Microsoft ROBOTICS STUDIO

로보틱스 튜토리얼 및 예제

로보틱스 튜토리얼 및 예제는 아래의 항목으로 구성됩니다.

목차

1. 하드웨어 설정

하드웨어 설정 개요

마이크로소프트 Robotics Studio 는 다양한 로보틱스 하드웨어와 같이 사용할 수 있습니다. 이 섹션에서 몇 가지 로봇 하드웨어 설정에 관한 정보를 알 수 있습니다. 추가적 하드웨어 플렛폼에 관한 정보는 마이크로소프트 Robotics Studio 파트너사, 마이크로소프트 Robotics Studio 개발자 커뮤니티 그리고 MSDN 포럼(아래에 참조를 보세요)에서 제공됩니다.

일반적으로, 마이크로소프트 Robotics Studio 가 지원하는 운영 체계가 설치된 PC 에 로봇 하드웨어 통신 설정에 다음의 사항을 따르십시오. 다음 정보는 이번 배포판 튜토리얼이 지원하는 로봇을 설정하는 데 도움을 줍니다. 로보틱스 튜토리얼 6(C#)에서-원격 접속된 로봇(Remotely Connected Robots)는 원격으로 접속된 로보틱스 하드웨어를 지원하는 서비스 개발방법을 단계별로 보여줍니다.

Fischertechnik

Robotics Studio 에 포함된 로보틱스 튜토리얼과 예제 중 몇 개는 USB 또는 RF 로 연결된 Fischertechnik 16 비트 Robo Interface(93293)로 작동됩니다(마이크로소프트 Robotics Studio 는 8 비트 Robo Interface 이하는 지원하지 않습니다).

fischertechnik Robo Interface 사용을 위해, PC 에 fischertechnik 드라이버를 설치합니다. 그리고 fischertechnik 문서에 있는 지시사항에 따라 PC 의 USB 포트로 Robo Interface 사이의 USB 연결하거나 fischertechnik RF 디바이스를 PC 와 연결하십시오. 이후 전원(적합한 배터리 팩 또는 AC 어댑터)를 Robo Interface 에 넣어주면 마이크로소프트 Robotics Studio 튜토리얼을 실행하기 위한 준비가 완료됩니다.

PC 소프트웨어와 Robo Interface 가 적절히 설치되고 PC 와 연결이 된 후 전원을 켜면, 마이크로소프트 Robotics Studio 튜토리얼을 사용할 수 있습니다. 튜토리얼은 센서와 모터를 Robo Interface 에 연결하는 방법을 기술하는 특별한 설정 파일(configuration file)을 필요로 합니다. 이 파일에 관해서 자세한 것은, 각 튜토리얼을 참조하십시오.

마이크로소프트 Robotics Studio 는 특정 로봇들의 물리적 설정을 위해 몇 개의 매니페스트를 제공합니다. 현재 Bionic Walker 과 RoboMobile 물리적 키트를 위한 설정 파일을 가지고 있습니다. 로봇이 동작하는지를 마이크로소프트 Robotics Studio 커맨드 실행창에서 다음 명령을 실행해 확인할 수 있습니다:

```
dsshost /p:50000 /t:50001 /m:"samples\config\FischerTechnik.RoboMobile.manifest.xml"
/m:"samples\config\RoboticsTutorial4.manifest.xml"
```

보다 상세한 것은 아래를 참고하세요.

- http://www.fischertechnik.de

iRobot Create 와 Roomba

로보틱스 튜토리얼 그리고 예제 일부는 시리얼 접속을 사용하는 PC 또는 Bluetooth 어댑터를 이용해 iRobot Roomba 와 Create 를 사용합니다. 마이크로소프트 Robotics Studio 를 iRobot Roomba 또는 Create 과 사용할 때 PC 와 로봇 사이에서 접속을 설정하기 위해 다음을 따라 주십시오.

로보틱스 튜토리얼을 4(C#)-Drive-By-Wire 같은 튜토리얼을 처음 시작할 때, 웹 브라우저 윈도우가 자동으로 열리고 거기서 하드웨어를 설정합니다. PC 와 접속된 COM 포트에 따라 SerialPort 값을 변경합니다. 만약 별도의 인바운드와 아웃바운드 시리얼 포트를 지정하는 Bluetooth 어댑터를 사용하고 있으면, **outbound** 포트를 사용하십시오.

로봇이 접속을 확인하기 위해, 마이크로소프트 Robotics Studio 커맨드 프롬프트로에서 다음 명령을 실행해 DSS 노드를 시작합니다:

iRobot SCI 커맨드 인터페이스로 표현된 기본 iRobot 서비스 시작:

```
dsshost /p:50000 /t:50001 /m:"samplesWconfigWiRobot.manifest.xml"
```

로보틱스 튜토리얼 4(C#)-Drive-By-Wire 의 일부인 드라이브 서비스를 실행합니다:

```
dsshost /p:50000 /t:50001 /m:"samplesWconfigWiRobot.Drive.manifest.xml"
/m:"samplesWconfigWRoboticsTutorial4.manifest.xml"
```

드라이브 서비스를 조이스틱 또는 마우스로 로봇을 조종하기 위해 사용될 수 있는 심플 대시보드 서비스를 시작합니다:

```
dsshost /p:50000 /t:50001 /m:"samplesWconfigWiRobot.DriveDashboard.manifest.xml"
```

상세한 것은, 다음 웹 사이트를 참고하십시오:

- http://www.irobot.com
- http://www.roombadevtools.com

LEGO Mindstorms RCX-Robotics Inventions 시스템

이 섹션은 LEGO Mindstorms 으로 마이크로소프트 Robotics Studio 를 사용하는 정보를 제공합니다.

LEGO RIS 2.0 소프트웨어 설치

LEGO 키트에 동봉된 CD 에 있는 RIS 2.0 소프트웨어는 윈도우 98, 윈도우 ME 와 윈도우 XP 에서 설치 및 실행됩니다. 하지만 LEGO 사용에 관한 일부 토론 포럼에 따르면, 소프트웨어가 자주 충돌되므로, 윈도우 XP 에서만 마이크로소프트 Robotics Studio 를 사용해 LEGO Mindstorms 을 프로그램하기를 강하게 권고합니다.

LEGO Mindstorms RIS 2.0 소프트웨어를 설치하고 난 후에, LEGO 에서 1~2 개의 소프트웨어 패치를 설치해야 할 것입니다.

모든 XP 시스템에서, LEGO RIS Software 시작할 때하는 PC 시스템이 재부팅 되는 버그가 있습니다. 따라서 LEGO 웹 사이트로부터 Download RIS20XPPatch.exe 패치를 설치하십시오.

Hyper-Threading(HT)를 사용하고 있는 PC 를 위해, 업데이트된 IR 타워 드라이버를 설치하십시오. 패치 파일은 또한 LEGO 웹 사이트에 찾을 수 있습니다. Tower164 패치를 실행한 뒤 스크린에 나타난 명령을 따라 설치합니다.

주의: 이 패치가 설치된 후에도 어떤 듀얼 코어 시스템들은 LEGO 소프트웨어를 실행하는데 문제가 있습니다.

LEGO RIS 2.0 소프트웨어의 첫 실행

USB 포트로 LEGO 적외선(IR)타워를 연결합니다. 하드웨어 마법사는 디바이스를 발견하고, 자동으로 설치합니다.
다음으로, 애플리케이션이 실행 중일 때 지시사항을 따를 수 있을 정도로 스피커 볼륨이 올립니다. 바탕 화면에 있는 LEGO RIS 2.0 바로가기 또는 Programs 메뉴에서 LEGO RIS 2.0 를 선택해 프로그램을 실행한 뒤 topmenu option 에서 Run 을 선택하십시오.

설치 동안, 다른 실행 프로그램으로 전환하지 않습니다. IR 타워와 RCX 브릭(brick) 통해 LEGO RCX 펌웨어의 다운로딩과 통신의 유효 범위 설정을 하게 됩니다. 자세한 정보는 LEGO 키트에 포함된 The Constructopedia 라는 소책자 4 페이지를 참고하십시오.

다음에 주의하십시오:

- RCX 브릭에 펌웨어(기본적인 LEGO 운영 체계) 다운로드를 수행하는 동안 다른 애플리케이션을 실행하지 마십시오.
- RCX 브릭에 펌웨어의 성공적 다운로드 전 몇 번의 다운로드 시도(3~4 번)는 정상입니다.
- 펌웨어의 다운로드는 대략 4 분이 걸립니다. 다운로드 수행은 RCX 브릭의 화면과 IR 타워와 녹색 LED 점멸로 확인합니다. 다운로드 실패 시 재시도 할 수 있습니다.
- 펌웨어의 다운로드를 완료한 뒤, LEGO 프로그램이 RCX 브릭에 5 개의 예제 애플리케이션을 다운로드합니다. 다운로드 실패 시 또한 재시도 할 수 있습니다.
- 만일 배터리 전압이 매우 낮으면 RCX 브릭 배터리 교체해야 하는 데, 이때 마다 펌웨어 다운로드를 다시 해야 합니다.

RIS 2.0 프로그래밍에 필요한 소프트웨어

마이크로소프트 윈도우 XP 컴퓨터에 비주얼 스투디오 Express 2005 Edition 을 다운로드하고 설치합니다. 비주얼 스투디오 설치는 마이크로소프트 .NET 2.0 을 같이 설치합니다. RIS 2.0 은 윈도우 98 을 위해 제작되었으나 윈도우 XP 를 지원합니다. 윈도우 2000 또는 윈도우 2003 Server 에서는 잘 동작하지 않습니다. 따라서 안정적인 Mindstorms RIS 2.0 개발을 위해 윈도우 XP 사용을 권고합니다.

다음으로, LEGO Mindstorms Software Development Kit(SDK) 2.5 를 설치해야 합니다. 이것은 LEGO 웹 사이트에서 무료로 다운로드할 수 있습니다.

Mindstorms SDK 가 설치된 후 RCX 브릭을 켜면, 마이크로소프트 Robotics Studio 튜토리얼을 사용할 수 있습니다. 튜토리얼은 RCX 브릭에 접속된 센서와 모터들을 기술하는 설정 파일을 필요로 합니다. 이 파일에 관한 자세한 정보는 튜토리얼을 참조하십시오.

LEGO Mindstorms 을 위한 LEGO 소프트웨어와 마이크로소프트 Robotics Studio 셋업 팁

다음은 LEGO Mindstorms 를 현재 버전의 마이크로소프트 Robotics Studio 와 사용할 때 일어날 수 있는 일입니다.

이미 알려진 문제점들:

- 몇 개의 새로운 컴퓨터는 LEGO IR 타워를 인식하지 못합니다. 예를 들면, Toshiba Protege 3500 Tablet PC 에서 IR 타워 드라이버를 설치가 불가능합니다.

- LEGO Mindstorms RIS 2.0 에 다른 관한 몇 개의 포럼 위의 기입은 듀얼 코어 프로세서인 일부 PC 에서 IR 타워로 통신하는 데 문제가 있습니다. Tower164.exe 패치는 일부의 경우를 해결해 준다는 보고가 있습니다.
- RCX 브릭이 긴 시간 동안 대기상태로 있거나 배터리를 변경할 때, LEGO 펌웨어를 다시 다운로드합니다. 패스워드를 입력하라고 할 경우에는 이름을 입력하고 엔터를 누릅니다.

적외선 통신의 문제점들

작성된 애플리케이션을 실행하고 있는 PC 와 그 명령에 따라 물리적 동작을 실행해야 하는 RCX 브릭 사이를 LEGO IR 타워는 무선 줄(생명선)로 동작합니다. 무선 연결이 끊어지면 프로그램 실행이 정지됩니다.

RCX 브릭이 PC 와 비교해 매우 단순한 컴퓨터기 때문에, 코드개발지 제약사항들을 명심할 필요가 있습니다. 예를 들면, RCX 브릭은 이벤트를 발생시키지 않습니다. 하드웨어 상태변화와 모아진 정보는 IR 타워가 폴링하여 마이크로소프트 Robotics Studio 프로그램에 의해 해석된 뒤 처리 가능한 이벤트가 됩니다. 비교적 느린 적외선 신호 형식을 사용하므로 컴퓨터에서 보내진 명령이 브릭에서 밀려서 실행됨을 경험합니다. 시험 응용 프로그램 개발에 있어 다음의 문제점이 발생가능 합니다:

- **통신 대기(Communications latency).**
  RCX 브릭에서 이벤트가 발생하고 PC 에서 확인 될 때 사이에 몇 초의 지연이 있을지도 모릅니다. 긴 지연의 경우 이벤트가 아주 소실될 수 있습니다. 직사광선과 같은 타워 간섭을 초래할 수 있는 제약 상황과 환경은 이런 경우에 중요할 수 있습니다. 합리적인 범위 내의 로봇을 유지하고, RCX 브릭과 타워 사이에서 장애물을 신호의 경로에 배치하지 않는 것이 좋습니다.
- **명령의 손실(Losing commands).**
  센서 이벤트 핸들러로부터 모터 명령을 내릴 때 가장 종종 발생했습니다.
- **센서 타입 설정(Setting the sensor type).**
  애플리케이션의 메인 폼 로드 이벤트에서 센서 타입을 설정할 것을 권고합니다.
- **대기된 명령이 계속 동작함.**
  PC 에서 어플리케이션이 멈춘 뒤에도 로봇이 대기된 명령들을 실행하기 위해 계속 작동할지도 모릅니다. 이 경우 브릭 전원을 끌 필요가 있습니다.

우리는 문제점들을 해결하려 하고 있지만, 애플리케이션을 제작과 디버그 시 이점들을 인식할 필요가 있습니다.

LEGO Mindstorms NXT

LEGO Mindstorms NXT 는 LEGO 사의 32 비트 로보틱스 키트를 사용하며 PC 와 NXT Intelligent 브릭 사이의 통신을 위해 USB 와 Bluetooth 를 제공합니다. 이 섹션은 LEGO NXT 를 마이크로소프트 Robotics Studio 에서 설정하는 방법을 기술합니다. 한번 설정이 완료된 후 LEGO NXT Combined Motor Control 예제 및 기본 로보틱스 튜토리얼들을 LEGO NXT 로 이용할 수 있습니다.

Bluetooth설정

마이크로소프트 Robotics Studio 에서 제공되는 튜토리얼들은 현재 Bluetooth 통신만을 지원합니다. 따라서 튜토리얼을 사용하기 위해 PC 에 Bluetooth 어댑터가 필요합니다.

튜토리얼 사용을 위해 NXT Intelligent Brick 에 최신 펌웨어가 설치되었는 지를 확인합니다. PC 의 Bluetooth 어댑터를 NXT 와 통신 가능하게 설정합니다. 통상 Bluetooth 소프트웨어를 이용해 PC 와 디바이스를 안전하게 접속합니다. 다음을 따라 설정하십시오.

만일 Bluetooth 접속에 문제가 있으면, 다음 단계를 수행해 보십시오:

1. Bluetooth 설정 애플릿을 엽니다. 보통 태스크 바에서 Bluetooth 아이콘을 더블 클릭하는 것에 의해 이 페이지를 열 수 있습니다.
2. Add a new connection 을 선택하거나 PC 가 범위에 있는 Bluetooth 디바이스를 탐색하도록 하십시오.
3. 디바이스가 조합 되면, 속성 또는 세부사항을 보고 어떤 COM 포트가 NXT 에 할당 되었는지 확인사십시오. 종종 두 개의 포트가 연결되면, 그 중 outbound 연결을 이용합니다.
4. 만일 Bluetooth 관리자가 COM 포트를 찾지 못하면, Bluetooth 관리자의 Ports(COM & LPT)를 보고 NXT 접속에 할당된 COM 포트를 알아보십시오.

LEGO NXT설정

마이크로소프트 Robotics Studio 가 NXT 로부터 센서 메시지를 받기 위해선, 센서를 감시하고 PC 에 메시지를 보내는 프로그램이 NXT 브릭에서 실행되어야 합니다. 이것을 NXT 'msrs' 프로그램이라고 부릅니다. 브릭에 이것을 다운로드하는 방법은 다음의 NXT 'msrs' 프로그램 설치를 보십시오.

NXT 위의 센서가 다양한 방법으로 설정될 수 있으므로 NXT 메인 프로그램은 설정 정보가 필요합니다. 이것은 NXT 서비스가 센서 설정 부분에 기술된 바와 같이 시작될 때 NXT 브릭에 작은 설정 파일을 송신하는 것으로 이루어집니다.

LEGO NXT 펌웨어는 한 개의 마스터와 슬레이브 부분을 요구합니다. Bluetooth 연결을 시작하는 쪽이 마스터가 됩니다. NXT 에 접속하는 방법은 마이크로소프트 Robotics Studio 에 NXT 연결 부분에 나와있습니다.

NXT 'MSRS' 프로그램 설치

NXT 'MSRS' 프로그램은 NXT 시각 언어로 작성되었으며, 브릭에 최적화 되어 실행됩니다. 이 프로그램은 센서를 감시하고, 변화가 생길 때 PC 로 메시지를 보냅니다. LegoNxt 서비스를 실행할 때 이 프로그램은 브릭에 자동으로 다운로드됩니다.

이 프로그램은 마이크로소프트 Robotics Studio 설치 폴더 아래의 samplesWPlatformsWLEGOWNXTWResourcesWMSRS006.rbt 입니다. msrs.rbt 를 제외한 다른 파일들은 모두 "My Block" 서브루틴입니다. 이들은 모두 참고로 고급 사용자를 위해 제공되며, 더 많은 센서 타입을 추가하고 싶어할 경우에 변경할 필요가 있을 것입니다.

- msrs.rbt
- Read3.rbt
- MOD.rbt
- GtrThan=.rbt
- LessThan=.rbt
- SendMsg.rbt
- DualThresh.rbt
- Process.rbt

센서 설정(Configuring Sensors)

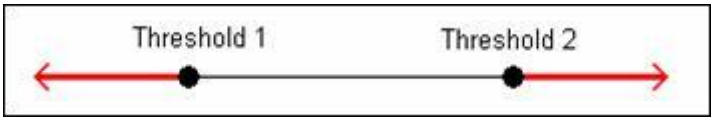
NXT 는 많은 종류의 센서 타입과 센서를 읽는 방법을 제공합니다. 마이크로소프트 Robotics Studio 는 NXT 를 이용할 때 다음의 센서들을 지원합니다:

이름	설명	허용가능 범위	허용가능 포트
Touch	Binary touch sensor	0-1	Sensor 1, 2, 3, 4
Sonar	Ultrasonic range sensor	0-250	Sensor 1, 2, 3, 4
Sound	Sound sensor	0-100	Sensor 1, 2, 3, 4
LightOn	Light sensor with red LED on	0-100	Sensor 1, 2, 3, 4
LightOff	Light sensor with red LED off	0-100	Sensor 1, 2, 3, 4
Angle	Motor shaft angle	-2147483647 to +2147483647	Motor 1, 2, 3
Encoder	Sampled down motor angle	User specified. 2-360	Motor 1, 2, 3
Button	NXT control buttons	0-1	Right, Left, and Enter Buttons
Null	No sensor connected	N/A	All

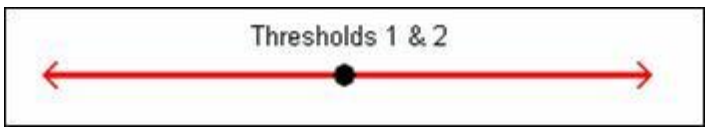
특정 센서의 종류를 특정 포트에 지정하는 것과 더불어 설정 파일은 또한 감시하거나 무시되는 센서 값의 특정 범위를 지정할 수 있습니다. 이는 내부와 외부 범위를 통해 지정되며, 각각의 임계치(threshold)들로 설정됩니다. 통상 센서의 전체 범위를 읽고 싶을 경우는 임계치들을 같게 설정합니다. 다음 그림에서와 같이 두꺼운 빨간 영역이 센서 변경을 위해 감시됩니다.

내부범위:

외부범위:



전체범위: (threshold 들이 같을때)



NXT 모터는 2 개의 센서 타입인 angle sensor 와 encoder 중의 하나로 설정될 수 있습니다. (물론 'null' 타입 또한 이용 가능합니다.). 모터가 angle sensor 로 설정되면 표적 범위는 다른 센서와 같습니다. 모터의 각도는 360°가 지난 경우에도 리셋 되지 않을 것입니다(-2147483647~+2147483647). 모터가 encoder 로 설정되면 해상도(회전당 틱수(ticks))를 설정할

필요가 있습니다. 이 값은 2 부터 360 사이로 설정합니다. 360 을 일정하게 나눌 수 있는 값을 사용하는 것이 바람직합니다. 이 값이 너무 높으면, 최대 속도에서 NXT 는 일부 encoder 정보가 손실될 수 있습니다. 해상도 6 이 NXT encoder 에 가장 적합합니다.

이 센서 설정은 다양하게 이루어집니다. 가장 손 쉬운 방법은 인터넷 탐색기(Explorer)입니다. LegoNxtBrickService(혹은 이를 파트너로 가질 경우)를 처음 시작하면, 멋진 웹 형식으로 된 COM 포트 설정과 센서 설정을 위한 LegoNxtBrickService 의 상태 페이지가 IE 를 통해 열립니다.

LegoNxtBrickService가 시작할 때마다 브릭에 설정 파일을 송신하지는 않습니다. COM 포트가 0 인경우 서비스는 LegoNxtBrickService 상태 페이지 에서 COM포트를 설정할 때까지 기다립니다. 매니페스트와 유효한 COM 포트(Tutorials 3과 4와 같이) 설정 파일을 가지고 서비스를 시작하면, 설정 파일은 이미 브릭에 있다고 가정하여 다시 보내지지 않습니다. 웹 브라우저의 LegoNxtBrickService 상태 페이지에서 설정 파일을 재전송할 수 있습니다.

고급 사용자를 위해: 네이티브 NXT 프로그램을 변경하고자 할 경우, 설정 파일에 관한 자세한 사항은 부록- 설정 파일 문서를 보십시오.

마이크로소프트 Robotics Studio 에 NXT 연결 (Connecting NXT to Microsoft Robotics Studio)

LEGO NXT에 접속하기 위한 절차가 이전의 CTP이후 단순화되었습니다. PC와 NXT 브릭 사이의 Bluetooth 접속은 PC에 설치된 Bluetooth 소프트웨어를 사용합니다.

블루투스 접속 과정에서 핀을 NXT 에서 선택하고, PC 에 같은 핀을 입력하게 됩니다.

이 과정이 성공적이면, 화면에서 다음과 유사한 창을 보게 됩니다. 이때 COM 포트를 기억하십시오.

다음 단계로 LEGO® NXT Brick(LegoNxt) 서비스를 시작합니다.

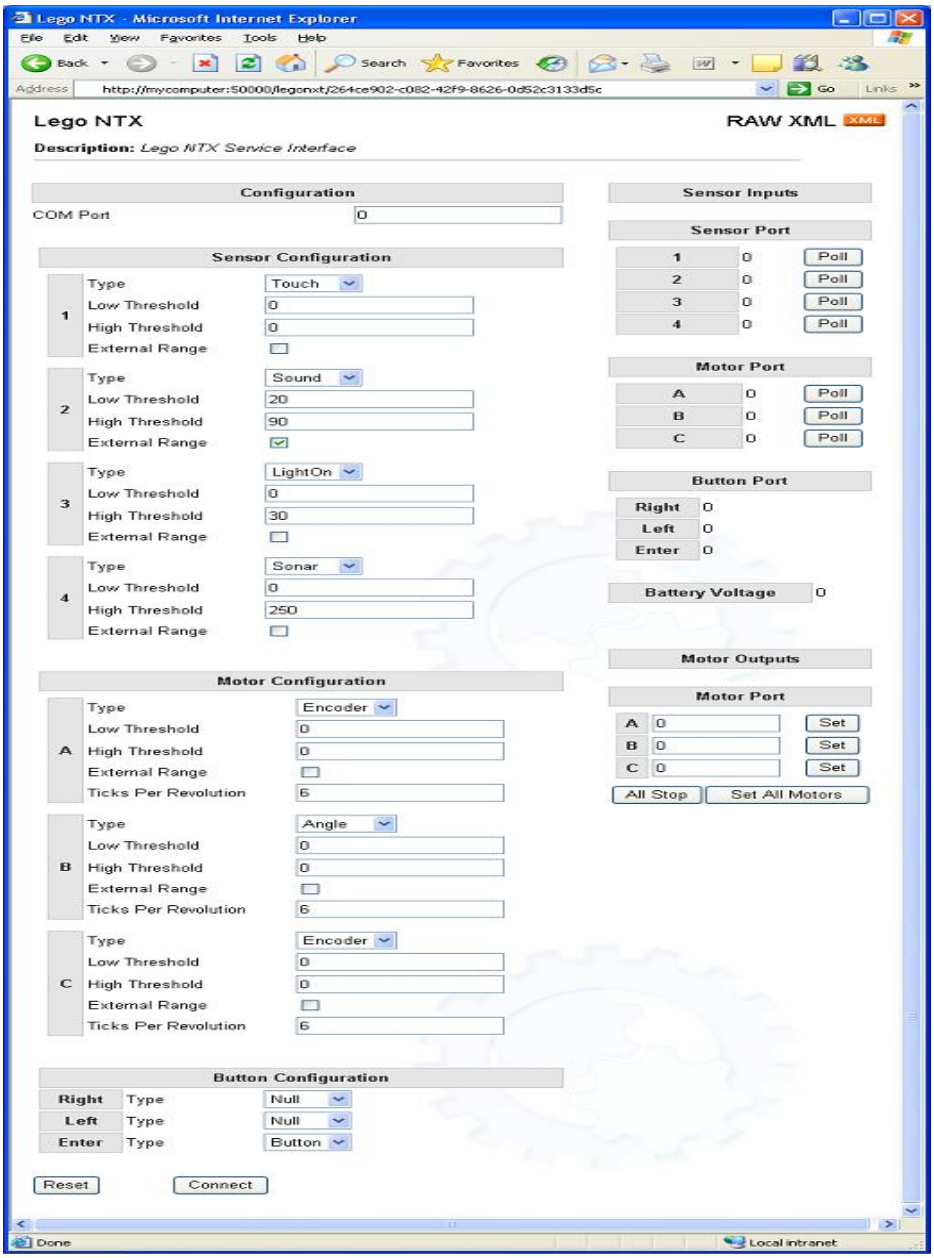
마이크로소프트 Robotics Studio 커맨드 프롬프트를 열어 다음을 입력해 LEGO NXT 매니페스트를 시작합니다:

```
dsshost /p:50000 /t:50001 /m:"samplesWConfigWLEGO.NXT.Brick.manifest.xml"
```

MSRS 프로그램이 NXT 에 전송될 때 콘솔은 설치 진행을 표시할 것입니다. 만약 서비스가 잘못된 COM 포트로 설정된 경우(0 과 같이) 인터넷 익스플로러가 열리며 여기서 알맞은 포트를 설정합니다. 브라우저가 열리지 않는 경우, 콘솔에서 URL 정보를 확인하고 웹브라우저를 엽니다: http://mycomputer:50000/legonxt/264ce902-c082-42f9-8626-0d52c3133d5c 과 유사하며 컴퓨터

이름과 /legonxt/ 뒤의 GUID 가 다릅니다. 웹브라우저는 다음 그림의 화면을 보여줄 것이며, 여기서 COM 포트와 센서를 설정한 뒤 Connect 를 누릅니다. 이것은 config.txt 파일을 레고 NXT 에 전송합니다.

웹 페이지는 또한 당신의 NXT robot.를 디버그하는 좋은 방법입니다. 쉽게 NXT의 모든 센서와 모터 값을 볼 수 있습니다.



메인 프로그램이 실행 중, 당신은 다음 화면을 볼 것입니다:

이 화면은 NXT 가 PC 로부터 설정 파일 또는 시작 명령을 받기 위해 기다리고 있는 것입니다.

만일 당신의 서비스가 유효한 COM 포트로 시작되면 단지 시작 메시지가 NXT 에 전송되며, NXT 는 이미 설정 파일을 가지고 있어야 합니다. 만일 Bluetooth 포트를 변경하거나, 로봇을 변경하면 로봇에 업데이트된 config.txt 파일을 전송해야 합니다. 이를 위해서는 서비스를 시작하고 상태 웹페이지에서 설정을 변경한 뒤 "Connect"를 누릅니다.

모든 것이 성공적이면, NXT 가 삑 하고 비프음을 울릴 것이며, 모래시계가 없어집니다. 이제 NXT 브릭은 설정 파일의 매개 변수에 따라 메시지를 전송하기 시작할 것입니다.

메인 프로그램이 실행되지 않거나 Bluetooth 경로가 틀린 방향의 설정된 경우 같이 만약 어떤 것이 잘못 설정된 경우 콘솔에서 오류 메시지를 볼 것입니다. 다음의 문제 해결방법을 따르십시오.

문제 해결방법

- 실행할 때마다 서비스가 NXT 에 접속하지 못합니다.
  - NXT 초기화의 부분 중 설정 파일을 brick 에 전송할 때 에러가 많습니다. 센서 설정을 변경는 것이 아니면 이 단계는 안전하게 스킵 될 수 있습니다. 단순히 서비스를 정상 COM 포트로 시작하면, LegoNxtService 는 설정파일을 전송하지 않게 됩니다.
  - NXT 와 컴퓨터가 연결된 후 몇 초를 기다린 뒤 서비스를 시작하는 것이 좀더 안정적입니다.
- 모터 명령은 전송되나, 센서 메시지를 수신할 수 없습니다.
  - NXT 설정 파일이 잘못된 경우일 수 있습니다. (예를 들면 모든 센서가 null 로 설정됨). 이경우 새로운 설정 파일을 브릭에 보내야 합니다.
- PC 부터 NXT 로 Bluetooth 접속할 수 없습니다
  - Bluetooth 서비스를 사용안함(disable) 설정한 뒤 다시 사용함(enable)으로 설정해 보십시오.

  - PC 를 다시 시작해 보십시오.
  - NXT 와 Bluetooth 접속정보를 지워 보십시오.
  - Bluetooth 동글을 사용하고 있으면 동봉된 CD 에 있는 드라이버를 사용하십시오. 이미 드라이버가 설치된 경우 이를 삭제하고 다시 설치해야 합니다.
- 기타 다른 동작에 문제가 있습니다.
  - 비주얼 스투디오의 출력창을 보십시오. 다른 서비스의 오류일 가능성이 있습니다.
  - 마이크로소프트 Robotics Studio Hardware Configuration and Troubleshooting 포럼에 질문하십시오.

부록- 설정 파일 문서

NXT 그래픽의 언어의 단순한 읽기 파일 조작 때문에, 설정 파일이 매우 엄격한 구조를 가집니다. NXT 는 한번에 한 라인을 해석합니다. 파일의 구조는 다음과 같습니다:

- Sensor port 1 sensor type
- Sensor port 1 threshold 1
- Sensor port 1 threshold 2
- Sensor port 2 sensor type
- Sensor port 2 threshold 1
- Sensor port 2 threshold 2
- Sensor port 3 sensor type
- Sensor port 3 threshold 1
- Sensor port 3 threshold 2
- Sensor port 4 sensor type
- Sensor port 4 threshold 1
- Sensor port 4 threshold 2
- Motor port 1 sensor type
- Motor port 1 threshold 1
- Motor port 1 threshold 2
- Motor port 2 sensor type
- Motor port 2 threshold 1
- Motor port 2 threshold 2

- Motor port 3 sensor type
- Motor port 3 threshold 1
- Motor port 3 threshold 2
- Right button sensor type
- Left button sensor type
- Center button sensor type

쓰는 시점에서, NXT 그래픽 언어는 텍스트 입력을 가진 서브루틴을 지원하지 않습니다. 따라서 모든 센서 이름은 번호로 나타납니다. 다음은 가능한 센서의 종류, 센서 번호, 범위와 포트의 목록입니다.

번호	이름	설명	허용가능 범위	허용가능 포트	문법
0	Null	Sensor not connected	N/A	All	0 0 0
1	Angle	Motor shaft angle	-2147483647 to +2147483647	Motor 1, 2, 3	1 threshold 1 threshold 2
2	Button	NXT control buttons	0-1	Right, Left, Enter Button	2
3	Encoder	Sampled down motor angle	User specified. 2-360	Motor 1, 2, 3	3 0 Ticks per revolution
4	LightOn	Light sensor with LED on	0-100	Sensor 1, 2, 3, 4	4 threshold 1 threshold 2
5	Sonar	Ultrasonic range sensor	0-250	Sensor 1, 2, 3, 4	5 threshold 1 threshold 2
6	Sound	Sound sensor	0-100	Sensor 1, 2, 3, 4	6 threshold 1 threshold 2
7	Touch	Binary touch sensor	0-1	Sensor 1, 2, 3, 4	7 0 0
8	LightOff	Light sensor with LED off	0-100	Sensor 1, 2, 3, 4	4 threshold 1 threshold 2

NOTE: 내부 범위를 지정하기 위해 임계치(threshold) 1 은 임계치 2 보다 작아야 합니다. 외부범위 지정을 위해 임계치 1 은 임계치 2 보다 더 커야 합니다. NXT 에 파일을 전송하기 전에 ParseConfig 함수에서 자동적으로 살펴집니다.

예를 들어, 다음과 같이 설정할 경우:

- port 1 에 touch sensor
- port 2 에 sonar sensor 를 그리고 30cm 보다 작을 때 알림 받기 원함
- port 3 에 light sensor 를 그리고 50 에서 60 사이 값일 때 알림 받기 원함
- port 4 에 없음
- motor 1 이 angle sensor 이고 100 에서 200 사이 범위가 아닐 때(외부범위) 알림 받기 원함
- motors 2 와 3 을 회전당 6 틱수인 encoder 로 설정
- NXT 중간 버튼만 듣기를 원함

설정 파일은 다음과 같습니다:

```
7
0
0
5
0
30
4
50
60
0
0
0
1
200
100
3
6
0
3
6
0
0
0
2
```

Output:

NXT 의 센서가 일정 범위 안에 있고 값이 변화되면 메시지가 PC 로 전송됩니다. 메시지의 형식은 다음과 같습니다

```
<Sensor Type> <Sensor Port> <Value>
```

이 데이터는 10 진의 ASCII 형식으로 전송 됩니다.

LEGO NXT 를 이용한 모터 제어 예제

이 튜토리얼은 세 개의 LEGO NXT 모터를 제어하는 서비스를 만드는 방법을 설명합니다.

시작하기

Robotics Studio 커맨트 프롬프트에서 다음과 같이 MotorTest 프로젝트를 만듭니다:

```
cd samples
dssnewservice /s:MotorTest
cd MotorTest
MotorTest.csproj
```

단계 1: 매니페스트 변경

만들어진 Motortest.manifest.xml 을 열어 그 내용을 아래와 같이 수정합니다.

1. 모터와 레고를 위한 xmlns 를 추가합니다:

```
xmlns:motor=http://schemas.microsoft.com/robotics/2006/05/motor.html
xmlns:leghttp://schemas.microsoft.com/robotics/2006/05/legonxt.html
```

2. 끝 부근 </CreateServiceList>위에 LEGO.NXT.Tribot.manifest.xml 에서 LEGOMotorSample 과 CreateMotorTestManifest 로 참조된 부분을 아래와 같이 추가합니다:

```
<!--Start LegoNXT Brick -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/robotics/2006/05/legonxt.html</dss
p:Contract>
        <dssp:PartnerList>
          <!--Initial LegoNXT config file -->
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.Brick.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
```

```
        <Name>lego:Brick1</Name>
      </ServiceRecordType>
```

브릭을 위해 <Name>lego:Brick1</Name>이 사용됨에 주의하십시오.

3. 다음 부분을 추가합니다:

```
<!-- Start Lego Motor A -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html</dssp:Co
ntract>
        <dssp:PartnerList>
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.MotorA.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
          <dssp:Partner>
            <dssp:Name>lego:Brick1</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>motor:MotorA</Name>
      </ServiceRecordType>
```

- 서비스 컨트랙트 확인하기:

```
<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html
</dssp:Contract>
```

- 서비스 이름:

```
<Name>motor:MotorA</Name>
```

- 설정 파일 정의:

```
LEGO.NXT.MotorA.Config.xml
```

- Brick1에 연결됨을 알려주기:

```
<dssp:Partner><dssp:Name>lego:Brick1</dssp:Name></dssp:Partner>
```

4. 위 단계 3 을 다른 두 모터 부분에서 적용합니다:

```
<!-- Start Lego Motor B -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html</dssp:Co
ntract>
        <dssp:PartnerList>
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.MotorB.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
          <dssp:Partner>
            <dssp:Name>lego:Brick1</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>motor:MotorB</Name>
      </ServiceRecordType>

      <!-- Start Lego Motor C -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html</dssp:Co
ntract>
        <dssp:PartnerList>
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.MotorC.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
          <dssp:Partner>
            <dssp:Name>lego:Brick1</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>motor:MotorC</Name>
      </ServiceRecordType>
```

5. <ServiceRecordType>위의...motortest.html 로 가서 모터에 대한 partnerlist 부분을 추가합니다:

```
<ServiceRecordType>
```

```
<dssp:Contract>http://schemas.tempuri.org/2006/10/motortest.html</dssp:Contrac
t>
          <dssp:PartnerList>
            <dssp:Partner>
              <dssp:Name>motor:MotorA</dssp:Name>
            </dssp:Partner>
            <dssp:Partner>
              <dssp:Name>motor:MotorB</dssp:Name>
            </dssp:Partner>
            <dssp:Partner>
              <dssp:Name>motor:MotorC</dssp:Name>
            </dssp:Partner>
          </dssp:PartnerList>
        </ServiceRecordType>
```

여러분의 매니페스트는 Motortest.manifest.xml 와 같을 것입니다:

```
<Manifest

    xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html"
    xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html"
    xmlns:motor="http://schemas.microsoft.com/robotics/2006/05/motor.html"
    xmlns:lego="http://schemas.microsoft.com/robotics/2006/05/legonxt.html"
    >
    <CreateServiceList>
        <ServiceRecordType>

<dssp:Contract>http://schemas.tempuri.org/2006/10/motortest.html</dssp:Contract>
          <dssp:PartnerList>
            <dssp:Partner>
              <dssp:Name>motor:MotorA</dssp:Name>
            </dssp:Partner>
            <dssp:Partner>
              <dssp:Name>motor:MotorB</dssp:Name>
            </dssp:Partner>
            <dssp:Partner>
              <dssp:Name>motor:MotorC</dssp:Name>
```

```
            </dssp:Partner>
          </dssp:PartnerList>
        </ServiceRecordType>


      <!--Start LegoNXT Brick -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/robotics/2006/05/legonxt.html</dssp:Contract
>
        <dssp:PartnerList>
          <!--Initial LegoNXT config file -->
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.Brick.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>lego:Brick1</Name>
      </ServiceRecordType>


      <!-- Start Lego Motor A -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html</dssp:Contract>
        <dssp:PartnerList>
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.MotorA.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
          <dssp:Partner>
            <dssp:Name>lego:Brick1</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>motor:MotorA</Name>
      </ServiceRecordType>
```

```
      <!-- Start Lego Motor B -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html</dssp:Contract>
        <dssp:PartnerList>
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.MotorB.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
          <dssp:Partner>
            <dssp:Name>lego:Brick1</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>motor:MotorB</Name>
      </ServiceRecordType>


      <!-- Start Lego Motor C -->
      <ServiceRecordType>

<dssp:Contract>http://schemas.microsoft.com/2006/06/legonxtmotor.html</dssp:Contract>
        <dssp:PartnerList>
          <dssp:Partner>
            <dssp:Service>LEGO.NXT.MotorC.Config.xml</dssp:Service>
            <dssp:Name>dssp:StateService</dssp:Name>
          </dssp:Partner>
          <dssp:Partner>
            <dssp:Name>lego:Brick1</dssp:Name>
          </dssp:Partner>
        </dssp:PartnerList>
        <Name>motor:MotorC</Name>
      </ServiceRecordType>

    </CreateServiceList>

</Manifest>
```

단계 2: 명령문으로 참조 추가

다음 파일에 대한 프로젝트 참조를 추가합니다:

- Legonxt.y2006.m05.proxy
- Legonxtservices.y2006.m06.proxy
- RoboticsCommon.proxy

Motortest.cs 를 열어 아래의 using 명령문을 추가합니다:

```
using lego = Microsoft.Robotics.Services.LegoNxt.Proxy;
using motor = Microsoft.Robotics.Services.Motor.Proxy;
using legomotor = Microsoft.Robotics.Services.LegoNxt.Motor.Proxy;
```

단계 3: 파트너 추가

다음과 같이 _mainPort 아래에 partner 를 추가합니다:

```
private MotorTestOperations _mainPort = new MotorTestOperations();


        [Partner(motor.Contract.Identifier + ":MotorA",
            Contract = legomotor.Contract.Identifier,
            Optional = false,
            CreationPolicy = PartnerCreationPolicy.UsePartnerListEntry)]
        private motor.MotorOperations _motorA = new motor.MotorOperations();


        [Partner(motor.Contract.Identifier + ":MotorB",
            Contract = legomotor.Contract.Identifier,
            Optional = false,
            CreationPolicy = PartnerCreationPolicy.UsePartnerListEntry)]
        private motor.MotorOperations _motorB = new motor.MotorOperations();


        [Partner(motor.Contract.Identifier + ":MotorC",
            Contract = legomotor.Contract.Identifier,
```

```
            Optional = false,
            CreationPolicy = PartnerCreationPolicy.UsePartnerListEntry)]
        private motor.MotorOperations _motorC = new motor.MotorOperations();
```

파트너 "name" 파라미터가 매니페스트와 일치하는지 확인합니다. motor.Contract.Identifier + ":MotorA" 는 매니페스트의 <Name>motor:MotorA</Name>와 일치되는 문자열을 만듭니다.

확장자 cs 파일에 있는 motor.Contract.Identifier 는 Roboticscommon.proxy 에 정의된 모터 컨트랙트를 참조합니다:

```
public const string Identifier = "http://schemas.microsoft.com/robotics/2006/05/motor.html";
```

이것은 매니페스트 맨 앞에 정의된 motor 와 일치합니다:

```
xmlns:motor="http://schemas.microsoft.com/robotics/2006/05/motor.html"
```

단계 4: Get 조작 변경

이제 웹 브라우저를 사용해 다른 모터로 바꾸기 위해 Get 조작을 변경하십시오:

1. Motortesttypes.cs 를 열고, Counter 를 상태에 추가합니다:

```
[DataContract()]
    public class MotorTestState
    {
        [DataMember]
        public int Counter;
    }
```

2. Motortest.cs 를 엽니다. Get 핸들러에서 서비스 상태를 변경할 것이므로 Exclusive 로 설정 하십시오:

```
[ServiceHandler(ServiceHandlerBehavior.Exclusive)]
public virtual IEnumerator<ITask> GetHandler(Get get)
```

3. Get 핸들러에 아래 내용을 추가합니다:

```
public virtual IEnumerator<ITask> GetHandler(Get get)
        {
            _state.Counter++;
            if (_state.Counter == 1)
            {
                _motorA.SetMotorPower(new motor.SetMotorPowerRequest(1.0));
            }
            else if (_state.Counter == 2)
            {
                _motorA.SetMotorPower(new motor.SetMotorPowerRequest(0.0));
                _motorB.SetMotorPower(new motor.SetMotorPowerRequest(1.0));
            }
            else if (_state.Counter == 3)
            {
                _motorB.SetMotorPower(new motor.SetMotorPowerRequest(0.0));
                _motorC.SetMotorPower(new motor.SetMotorPowerRequest(1.0));
            }
            else if (_state.Counter == 4)
            {
                _motorC.SetMotorPower(new motor.SetMotorPowerRequest(0.0));
                _state.Counter = 0;
            }

            get.ResponsePort.Post(_state);
            yield break;
        }
```

실행해 보기

NXT 를 켜고 PC 와 연결합니다.

비주얼 스투디오에서 F5 를 누릅니다.

주의사항: 처음 웹 브라우저가 열리면 LEGO NXT 를 올바른 포트에 연결합니다. 이를 수행하고 서비스를 종료합니다.

커맨트 프롬프트에서 Motortest 디렉터리에 있는 새로 생성된 Motor config 파일을 찾습니다:

```
dir lego.nxt.motor*10/19/2006   07:00 AM   848   LEGO.NXT.MotorA.Config.xml
10/19/2006   07:07 AM   848   LEGO.NXT.MotorB.Config.xml
10/19/2006   07:07 AM   848   LEGO.NXT.MotorC.Config.xml
```

여기서 Motor B 와 Motor C 의 identifier 를 2 와 3 으로 바꿉니다:

```
notepad LEGO.NXT.MotorB.Config.xml
    <Identifier>2</Identifier>

notepad LEGO.NXT.MotorC.Config.xml
    <Identifier>3</Identifier>
```

서비스를 다시 실행하고 LEGO 에서 비프음이 들리면 웹 브라우저에서 http://localhost:50000/motortest 를 열고 모터를 바꾸기 위해 여러 번 새로고침을 해 봅니다.

KHR-1 서비스 예제

KHR-1 서비스는 Kondo KHR-1 로봇에 DSS 서비스 인터페이스를 제공합니다.

단계 1: 예제의 시작과 실행

시작 > 모든 프로그램 메뉴로에서 마이크로소프트 Robotics Studio 커맨드 프롬프트를 시작하십시오.

DssHost 노드를 시작하고, 서비스의 인스턴스를 다음 명령을 입력해 만듭니다:

```
dsshost /p:50000 /m:"samples₩config₩khr1service.manifest.xml"
```

서비스를 시작한 뒤 다음과 같은 응답을 보게 됩니다:

```
Initializing Dss Node with manifesfile:///C:/mri/config/khr1service.manifest.xml
```

단계 2: 서비스 조사

인터넷 익스플로러(IE)를 시작하고, Address 바에서 다음 URI 를 입력합니다:

```
http://localhost:50000/khr1
```

이것은 KHR1 서비스를 위해 웹 페이지를 표시합니다. 서비스 상태의 XML 표현을 보기 위해, 다음 URI 를 사용하십시오:

```
http://localhost:50000/khr1/raw
```

단계 3: 서로 다른 KHR1 서비스 상태 요소들 이해하기

KHR1 서비스 상태는 다음 구성 요소를 포함합니다:

- KHR1 에 접속하기 위해 사용되는 COM 포트.
- 2 개의 서보 제어 보드 상태 각각의 목록:
    - ID.
    - 스피드(Speed).
    - 12 개의 채널(각각의 위치 상태).

단계 4: KHR1 서비스 사용

KHR1 서비스는 서브스크립션을 지원하지 않습니다.

KHR1 서비스에 다음 메시지를 보낼 수 있습니다:

- PlayMotion- 미리 설정된 움직임을 재생합니다.
- SetServoPosition-1 개의 제어 보드에 서보 스피드와 위치를 설정합니다.

단계 5: KHR1 웹 인터페이스 사용

KHR1 서비스는 웹 인터페이스를 이용해 서보 위치를 변경하고, 움직임을 재생하고, 서보를 때때로 돌리고, 홈 위치를 설정하고, 현재 서보 위치를 읽는 것을 허락합니다.

현재의 서보 위치를 변경합니다:

1. 변경할 속도를 지정하십시오. 0 에서 7 사이 값으로 0 이 빠르고 7 이 느림
2. 서보 위치를 필요에 따라서 설정하십시오. 서보 위치는 0 과 180 사이 값으로 표현됩니다.
3. Change 버튼을 누르십시오. 해당되는 서보 보드에 이 정보를 보냅니다.

움직임을 재생합니다:

텍스트 박스에서 현재 재생할 움직임(Motion)을 입력하고 Play 버튼을 클릭합니다.

서보를 켜고 끕니다.:

Sleep 와 Motion 버튼은 각각 서보를 켜고 끕니다. 로봇이 사용 중이 아닐 때 배터리 전원 보존에 유용합니다.

홈 위치를 설정합니다:

홈 버튼을 누르는 것은 KHR1 의 현재의 서보 위치를 홈 위치로 저장하게 합니다. 전원을 켜게 될 때 이 위치를 홈 위치로 가정합니다.

현재의 서보 위치를 읽습니다:

Read 버튼을 현재의 서보 위치를 읽기 위해 클릭하십시오. 서비스의 상태가 현재의 서보 위치와 동기 되어 있지 않을 때 유용할 수 있습니다.

단계 6: 서보 할당 이해하기

Kondo KHR1 의 서보 할당은 로봇의 조립에 의존합니다. Kondo 에 의해 제공된 명령을 따르면, 서보는 다음과 같이 할당 됩니다:

보드	채널	할당
0	1	Left Shoulder

보드	채널	할당
0	2	Left Elbow
0	3	Left Wrist
0	4	unused
0	5	unused
0	6	Head
0	7	Right Shoulder
0	8	Right Elbow
0	9	Right Wrist
0	10	unused
0	11	unused
0	12	unused
1	1	Left Pelvis
1	2	Left Hip
1	3	Left Knee
1	4	Left Ankle
1	5	Left Foot
1	6	unused
1	7	Right Pelvis
1	8	Right Hip
1	9	Right Knee
1	10	Right Ankle
1	11	Right Foot
1	12	unused

MobileRobots Pioneer P3DX

P3DX 로봇이 마이크로소프트 Robotics Studio 를 직접 실행하기 위한 올바른 환경을 가지는 지를 확인합니다. P3DX 로봇이 마이크로소프트 Robotics Studio 실행할 수 있는지 확인하기 위해 DSS Deploy 툴(DssDeploy.exe)을 사용해 로보틱스 튜토리얼 5 (C#)에 기술된 Explorer 서비스를 자동실행 배포(deploy) 패키지로 만듭니다.

배포 패키지를 만들기 위해 마이크로소프트 Robotics Studio 커맨드 프롬프트에서 다음을 실행하십시오:

```
dssdeploy /p /m:"samplesWconfigWexplorer.manifest.xml" ExplorerDeployment.exe
```

생성된 배포 패키지 파일(ExplorerDeployment.exe)를 로봇에 내장된 PC 에 복사합니다.

DOS 창에서 다음과 같이 배포 패키지를 설치하십시오:

```
dssdeploy /u /t:"C:WMSRS" ExplorerDeployment.exe
```

이 명령은 로봇에 서비스를 실행하는 데 필요한 파일들을 C: WMSRS 디렉터리에 설치합니다.

P3DX 로봇에서 Explorer 서비스 실행을 위를 다음명령을 DOS 창에 입력합니다. 빠른 실행을 위해 다음을 배치파일로 만들어 두면 편리합니다.

```
cd /d C:WMSRS
dsshost /p:50000 /t:50001 /m:"samplesWconfigWexplorer.manifest.xml"
```

SICK Laser Range Finder 는 시작 하는데 시간이 필요합니다. 1,2 분 이후 로봇이 자동으로 방주위를 네비게이션하는 것을 볼 수 있습니다.

2. 기본 로보틱스 튜토리얼

로보틱스 튜토리얼 1(C#) - 서비스 접근하기

마이크로소프트 Robotics Studio 를 사용해 애플리케이션을 만드는 것은 서비스들 사이의 입출력을 오케스트레이션하는 것입니다. 서비스는 소프트웨어 또는 하드웨어에 인터페이스를 표현하며 특정 기능을 수행하는 프로세스들 사이의 통신을 가능하게 합니다.

이 튜토리얼은 컨택트 센서(범퍼로 이 튜토리얼의 안에서 언급했습니다)의 출력을 읽고, 콘솔 창에 메시지를 표시하는 기본 서비스를 어떻게 사용하는지를 가르칩니다.

그림 1 - 단순한 범퍼 서비스

이 튜토리얼에서 만드는 서비스는 범퍼를 감시하여 눌려지면 콘솔에 정보를 표시합니다. 이 튜토리얼은 알림 메시지를 수신하기 위해 범퍼 센서를 연결하고 등록하는 것을 가르칩니다. 이 튜토리얼은 범퍼로부터 정보를 얻는 것에 초점을 맞추지만, 여기서 만들어진 서비스는 다른 센서 장치에 적용될 수 있습니다.

시작하기

서비스를 만들어, 이 튜토리얼로 시작되기 위해. The DssNewService 도구는 Decentralized Software Services Protocol(DSSP)동작을 지원하는 기본 서비스를 만듭니다.

새 서비스 만들기

1. Robotics Studio 커맨드 프롬프트를 여십시오:
    - 시작 메뉴에서 모든 프로그램, 마이크로소프트 Robotics Studio, 그 다음 마이크로소프트 Robotics Studio 커맨드 프롬프트를 선택하십시오.
2. Samples 디렉터리로 이동합니다.

```
cd samples
```

3. DssNewService 도구를 /service:MyTutorial1 매개 변수를 입력해 실행하여 MyTutorial1 이름의 서비스를 생성합니다:

```
dssnewservice /service: MyTutorial1
```

4. 생성된 MyTutorial1 디렉터리로 이동합니다.

5. C#에디터를 사용해 MyTutorial1 솔루션을 로드합니다.

일반적으로, 서비스를 사용하기 위해 다음과정을 따라야 합니다:

1. 사용하고자 하는 서비스의 프록시 DLL 을 프로젝트 참조(project reference)에 추가합니다.
2. using 지시자를 이용해 서비스를 참조 하십시오.
3. 서비스와 통신을 위해 포트를 셋업하십시오. 포트는 우리가 사용하고 있는 서비스에 의해 정의되고, strongly typed 인터페이스를 서비스와 상호 작용하기 위해 제공합니다.

단계 1: 참조 추가

DssNewService 도구를 이용해 MyTutorial 서비스를 만들면, 몇 개의 로보틱스 DLL 파일 참조가 서비스 프로젝트에 디폴트로 추가됩니다. 이 튜토리얼은 RoboticsCommon 라이브러리에 있는 콘택트 센서를 필요로 합니다. Robotics Common 라이브러리의 참조는 그 프록시 DLL 인, RoboticsCommon.Proxy.dll 을 이용 합니다.

RoboticsCommon.Proxy.dll File 을 참조로 추가

1. 비주얼 스투디오의 솔루션 탐색기에서 있는 참조를 오른쪽 버튼을 클릭하십시오. 그리고 참조추가를 선택하십시오. 참조추가 대화 상자는 열립니다.
2. .NET 탭에서, RoboticsCommon.Proxy 를 선택하십시오.

RoboticsCommon 은 콘택트 센서에 대한 제네릭 컨트랙트를 정의합니다. 이를 사용하면 사전에 어느 하드웨어가 접속하게 될 것인지 알 필요가 없습니다.

다음과 같이 MyTutorial1.cs 의 맨 위에 범퍼 서비스의 토대인 ContactSensor 의 제네릭 컨트랙트 사용을 위해 using 지시자를 이용합니다.

```
using bumper = Microsoft.Robotics.Services.ContactSensor.Proxy;
```

단계 2: 파트너 관계 생성

이제 MyTutorial1 서비스와 범퍼 서비스 사이의 파트너 관계를 만듭니다. 서비스 파트너는 서비스가 의존하는 다른 서비스입니다. 범퍼 서비스와 통신하기 위해, 우리는 ContactSensorArrayOpearations 의 포트를 셋업하고, Partner 어트리뷰트를 사용하는 것으로 이를 파트너와 동일시합니다. _mainPort 를 정의하는 라인 이후 다음 코드를 당신의 서비스에 추가하십시오:

```
[Partner("bumper", Contract = bumper.Contract.Identifier,
    CreationPolicy = PartnerCreationPolicy.UseExisting)]
private bumper.ContactSensorArrayOperations _bumperPort = new
```

```
bumper.ContactSensorArrayOperations();
```

솔루션 탐색기에 리스트된 프로젝트 파일을 보면, MyTutorial1 서비스가 만들질 때 같이 생성 된, 매니페스트 파일인 MyTutorial1.manifest.xml 를 볼 수 있습니다. 매니페스트는 당신의 애플리케이션이 실행할 때 시작되는 서비스 또는 서비스들을 지정합니다.

서비스 파트너를 여러분의 하드웨어에 연결하는 가장 간단한 방법은 하드웨어를 위한 서비스 컨트랙트를 매니페스트에 추가해 시작하는 것입니다. 다음 라인은 비주얼 스투디오 실행 시 프로젝트 속성의 디버그에 있는 명령줄 인수에 LEGO NXT 범퍼를 추가하는 방법입니다. 이 튜토리얼에서 지원하는 하드웨어에 대한 컨트랙트를 찾기 위해, 마이크로소프트 Robotics Studio 설치 디렉터리의 ₩samples₩config₩ 디렉터리에서 .MotorTouchSensor.manifest.xml 로 끝나는 매니페스트들 중 여러분의 로봇과 통신하는 하나를 발견하십시오. 그리고 비주얼 스투디오 Debug 탭에 있는 Command Line Arguments 에 하드웨어를 위한 매니페스트를 추가합니다.

```
/p:50000 /t:50001 /m:"samples/MyTutorial1/MyTutorial1.manifest.xml"
```

를 다음과 같이 변경합니다.

```
/p:50000 /t:50001
/m:"samples/MyTutorial1/MyTutorial1.manifest.xml"
/m:"samples/config/LEGO.NXT.MotorTouchSensor.manifest.xml"
```

주의:

만일 완성된 튜토리얼인 samples₩Robotics Tutorials₩Tutorial1₩CSharp₩RoboticsTutorial1.csproj 를 사용할 경우 반드시 Debug 탭의 아래에서 Command Line Arguments 를 당신의 하드웨어를 위해 다음과 같이 RoboticsTutorial1.manifest.xml 매니페스트와 하드웨어 매니페스트를 사용하게 변경합니다.

/p:50000 /t:50001 /m:"samples/config/RoboticsTutorial1.manifest.xml" /m:"samples/config/LEGO.NXT.MotorTouchSensor.manifest.xml"

단계 3: 서비스 초기화와 시작

모든 서비스는 Start()를 선언해야 합니다메서드. Start() (서비스가 초기화 후에)메서드는 자동적으로 불립니다. 범퍼 서비스에 서브스크립션하고, 콘택트 센서 알림을 듣기 시작하도록 Start()를 메서드를 변경합니다.

Start()메서드 끝에 SubscribeToBumpers 메서드를 추가해 범퍼 서비스에 서브스크립션합니다. 다음 단계와 같이 SubscribeToBumpers 를 정의합니다.

```
// Start listening for bumpers.
SubscribeToBumpers();
```

Start() 에서 있는 base.Start 호출은 DssNewService 도구로 만들어졌는 템플릿의 한 부분입니다. 이 메서드는 서비스가 메인 포트에 보내지는 메시지를 위해 서비스 핸들러를 가동시킵니다. 다른 서비스들이 이 서비스를 발견할 수 있게 로컬 서비스 디렉터리에 서비스를 공지합니다.

단계 4: 서브스크립션 작성

Start()메서드 이후 SubscribeToBumpers()메소드를 추가합니다.

```
/// <summary>
/// Subscribe to the Bumpers service
/// </summary>
void SubscribeToBumpers()
{
    // Create the bumper notification port.
    bumper.ContactSensorArrayOperations bumperNotificationPort = new
bumper.ContactSensorArrayOperations();

    // Subscribe to the bumper service, receive notifications on the
bumperNotificationPort.
    _bumperPort.Subscribe(bumperNotificationPort);

    // Start listening for updates from the bumper service.
    Activate(
        Arbiter.Receive<bumper.Update>
            (true, bumperNotificationPort, BumperHandler));
}
```

SubscribeToBumpers()메서드 처음의 태스크는 범퍼 서비스로부터 알림을 수신하는 알림 포트를 만듭니다. 알림 포트를 ContactSensorArrayOperations 의 인스턴스로 부터 만듭니다.

```
bumper.ContactSensorArrayOperations bumperNotificationPort = new
bumper.ContactSensorArrayOperations();
```

_bumperPort 포트를 서브스크립션합니다. 그리고 알림이 bumperNotificationPort 에 보내어짐을 지정합니다.

```
_bumperPort.Subscribe(bumperNotificationPort)
```

Activate()메서드를 사용해 핸들러가 범퍼로부터 알림을 수신도록 설정합니다. Activate() 포트와 아비터(arbiter) 사이의 관계를 등록하는 일반적인 메서드입니다. 다음 코드 부분에서 사용되는 Receive 중재자는 범퍼부터 범퍼 핸들러로 알림 메시지를 보냅니다.

```
Activate(
        Arbiter.Receive<bumper.Update>
            (true, bumperNotificationPort, BumperHandler));
```

이것으로 서비스가 범퍼 서비스로부터 범퍼 알림을 받을 때마다 BumperHandler 메서드가 호출됩니다.

단계 5: 범퍼 핸들러 생성

MyTutorial1.cs 파일의 끝부분에 BumperHandler() 메서드를 추가합니다. 범퍼가 눌렸다라고 알림받으면 BumperHandler()메서드에서 콘솔 화면에 메시지를 출력합니다.

```
/// <summary>
/// Handle Bumper Notifications
/// </summary>
/// <param name="notification">Update notification</param>
private void BumperHandler(bumper.Update notification)
{
    if (notification.Body.Pressed)
        LogInfo(LogGroups.Console, "Ouch - the bumper was pressed.");
}
```

실행해 보기

MyTutorial1 서비스를 빌드하고, 실행하기 위해, 디버그 메뉴에서 디버깅시작 (F5 를 누름)를 선택합니다.

범퍼를 눌릴 때마다 콘솔 화면에 메시지가 출력됩니다.

그림 2 - 범퍼가 눌렸을 때 콘솔 출력

완성된 튜토리얼은 samplesWRoboticsTutorialsWTutorial1WCSharpW 디렉터리에서 있습니다.

로보틱스 튜토리얼 2(C#) - 서비스 제어하기

로보틱스 튜토리얼 1(C#)에서 단순한 서비스를 액세스하는 방법과 다른 서비스에서 발생하는 이벤트의 알림을 요청할 수 있는 방법을 배웠습니다. 이 튜토리얼은 로보틱스 튜토리얼 1 에 이어서 콘택트 센서를 사용해 모터를 멈추는 방법을 보입니다. 로보틱스 튜토리얼 1(C#)에서 만든 MyTutorial1을 변경하여 모터가 콘택트 센서가 눌리는 것에 따라 돌거나 멈추게 합니다.

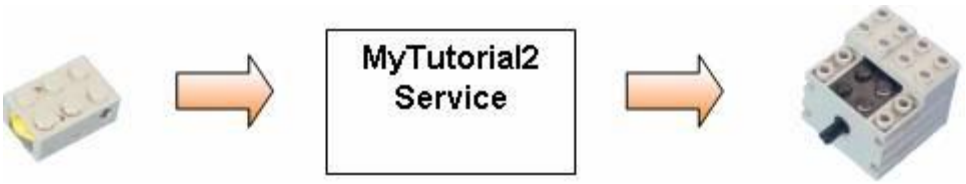


그림 1 - 범퍼 서비스와 모터 서비스의 간단한 조정

단계 1: 참조 추가

로보틱스 튜토리얼 1(C#)에서 프로젝트에 RoboticsCommon.Proxy 참조를 추가했습니다. 튜토리얼 1 에서 만들어진 서비스에서 이 참조는 그래도 남아있어야 합니다. MyRoboticsTutorial1의 솔루션탐색기 속성에서 이를 확인하십시오:

```
RoboticsCommon.Proxy
```

MyRoboticsTutorial1.cs 파일에서 참조 된 서비스를 액세스하기 위해 using 지시자를 이용했습니다. 이번 튜토리얼의 서비스에도 하나 이상의 참조를 필요로 합니다. 모터 서비스를 위한 프록시 DLL 인 Microsoft.Robotics.Services.Motor.Proxy 와 그 에일리어스(motor)는 다음과 같습니다.

```
using motor = Microsoft.Robotics.Services.Motor.Proxy;
```

다음 단계는 MyTutorial1 서비스와 모터 서비스 사이의 파트너 관계 생성입니다. 서비스 관계는 협력은 현재 서비스가 의존하는 서비스와 동일시 됩니다. 모터 서비스와 통신하기 위해, MotorOperations 포트를 셋업하고, 다음 어트리뷰트를 사용하는 것에 의해 파트너와 동일시합니다. 범퍼 파트너 선언 이후 다음 코드를 서비스에 추가하십시오.

```
[Partner("motor", Contract = motor.Contract.Identifier, CreationPolicy =
PartnerCreationPolicy.UseExisting)]
private motor.MotorOperations _motorPort = new motor.MotorOperations();
```

단계 2: 범퍼 서비스에 서브스크립션

MyTutorial1 서비스는 범퍼 서비스에 다음과 같이 서브스크립션 합니다.

```
/// <summary>
/// Subscribe to notifications when bumpers are pressed
/// </summary>
void SubscribeToBumpers()
{
    // Create bumper notification port
    bumper.ContactSensorArrayOperations bumperNotificationPort = new
bumper.ContactSensorArrayOperations();

    // Subscribe to the bumper service, send notifications to bumperNotificationPort
    _bumperPort.Subscribe(bumperNotificationPort);

    // Start listening for Bumper Replace notifications
    Activate(
        Arbiter.Receive<bumper.Update>
            (true, bumperNotificationPort, BumperHandler));
}
```

단계 3: 범퍼 핸들러 수정

이 단계는 MyTutorial1 가 범퍼 상태로 모터를 때때로 돌리기 위해 범퍼 핸들러를 변경하는 방법을 설명합니다.

MyTutorial1Types.cs 파일을 열어 데이터 구성 요소를 서비스의 StateType 클래스에 추가하십시오. StateType 클래스는 서비스에 대한 디폴트 데이터 컨트랙트를 제공하고, 서비스를 조작하기 위해 필요한 어떤 상태라도 포함합니다. 만일 모터의 동작 상태가 온 또는 오프라면 모터의 조작 가능 상태를 추적하기 위해 bool 값을 포함하는 MotorOn 을 데이터 구성 요소로 추가하십시오.

```
[DataContract]
public class RoboticsTutorial2State
{
    [DataMember]
    public bool MotorOn;
}
```

다음 태스크는 범퍼 핸들러(BumperHandler) 메서드를 MotorOn 데이터 구성 요소의 현재 설정치에 따라 모터를 구동하도록 다음과 같이 수정합니다:

```
/// <summary>
/// Handle Bumper Notifications
/// </summary>
/// <param name="notification"></param>
private void BumperHandler(bumper.Update notification)
{
    string message;

    if (!notification.Body.Pressed)
        return;

    // Toggle the motor state
    _state.MotorOn = !_state.MotorOn;

    // Create a motor request
    motor.SetMotorPowerRequest motorRequest = new motor.SetMotorPowerRequest();

    if (_state.MotorOn)
    {
        // Set the motor power on
        motorRequest.TargetPower = 1.0;
        message = "Motor On";
    }
    else
    {
        // Set the motor power off
        motorRequest.TargetPower = 0.0;
        message = "Motor Off";
    }

    // Send the motor request
    _motorPort.SetMotorPower(motorRequest);

    // Show the motor status on the console screen
    LogInfo(LogGroups.Console, message);
}
```

BumperHandler 메서드에 추가된 것들은 다음과 같습니다:

1. 범퍼가 눌렸는 지를 확인합니다.
   만일 발생한 이벤트가 범퍼 릴리즈라면, BumperHandler 메서드는 리턴합니다.
2. 만일 어떠한 범퍼 릴리즈도 없으면, BumperHandler 메서드는 모터의 상태를 바꿉니다.
3. 새로운 모터 상태 값에 따라, BumperHandler 메서드는 그 다음 SetMotorPower 요구 메시지를 모터 전원을 0%또는 100%에 설정합니다.
4. BumperHandler 는 SetMotorPower 로 메시지를 전송합니다.
5. 그리고 콘솔 창에 동작을 표시합니다.

실행해보기

만일 로보틱스 튜토리얼 1 에서 시작한 경우, 로봇을 위한 하드웨어 매니페스트가 벌써 추가되었을 것입니다. 디버그 탭 아래에서 당신의 프로젝트 설정을 여십시오. 그리고 명령줄 인수가 여러분의 하드웨어를 위한 매니페스트를 참조하게 변경 되었는지 확인하십시오.

주의:
만일 완성된 튜토리얼인 samplesWRoboticsTutorialsWTutorial2WCSharpWRoboticsTutorial2.csproj 를 사용하면 반드시 디버그 탭의 아래의 명령줄 인수를 변경해 여러분의 하드웨어를 위한 매니페스트를 참조하도록 하십시오. 튜토리얼 서비스의 매니페스트는 이 경우 RoboticsTutorial2.manifest.xml 일 것입니다.

```
-port:50000 -tcpport:50001 -manifest:"samplesWConfigWRoboticsTutorial2.manifest.xml" -
manifest:"samplesWconfigWLEGO.NXT.MotorTouchSensor.manifest.xml"
```

프로젝트를 저장하십시오.

MyTutorial2 서비스를 빌드하고, 실행하기 위해, 디버그 메뉴에서 디버깅시작 (F5 를 누름)를 선택합니다.

이제 범퍼를 여러 차례 누르십시오. 모터는 돌거나 멈추어야 합니다. 또한 현재 동작을 표시하는 메시지는 콘솔 창에서 표시됩니다:

```
Motor On
Motor Off
Motor On
Motor Off
```

여러분이 작성한 코드를 완성된 튜토리얼이 있는 samplesWRoboticsTutorialsWTutorial2W 디렉터리의 것과 비교해 볼 수 있습니다.

로보틱스 튜토리얼 3(C#)- 재사용 가능한 오케스트레이션 서비스 생성하기

마이크로소프트 Robotics Studio 는 서비스 개발에 있어 재사용 가능한 디자인을 제공합니다. 이 디자인은 한 번 서비스를 공통 하드웨어 사양에서 작성한 뒤 여러 가지 하드웨어 로봇 플랫폼에 두루 개발된 서비스를 사용하는 것이 가능합니다.

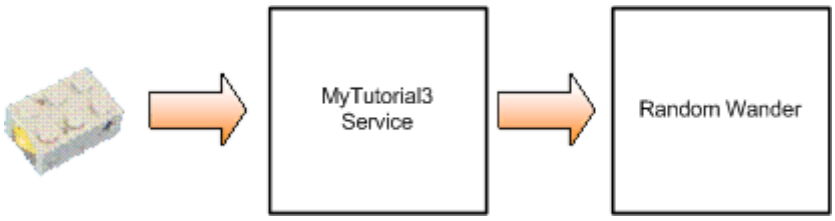


그림 1- 간단한 랜덤주행 서비스

이 튜토리얼은 하드웨어 서비스의 추상화와 기본 정의들을 가진 파트너들을 사용하는 서비스를 어떻게 만드는 지 보여줍니다. 설정 파일(매니페스트)에 의거하여, 런타임에 서비스는 이런 하드웨어 서비스들을 묶습니다. 튜토리얼은 매우 기본적인 떠돌이 동작을 실행합니다:

- 만일 전면 범퍼가 눌리면, 임의로 회전하고 같은 모터 출력을 역방향으로 양쪽 바퀴에 적용합니다.
- 만일 후방 범퍼가 눌리면, 임의로 회전하고 같은 모터 출력을 순방향으로 양쪽 바퀴에 적용합니다

이 매우 기초적인 동작은 바퀴를 가진 로봇이 물건에 충돌하고 회전하며 움직이는 결과를 보여줍니다. 더 많은 지적인(그리고 점더 고급의 조정 서비스) 기능은 로보틱스 튜토리얼 5(C#)를 참고하십시오.

시작하기

이 튜토리얼을 위해 서비스를 새로 만들지 않을 것입니다. 대신, 우리는 samples₩Robotics Tutorials₩Tutorial3₩CSharp 에 설치된 기존 서비스를 이용합니다.

개발 환경에서 RoboticsTutorial3.csproj 을 엽니다. 비주얼 스투디오를 사용하고 있으면, 솔루션탐색기에서 다음을 볼 수 있습니다. 이 서비스는 공통 정의가 들어있는 RoboticsCommon.Proxy.dll 를 사용합니다. 이 DLL 은 프로젝트 참조로 포함되어 있습니다.

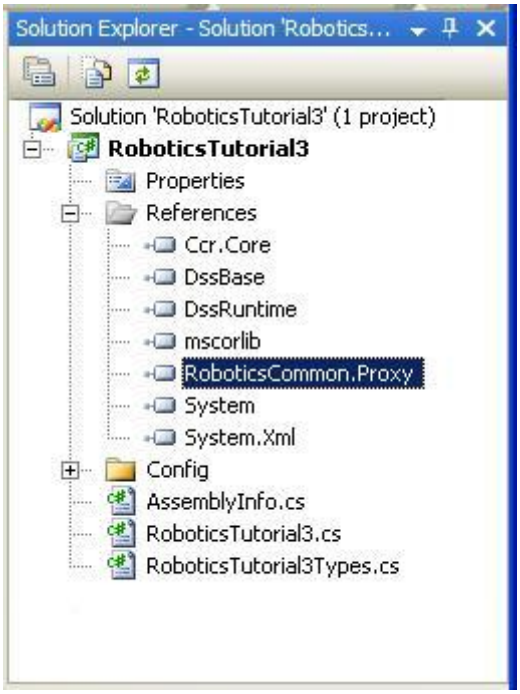


그림 2 - RoboticsCommon.Proxy.dll 이 서비스 프로젝트에 참조되었음을 확인합니다.

RoboticsTutorial3.cs 파일을 여십시오.

단계 1: 추상화 서비스에 대한 파트너 관계 사용

RoboticsTutorial3.cs 의 맨위에 다음의 using 명령 줄이 추가 되었습니다.

```
using contactsensor = Microsoft.Robotics.Services.ContactSensor.Proxy;
using drive = Microsoft.Robotics.Services.Drive.Proxy;
using motor = Microsoft.Robotics.Services.Motor.Proxy;
```

다음의 파트너 관계는 서비스 클래스 정의에서 미리 추가되어 있습니다:

```
[Partner("Bumper",
    Contract = contactsensor.Contract.Identifier,
    CreationPolicy = PartnerCreationPolicy.UseExisting)]
contactsensor.ContactSensorArrayOperations _contactSensorPort = new
contactsensor.ContactSensorArrayOperations();

[Partner("Drive",
    Contract = drive.Contract.Identifier,
    CreationPolicy = PartnerCreationPolicy.UseExisting)]
drive.DriveOperations _drivePort = new drive.DriveOperations();
```

이전의 파트너 관계 선언으로부터의 한가지 차이는 PartnerCreationPolicy 값입니다:

```
CreationPolicy = PartnerCreationPolicy UseExisting
```

ContactSensorArray 는 추상적인 서비스 정의입니다. 이것을 위한 어떠한 디폴트 서비스 구현이 존재하지 않습니다. 대신, 그것이 대체 컨트랙트(alternate contract)로 콘택트 센서 컨트랙트를 실행하는(LegoRCX Bumper 서비스와 같다) 특정의 서비스 구현에 의지합니다. 서비스가 대체 컨트랙트를 실행할 때, 서비스 인스턴스 디렉터리에서 자신의 컨트랙트로 처음 검색되고 대체 컨트랙트 역할을 하는 인스턴스로 검색됩니다. 마이크로소프트 Robotics Studio 의 일부로서 제공된 예제 서비스 중의 대부분은 대체 컨트랙트로 구현되었습니다.

단계 2: 범퍼로 콘택트 센서를 이용한 행위 유발

떠돌아다니는 동작을 시작하기 위해, 범퍼 중의 하나를 눌려야 합니다. 범퍼 알림을 수신하기 위해 콘택트 센서 포트에 서브스크립션하고, 알림은 목표하는 포트에 공급됩니다.

다음은 범퍼의 알림을 위한 목표하는 포트 선언입니다.

```
// target port for bumper notifications
contactsensor.ContactSensorArrayOperations _contactNotificationPort = new
contactsensor.ContactSensorArrayOperations();
```

SubscribeToBumperSensors()를 정의학고 이를 Start()에서 호출합니다:

```
/// <summary>
/// Service Start
/// </summary>
protected override void Start()
{

    // Listen on the main port for requests and call the appropriate handler.
    // Publish the service to the local Node Directory
    base.Start();

    SubscribeToBumperSensors();
}

/// <summary>
/// Subscribe to sensors and send notifications
/// to BumperNotificationHandler
```

```
/// </summary>
void SubscribeToBumperSensors()
{
    _contactSensorPort.Subscribe(_contactNotificationPort);

    // attach handler to update notification on bumpers
    Activate(Arbiter.Receive<contactsensor.Update>(true,      _contactNotificationPort,
BumperNotificationHandler));
}
```

SubscribeToBumperSensors()메서드는 _contactSensorPort 에 서브스크립션하여 알림 메시지를 받처록 설정합니다. 그 다음 알림 포트에서 핸들러를 가동시킵니다. 핸들러는 두개의 루틴을 실행합니다. 즉 전면 범퍼가 눌리면 핸들러는 TurnAndGoBackwards 메서드를 실행하고 후방 범퍼가 눌리면 그것은 TurnAndGoForward 를 실행합니다.

```
/// <summary>
/// Handler for Contact Sensor Notifications
/// </summary>
/// <param name="updateNotification"></param>
void BumperNotificationHandler(contactsensor.Update updateNotification)
{
    // Since we are writing a generic wander service we dont really know
    // which bumper is front, side, rear etc. We expect a navigation service to be
tuned
    // to a robot platform or read configuration through its initial state.
    // here, we just assume the bumpers are named intuitively so we search by name

    contactsensor.ContactSensor s = updateNotification.Body;
    if (!s.Pressed)
        return;

    if (!string.IsNullOrEmpty(s.Name) &&
        s.Name.ToLowerInvariant().Contains("front"))
    {
        SpawnIterator<double>(-1,BackUpTurnAndMove);
        return;
    }
```

```
    if (!string.IsNullOrEmpty(s.Name) &&
        s.Name.ToLowerInvariant().Contains("rear"))
    {
        SpawnIterator<double>(1,BackUpTurnAndMove);
        return;
    }
}
```

단계 3: Turn 과 Move 로 동장 명령 보냄

범퍼가 눌렸을 때 유발한 방법 중의 하나인 BackUpTurnAndMove 메서드는 다음의 동작을 수행합니다:

1. 1500 ms 동안 역 방향으로 이동합니다.
2. 두 바퀴의 모터에서 같은 세기 정반대 극성의 힘을 적용하여 로봇을 회전시킵니다.
3. 무작위 기간을 동안 양쪽 모터에 같은 힘을 로봇을 움직이기 위해 적용하십시오.

```
Random _randomGen = new Random();

/// <summary>
/// Implements a simple sequential state machine that makes the robot wander
/// </summary>
/// <param name="polarity">Use 1 for going forward, -1 for going backwards</param>
/// <returns></returns>
IEnumerator<ITask> BackUpTurnAndMove(double polarity)
{
    // First backup a little.
    yield return Arbiter.Receive(false,
        StartMove(0.4*polarity),
        delegate(bool result) { });

    // wait
    yield return Arbiter.Receive(false, TimeoutPort(1500), delegate(DateTime t) { });

    // now Turn
    yield return Arbiter.Receive(false,
        StartTurn(),
        delegate(bool result) { });
```

```
    // wait
    yield return Arbiter.Receive(false, TimeoutPort(1500), delegate(DateTime t) { });

    // now reverse direction and keep moving straight
    yield return Arbiter.Receive(false,
        StartMove(_randomGen.NextDouble()*polarity),
        delegate(bool result) { });

    // done
    yield break;
}

Port<bool> StartTurn()
{
    Port<bool> result = new Port<bool>();
    // start a turn
    SpawnIterator<Port<bool>>(result, RandomTurn);
    return result;
}

Port<bool> StartMove(double powerLevel)
{
    Port<bool> result = new Port<bool>();
    // start movement
    SpawnIterator<Port<bool>, double>(result, powerLevel, MoveStraight);
    return result;
}

IEnumerator<ITask> RandomTurn(Port<bool> done)
{
    // we turn by issuing motor commands, using reverse polarity for left and right
    // We could just issue a Rotate command but since its a higher level function
    // we cant assume (yet) all our implementations of differential drives support it
    double randomPower = _randomGen.NextDouble();
    drive.SetDrivePowerRequest setPower = new drive.SetDrivePowerRequest(randomPower,
-randomPower);
```

```
    bool success = false;
    yield return
        Arbiter.Choice(
            _drivePort.SetDrivePower(setPower),
            delegate(DefaultUpdateResponseType rsp) { success = true; },
            delegate(W3C.Soap.Fault failure)
            {
                // report error but report done anyway. we will attempt
                // to do the next step in wander behavior even if turn failed
                LogError("Failed setting drive power");
            });

    done.Post(success);
    yield break;
}

IEnumerator<ITask> MoveStraight(Port<bool> done, double powerLevel)
{
    drive.SetDrivePowerRequest setPower = new drive.SetDrivePowerRequest(powerLevel,
powerLevel);

    yield return
        Arbiter.Choice(
        _drivePort.SetDrivePower(setPower),
        delegate(DefaultUpdateResponseType success) { done.Post(true); },
        delegate(W3C.Soap.Fault failure)
        {
            // report error but report done anyway. we will attempt
            // to do the next step in wander behavior even if turn failed
            LogError("Failed setting drive power");
            done.Post(false);
        });
}
```

단계 4: 로보틱스 튜토리얼 3의 하드웨어 매니페스트 사용

로보틱스 튜토리얼 서비스는 단독으로는 실행할 수 없습니다. 그것은 드라이브, 모터와 콘택트 센서 추상 서비스의 실제 구현이 필요하기 때문입니다. 서비스를 실행하기 위해 선언한 파트너들의 인스턴스들이 존재해야 합니다.

서비스 파트너를 여러분의 하드웨어와 연결하는 단순한 방법은 각 하드웨어를 위해 서비스 컨트랙트를 포함하는 부가의 매니페스트를 같이 시작하는 것입니다. 다음 라인은 DssNewService 에 커맨드 라인 인수로 LEGO NXT Tribot 를 위해 부가 매니페스트를 포함시키는 방법입니다. 이 튜토리얼에서 지원하는 하드웨어를 위한 컨트랙트는 마이크로소프트 Robotics Studio 설치 디렉터리의 WSamplesWConfigW 디렉터리에서 .manifest.xml 로 끝나는 매니페스트들 중에서 찾아 볼 수 있습니다. 비주얼 스투디오의 디버그 탭 아래에 설정된 명령줄 인수에 여러분의 하드웨어를 위한 매니페스트를 추가로 입력하십시오.

예로 LEGO.NXT.TriBot 매니페스트를 실행하고자 할 때, 명령중 인수는 다음과 같습니다:

```
/p:50000 /t:50001 /m:"samplesWconfigWLEGO.NXT.TriBot.manifest.xml"
/m:"samplesWconfigWRoboticsTutorial3.manifest.xml"
```

실행해보기

마이크로소프트 Robotics Studio 커맨드 프롬프트에서 다음을 입력하거나 비주얼 스투디오(메뉴 > 빌드 > 빌드 솔루션)를 이용해 서비스를 컴파일 하십시오:

```
msbuild -v:m "samplesWRoboticsTutorialsWTutorial3WCSharpWRoboticsTutorial3.csproj"
```

프로젝트를 컴파일하고 난 후에, 마이크로소프트 Robotics Studio 커맨드 프롬프트에서 그것을 실행할 수 있습니다. 로봇 매니페스트를 여러분의 하드웨어 매니페스트로 교체하십시오. 만일 여러분의 하드웨어가 Robotics Studio 에서 지원 되지 않으면, 시뮬레이션 매니페스트를 다음과 같이 사용하십시오:

```
dsshost /p:50000 /t:50001 /m:"samplesWconfigWRoboticsTutorial3.manifest.xml"
/m:"samplesWconfigWMobileRobots.P3DX.Simulation.manifest.xml"
```

이제 로봇에 있는 범퍼를 눌러보십시오. 임의로 돌아다니고 있는 로봇을 보게 될 것입니다.

시뮬레이션:

시뮬레이션 매니페스트를 사용하고 있으면, 시뮬레이션하게 되는 가상 카메라를 사용해 로봇 범퍼를 누를 수 있습니다. 첫번째로 Alt-P(Menu/Physics)/Settings 하여 카메라 설정을 [x] Enable rigid body for default camera 를 사용하도록 설정하고/ [ok]를 선택합니다.

이것은 카메라 앞에서 작은 구를 만듭니다. 이제 F2 를 두번 눌러 물리적 시야(physics view)로 만든 뒤 로봇 범퍼에 접촉할 때까지 카메라를 움직입니다.

주의:

여러분 로봇의 하드웨어 설정을 위해 하드웨어 설정 개요를 참고하십시오.

로보틱스 튜토리얼 4(C#) - 조종 UI 를 통한 로봇 제어(Drive-By-wire)

마이크로소프트 Robotics Studio 는 서비스 개발에 있어 재사용 가능한 디자인을 제공합니다. 이 디자인은 한 번 서비스를 공통 하드웨어 사양에서 작성한 뒤 여러 가지 하드웨어 로봇 플랫폼에 두루 개발된 서비스를 사용하는 것이 가능합니다.

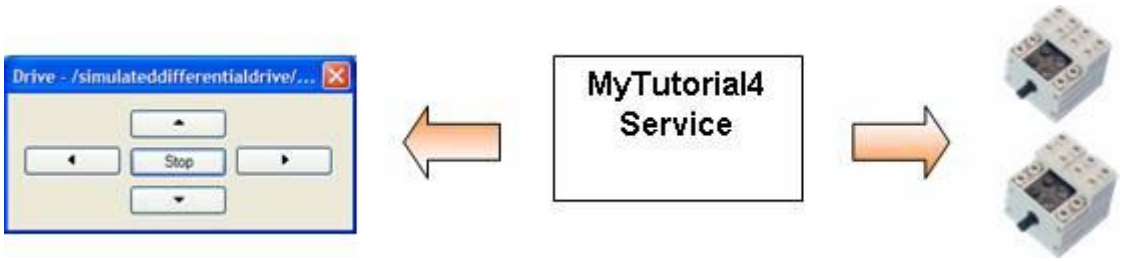


그림 1 - 로보틱스 튜토리얼 4

이 튜토리얼은 하드웨어 서비스의 추상화와 기본 정의들을 가진 파트너들을 사용하는 서비스를 어떻게 만드는 지 보여줍니다. 설정 파일(매니페스트)에 의거하여, 런타임에 서비스는 이런 하드웨어 서비스들의 특정된 구현들을 묶습니다. 이 튜토리얼은 윈도우 유저 인터페이스로 로봇 움직임을 제어하는 것을 보여줍니다.

시작하기

이 튜토리얼을 위해 새 서비스를 만들지 않을 것입니다. 대신 samplesWRoboticsTutorialsWTutorial4WCSharp 에 설치된 서비스를 이용합니다.

개발 환경에서 RoboticsTutorial4.csproj 을 불러 시작하십시오. 비주얼 스투디오를 사용한다면 솔루션 탐색기 창에서 다음을 볼 수 있습니다:

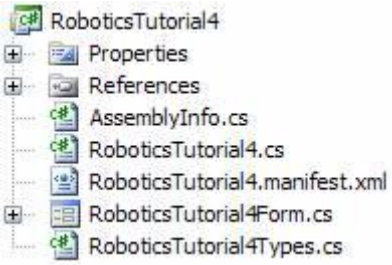


그림 2- 로보틱스 튜토리얼 4 프로젝트 파일

RoboticsTutorial4.cs 파일을 여십시오.

단계 1: 서비스 조작 정의

RoboticsTutorial4Types.cs 에는 서비스가 지원하는 조작(operation)을 정의합니다. 클래스 RoboticsTutorial4Operations 는 DssNewService 에 의해 생성된 조작에 더하여 5 개의 새로운 조작을 정의합니다.

```
[ServicePort]
public class RoboticsTutorial4Operations : PortSet<
    DsspDefaultLookup,
    DsspDefaultDrop,
    Get,
    Replace,
    Stop,
    Forward,
    Backward,
    TurnLeft,
    TurnRight>
{
}
```

이 조작의 각각은 메시지 몸체(Body) 타입을 위한 클래스 정의와 각 조작에 대한 클래스 정의를 가집니다. 아래는 Stop 조작을 위한 메시지는 타입과 요청(request) 타입입니다:

```
public class Stop : Submit<StopRequest, PortSet<DefaultSubmitResponseType, Fault>>
{
    public Stop()
        : base(new StopRequest())
    {
    }
}

[DataContract]
public class StopRequest { }
```

이것은 제네릭 타입 Submit<TBody TResponse>에서 파생되고 몸체 타입으로 StopRequest 를 가지는 조작 클래스인 Stop 을 정의합니다. Stop 이 정의된 것과 마찬가지로, 이 파일에는 Forward, Backward, TurnLeft 와 TurnRight 를 위한 정의를 포함합니다. 서비스는 드라이브 파트너에게 명령을 보내는 것으로 이 조작들에 응답합니다.

RoboticsTutorial4.cs 안에 메인 서비스 구현 클래스인 RoboticsTutorial4 에는 이들 조작 각각에 매칭되는 핸들러가 있습니다. Forward 조작에 대한 핸들러는 ForwardHandler 입니다.

ForwardHandler

만일 모터가 사용 가능하지 않으면 ForwardHandler 는 EnableMotor 를 호출해 드라이브에 모터를 사용 가능하게 하는 메시지를 보냅니다.

로봇을 전진하게 하기 위해, ForwardHandler 는 SetDrivePowerRequest 를 생성하고, SetDrivePower 헬퍼 함수를 사용해 _drivePort 에 이 요청을 포스트합니다.

```
[ServiceHandler(ServiceHandlerBehavior.Concurrent)]
public virtual IEnumerator<ITask> ForwardHandler(Forward forward)
{
    if (!_state.MotorEnabled)
    {
        yield return EnableMotor();
    }

    // This sample sets the power to 75%.
    // Depending on your robotic hardware,
    // you may wish to change these values.
    drive.SetDrivePowerRequest request = new drive.SetDrivePowerRequest();
    request.LeftWheelPower = 0.75;
    request.RightWheelPower = 0.75;

    yield return Arbiter.Choice(
        _drivePort.SetDrivePower(request),
        delegate(DefaultUpdateResponseType response) { },
        delegate(Fault fault)
        {
            LogError(null, "Unable to drive forwards", fault);
        }
    );
```

```
}
```

_drivePort 필드는 추상 드라이브 서비스 정의를 사용하는 파트너의 조작 포트입니다. 자세한 사항은 로보틱스 튜토리얼 3 에 기술되어 있습니다.

다른 4 개의 메시지들을 위한 핸들러들도 유사하게 정의됩니다.

EnableMotor

만일 서비스 상태의 MotorEnabled 속성이 false 로 지정된 경우, Forward, Backward, LeftTurn 과 RightTurn 을 위한 핸들러는 EnableMotor 유틸리티 메서드로 모터를 사용 가능하게 합니다.

이 메서드는 드라이브 서비스 파트너에 EnableDrive 업데이트(update) 메시지를 보냅니다. Choice 타입을 리턴하는 것으로 호출자는 함수의 결과를 리턴합니다.

```
private Choice EnableMotor()
{
    drive.EnableDriveRequest request = new drive.EnableDriveRequest();
    request.Enable = true;

    return Arbiter.Choice(
        _drivePort.EnableDrive(request),
        delegate(DefaultUpdateResponseType response) { },
        delegate(Fault fault)
        {
            LogError(null, "Unable to enable motor", fault);
        }
    );
}
```

NotifyDriveUpdate 핸들러

모터가 사용 가능한지를 계속 확인 할 수 있게 이 서비스는 드라이브 서비스를 파트너로 서브스크립션합니다. 일부 로봇에는 배터리를 보존하고, 비정상적인 동작을 막기 위해 모터가 불능화(disabled) 될 수 있습니다. Start 메서드에, 서브스크립션이 시작되고 NotifyDriveUpdate 메서드는 드라이브 서비스로부터 Update 알림을 처리하기 위해 활성화 됩니다:

```
_drivePort.Subscribe(_driveNotify);
Activate(Arbiter.Receive<drive.Update>(true, _driveNotify, NotifyDriveUpdate));
```

NotifyDriveUpdate 메시지는 update 메시지의 바디에서 IsEnabled 속성을 StateType 변수의 MotorEnabled 에 복사합니다. 이 후 StateType 변수는 Replace message 를 메인 포트에 포스트함으로 서비스 상태를 변화시킵니다. StateType 클래스는 RoboticsTutorial4Types.cs 에 정의됩니다:

```
private void NotifyDriveUpdate(drive.Update update)
{
    RoboticsTutorial4State state = new RoboticsTutorial4State();
    state.MotorEnabled = update.Body.IsEnabled;

    _mainPort.Post(new Replace(state));
}
```

단계 2: 폼 시작

윈폼(WinForm)은 그것이 시작된 스레드에서만 사용됩니다. 서비스가 폼이 시작된 쓰레드를 추적할 수 있게 하기 위해 그리고 정확하게 해당 폼에 인보케이션(invocation)을 연결하기 위해, 서비스는 WinFormsServicePort 에 메시지를 보냅니다. 이것은 서비스 구현 클래스가 정의된 DsspServiceBase 클래스에서 정의된 내부 포트입니다.

폼을 표시하기 위해 RunForm 메시지에 포스트해야 합니다. 폼을 제어할 코드를 부르는 FormInvoke 메시지를 포스트합니다.. 서비스 구현 클래스에서 Start 메서드는 이 방법을 보여줍니다.

```
protected override void Start()
{
    base.Start();

    WinFormsServicePort.Post(new RunForm(StartForm));
    _drivePort.Subscribe(_driveNotify);
    Activate(Arbiter.Receive<drive.Update>(true, _driveNotify, NotifyDriveUpdate));
}
```

RunForm 메시지는 생성자에 하나의 매개 변수로 폼을 만들기 위해 호출되는 delegate 를 가집니다. 그 delegate 는 System.Windows.Forms.Form 객체를 리턴합니다. 이 객체는 폼 인스턴스를 생성하고 새롭게 만들어지는 객체를 리턴합니다. 아래 코드에서 폼의 타이틀은 사용되고 있는 드라이브 서비스의 경로로 설정됩니다. 이것은 FormInvoke 메시지를 사용하는 방법을 보여줍니다.

```
private System.Windows.Forms.Form StartForm()
```

```
{
    RoboticsTutorial4Form form = new RoboticsTutorial4Form(_mainPort);

    Invoke(delegate()
        {
            PartnerType partner = FindPartner("Drive");
            Uri uri = new Uri(partner.Service);
            form.Text = string.Format(
                Resources.Culture,
                Resources.Title,
                uri.AbsolutePath
            );
        }
    );

    return form;
}
```

이 헬퍼 메서드는 WinFormsServicePort 에 FormInvoke 메시지를 보냅니다. 이것은 폼을 변경하는 코드가 정확한 스레드에서 실행됨을 보여줍니다.

```
private void Invoke(System.Windows.Forms.MethodInvoker mi)
{
    WinFormsServicePort.Post(new FormInvoke(mi));
}
```

단계 3: 폼에서 서비스 호출

위에 Step 2 에, 폼은 서비스 시작의 일부로서 만들어졌습니다. 폼이 만들어졌을 때, _mainPort 필드는 폼 생성자에 매개 변수로 넘겨 집니다. 이는 폼이 서비스에 메시지를 보내는 것을 가능하게 합니다. 폼은 단계 1 에서 정의된 5 개의 조작에 부합하는 5 개의 가집니다.

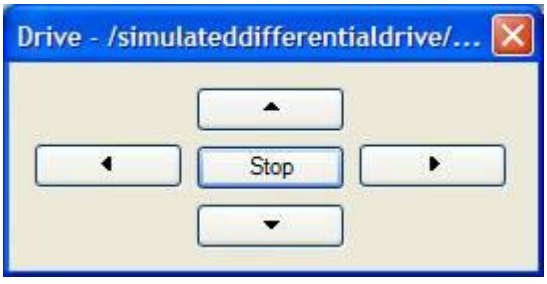


그림 3 - 로보틱스 튜토리얼 4 Direction Dialog

버튼이 눌릴 때, 이벤트는 서비스 메인 포트에 적합한 조작을 포스팅하는 것으로 처리됩니다. 다음은 Stop 버튼을 위한 이벤트 핸들러 함수입니다.

```
private void btnStop_Click(object sender, EventArgs e)
{
    _mainPort.Post(new Stop());
}
```

다른 버튼에서도 유사한 방법을 사용합니다.

폼이 닫혀질 때, 서비스는 더 이상 그 주요 기능을 수행하지 못합니다. 폼을 닫을 때 폼 코드는 서비스 메인 포트에 Drop 메세지를 보냅니다.

```
protected override void OnClosed(EventArgs e)
{
    _mainPort.Post(new DsspDefaultDrop(DropRequestType.Instance));

    base.OnClosed(e);
}
```

단계 4: 서비스 실행

로보틱스 튜토리얼 3 서비스와 같이 이 서비스는 추상 드라이브 컨트랙트를 사용합니다. 이것은 컨트랙트가 이를 구현하는 어떤 서비스라도 구동하는 데 사용될 수 있다는 것을 의미합니다.

SamplesWConfig 디렉터리에 .manifest.xml 확장자를 가진 몇 개의 파일이 있습니다. 로보틱스 튜토리얼 3 에 기술된 바와 같이 이 파일들은 DSS 노드가 시작되는 때 실행할 서비스들과, 선택적으로 각각을 위한 설정 정보의 리스트를 가지고 있습니다. 이 튜토리얼을 시뮬레이션된 드라이브와 같이 실행하기 위해 커맨드 라인에 다음 명령을 입력합니다:

```
dsshost /p:50000 /t:50001 /m:"samplesWconfigWMobileRobots.P3DX.Simulation.manifest.xml"
/m:"samplesWconfigWRoboticsTutorial4.manifest.xml"
```

로보틱스 튜토리얼 5 (C#) - 고급 서비스 기능 사용하기
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.

- Robotics Tutorial 5 (C#) - Using Advanced Services

로보틱스 튜토리얼 6 (C#) - 원격 연결 로봇
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.

- Robotics Tutorial 6 (C#) - Remotely Connected Robots

로보틱스 튜토리얼 7 (C#) - 음성과 비전 처리
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.

- Robotics Tutorial 7 (C#) - Speech and Vision in Robots

기타 예제

로보틱스 기타 예제 설명
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Robotics Samples

Microsoft GPS 서비스
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Microsoft GPS Service

Text-To-Speech 서비스
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Text-To-Speech Service

음성인식 서비스
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Speech Recognition Sample

간단한 비전 처리 예제
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Simple Vision Sample

비전 기반 트래킹 예제
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Vision Based Tracking Sample

웹캠 서비스
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- WebCam Service

Windows 메시지 예제
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Windows Messages Sample

Atom/RSS 신디케이션 서비스 개요
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Atom/RSS Syndication Service Overview

Atom/RSS 신디케이션 서비스 튜토리얼
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Atom/RSS Syndication Service Tutorial

SQL 서버의 데이터를 다루기 위한 저장 및 조회하기
본 자료의 세부적인 내용은 아래 영문 내용을 참고하시기 바랍니다.
- Storing, Retrieving, and Manipulating Data in a SQL Server