .
- **라이브러리 함수 호출** 대화 상자에서 **파라미터** 탭의 **최소 크기** 풀다운 메뉴에 문자열 크기를 지정합니다 .

최소 크기 컨트롤을 사용하여 LabVIEW 실행시 C 문자열 포인터에 할당된 메모리가 적어도 **최소 크기**인지 확인하도록 합니다 . 문자열의 **최소 크기**를 나타내기 위해 숫자형 값을 입력할 수 있습니다 . 또는 **파라미터** 리스트에서 정수 파라미터를 설정한 경우 , 풀다운 메뉴에서 이 파라미터를 선택할 수 있습니다 . 이 옵션은 C 문자열 포인터에서만 사용 가능합니다 .

노트 **최소 크기**보다 작은 문자열을 전달하는 경우 , LabVIEW 는 문자열의 크기를 최소 크기로 확대합니다 . 그러나 최소 크기보다 큰 문자열을 전달하는 경우 , 문자열의 사이즈는 그대로 유지됩니다 .

문자열 옵션

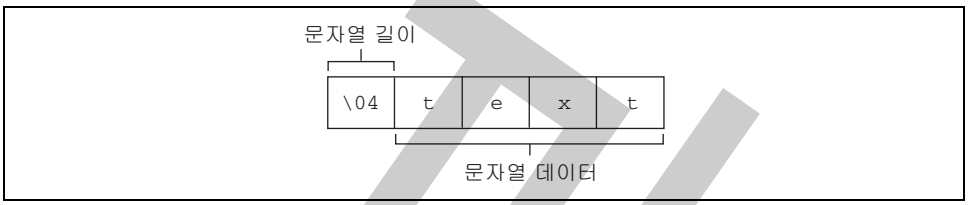
LabVIEW 는 문자열을 배열 , 즉 핸들이 포인트하는 구조로 저장합니다 . 라이브러리 함수 호출 노드는 C 및 파스칼 스타일 문자열 포인터 또는 LabVIEW 문자열 핸들에서 작동합니다 . 다음 그림은 LabVIEW 문자열 핸들의 예를 보여줍니다 .



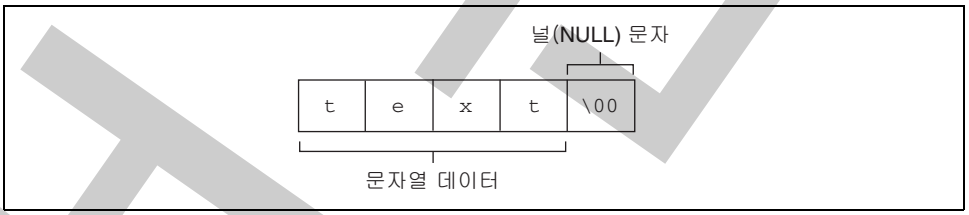
문자열을 문자의 배열로 생각하면 이해가 쉽습니다 . 문자를 순서대로 병합하면 문자열이 형성됩니다 . LabVIEW 에서는 특수한 형식으로 문자열을 저장합니다 . 이 형식은 문자 배열의 처음 4 바이트를 이용해 얼마나 많은 문자가 문자열에 나타나는지를 저장하는 부호있는 32 비트 정수를 형성합니다 . 따라서 n 개의 문자로 된 문자열을 메모리에 저장하려면 n + 4 바이트가 필요합니다 . 예를 들어 , `text` 라는 문자열에는 4 개의 문자가 들어 있는데 , LabVIEW 에서 이 문자열을 저장할 때에는 처음 4 바이트에 값 4 가 부호있는 32 비트 숫자로 들어가고 그 다

음의 4 바이트에 문자열의 문자가 들어갑니다 . 이 타입으로 문자열을 저장할 경우에는 문자열에 널 문자를 사용할 수 있다는 장점이 있습니다 . 문자열은 최대 2^{31} 개의 문자까지 가능하므로 , 길이에 거의 제한을 받지 않습니다 .

파스칼 문자열 형식은 LabVIEW 문자열 형식과 비슷하지만 , 문자열의 길이가 부호있는 32 비트 정수가 아니라 부호없는 8 비트 정수로 저장됩니다 . 따라서 파스칼 스타일 문자열의 길이는 255 문자로 제한됩니다 . 파스칼 문자의 그래픽 표시는 다음 그림과 같습니다 . 길이가 *n* 문자인 파스칼 문자열을 저장하려면 *n* +1 바이트의 메모리가 필요합니다 .



C 문자열은 가장 자주 다루어지는 문자열 타입입니다 . C 문자열을 `char *` 로 선언할 경우 , C 스타일 문자열은 C 의 일반적인 숫자형 배열과 매우 유사합니다 . C 문자열에는 LabVIEW 나 파스칼 문자열과는 다르게 문자열 길이를 나타내는 정보가 없습니다 . 그 대신 C 문자열은 다음 그림과 같이 널 (Null) 문자라는 특수 문자를 사용하여 문자열의 끝을 나타냅니다 . 널은 ASCII 문자 세트에 제로값을 갖습니다 . 여기서 이 제로 값은 문자 `0` 이 아닌 숫자 0 이라는 데 유의하십시오 .



C 에서 *n* 개의 문자를 가진 문자열을 저장하려면 모두 *n +1* 바이트의 메모리가 필요합니다 . 즉 , 문자열의 문자를 저장할 *n* 바이트에 널 (Null) 종료 문자를 위해 1 바이트가 추가됩니다 . C 스타일 문자열의 장점은 사용 가능한 메모리 용량만이 크기에 제한을 준다는 점입니다 . 그러나 , 시리얼 또는 GPIB 인스트루먼트에서처럼 숫자형 데이터를 2 진 문자열로 반환하는 인스트루먼트에서 데이터를 수집할 경우 , 문자열에 0 값이 있을 수 있습니다 . 널 (Null) 이 존재할 수 있는 2 진 데이터에서는 8 비트 부호없는 정수의 배열 또는 문자열 핸들을 사용하십시오 . 문자열을 C 스타일 문자열로 처리하면 , 인스트루먼트가 숫자값인 0 을 반환할 때 문자열의 끝에 도달한 것으로 잘못 인식할 수 있습니다 .

웨이브폼

웨이브폼 데이터 형식을 포함하는 공유 라이브러리를 호출할 경우 , **데이터 타입** 풀다운 메뉴에서 숫자 값을 지정하지 않아도 됩니다 . 기본적으로 **8 바이트 배정도** 로 지정됩니다 . 그러나 , **차원**은 지정해 주어야 합니다 . 파라미터가 단일 웨이브폼이면 , **차원**을 0 으로 지정합니다 . 파라미터가 웨이브폼 배열이면 , **차원**을 1 로 지정합니다 . LabVIEW 는 1 차원보다 높은 차원의 웨이브폼 배열은 지원하지 않습니다 .

디지털 웨이브폼

파라미터가 단일 디지털 웨이브폼이면 **차원**을 0 으로 지정하고 , 파라미터가 디지털 웨이브폼 배열이면 **차원**을 1 로 지정하십시오 . LabVIEW 는 1 차원보다 높은 차원의 디지털 웨이브폼의 배열은 지원하지 않습니다 .

디지털 데이터

파라미터가 디지털 데이터의 배열이면 **차원**을 1 로 지정하고 , 그렇지 않으면 0 으로 지정하십시오 . LabVIEW 는 1 차원보다 높은 차원의 디지털 데이터의 배열은 지원하지 않습니다 .

노트 공유 라이브러리를 사용하면 웨이브폼 , 디지털 웨이브폼 , 디지털 데이터를 전달할 수 있지만 , 공유 라이브러리에 있는 데이터에 접근할 수는 없습니다 .

ActiveX

데이터 타입 풀다운 메뉴에서 다음 아이템 중 하나를 선택합니다 :

- **ActiveX 배리언트 포인터** —ActiveX 데이터에 대한 포인터를 전달합니다 .
- **IDispatch* Pointer**—ActiveX 오토메이션 서버의 IDispatch 인터페이스에 대한 포인터를 전달합니다 .
- **IUnknown* Pointer**—ActiveX 오토메이션 서버의 IUnknown 인터페이스에 대한 포인터를 전달합니다 .

타입에 적용

타입에 적용을 사용하여 임의의 LabVIEW 데이터를 DLL 에 전달하십시오 . 임의의 LabVIEW 데이터 타입은 다음과 같은 방법으로 DLL 에 전달됩니다 :

- 스칼라는 참조에 의해 전달됩니다 . 스칼라에 대한 포인터가 라이브러리에 전달됩니다 .
- 배열과 문자열은 **데이터 포맷** 설정에 따라 전달됩니다 . 다음 **데이터 포맷** 설정 중에서 선택할 수 있습니다 :
  - **값에 의한 핸들**은 라이브러리 핸들값을 전달합니다 . 이 핸들값은 널 (Null) 이 아닙니다 .
  - **핸들의 포인터**는 핸들의 포인터를 라이브러리에 전달합니다 . 핸들이 널 (Null) 이면 핸들을 빈 문자열이나 빈 배열로 처리합니다 . 핸들이 널 (Null) 일 때 값을 설정하려면 , 새 핸들을 할당해야 합니다 .
  - **배열 데이터 포인터**는 배열의 첫 번째 원소에 포인터를 전달하여 , 호출된 라이브러리가 배열 데이터의 데이터 타입에 따라 배열 데이터에 접근할 수 있도록 합니다 .
- 클러스터는 참조에 의해 전달됩니다 .
- 배열 또는 클러스터의 스칼라 원소는 인라인 (in line) 입니다 . 예를 들어 , 숫자형이 있는 클러스터는 숫자형이 있는 구조에 포인터로 전달됩니다 .
- 배열 내의 클러스터는 인라인 (in line) 입니다 .
- 클러스터 내의 문자열과 배열은 핸들에 의해 참조됩니다 .

노트 DLL 에서 호출하려는 함수에서 LabVIEW 에 없는 타입의 파라미터가 하나 이상 존재하는 경우 , 각 파라미터가 DLL 에서 데이터를 정확히 해석할 수 있는 방법으로 함수에 전달되어야 합니다 . 라이브러리 함수 호출 노드의 현재 설정으로부터 기본이 되는 .c 파일을 만드십시오 . .c 파일을 보면 LabVIEW 가 DLL 함수와 호환되는 방법으

로 데이터를 전달할지 여부를 알 수 있습니다 . 그런 다음 필요한 대로 변경할 수 있습니다 .

인스턴스 데이터 포인터

인스턴스 데이터 포인터를 사용하여 라이브러리 함수 호출 노드의 각 인스턴스에 할당된 데이터에 접근할 수 있습니다 . **인스턴스 데이터 포인터**는 임의로 사용가능한 포인터 크기의 할당을 참조합니다 . 또한 이 할당은 **콜백** 탭의 각 콜백 함수로 전달됩니다 .

지원되지 않는 데이터 타입으로 작업하기

라이브러리 함수 호출 노드에서 전달할 수 없는 형식의 데이터를 예상하는 함수가 있을 수 있습니다 . 특히 라이브러리 함수 호출 노드는 다른 데이터의 포인터가 있는 구조나 배열 , 또는 크기가 변할 수 있는 단순 배열이 있는 구조를 지원하지 않습니다 . 지원되지 않는 데이터 타입을 예상하는 함수는 다음과 같은 방법으로 호출할 수 있습니다 :

- 데이터에 포인터가 없으면 , (패턴화된 문자열로) 함수를 사용하여 필요한 데이터의 바이너리 이미지를 포함하는 문자열을 만들고 , 이 문자열을 C 문자열 포인터로 전달할 수 있습니다 . 아마도 (패턴화된 문자열로) 의 **바이트 순서** 입력을 사용하여 데이터를 고유 바이트 순서로 패턴화되도록 지정할 것입니다 .
- LabVIEW 가 사용하는 형식으로 데이터를 허용하는 라이브러리 함수를 작성하고 , 다른 라이브러리에서 예상하는 데이터 구조를 만듭니다 . 그 후 , 이 함수에서 다른 라이브러리를 호출할 수 있으며 , 복귀하기 전에 반환된 값을 가져올 수 있습니다 . 이 함수는 블록다이어그램의 데이터를 (타입에 적용) 으로 허용할 것이므로 모든 블록다이어그램 데이터 타입을 전달할 수 있습니다 .

스레드 안전한 (Thread-Safe) DLL 및 스레드 불안전한 (Thread-Unsafe) DLL

라이브러리 함수 호출 대화 상자의 **함수** 탭에 있는 **스레드** 섹션에서 라이브러리 호출을 실행할 스레드를 선택할 수 있습니다 . 이때 스레드 옵션은 **UI 스레드에서 실행**과 **모든 스레드에서 실행**입니다 . **UI 스레드에서 실행**을 선택하면 , 라이브러리 함수 호출 노드가 현재 VI 가 실행되는 스레드에서 LabVIEW 사용자 인터페이스 스레드로 전환됩니다 . **모든 스레드에서 실행**을 선택하면 , 라이브러리 함수 호출 노드는 계속해서 현재 실행되는 스레드에 있게 됩니다 . 기본적으로 모든 라이브러리 함수 호출 노드는 사용자 인터페이스 스레드에서 실행됩니다 .

라이브러리 함수 호출 노드가 모든 스레드에서 실행되도록 설정하기 전에 , 여러 스레드가 동시에 함수를 호출할 수 있어야 합니다 . 공유 라이브러리에서 , 다음과 같은 경우 코드는 스레드 안전한 (thread-safe) 코드로 간주됩니다 :

- 글로벌 변수 , 디스크의 파일과 같은 글로벌 데이터를 저장하지 않습니다 .
- 어떠한 하드웨어에도 접근하지 않습니다 . 즉 , 코드에는 레지스터 레벨 프로그래밍이 없습니다 .
- 스레드에 안전하지 않은 함수 , 공유 라이브러리 , 또는 드라이버를 호출하지 않습니다 .
- 세마포어나 뮤텍스 (mutex) 를 사용하여 글로벌 리소스에 대한 접근을 제한합니다 .
- 재호출되지 않는 하나의 VI 에 의해서만 호출됩니다 .

재호출에 대한 자세한 내용은 *LabVIEW 도움말*의 *실행 프로퍼티 페이지* 항목을 참조하십시오 . LabVIEW 의 멀티스레딩에 대한 자세한 내용은 *LabVIEW 도움말*의 *멀티스레드 어플리케이션의 장점* 항목을 참조하십시오 .

연습문제 1-1 을 (를) 통해 이 섹션의 개념을 익힙니다 .

C. 공유 라이브러리 반입 마법사 사용

공유 라이브러리 반입 마법사를 사용하여 `.dll` 파일 (Windows), `.so` 파일 (Linux), 또는 `.framework` 파일 (Macintosh) 의 함수에 대한 래퍼 (Wrapper) VI 의 LabVIEW 프로젝트 라이브러리를 만들거나 업데이트하십시오 . **공유 라이브러리 반입** 마법사는 대부분의 C 와 C++ 헤더 파일을 지원합니다 . 마법사가 생성하는 래퍼 VI 는 라이브러리 함수 호출 노드를 사용합니다 . 이 때 사용하는 노드는 C++ `this` 포인터나 C++ 클래스에서 메소드 호출을 지원하지 않습니다 .

노트 인스트루먼트 드라이버에 대한 공유 라이브러리 파일을 반입하려면 , `ni.com` 의 인스트루먼트 드라이버 네트워크에서 LabVIEW 인스트루먼트 드라이버 반입 마법사를 다운로드받을 수 있습니다 .

공유 라이브러리 반입 마법사는 헤더 파일을 분석하고 , 공유 라이브러리에 함수를 나열하고 , 공유 라이브러리의 데이터 타입을 LabVIEW 데이터 타입으로 변환하며 , 각 함수에 대한 래퍼 VI 를 생성합니다 . 편집가능한 LabVIEW 프로젝트 라이브러리에 VI 를 저장합니다 . 또한 마법사를 종료할 때 시작할 수 있는 생성된 라이브러리에 대해 HTML 리포트를 생성합니다 .

마법사에서 경로와 선행 처리기 (preprocessor) 정의를 포함하고 , 각 함수를 감싸는 VI 를 설정하고 , 메모리 할당과 에러 처리를 설정할 수 있습니다 . 또한 마법사는 원래 함수를 구성하고 있는 각 요소에 대해 사용자 정의 컨트롤을 생성하여 , 이 컨트롤을 프로젝트 라이브러리에 추가합니다 . 사용자 정의 컨트롤을 사용하여 라이브러리에서 해당 데이터 타입을 포함하는 모든 VI 를 수정할 수 있습니다 .

한 공유 라이브러리 파일에서 마법사를 여러 번 실행할 수 있습니다 . **생성 또는 업데이트 모드 지정** 페이지의 **공유 라이브러리를 위한 VI 업데이트** 옵션을 선택하면 , 마법사가 이전 버전의 프로젝트 라이브러리 파일과 이 파일에 있는 기존의 VI 를 덮어씁니다 . 프로젝트 라이브러리 파일의 생성된 VI 를 다시 반입하지 않기로 선택하면 , 이 VI 는 디렉토리에서 변하지 않습니다 . 마법사는 공유 라이브러리의 각 함수에 대한 가장 최신 셋팅을 유지합니다 . 예를 들어 , 3 개의 함수가 있는 공유 라이브러리에서 , 두 번째 함수만을 업데이트했을 경우를 생각해봅니다 . 이 후 이 공유 라이브러리 파일에서 마법사를 실행하면 , 마법사는 첫 번째와 세 번째 함수에서는 기존의 셋팅을 , 두 번째 함수에서는 새로운 셋팅을 유지합니다 .

도구≫반입≫공유 라이브러리를 선택하여 **공유 라이브러리 반입** 마법사를 시작합니다 . 요청에 따라 공유 라이브러리 파일에 대한 래퍼 VI 를 생성합니다 . 분석하려는 마법사의 공유 라이브러리 파일과 헤더 (`.h`) 파일의 이름을 입력해야 합니다 .

공유 라이브러리 반입과 래퍼 VI 생성에 대한 단계적 설명은 *LabVIEW 도움말*의 *공유 라이브러리 파일에서 함수 반입하기* 항목을 참조하십시오 .

공유 라이브러리 반입 마법사 사용에 대한 예를 보려면 `labview\examples\dll\regexpr` 디렉토리에서 Import Shared Library Tutorial GUI VI 를 참조하십시오 .

복습 : 퀴즈

1. 다음 중 공유 라이브러리에서 함수를 호출하기 위해 알아야 할 사항은 무엇입니까 ? (복수 정답 가능)
   a. 함수가 반환하는 데이터 타입
   b. 호출 형식
   c. 공유 라이브러리를 작성한 개발 환경
   d. 함수로 전달되는 파라미터의 순서

2. 하드웨어에 접근하며 , 스레드 안전한 것으로 알려진 DLL 을 인계받았습니다 . 다음 중 사용자의 라이브러리 함수 호출 노드에 선택해야 할 설정은 무엇입니까 ?
   a. UI 스레드에서 실행
   b. 모든 스레드에서 실행

3. 다음 문장은 참입니까 , 거짓입니까 ? **공유 라이브러리 반입 마법사**에서는 공유 라이브러리 파일만이 필요합니다 .

복습 : 퀴즈의 답

1. 다음 중 공유 라이브러리에서 함수를 호출하기 위해 알아야 할 사항은 무엇입니까 ? (복수 정답 가능)

   a. 함수가 반환하는 데이터 타입

   b. 호출 형식

   c. 공유 라이브러리를 작성한 개발 환경

   d. 함수로 전달되는 파라미터의 순서

2. 하드웨어에 접근하며 , 스레드 안전한 것으로 알려진 DLL 을 인계받았습니다 . 다음 중 사용자의 라이브러리 함수 호출 노드에 선택해야 할 설정은 무엇입니까 ?

   a. UI 스레드에서 실행

   b. 모든 스레드에서 실행

3. 다음 문장은 참입니까 , 거짓입니까 ? **공유 라이브러리 반입 마법사**에서는 공유 라이브러리 파일만이 필요합니다 .

   거짓 . 공유 라이브러리 반입 마법사에서는 공유 라이브러리 파일과 헤더 파일이 모두 필요합니다 .

노트