표준 데이터 로거 내보내기 모듈(Export DLL) 제작 가이드

1. **들어가기 전에**

이 문서는 표준 데이터 로깅 프로그램을 통해 측정된 센서 데이터를 내보내는(Export) DLL 형태의 모듈을 작성하는 방법을 설명합니다.

Export DLL 을 제작함으로, 센서 데이터를 파일로 저장하거나, 네트워크를 통해 서버 또는 다른 프로그램으로 전송하게끔 할 수 있습니다.

Export DLL 은 하나의 데이터 로거에 여러 개의 모듈을 적용할 수 있어서, 파일로 저장하면서 서버로 전송하는 등의 작업을 동시에 수행할 수 있습니다.

이 예제에서는 로깅 프로그램에서 수집된 데이터를 CSV 형식으로 내보내는 DLL 을 만들어 볼 것입니다.

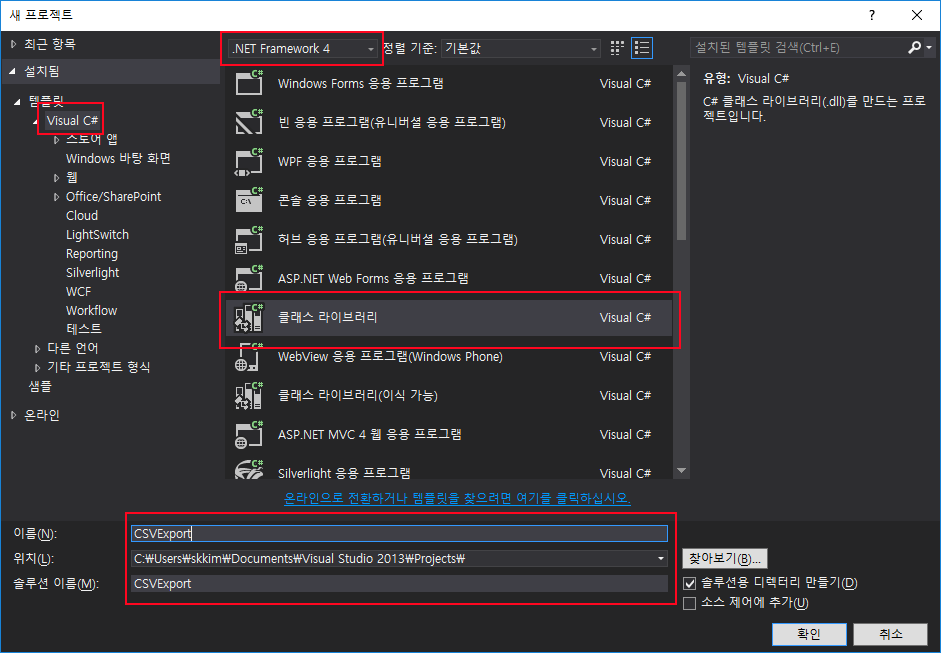
표준 데이터 로깅 프로그램은 C# 언어로 제작되었으며, .NetFramework 4.0 환경으로 개발되었습니다.

1. **개발 환경 준비**

OS : Windows Vista 이상 버전이 필요합니다.  
IDE : Microsoft Visual Studio 2010 이상 버전이 필요합니다.(Express Edition, Community Edtion 가능)

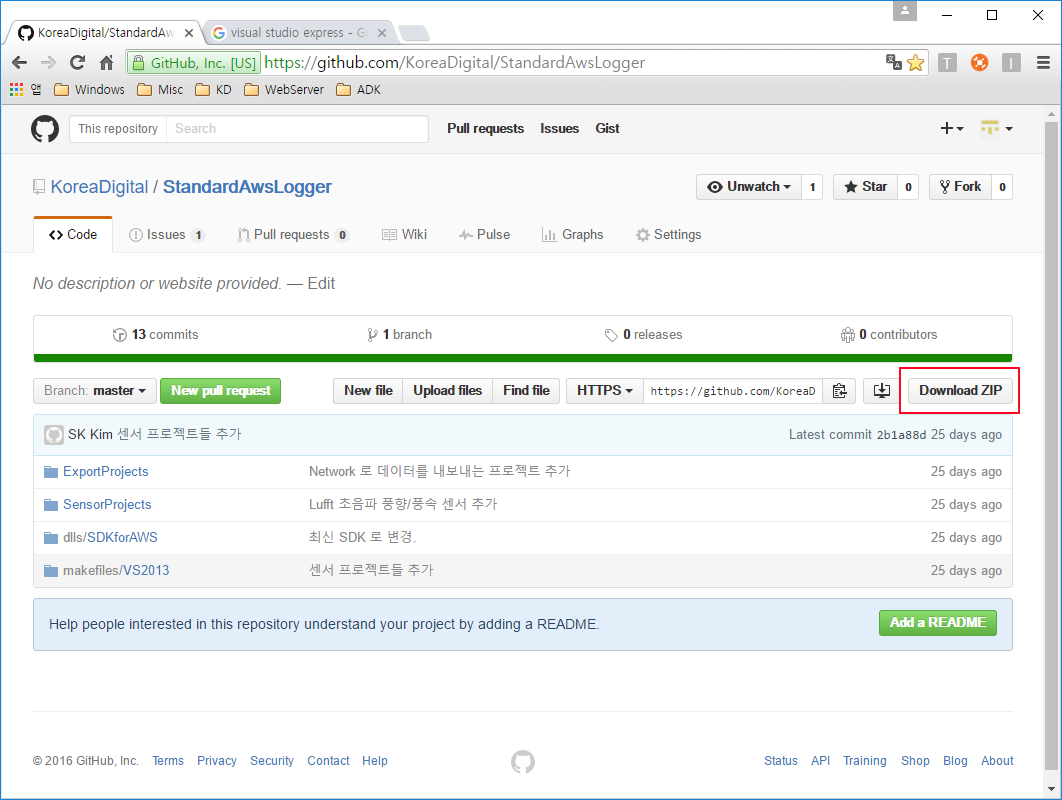
1. **프로젝트 생성**

Visual Studio 를 실행하고, C# 새 프로젝트를 생성합니다. 대상 Framework 는 4.0, 프로젝트 형식은 클래스 라이브러리로 지정하고 프로젝트 이름을 적습니다.



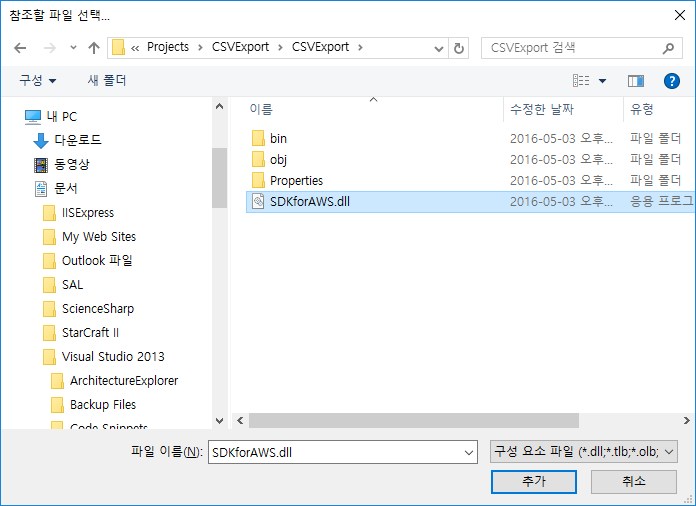
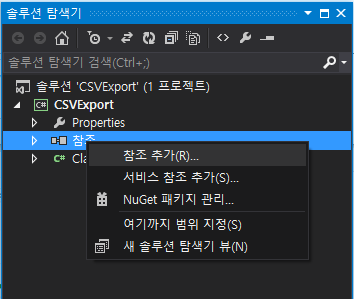
1. **소스코드 다운로드**

표준 데이터 로거의 GutHub 오픈 소스 프로젝트 사이트(<https://github.com/KoreaDigital/StandardAwsLogger>)에 접속하여, 소스코드를 다운로드 합니다.  
  
다운로드한 소스파일의 압축을 해제하고,  
StandardAwsLogger-master\dlls\SDKforAWS\Debug\ 아래 있는 SDKforAWS.dll 과 SDKforAWS.pdb 파일을 새로 생성한 프로젝트 폴더로 복사합니다.



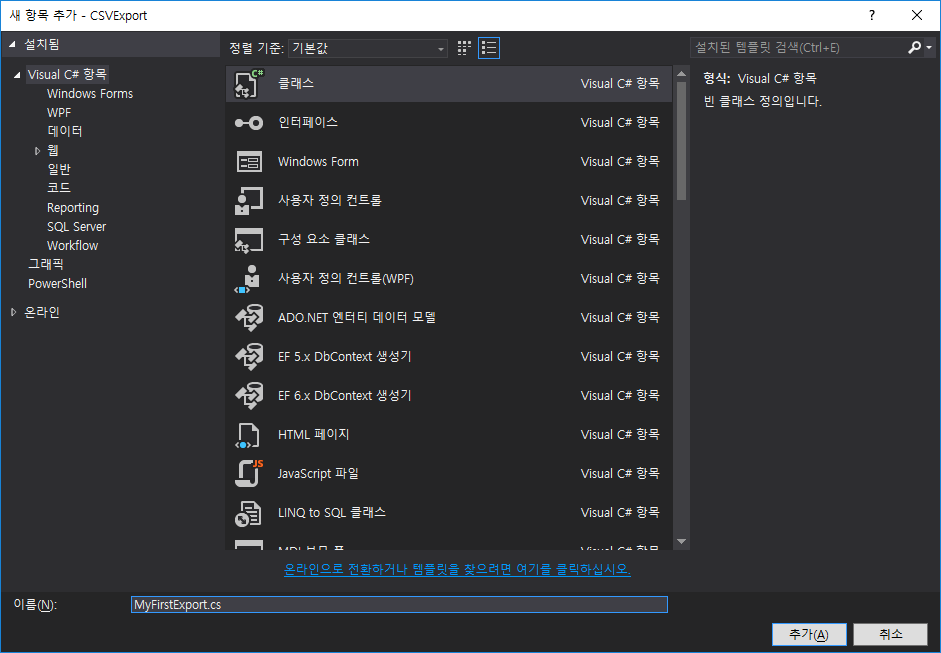
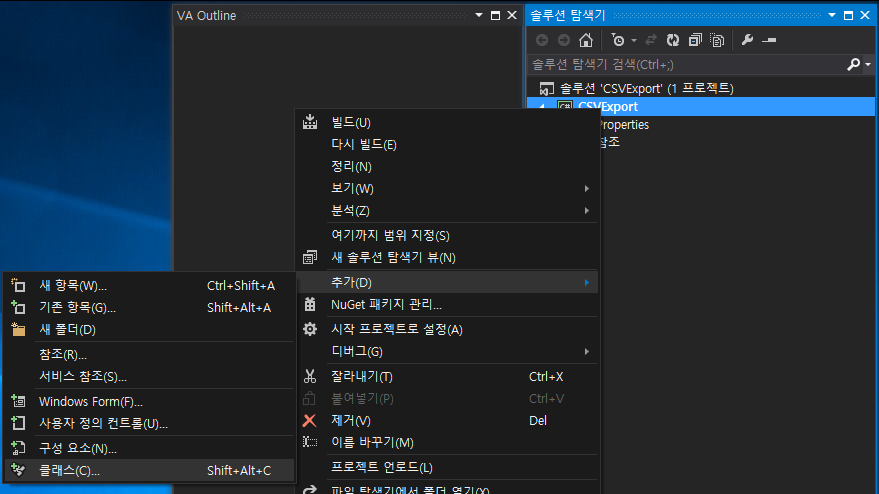
1. **프로젝트에 SDK DLL 참조 추가**

Visual Studio 에서 솔루션 탐색기 창을 열고, 생성한 프로젝트의 참조에서 참조 추가를 선택합니다.  
  
Step 3 에서 복사한 DLL 파일을 선택합니다.



1. **클래스 파일 추가 하기**

프로젝트를 생성할 때, 자동 생성된 Class1.cs 파일은 삭제하고, MyFirstExport 클래스를 추가합니다.



1. **코드 작성하기**

SDK 를 사용하기 위한 using 선언문을 추가합니다.

using SDKforAWS;

Export 모듈을 만들기 위해, 생성한 MyFirstExport 클래스가 SDKforAWS의 WExport 를 상속하도록 합니다. WExport 는 내보내기 모듈의 동작을 정의하는 추상 클래스 입니다.

public class MyFirstExport : WExport

WExport 클래스는 다음과 같이 정의되어 있습니다.

namespace SDKforAWS

{

public abstract class WExport

{

public WExport(List<WSensor> sensorlistclon);

public string getId();

/\*

\* @return DLL 에 대한 설명을 리턴해야 합니다.

\*/

public abstract string getDescription();

/\*

\* 해당 모듈이 해제되기 전에 호출됩니다.

\* 사용하던 모든 자원을 해제해야 합니다.

\*/

public abstract void Destroy();

/\*

\* 센서값이 업데이트 될때마다 호출됩니다.

\* 업데이트 주기는, 로거에서 사용자가 지정한 시간 간격입니다.

\* @param time 데이터가 수집된 시각입니다.

\* OS의 timezone 설정에 따른, 해당 timezone 의 현지 시각값입니다.

\* @param sensorList 센서 정보 및 수집된 센서의 값을 가지고 있는 WSensor 의 목록입니다.

\*/

public abstract void Update(DateTime time, List<WSensor> sensorList);

public abstract void Update(KmaDataStructure data);

/\*

\* 사용자가 모듈의 설정 메뉴를 누르면, 호출됩니다.

\* UI thread 에서 호출되므로, Form 을 사용할 수 있습니다.

\*/

public abstract void Setup();

}

}

주요 함수들을 보면 다음과 같습니다.

|  |  |
| --- | --- |
| 함수 원형 | string getDescription() |
| 설명 | Export 모듈에 대한 설명을 문자열로 리턴합니다. 사용자에게 표시되는 문자열이므로, 모듈의 특징을 요약해서 적어야 합니다. |
| 함수 원형 | void Destroy() |
| 설명 | 모듈이 해제될 때 호출되는 콜백 함수입니다. 모듈에서 사용했던 리소스들은 이 함수에서 해제해야 합니다. |
| 함수 원형 | void Update(DateTime time, List<WSensor> sensorList) |
| 설명 | 사용자가 지정한 Interval 간격으로, 데이터가 수집될 때마다 호출되는 콜백 함수입니다. 데이터가 수집된 시각 및 센서와 센서값들 정보가 인자로 넘어옵니다. |
| 함수 원형 | void Update(KmaDataStructure data) |
| 설명 | 위의 Update 와 유사하나, 기상청 고시 표준 형식으로 된 데이터가 넘어옵니다. |
| 함수 원형 | void Setup() |
| 설명 | 사용자가 모듈을 추가할 때, 모듈별 설정값들을 지정할 때 호출되는 콜백 함수입니다. UI Thread 에서 호출되므로, WinForm 을 사용해도 됩니다. 주로 Form 을 띄우고, 사용자로부터 설정값 입력을 받습니다. |

모든 Export 모듈은 List<WSensor> 형식의 인자를 받는 생성자를 제공해야 하며, WExport 의 동일한 생성자를 호출해줘야 합니다.

public MyFirstExport(List<WSensor> sensorList)

: base(sensorList)

{

}

Export 모듈에 대한 설명을 리턴하는 함수 구현입니다.

public override string getDescription()

{

return "데이터를 CSV 형식으로 내보내는 모듈입니다";

}

지금 만들려는 모듈은 사용자가 설정할 수 있는 부분이 없으므로 빈 함수로 남겨둡니다.

public override void Setup()

{

}

파일로 데이터를 저장하는 것이므로, 기상청 포맷과는 무관합니다. 기상청 포맷으로 데이터가 오는 함수도 빈 함수로 남겨둡니다.

public override void Update(KmaDataStructure data)

{

}

이제 데이터가 들어올 때마다 호출되는 Update 함수와, 모듈이 해제될 때 호출되는 Destroy 함수만 남았습니다. CSV 파일은 C:\test.csv 파일로 저장하면 될 것이고, 모듈이 load 될 때 파일을 열었다가, 모듈이 해제될 때 파일을 닫으면 되므로, 각각 생성자와 Destroy 함수에서 그 역할을 수행하면 됩니다.

파일에 데이터를 쓰기 위해, 멤버 변수로 System.IO.StreamWriter 객체를 선언하고, 생성자를 아래와 같이 수정합니다.

private System.IO.StreamWriter mFile;

public MyFirstExport(List<WSensor> sensorlistclon)

: base(sensorlistclon)

{

mFile = new System.IO.StreamWriter(@"C:\test.csv");

mFile.AutoFlush = true;

}

모듈이 해제될 때는 파일을 닫아줘야 합니다.

public override void Destroy()

{

mFile.Flush();

mFile.Close();

}

이제 데이터가 업데이트 될 때 마다, 파일에 데이터를 써 주면 됩니다.  
  
WSensor 클래스는 하나의 센서 모듈을 표현하는 클래스이고, 하나의 센서 모듈에는 여러 개의 데이터들이 포함되어 있을 수 있습니다. 하나의 데이터를 표현하는 클래스는 WSensor.Unit 클래스이며, 위의 예제에서처럼 WSensor 객체의 units 라는 멤버를 통해 접근할 수 있습니다. units 는 List<WSensor.Unit> 타입으로 선언되어 있습니다.  
예를 들어, 온습도 센서 모듈에는 온도와 습도 2개의 Unit 이 들어있습니다.

public override void Update(DateTime time, List<WSensor> sensorList)

{

String msg = time.ToString();

foreach( WSensor sensor in sensorList )

{

foreach( WSensor.Unit unit in sensor.units )

{

msg += ',' + unit.format(false);

}

}

mFile.WriteLine(msg);

}

완성된 전체 소스입니다.

using SDKforAWS;

using System;

using System.Collections.Generic;

namespace CSVExport

{

public class MyFirstExport : WExport

{

private System.IO.StreamWriter mFile;

public MyFirstExport(List<WSensor> sensorlistclon)

: base(sensorlistclon)

{

mFile = new System.IO.StreamWriter(@"C:\test.csv");

mFile.AutoFlush = true;

}

public override string getDescription()

{

return "데이터를 CSV 형식으로 내보내는 모듈입니다";

}

public override void Setup()

{

}

public override void Update(DateTime time, List<WSensor> sensorList)

{

String msg = time.ToString();

foreach( WSensor sensor in sensorList )

{

foreach( WSensor.Unit unit in sensor.units )

{

msg += ',' + unit.format(false);

}

}

mFile.WriteLine(msg);

}

public override void Update(KmaDataStructure data)

{

}

public override void Destroy()

{

mFile.Flush();

mFile.Close();

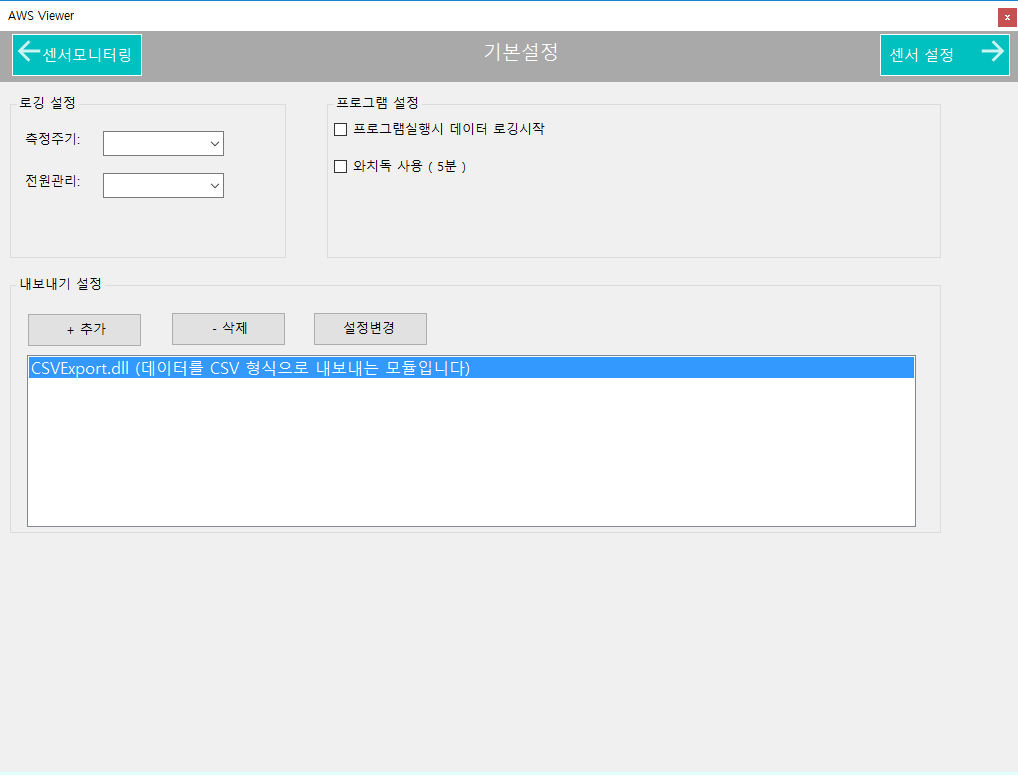
}

}

}

1. **실행**

프로젝트를 빌드하고, 프로젝트 폴더\bin\release\CSVExport.dll 파일을 로깅 프로그램과 동일한 폴더에 넣고, 로깅 프로그램을 실행합니다.  
로깅 프로그램의 기본 설정에서, 내보내기 설정 란에, 추가 버튼을 누르고,  
제작한 DLL 파일은 선택합니다.



모니터링 화면으로 돌아가서, 데이터를 수집하고, C:\test.csv 파일에 정상적으로 데이터가 기록되고 있는지 확인합니다.