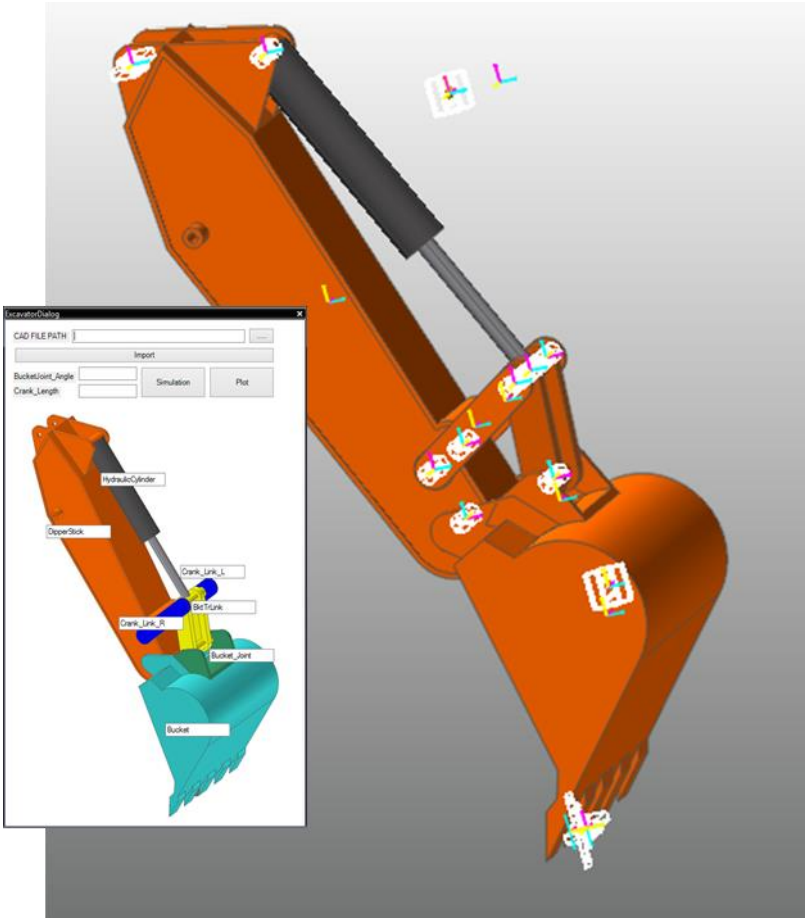
RECURDYN

Dipper Stick with Bucket Tutorial (ProcessNet General)



FunctionBay

Copyright © 2020 FunctionBay, Inc. All rights reserved.

User and training documentation from FunctionBay, Inc. is subjected to the copyright laws of the Republic of Korea and other countries and is provided under a license agreement that restricts copying, disclosure, and use of such documentation. FunctionBay, Inc. hereby grants to the licensed user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the FunctionBay, Inc. copyright notice and any other proprietary notice provided by FunctionBay, Inc. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of FunctionBay, Inc. and no authorization is granted to make copies for such purpose.

Information described herein is furnished for general information only, is subjected to change without notice, and should not be construed as a warranty or commitment by FunctionBay, Inc. FunctionBay, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the Republic of Korea and other countries. UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.

Registered Trademarks of FunctionBay, Inc. or Subsidiary

RecurDyn is a registered trademark of FunctionBay, Inc.

RecurDyn/Professional, RecurDyn/ProcessNet, RecurDyn/Acoustics, RecurDyn/AutoDesign, RecurDyn/Bearing, RecurDyn/Belt, RecurDyn/Chain, RecurDyn/CoLink, RecurDyn/Control, RecurDyn/Crank, RecurDyn/Durability, RecurDyn/EHD, RecurDyn/Engine, RecurDyn/eTemplate, RecurDyn/FFlex, RecurDyn/Gear, RecurDyn/DriveTrain, RecurDyn/HAT, RecurDyn/Linear, RecurDyn/Mesher, RecurDyn/MTT2D, RecurDyn/MTT3D, RecurDyn/Particleworks I/F, RecurDyn/Piston, RecurDyn/R2R2D, RecurDyn/RFlex, RecurDyn/RFlexGen, RecurDyn/SPI, RecurDyn/Spring, RecurDyn/TimingChain, RecurDyn/Tire, RecurDyn/Track_HM, RecurDyn/Track_LM, RecurDyn/TSG, RecurDyn/Valve

are trademarks of FunctionBay, Inc.

Edition Note

This document describes the release information of **RecurDyn V9R4**.

목차

Chapter 1

개요

목적

ProcessNet 은 .Net Frame Work 기반의 프로그램 툴로써, Microsoft Visual Studio 에서 사용하는 방법과 마찬가지로 클래스(Class), 변수(Variable) 등을 사용할 수 있습니다. 따라서 ProcessNet 을 사용하면 방대한 영역의 프로그램 기법을 사용 가능합니다. 이번 튜토리얼에서는 ProcessNet 을 사용해서 .Net Frame Work 의 사용자 인터페이스를 만드는 방법 중 하나인 Winform 을 만드는 방법을 배울 것입니다. 사용자는 Winform 을 사용해서 RecurDyn 의 모델링, 해석, 플로팅(Plotting)의 3 가지 과정을 자동화를 할 수 있습니다.

- Winform 을 사용한 사용자 인터페이스 생성
- CAD 파일을 이용한 모델 생성의 자동화
- ThisApplication 이 아닌 다른 Class 에서 ProcessNet 함수 사용
- 다이얼로그에서 모델링, 해석, 플로팅 자동화
- VSTA 가 아닌 Visual Studio 를 사용

사전 학습 내용

이 튜토리얼에서 사용된 모델은 RecurDyn 의 튜토리얼에 포함된 DOE&Batch Simulation 튜토리얼인 Dipper Stick with Bucket 을 기반으로 작성이 되었습니다. 따라서 이 튜토리얼을 시작하기에 앞서, RecurDyn 의 튜토리얼 중 하나인 Dipper Stick with Bucket 을 따라 해보고 ProcessNet VSTA 튜토리얼을 진행 후 튜토리얼을 학습하시기 바랍니다.

필요 요건

- **ProcessNet VSTA** 와 **Dipper Stick with Bucket** 튜토리얼을 익혔거나 이에 상용하는 작업을 해 본 것을 전제로 하며, 물리학에 대한 기본 지식이 있어야 합니다.

과정

이 튜터리얼은 다음의 과정들로 구성되어 있습니다. 각 과정을 완성하기까지 걸리는 시간은 아래의 표와 같습니다.

과정	시간(분)
ProcessNet 시작	5
다이얼로그 생성	15
자동 모델 생성 코드	30
모델 해석	10
Plot 자동 생성	10
총합	**70**

 예상 소요 시간

70 분

Chapter 2

ProcessNet General 시작

목적

ProcessNet General 을 사용하기 위해 **Visual Studio** 에서 **ProcessNet** 을 어떻게 시작하는지를 살펴봅시다.

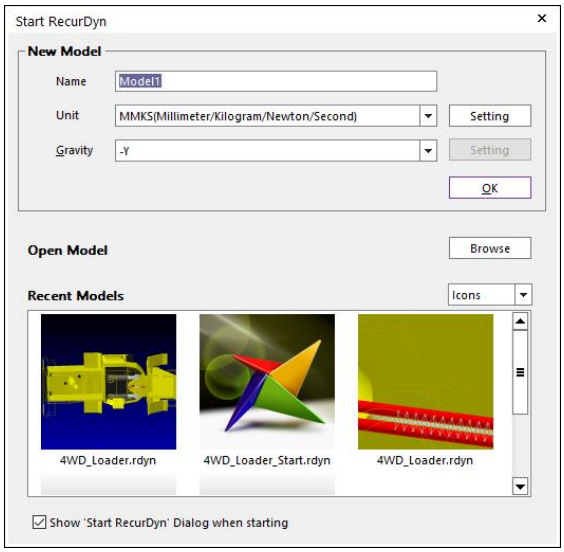
예상 소요 시간

5 분

RecurDyn 시작하기

RecurDyn 을 시작하기

1. **RecurDyn** 을 실행합니다.
2. **RecurDyn** 이 실행되면서 **Start RecurDyn** 대화상자가 나타납니다.
3. **Model Name** 입력란에 **Excavator** 을 입력합니다.
4. **Unit** 을 **MMKS** 단위계로 변경합니다.
5. **OK** 를 눌러 새 모델을 생성합니다.



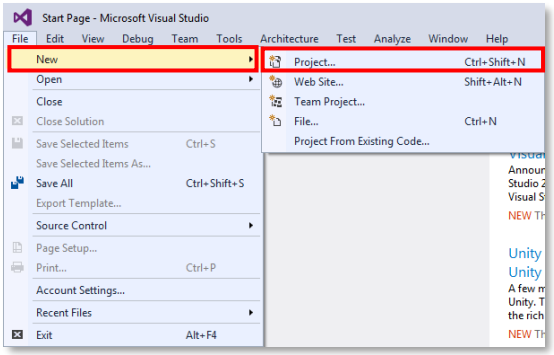
모델 저장하기

- **File** 메뉴에서, **Save As** 를 클릭하여 원하는 위치에 **Excavator.rdyn** 으로 저장합니다.

ProcessNet 시작하기

ProcessNet 시작한 후 초기화 하기

1. **ProcessNet** 의 통합 개발환경(IDE)로 들어가기 위해서 **Visual Studio 를 실행합니다**. (튜토리얼은 **Visual Studio 2015 Express** 버전을 기반으로 작성 되었습니다.)

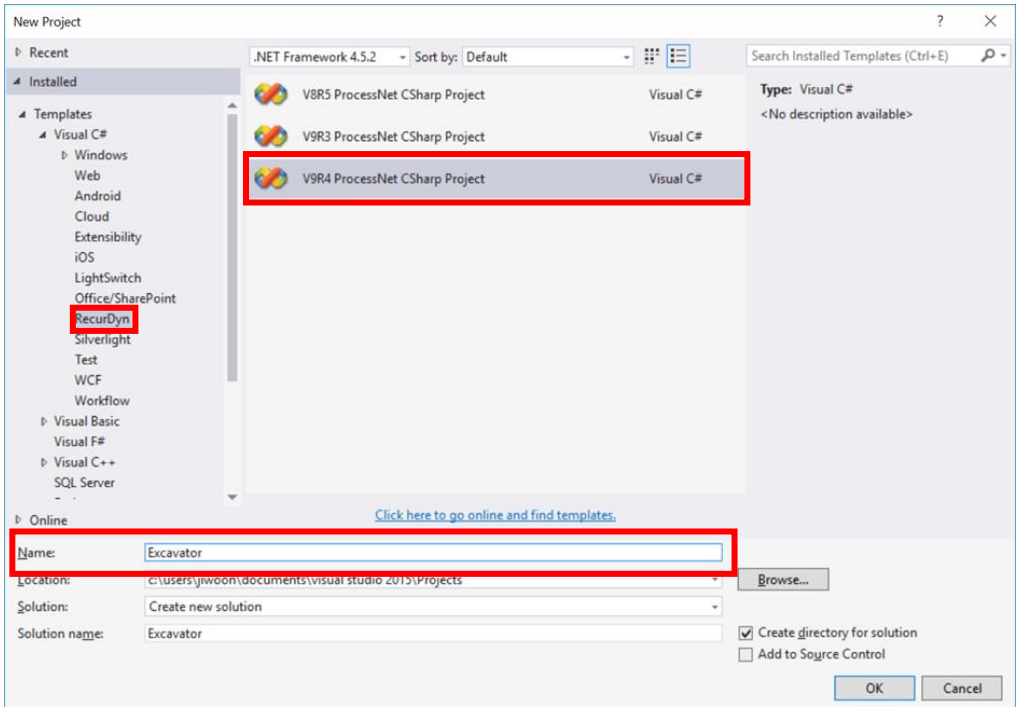


2. **Visual Studio** 가 실행되면은 **File** 메뉴에서 **New 버튼을 클릭 후 Project** 를 클릭합니다.
3. **New Project** 다이얼로그가 발생합니다 여기서 **RecurDyn** 의 버전과 맞는 **Template** 를 선택합니다.
   - 이 튜토리얼은 **V9R4** 기반이기 때문에 **V9R4** 를 선택합니다.

Note : ProcessNet Project 는 항상 RecurDyn 의 버전과 맞춰서 사용해야 합니다. 만약 버전이 다른 경우 ProcessNet 애플리케이션이 실행이 안 되는 경우가 발생합니다. 설치한 RecurDyn 의 버전에 맞춰서 Templates 이 보여집니다.

---

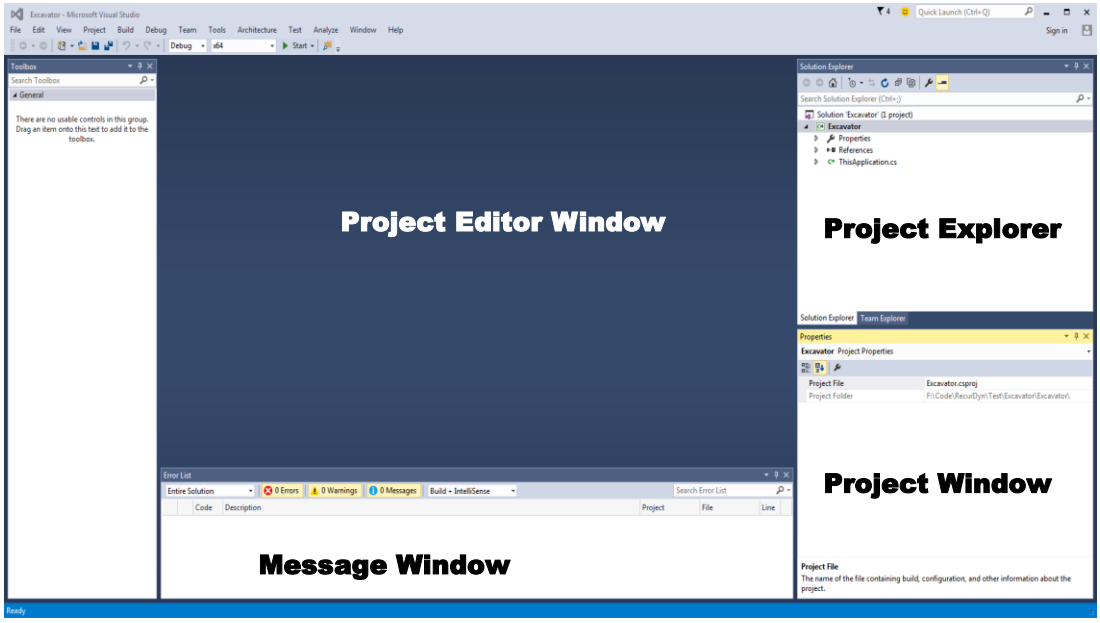
4. **Templates -> Visual C# -> RecurDyn** 을 선택을 합니다. 그러면 ProcessNet Template 인 **<RecurDyn Version> ProcessNet CSharp Project** 아이콘이 보이는데



이를 클릭합니다.

5. **Name, Solution name** 에 Excavator 을 입력하고 **Location** 은 임의의 위치를 입력해준 다음 OK 를 누릅니다.

   그리하면 아래와 같이 **Excavator Project** 가 생성이 됩니다.



6. **File - Save Excavator** 를 클릭해서 **ProcessNet** 프로젝트를 저장합니다.

   이제 ProcessNet 애플리케이션을 개발하기 위한 준비가 완료되었습니다.

Chapter

3

다이얼로그 생성

이 장에서는 다이얼로그 윈도우를 생성하고 그 디자인을 구성할 것입니다. 이 작업은 다이얼로그의 레이아웃의 설계 및 **ProcessNet** 에서 다이얼로그를 호출하기 위한 코드를 추가하는 작업을 포함합니다.

목적

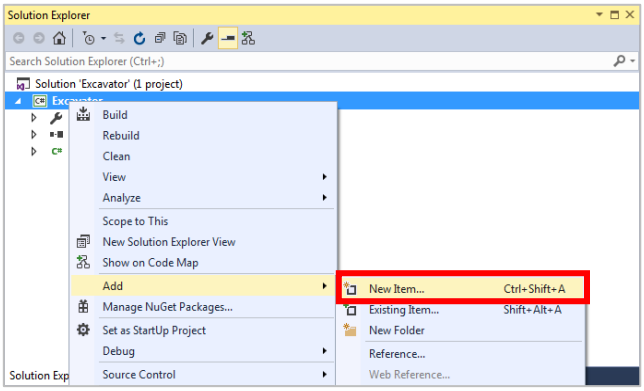
ProcessNet 에서 다이얼로그 윈도우를 생성하는 방법과 **ProcessNet** 에서 다이얼로그 윈도우를 호출하는 함수를 생성하는 방법을 살펴볼 것입니다.

예상 소요 시간

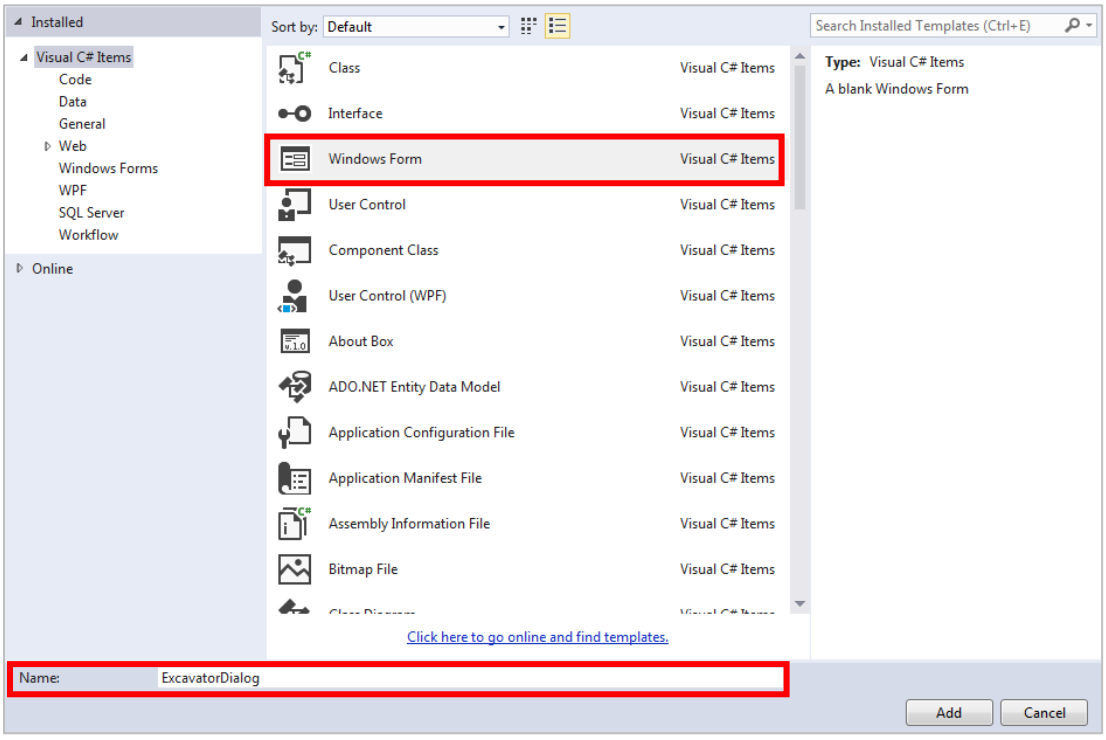
15 분

다이얼로그 생성하기

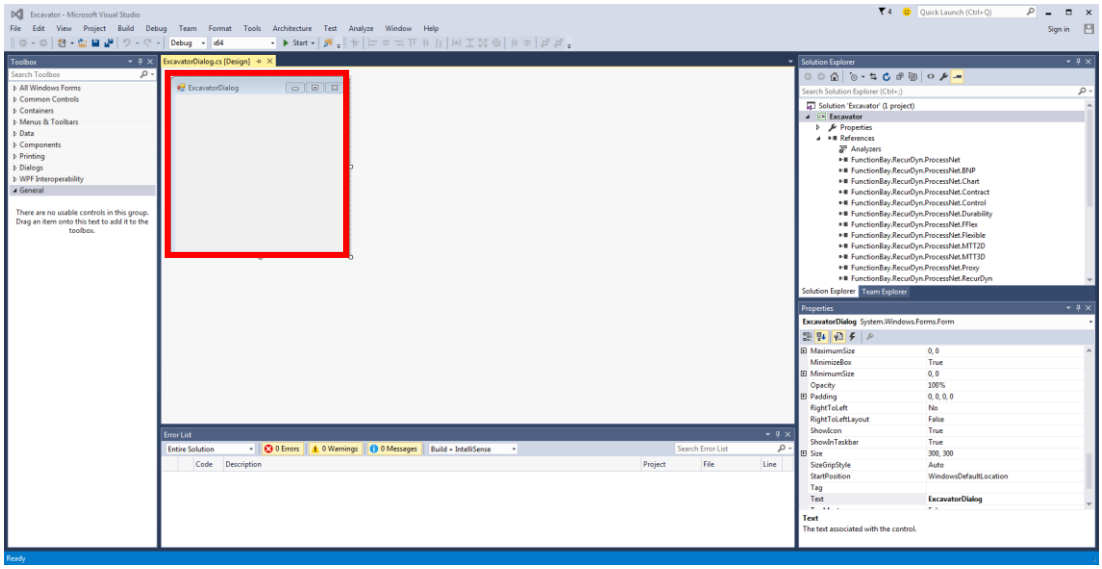
새 다이얼로그 윈도우 생성하기



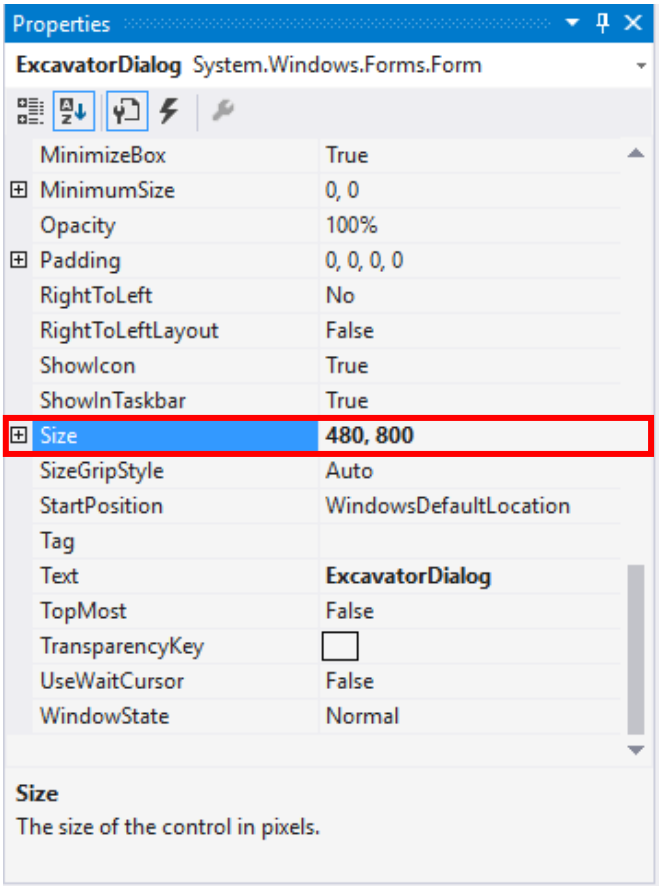
1. **Visual Studio** 의 **Project Explorer** 창에서 마우스의 오른쪽을 클릭합니다.
2. **Add – New Item** 을 클릭합니다.
3. 다음과 같이 **Add New Item** 다이얼 로그 창이 나타나면 **Windows Form** 아이콘을 클릭하고 Name 을 **ExcavatorDialog** 로 입력합니다.
4. **Add** 버튼을 클릭합니다.

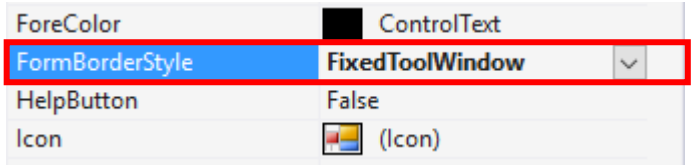


5. **Windows Form** 을 위한 설계 윈도우인 **ExcavatorDialog.cs[Design]**이 **Visual Studio Project Editor** 윈도우에 나타납니다.



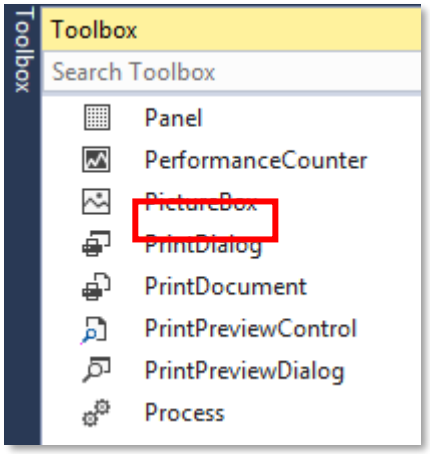
6. 화면 왼쪽상단에 있는 **ExcavatorDialog** 다이얼로그를 클릭합니다.
7. **화면 오른쪽 하단의 Properties Windows** 에 **ExcavatorDialog** 의 정보가 나타나는데 여기서 **size** 를 480, 800 으로 수정합니다.
8. 마찬가지로 **FromBorderStype** 을 **FixedToolWindow** 로 수정합니다.





9. 화면 왼쪽 상단에서 **Toolbox** 

위로커서를 이동시킵니다. 그러면 다이얼로그 윈도우 및 버튼, 기타 비슷한 제어기능을 추가할 수 있는 메뉴가 보여집니다.



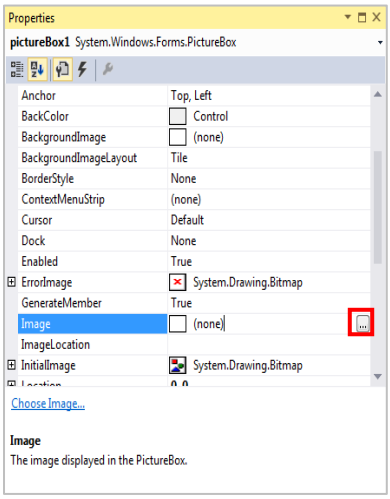
Note : Toolbox 가 보이지 않는 경우 View – Toolbox 를 클릭합니다.

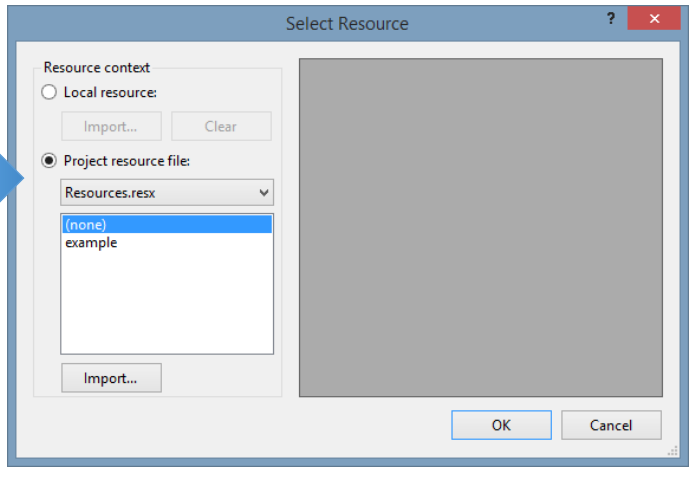
10. **Common Controls** 목록에서, **PictureBox** 을 클릭한 후 설계하고자 하는 다이얼로그의 왼쪽 상단구역으로 **Label** 을 Drag & Drop 을 수행합니다.

11. 생성한 **PicureBox** 를 선택합니다.

12. 아래의 그림처럼 오른쪽 하단의 **Properties** 창에서 **Image** 입력 창 옆에 있는 **...**를 클릭합니다.

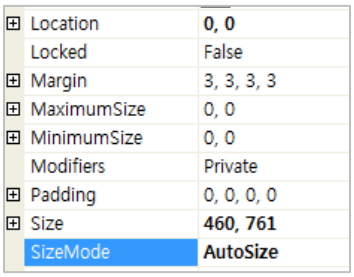
Select Resource 창이 발생합니다.



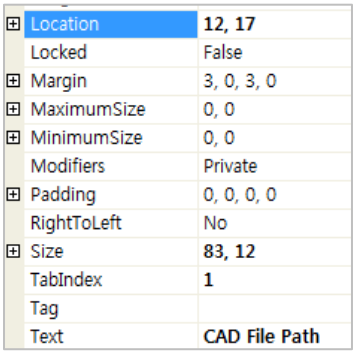


13. **Select Resource** 창에서 **Local resource** 버튼을 클릭하고 **Import** 버튼을 클릭합니다.

14. **Open** 다이얼로그가 나타나면 다이얼로그에 사용할 그림 파일을 선택합니다. (파일은 "<InstallDir>/Help/Tutorial/ProcessNet/General/Excavator/Excavator" 에 Excavator_1.png 로 존재합니다.)
15. 선택한 그림이 선택이 된 것을 확인하였으면 OK 버튼을 클릭합니다.
16. **Properies** 창에서 **SizeMode** 는 **AutoSize** 로 변경하고, **Location** 은 0,0 으로 입력합니다.



17. **Toolbox** 를 선택합니다.
18. **Common Controls** 목록에서, **Label** 을 클릭한 후 설계하고자 하는 다이얼로그의 왼쪽 상단구역으로 **Label** 을 Drag & Drop 을 수행합니다.
19. 생성한 **Label** 을 선택하고 **Properties** 창에서 **Text** 의 값에 **CAD File Path** 를 입력하고 **Location 은** 12,17 을 입력합니다.



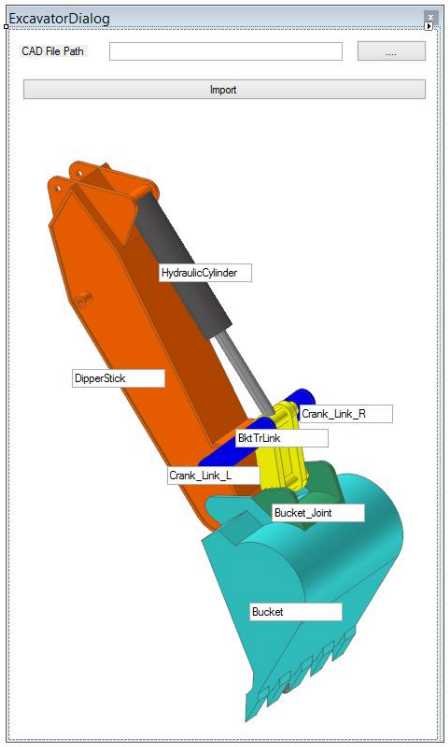
20. 다시 **ToolBox** 를 선택한 후 **Common Controls** 목록에 있는 **TextBox** 를 클릭한 후 **Label** 의 오른쪽위치에 Drag & Drop 을 수행합니다.
21. **오른쪽 하단의 Textbox1** 의 **Properties** 창에서 **Location** 은 108, 14 를 **Name** 은 **tbPath** 를 입력하고 **Size** 는 260,20 을 입력합니다.
22. **Button1, Button2, TextBox1, TextBox2, TextBox3, TextBox4, TextBox5, TextBox6** 에 대해서도 앞의 과정과 동일한 방법으로 다음 표를 참조하여 **Text** 와 **Name** 의 값을 수정합니다.

Dialog Element	Text	Name	Location	Size
Button1	...	btSearchPath	373, 12	75, 23

Button2	Import	btImport	15, 53	433, 23
TextBox1	HydraulicCylinder	tbHydraulicCylinder	161, 251	110, 20
TextBox2	DipperStick	tbDipperStick	68, 364	110, 20
TextBox3	Crank_Link_L	tbCrank_Link_L	312, 402	100, 20
TextBox4	Crank_Link_R	tbCrank_Link_R	170, 468	100, 20
TextBox5	BktTrLink	tbBktTrLink	243, 428	100, 20
TextBox6	Bucket_Joint	tbBucket_Joint	282, 508	100, 20
TextBox7	Bucket	tbBucket	258, 614	100, 20

23. 위의 과정을 수행하면 아래의 그림과 같이 다이얼로그가 완성된 것을 확인할 수 있습니다.

24. **File – Save ExcavatorDialog.cs** 버튼을 클릭해서 **Save** 를 수행합니다.



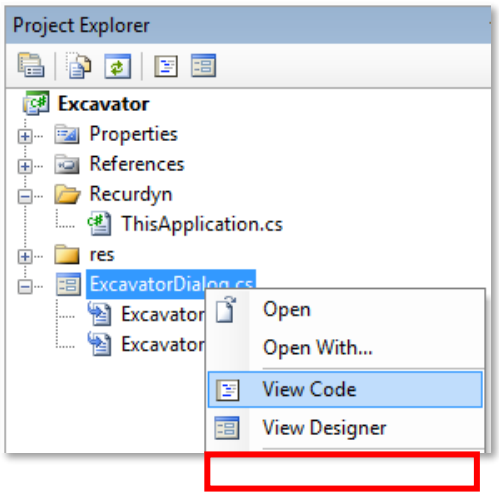
Note: Winform 으로 작성한 다이얼로그는 사용자의 PC 환경에 따라서 크기나 위치가 다소 다르게 나타날 수 있습니다.

다이얼로그의 초기환경 정의하기

지금까지 다이얼로그의 외관을 구성하였습니다. 이제 텍스트박스에 사용자가 값을 입력할 수 있도록 변수를 추가하고 버튼을 클릭 시 발생하는 이벤트를 정의할 것입니다. 그리고 ProcessNet 함수를 다이얼로그에서 사용할 수 있는 방법을 배울 것입니다.

다이얼로그 윈도우의 초기환경 정의하기

1. **Project Explorer** 에서 **ExcavatorDialog.cs** 을 선택한 후, 오른쪽 버튼을 클릭합니다. **View Code** 를 선택하면 **ExcavatorDialog.cs** 의 소스 코드가 **Project** 편집창에 나타납니다.



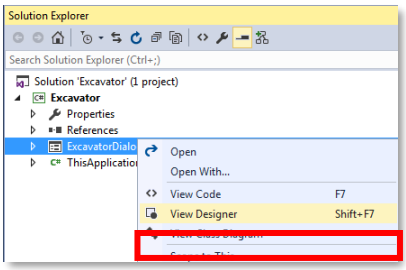
2. 편집창에 다이얼로그 윈도우에 사용할 변수를 입력합니다.

   - **FunctionBay.RecurDyn.ProcessNet** 는 **ProcessNet** 의 함수를 사용하기 위한 **Reference** 입니다.
   - **IApplication** 은 RecurDyn 을 인식하는데 사용하는 **Interface** 입니다.
   - **strFilePath 와 StrExcavatorPartName 은 Excavator 의 부재의 이름과 부재에** 대응하는 CAD 파일 혹은 Subsystem 파일의 경로입니다.

```
using System.Windows.Forms;
using FunctionBay.RecurDyn.ProcessNet;

namespace Excavator
{
	public partial class ExcavatorDialog : Form
	{
		IApplication application;
		string strFilePath;
		string[,] strExcavatorPartName = new string[7, 2];
		public ExcavatorDialog(IApplication app)
		{
			InitializeComponent();
			application = app;
		}
```

3. **Project Explorer** 의 **ExcavatorDialog.cs** 파일을 클릭하고 오른쪽 버튼을 클릭 후, **View Designer** 를 클릭하면 앞서 작성한 다이얼로그가 나타납니다.



4. 다이얼로그 윈도우 상에서 **...** 버튼을 더블 클릭하여 버튼을 더블 클릭할 경우 호출되는 함수를 생성합니다.
5. 자동으로 생성된 새로운 함수 안에 아래와 같이 코드를 삽입합니다.
   - **Folder Dialog** 창에서 Folder 의 경로를 가지고 오는 코드입니다.
   - **...** 버튼을 클릭하면 **btSearchPatch_Click()** 함수가 실행하면서 **Folder Dialog** 창이 발생합니다.

```
private void btSearchPath_Click(object sender, EventArgs e)
{
        FolderBrowserDialog dialog = new FolderBrowserDialog();
        dialog.ShowDialog();
        this.tbPath.Text = dialog.SelectedPath;
}
```

6. **Project Explorer** 의 **ExcavatorDialog.cs** 파일을 클릭하고 오른쪽 버튼을 클릭 후, **View Designer** 버튼을 클릭한 후, 다이얼로그 윈도우 상의 **Import** 버튼을 더블 클릭하여 **btImport_Click() 함수**를 생성합니다.
7. **btImport_Click ()** 함수 아래에 **UpdateDB()**라는 이름의 새로운 함수를 생성한 후, 아래의 소스코드와 동일하게 입력합니다.
   - **UpdateDB()**는 **다이얼로그 윈도우**에 있는 **textbox** 에 입력된 변수를 저장하는 함수입니다.
   - **This.tbPath.text 는 tbPath 라는 이름의 textbox 에 입력된 값을 가지고 오는 프로시저입니다.**

- **strExcavatorPartName** 는 2 차원 배열로 CAD 파일 및 Subsystem 파일의 경로와 이름을 저장합니다.

```
private void UpdateDB()
{
            strFilePath = this.tbPath.Text;
            strExcavatorPartName[0, 0] = this.tbDipperStick.Text.ToString();
            strExcavatorPartName[0, 1] = strFilePath + @"\" + this.tbDipperStick.Text.ToString() + ".x_t";
            strExcavatorPartName[1, 0] = this.tbCrank_Link_L.Text.ToString();
            strExcavatorPartName[1, 1] = strFilePath + @"\" + this.tbCrank_Link_L.Text.ToString() + ".x_t";
            strExcavatorPartName[2, 0] = this.tbCrank_Link_R.Text.ToString();
            strExcavatorPartName[2, 1] = strFilePath + @"\" + this.tbCrank_Link_R.Text.ToString() + ".x_t";
            strExcavatorPartName[3, 0] = this.tbBucket.Text.ToString();
            strExcavatorPartName[3, 1] = strFilePath + @"\" + this.tbBucket.Text.ToString() + ".x_t";
            strExcavatorPartName[4, 0] = this.tbBucket_Joint.Text.ToString();
            strExcavatorPartName[4, 1] = strFilePath + @"\" + this.tbBucket_Joint.Text.ToString() + ".x_t";
            strExcavatorPartName[5, 0] = this.tbBktTrLink.Text.ToString();
            strExcavatorPartName[5, 1] = strFilePath + @"\" + this.tbBktTrLink.Text.ToString() + ".rdsb";
            strExcavatorPartName[6, 0] = this.tbHydraulicCylinder.Text.ToString();
            strExcavatorPartName[6, 1] = strFilePath + @"\" + this.tbHydraulicCylinder.Text.ToString() +
".rdsb";
}
```

8. 자동으로 생성된 새로운 함수 안에 아래와 같이 코드를 입력합니다.

   - 다이얼로그에서 **Import** 버튼을 클릭 시 생성한 **UpdateDB()** 함수를 수행합니다.

```
private void btImport_Click(object sender, EventArgs e)
{
            UpdateDB();
}
```

9. **File** 메뉴에서 **Save ExcavatorDialog.cs** 를 클릭해서 **Save** 를 수행합니다.

애플리케이션을 실행했을 때 다이얼로그 윈도우 나타내기

RecurDyn 에서 ProcessNet 애플리케이션을 실행할 때 다이얼로그가 나타나게 하는 법과 해당 다이얼로그가 RecurDyn 에 종속되게 하는 법을 배울 것입니다.

애플리케이션을 실행했을 때 다이얼로그 윈도우 나타내기

1. **Project Explorer** 창에서 **ThisApplication.cs** 파일을 더블 클릭합니다.

2. **ThisApplication.cs** 에서 다음과 같이 줄이 그어진 **HelloProcessNet()** 함수와 **CreateBodyExample()** 함수를 삭제합니다. (예제로서 자동생성된 함수입니다.)

```
public void HelloProcessNet()
{
        //application is assigned at Initialize() such as
        //application = RecurDynApplication as IApplication;
        application.PrintMessage("Hello ProcessNet");
        application.PrintMessage(application.ProcessNetVersion);
}

public void CreateBodyExample()
{
        refFrame1 = modelDocument.CreateReferenceFrame();
        refFrame1.SetOrigin(100, 0, 0);

        refFrame2 = modelDocument.CreateReferenceFrame();
        refFrame2.SetOrigin(0, 200, 0);

        IBody body1 = model.CreateBodyBox("body1", refFrame1, 150, 100, 100);
        application.PrintMessage(body1.Name);
        IBody body2 = model.CreateBodySphere("body2", refFrame2, 50);
        application.PrintMessage(body2.Name);
}
```

3. 아래와 같이 **Run()** 함수를 작성합니다.

   - **ExcavatorDialog** 의 새로운 인스턴스를 생성하였습니다.
   - Application 과 MainWindow 의 값을 **ExcavatorDialog** 클래스에 전달해줍니다.

Note: application 과 MainWindow 를 전달해야 Winform 에서 ProcessNet 매서드를 사용할 수 있습니다.

```
public void Run()
{
        ExcavatorDialog DialogRun = new ExcavatorDialog(application);
        DialogRun.ShowDialog();
}
```

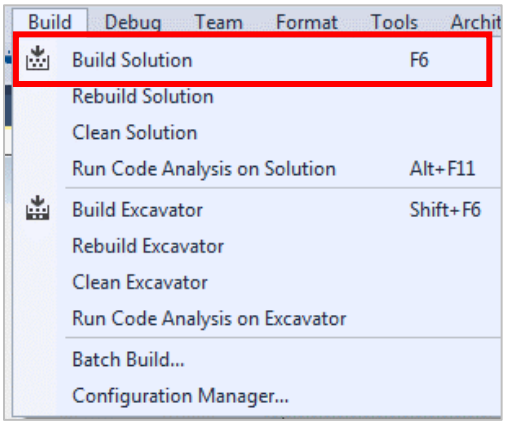
4. **File** 메뉴에서 **Save ThisApplication.cs** 를 클릭해서 **Save** 를 수행합니다.

다이얼로그 윈도우의 테스트

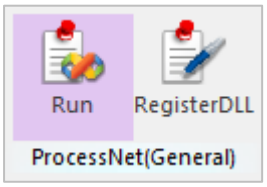
지금까지 작성한 애플리케이션이 정상적으로 작동하는지 테스트합니다.

애플리케이션 실행하기

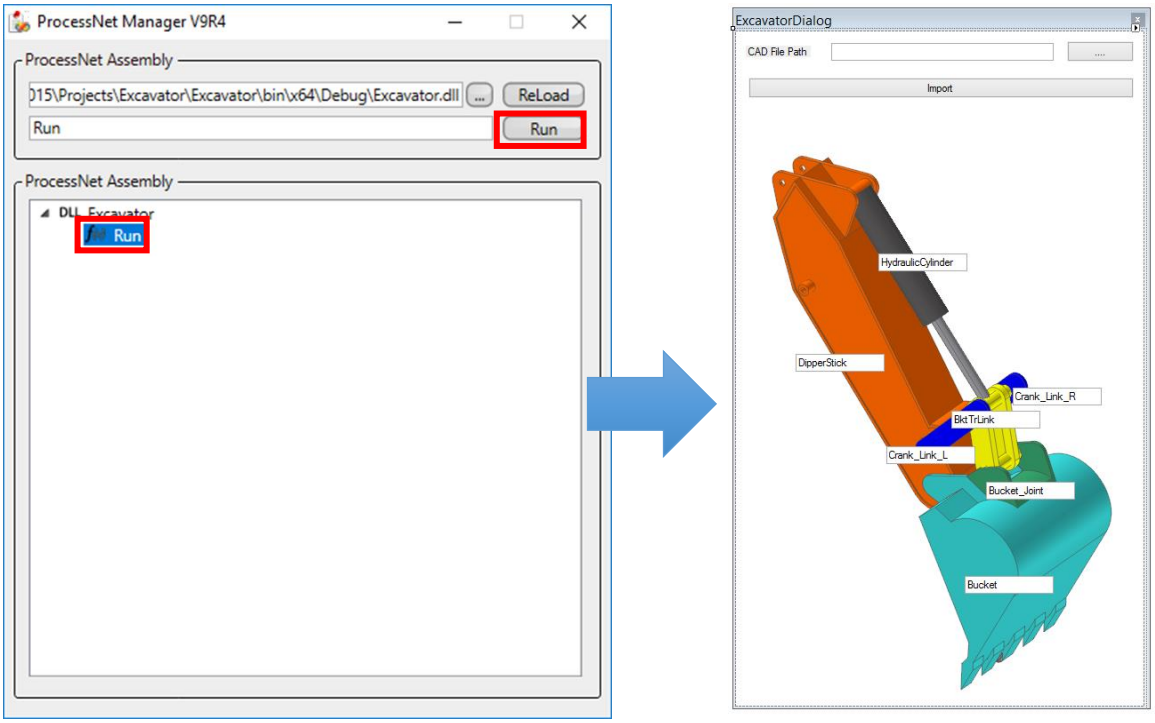
1. **IDE** 창 하단의 **Error List** 창에서 에러나 경고가 없는지를 확인합니다. 에러나 경고가 있다면, 목록을 확인하여 수정합니다. **Build** 매뉴에서 **Build Solution** 을 선택합니다.



2. **RecurDyn** 의 **Customize** 탭의 **ProcessNet(General)** 그룹에서 **Run** 을 선택합니다.



3. **ProcessNet Manager** 다이얼로그 하단의 트리컨트롤에서 **Excavator** 하위의 **Run** 을 클릭합니다.
4. **ProcessNet Manager** 다이얼로그에 있는 **Run** 버튼을 클릭합니다.



5. 생성한 다이얼로그가 보여집니다.
6. 실행이 되는 것을 확인하였으면 다이얼로그를 닫습니다.
7. **ProcessNet Manager** 다이얼로그를 닫습니다.

Chapter 4

자동 모델 생성 코드

목적

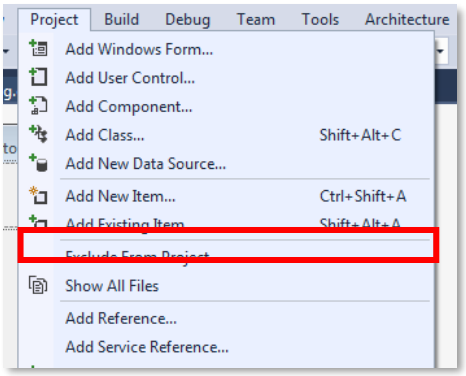
이 장에서는 새로운 Class 를 만들고 여기에 **ProcessNet** 함수를 작성합니다. 그 후, 이 함수를 이전 장에서 생성한 다이얼로그에서 호출하는 방법을 살펴볼 것입니다.

예상 소요 시간

30 분

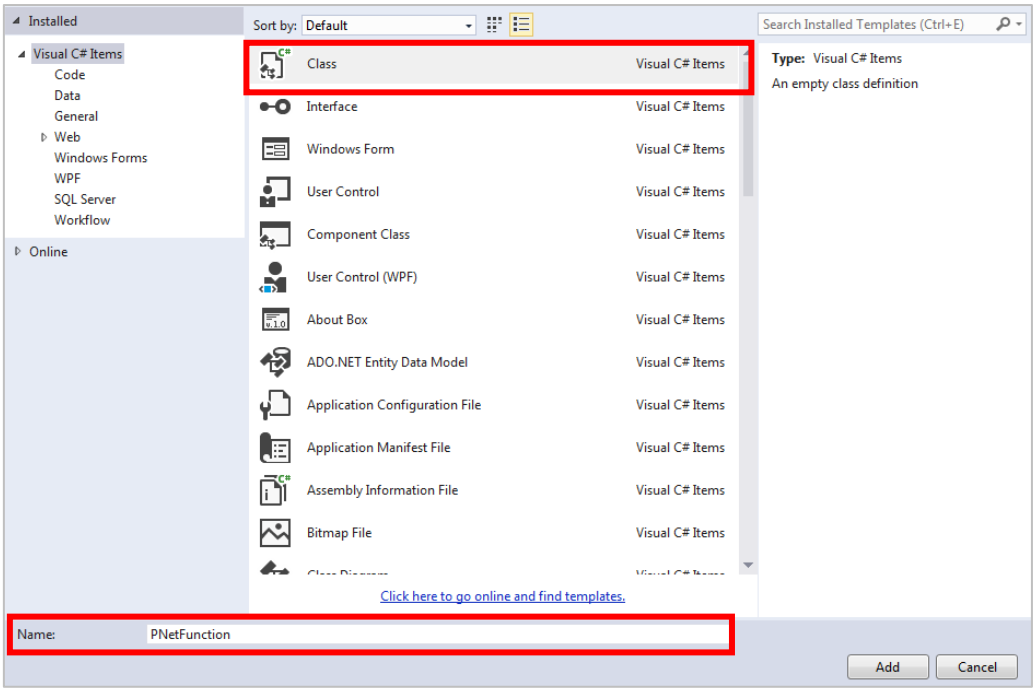
새로운 **Class** 생성

새로운 Class 를 생성하고 Class 내부에
ProcessNet 함수를 입력하는 방법을 배울 것입니다.



새로운 **Class** 생성

1. **Visual Studio** 에서 **Project – Add New Item** 버튼을 클릭합니다.
2. **Add New Item** 다이얼로그가 나타나면 **Templates** 에서 **Class** 를 선택하고 **Name** 은 **PNetFunction** 으로 입력한 후 Add 를 누릅니다.



3. **Visual Studio Project** 편집창에 **PNetFunction.cs** 가 나타나면 아래의 코드를 입력합니다. . 입력된 코드는 **ProcessNet** 함수를 실행할 때 쓰이는 기본적인 변수를 초기화 시켜줍니다.
   - **IApplication**: RecurDyn 을 인식하는데 사용
   - **IModelDocument**: RecurDyn 에서의 모델 문서
   - **ISubsystem**: 모델 문서에서 사용되고 있는 Subsystem
   - **IReferenceFrame**: RecurDyn 의 RefernceFrame

- **IPlotDocument**: RecurDyn Plot 문서

```
using FunctionBay.RecurDyn.ProcessNet;
namespace Excavator
{
	class PNetFunction
	{
		static public IApplication application;
		public IModelDocument modelDocument = null;
		public IPlotDocument plotDocument = null;
		public ISubSystem model = null;

		public IReferenceFrame refFrame1 = null;
		public IReferenceFrame refFrame2 = null;

		public PNetFunction(IApplication app)
		{
			application = app;
		}
	}
}
```

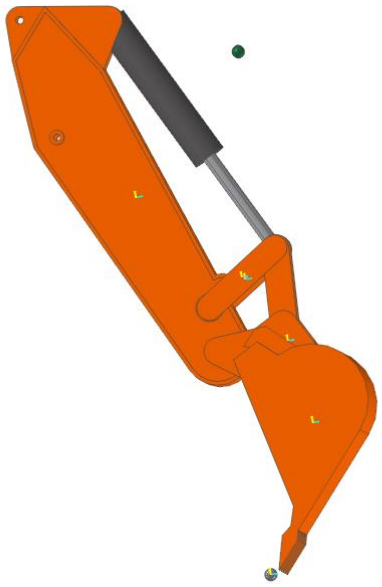
4. **File** 메뉴에서 **Save PNetFunction.cs** 를 클릭해서 save 를 수행합니다.

모델의 생성

Excavator 모델을 자동으로 생성하는 방법을 살펴볼 것입니다.

Body Import 하기

1. **CAD** 파일과 **Subsystem** 파일을 사용해서 아래와 같은 **Excavator** 모델을 생성하는 코드를 작성합니다.



```
public void Import(string[,] strExcavatorPartName)
{
}
```

2. **Import()** 함수를 생성합니다.

3. 다음으로 방금 생성한 **Import()**함수에 다음의 변수 선언을 삽입합니다.

```
modelDocument = application.ActiveModelDocument;
model = modelDocument.Model;

refFrame1 = modelDocument.CreateReferenceFrame();
refFrame1.SetOrigin(0, 0, 0);
refFrame2 = modelDocument.CreateReferenceFrame();
refFrame2.SetOrigin(0, 0, 0);
```

4. 방금 추가한 변수 선언 다음에 모델의 각 부품의 CAD 파일인 *.x_t 및 Subsystem 파일인 *.rdsb 을 Import 하기 위한 Loop 문을 추가합니다.

   - **FileImport()**는 CAD 파일 및 Subsystem 파일을 Import 해주는 함수입니다.
   - **Cank Link R** 은 CAD 파일로 Import 하지 않고 생성해주기 때문에 continue 를 사용해서 다음 Loop 로 넘어갑니다.

```
for(int iCount = 0; iCount < 7; iCount++)
{
        if( iCount == 2)
        continue;
        model.FileImport(strExcavatorPartName[iCount,1]);
}
```

5. SubSystemCollection Property 를 사용해서 모델 내에 있는 Subsystem 의 List 를 작성하고

```
ISubSystemCollection SubCollection01 = model.SubSystemCollection;
ISubSystem Sub01 = SubCollection01[0];
ISubSystem Sub02 = SubCollection01[1];
```

   Sub01, Sub02 로 선언하는 코드를 입력합니다.

6. **GetEntity()** 함수를 사용해서 Import 된 Body 를 검색하는 코드를 추가합니다.

   - **GetEntity()** 함수는 **RecurDyn** 의 Entity 들을 검색해줍니다.

- 기본적으로 **IGeneric** 으로 반환되며 원하는 Entity 의 종류에 맞춰서 형 변환을 해줄 수 있습니다.

```
IBody BodyDipperStick = model.GetEntity(strExcavatorPartName[0,0]) as IBody;
IBody BodyCrankLinkL = model.GetEntity(strExcavatorPartName[1, 0]) as IBody;
IBody BodyBucket = model.GetEntity(strExcavatorPartName[3, 0]) as IBody;
IBody BodyJoint = model.GetEntity(strExcavatorPartName[4, 0]) as IBody;

IBody BodyBktTrLink_CylRod_Cylinder = Sub01.GetEntity("BktTrLink_CylRod_Cylinder") as IBody;
IBody BodyBktTrLink_Bucket_BktTrLink_Cylinder = Sub01.GetEntity("Bucket_BktTrLink_Cylinder") as
IBody;
IBody BodyBktTrLink_Right_Link = Sub01.GetEntity("Right_Link") as IBody;
IBody BodyHydraulicCylinder_Cylinder =  Sub02.GetEntity("Cylinder") as IBody;
IBody BodyHydraulicCylinder_Rod = Sub02.GetEntity("Rod") as IBody;
IBody BodySub02Mother = Sub02.GetEntity("MotherBody") as IBody;
```

7. **Crank_Link_R** 라는 이름의 **Link Body** 와 **Bucket_Joint Body** 에 Marker 를 생성하는 코드를 입력합니다.

   - **Crank_Link_R** 의 Second Point 의 값에 PP 를 입력해서 사용자가 자세를 조절할 수 있도록 해야 하므로 CAD 파일을 Import 하는 대신에 Marker 를 새로 생성해줍니다.

```
IBody BodyCrankLinkR = model.CreateBodyLinkWithRadius(strExcavatorPartName[2, 0], new double[]
{ 5506.1017, -495.8525, 2231.9958 }, new double[] { 5606.1017, -495.8525, 2231.9958 },100,100, 35);
BodyCrankLinkR.Graphic.Color = 26367;

refFrame1.SetOrigin(6060.3717, -207.8525, 1993.4767);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 180, 90, 90);
IMarker Marker01 = BodyJoint.CreateMarker("Marker1", refFrame1);
```

8. **Assembly** 모드의 **Ground** 를 **BodyGround** 로 선언해 줍니다.

```
IBody BodyGround = model.Ground;
```

9. **DummyBody** 를 생성합니다.

```
refFrame1.SetOrigin(5579.2685, -207.8525, 62.560441);
IBody BodyDummyBucketTip = model.CreateBodyEllipsoid("BucketTip", refFrame1, 50, 50, 50);
refFrame1.SetOrigin(6100, -207.8525, 4200);
IBody BodyDummyDrivingForceBody = Sub02.CreateBodyEllipsoid("DrivingForceBody", refFrame1, 50, 50,
50);
```

10. **File** 메뉴에서 **Save PNetFunction.cs** 를 클릭해서 save 를 수행합니다.

SubEntity 생성

1. **Excavator** 에서 사용할 **SubEntity** 를 생성하겠습니다.
2. 방금 전 과정에서 작성한 Import()함수에 아래의 PV 를 생성하는 코드를 추가합니다.

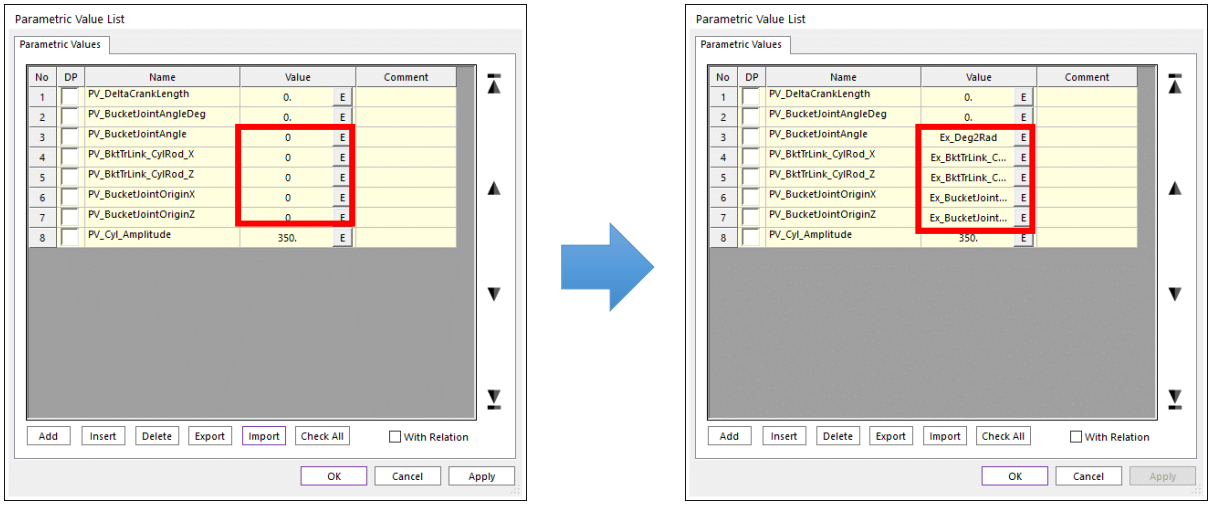
   (SubEntity 생성부터 뒤쪽의 Variable Equation 생성까지는 모두 Import()함수 내에 이어서 코드를 추가합니다.)

```
IParametricValue PV_DeltaCrankLength = model.CreateParametricValue("PV_DeltaCrankLength", 0);
IParametricValue PV_BucketJointAngleDeg = model.CreateParametricValue("PV_BucketJointAngleDeg", 0);
IParametricValue PV_BucketJointAngle = model.CreateParametricValue("PV_BucketJointAngle", 0);
IParametricValue PV_BktTrLink_CylRod_X = model.CreateParametricValue("PV_BktTrLink_CylRod_X", 0);
IParametricValue PV_BktTrLink_CylRod_Z = model.CreateParametricValue("PV_BktTrLink_CylRod_Z", 0);
IParametricValue PV_BucketJointOriginX = model.CreateParametricValue("PV_BucketJointOriginX", 0);
IParametricValue PV_BucketJointOriginZ = model.CreateParametricValue("PV_BucketJointOriginZ", 0);
IParametricValue PV_Cyl_Amplitude = model.CreateParametricValue("PV_Cyl_Amplitude", 350);
```

3. PV 를 생성하는 코드 아래에 아래의 Expression 를 생성하는 코드를 추가합니다

```
IExpression Ex_Deg2Rad = model.CreateExpression("Ex_Deg2Rad", "PV_BucketJointAngleDeg*PI/180");
IExpression Ex_BktTrLink_CylRod_X = model.CreateExpression("Ex_BktTrLink_CylRod_X",
"COS(0.291724-ACOS(((704.58305+PV_DeltaCrankLength)*(704.58305+PV_DeltaCrankLength)-
10996911)/(-10963694)))*965.471+SIN(0.291724-
ACOS(((704.58305+PV_DeltaCrankLength)*(704.58305+PV_DeltaCrankLength)-10996911)/(-
10963694)))*(-2034.522)+5139.0782");
IExpression Ex_BktTrLink_CylRod_Z = model.CreateExpression("Ex_BktTrLink_CylRod_Z", "-
SIN(0.291724-ACOS(((704.58305+PV_DeltaCrankLength)*(704.58305+PV_DeltaCrankLength)-
10996911)/(-10963694)))*965.471+COS(0.291724-
ACOS(((704.58305+PV_DeltaCrankLength)*(704.58305+PV_DeltaCrankLength)-10996911)/(-
10963694)))*(-2034.522)+4638.4021");
IExpression Ex_BucketJointOriginX = model.CreateExpression("Ex_BucketJointOriginX", "(1-
COS(PV_BucketJointAngle))*6191.0835-SIN(PV_BucketJointAngle)*1340.8818");
IExpression Ex_BucketJointOriginZ = model.CreateExpression("Ex_BucketJointOriginZ",
"SIN(PV_BucketJointAngle)*6191.0835+(1-COS(PV_BucketJointAngle))*1340.8818");
IExpression Ex_BucketTipLoad = model.CreateExpression("Ex_BucketTipLoad", "0");
IExpression Ex_DrivingForce = Sub02.CreateExpression("Ex_DrivingForce", "0");
Ex_DrivingForce.Arguments = new string[] { "Cylinder.Marker1@HydraulicCylinder",
"Rod.Marker1@HydraulicCylinder" };
Ex_DrivingForce.Text = "FZ(1,2,2)";
```

4. Expression 를 생성하는 코드 아래에 PV 의 Value 을 생성하는 코드를 추가합니다.
   - Expression 을 생성한 뒤 PV 의 Value 값에 입력합니다. 아래의 코드가 실행되면 아래의 그림과 같이 PV 의 Value 값이 Expression 으로 변경됩니다.




```
PV_BucketJointAngle.Text = "Ex_Deg2Rad";
PV_BktTrLink_CylRod_X.Text = "Ex_BktTrLink_CylRod_X";
PV_BktTrLink_CylRod_Z.Text = "Ex_BktTrLink_CylRod_Z";
PV_BucketJointOriginX.Text = "Ex_BucketJointOriginX";
PV_BucketJointOriginZ.Text = "Ex_BucketJointOriginZ";
```

5. PV 의 Value 를 설정하였으면 PP 를 생성하는 코드를 추가합니다.

```
IParametricPoint PP_CrankL_BktTrLink = model.CreateParametricPointWithText("PP_CrankL_BktTrLink",
"PV_BktTrLink_CylRod_X,80.147498,PV_BktTrLink_CylRod_Z", null);
IParametricPoint PP_Bucket_BktTrLink = model.CreateParametricPoint("PP_Bucket_BktTrLink", new double[] { 0,
0, 0 }, null);
IParametricPoint PP_BktTrLink_Rod = model.CreateParametricPointWithText("PP_BktTrLink_Rod",
"PV_BktTrLink_CylRod_X,-207.85255,PV_BktTrLink_CylRod_Z", null);
IParametricPoint PP_DipperStick_Cyl = model.CreateParametricPoint("PP_DipperStick_Cyl", new double[]
{ 5139.0782, -207.85255, 4638.4021 }, null);
IParametricPoint PP_BucketJointOrigin = model.CreateParametricPointWithText("PP_BucketJointOrigin", "
PV_BucketJointOriginX,0.,PV_BucketJointOriginZ", null);
IParametricPoint PP_CrankR_BktTrLink = model.CreateParametricPointWithText("PP_CrankR_BktTrLink",
"PV_BktTrLink_CylRod_X,-495.8525, PV_BktTrLink_CylRod_Z", null);
PP_Bucket_BktTrLink.RefMarker = Marker01;
```

6. PP 를 생성하는 코드 아래에 아래의 PPC 와 PVC 를 생성하는 코드를 추가합니다.

```
IParametricPointConnector PPC_Bucket_BktTrLink =
model.CreateParametricPointConnector("PPC_Bucket_BktTrLink");
PPC_Bucket_BktTrLink.Point.ParametricPoint = PP_Bucket_BktTrLink;
IParametricPointConnector PPC_BktTrLink_CylRod =
model.CreateParametricPointConnector("PPC_BktTrLink_CylRod");
PPC_BktTrLink_CylRod.Point.ParametricPoint = PP_BktTrLink_Rod;
IParametricPointConnector PPC_Cyl_End = model.CreateParametricPointConnector("PPC_Cyl_End");
PPC_Cyl_End.Point.ParametricPoint = PP_DipperStick_Cyl;
IParametricPointConnector PPC_Rod_End = model.CreateParametricPointConnector("PPC_Rod_End");
PPC_Rod_End.Point.ParametricPoint = PP_BktTrLink_Rod;

IParametricValueConnector PVC_Cyl_Amplitude = model.CreateParametricValueConnector("PVC_Cyl_Amplitude");
PVC_Cyl_Amplitude.Value.ParametricValue = PV_Cyl_Amplitude;
```

7. PPC 와 PVC 를 생성하는 코드 아래에 Cank_Link_R 의 Second Point 와 Normal Direction 를 수정하는 코드를 추가합니다.

```
IGeometryLink GeoLink1 = BodyCrankLinkR.GetEntity("Link1") as IGeometryLink;
GeoLink1.SecondParametricPoint = PP_CrankR_BktTrLink;
GeoLink1.SetNormalDirection(0, 1, 0);
```

8. **File** 메뉴에서 **Save PNetFunction.cs** 를 클릭해서 save 를 수행합니다..

Joint 생성

1. **Excavator** 에서 사용할 **Joint** 를 생성하겠습니다.

이전 과정에서 작성한 **Import()** 함수에 아래의 **Fixed Joint** 를 생성하는 코드를 추가합니다.

```
refFrame1.SetOrigin(4440.16, -387.85255, 4768.1811);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0, 90, 0);
IJointFixed FixedJoint_Dipper_Ground = model.CreateJointFixed("Fixed_Dipper_Ground", BodyGround,
BodyDipperStick, refFrame1);

refFrame1.SetOrigin(6191.0835, -207.8525, 1340.8818);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZYX, 0, 0, 0);
IJointFixed FixedJoint_Bucket_BucketJoint = model.CreateJointFixed("Fixed_Bucket_BucketJoint", BodyBucket,
BodyJoint, refFrame1);
FixedJoint_Bucket_BucketJoint.BaseMarker.RefFrame.EulerAngle.Beta.ParametricValue = PV_BucketJointAngle;

refFrame1.SetOrigin(5679.2685, -207.8525, 62.560441);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0, 90, 0);
IJointFixed FixedJoint_BucketTip_Bucket = model.CreateJointFixed("Fixed_BucketTip_Bucket",
BodyDummyBucketTip, BodyBucket, refFrame1);

refFrame1.SetOrigin(6100, -207.8525, 4200);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0, 0, 0);
IJointFixed FixedJoint_DrivingForceBody = model.CreateJointFixed("Fixed_DrivingForceBody", BodyGround,
BodyDummyDrivingForceBody, refFrame1);
```

2. **Revolute Joint** 를 생성하는 코드를 추가합니다.

```
refFrame1.SetOrigin(5506.1017, 62.147449, 2231.9959);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 180, 90, 90);
IJointRevolute RevJoint_Dipper_Crank_L = model.CreateJointRevolute("Rev_Dipper_Crank_L", BodyCrankLinkL,
BodyDipperStick, refFrame1);

refFrame1.SetOrigin(5504.8615, -207.8525, 1879.9098);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 180, 90, 90);
IJointRevolute RevJoint_Dipper_Bucket = model.CreateJointRevolute("Rev_Dipper_Bucket", BodyDipperStick,
BodyBucket, refFrame1);

refFrame1.Origin.ParametricPoint = PP_CrankL_BktTrLink;
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 180, 90, 90);
IJointRevolute RevJoint_BktTrLink_Crank_L = model.CreateJointRevolute("Rev_BktTrLink_Crank_L",
BodyCrankLinkL, BodyBktTrLink_CylRod_Cylinder, refFrame1);
RevJoint_BktTrLink_Crank_L.ActionMarker.RefFrame.Origin.ParametricPoint = PP_CrankL_BktTrLink;
RevJoint_BktTrLink_Crank_L.BaseMarker.RefFrame.Origin.ParametricPoint = PP_CrankL_BktTrLink;

refFrame1.Origin.ParametricPoint = PP_Bucket_BktTrLink;
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 180, 90, 90);
IJointRevolute RevJoint_Bucket_BktTrLink = model.CreateJointRevolute("Rev_Bucket_BktTrLink", BodyJoint,
BodyBktTrLink_Bucket_BktTrLink_Cylinder, refFrame1);
RevJoint_Bucket_BktTrLink.ActionMarker.RefFrame.Origin.ParametricPoint = PP_Bucket_BktTrLink;
RevJoint_Bucket_BktTrLink.BaseMarker.RefFrame.Origin.ParametricPoint = PP_Bucket_BktTrLink;

refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0, 90, 0);
IJointRevolute RevJoint_Dipper_Crank_R = model.CreateJointRevolute("Rev_Dipper_Crank_R", BodyCrankLinkR,
BodyBktTrLink_Right_Link, refFrame1);
RevJoint_Dipper_Crank_R.ActionMarker.RefFrame.Origin.ParametricPoint = PP_CrankR_BktTrLink;
RevJoint_Dipper_Crank_R.BaseMarker.RefFrame.Origin.ParametricPoint = PP_CrankR_BktTrLink;

refFrame1.SetOrigin(5506.1017, -477.85255, 2231.9959);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0,90,0);
IJointRevolute RevJoint7 = model.CreateJointRevolute("RevJoint3", BodyDipperStick, BodyCrankLinkR,
refFrame1);

refFrame1.Origin.ParametricPoint = PP_DipperStick_Cyl;
IJointRevolute RevJoint8 = model.CreateJointRevolute("RevJoint4", BodyDipperStick,
BodyHydraulicCylinder_Cylinder, refFrame1);

refFrame1.Origin.ParametricPoint = PP_BktTrLink_Rod;
IJointRevolute RevJoint9 = model.CreateJointRevolute("RevJoint5", BodyHydraulicCylinder_Rod,
BodyBktTrLink_CylRod_Cylinder, refFrame1);
```

3. **File** 메뉴에서 **Save PNetFunction.cs** 를 클릭해서 save 를 수행합니다.

Force 생성

1. **Excavator** 에서 사용할 **Force** 를 생성하겠습니다.
2. 지금까지 작성한 코드 아래에 Force 를 생성하는 코드를 입력합니다.

```
refFrame1.SetOrigin(5679.2685, -207.8525, 62.560441);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 90,90,-90);
refFrame2.SetOrigin(5579.2685, -207.8525, 62.560441);
refFrame2.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 90, 90, -90);

IForceAxial ForAxial1 = model.CreateForceAxial("BucketTipLoad", BodyDummyBucketTip, BodyBucket, refFrame2,
refFrame1);
ForAxial1.ForceDisplay = ForceDisplay.Action;
ForAxial1.Expression = Ex_BucketTipLoad;

refFrame1.SetOrigin(6400, -207.8525, 4200);
refFrame1.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0, 90, 0);
IMarker Marker02 = Sub02.CreateMarker("Marker1", BodySub02Mother, refFrame1);

refFrame2.SetOrigin(6100, -207.8525, 4200);
refFrame2.SetEulerAngleDegree(EulerAngle.EulerAngle_ZXZ, 0, 90, 0);
IForceAxial ForceAxial02 = Sub02.CreateForceAxial("Ex_Rq_CylPow", BodyDummyDrivingForceBody,
BodySub02Mother, refFrame2, refFrame1);
ForceAxial02.ForceDisplay = ForceDisplay.Base;
```

3. **File** 메뉴에서 **Save PNetFunction.cs** 를 클릭해서 save 를 수행합니다.

Variable Equation 생성

1. **Excavator** 에서 사용할 **Variable Equation 및 Request** 를 생성하겠습니다.
2. **지금까지 작성한 코드 아래에 Variable Equation 및 Request 를 생성하는 코드를 입력합니다.**
   - model.Redraw() 함수는 **Working Window** 에 있는 Graphic 정보를 다시 그려주는 역할을 합니다.

```
IExpression Ex_MaxPosRot = model.CreateExpression("Ex_MaxPosRot", "0");
IVariableEquation VE_MaxPosRot = model.CreateVariableEquation("VE_MaxPosRot", Ex_MaxPosRot);
Ex_MaxPosRot.Arguments = new string[] { "Bucket.Marker3", "DipperStick.Marker3", "VE_MaxPosRot" };
Ex_MaxPosRot.Text = "IF(VARVAL(3)-AZ(1,2):AZ(1,2),VARVAL(3),VARVAL(3))";

IExpression Ex_MaxNegRot = model.CreateExpression("Ex_MaxNegRot", "0");
IVariableEquation VE_MaxNegRot = model.CreateVariableEquation("VE_MaxNegRot", Ex_MaxNegRot);
Ex_MaxNegRot.Arguments = new string[] { "Bucket.Marker3", "DipperStick.Marker3", "VE_MaxNegRot" };
Ex_MaxNegRot.Text = "IF(AZ(1,2)-VARVAL(3):AZ(1,2),VARVAL(3),VARVAL(3))";

Ex_BucketTipLoad.Arguments = new string[] { "Bucket.Marker3", "DipperStick.Marker3" };
Ex_BucketTipLoad.Text = "50000*IF(WZ(1,2,2):0,0,1)";
IExpression Ex_CylinderPower = model.CreateExpression("Ex_CylinderPower", "0");
Ex_CylinderPower.Arguments = new string[] { "Ground.Marker2",
"DrivingForceBody.Marker1@HydraulicCylinder", "Rod.Marker1@HydraulicCylinder",
"Cylinder.Marker1@HydraulicCylinder" };
Ex_CylinderPower.Text = "FX(1,2,2)*VZ(3,4,4)";
IRequestExpression ExRq_CylPow = Sub02.CreateRequestExpression("ExRq_CylPow", Ex_CylinderPower,
Ex_MaxPosRot, null, null, null, null);

model.Redraw();
```

3. **File** 메뉴에서 **Save PNetFunction.cs** 를 클릭해서 save 를 수행합니다.
4. **Build** 매뉴에서 **Build Excavator** 을 선택하여 빌드를 수행합니다. **IDE** 창 하단의 **Error List** 창에서 에러나 경고가 없는지를 확인합니다. 에러나 경고가 있다면, 목록을 확인하여 수정합니다.

다이얼로그에 함수 연결

다이얼로그창에서 Import 버튼을 클릭했을 때, 지금까지 작성한 Import() 함수가 호출되도록 하는 방법을 살펴보겠습니다.

다이얼로그에 함수 연결

1. **Project Explorer 창에서 ExcavatorDialog.cs** 를 선택합니다.
2. 마우스 오른쪽 버튼을 클릭하고 **View Code** 를 클릭합니다.
3. 아래의 코드를 입력합니다. (굵게 표시된 부분을 입력합니다.)

```
public partial class ExcavatorDialog : Form
{
        IApplication application;
        string strFilePath;
        string[,] strExcavatorPartName = new string[7, 2];
        PNetFunction Function;

        public ExcavatorDialog(IApplication app)
        {
                InitializeComponent();
                application = app;
                Function = new PNetFunction(application);
        }
```

   - Import 함수를 사용하기 위해서 **PNetFunction** 인스턴스를 생성합니다.

4. **btImport_Click()** 함수 내에 앞에서 생성한 **PNetFuction** 인스턴스를 이용하여 **Import()**함수를 호출하는 코드를 입력합니다. (굵게 표시된 부분을 입력합니다.)

```
□□□□ private void btImport_Click(object sender, EventArgs e)
{
        UpdateDB();
        Function.Import(strExcavatorPartName);
}
```

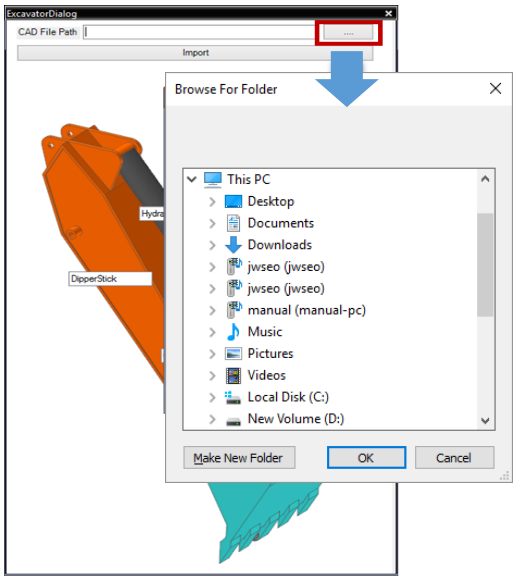
5. **File** 메뉴에서 **Save ExcavatorDialog.cs** 를 클릭해서 save 를 수행합니다

다이얼로그 윈도우의 테스트

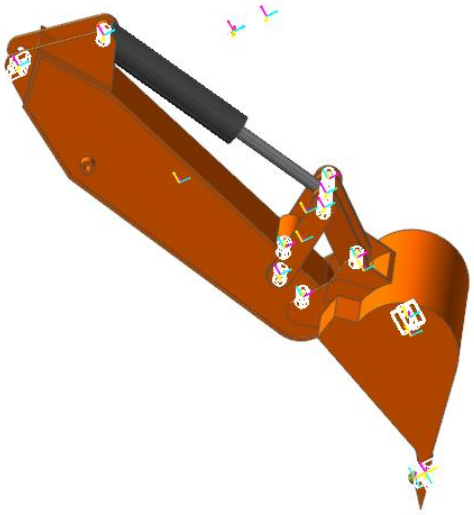
생성한 애플리케이션이 정상적으로 작동하는지 테스트합니다.

애플리케이션 실행하기

1. **Build** 매뉴에서 **Buid Solution** 을 선택합니다. **IDE** 창 하단의 **Error List** 창에서 에러나 경고가 없는지를 확인합니다. 에러나 경고가 있다면, 목록을 확인하여 수정합니다.
2. **RecurDyn** 의 **Customize** 탭의 **ProcessNet(General)**그룹에서 **Run** 을 선택합니다.
3. **ProcessNet Manager** 다이얼로그 하단의 트리컨트롤에서 **Excavator** 하위의 **Run** 을 클릭합니다.
4. **ProcessNet Manager** 다이얼로그에 있는 **Run** 버튼을 클릭합니다.
5. 옆의 그림과 같은 다이얼로그 박스가 나타납니다.
6. 나타난 다이얼로그에서 **...** 버튼을 클릭합니다.
7. Browse For Folder 다이얼로그가 열리면 Import 할 파일이 있는 위치를 선택합니다. (파일 위치는 "<InstallDir>/Help/Tutorial/ProcessNet/General/Excavator/Excavator" 에 존재합니다.)
8. CAD File Path 에 파일경로가 입력이 된 것을 확인하였으면 **Import** 버튼을 클릭합니다.
9. 아래의 그림과 같이 Excavator 모델이 자동으로 생성이 되는 것을 확인할 수 있습니다.
10. **다이얼로그 우측 상단의 x 를 눌러 ExcavatorDialog** 창을 닫습니다.



11. **ProcessNet Manager** 다이얼로그를 닫습니다.



Chapter 5

모델 해석

목적

이번 장에서는 유저가 다이얼로그에서 원하는 Entity 의 정보를 변경하면 변경된 값이 모델에 적용될 수 있는 함수를 작성하고 다이얼로그에서 모델 해석을 수행하는 방법을 살펴볼 것입니다.

예상 소요 시간

10 분

다이얼로그의 디자인 수정하기

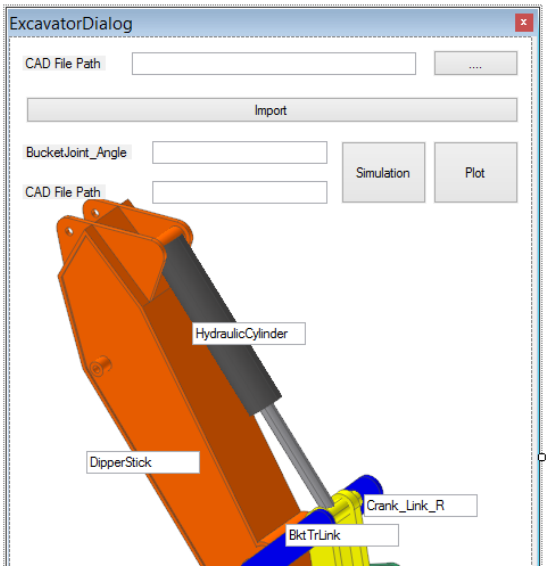
사용자가 다이얼로그에서 모델해석과 플로팅(Plotting)을 할 수 있도록 텍스트 박스와 버튼을 추가할 것입니다.

다이얼로그 윈도우의 디자인 수정하기

1. **Project Explorer** 에서 **ExcavatorDialog.cs** 을 마우스로 더블 클릭합니다. **ExcavatorDialog.cs** 의 다이얼로그가 **Project** 편집 창에 보여집니다.
2. **ToolBox** 를 선택한 후, **Common Contorls** 목록에서 다음과 같은 컨트롤들을 추가하고 값을 변경합니다.

Dialog Element	Text	Name	Location	Size
Button1	Simulation	btSimulation	292, 90	75, 57
Button2	Plot	btPlot	373, 90	75, 57
TextBox1		tbBucketJointAngle	126, 92	154, 20
TextBox2		tbCrankLength	126, 127	154, 20
Label1	BucketJoint_Angle	lbBucketJointAngle	12, 95	83, 12
Label2	Crank_Length	lbCrankLength	12, 130	83, 12

3. **File** 메뉴에서 **Save ExcavatorDialog.cs** 를 클릭해서 **Save** 를 수행합니다.


ExcavatorDialog
CAD File Path
....
Import
BucketJoint_Angle
Simulation
Plot
CAD File Path
HydraulicCylinder
DipperStick
Crank_Link_R
BktTrLink

모델 해석 함수

이번 과정에서는 실린더의 길이와 bucket 의 각도를 사용자가 다이얼로그에 입력하면 모델에 반영이 되고, 버튼을 클릭하는 경우 해석을 수행하는 함수를 생성하겠습니다.

모델 해석 함수

1. Project Explorer 에서 PnetFunction.cs 를 더블 클릭합니다.
2. 앞에서 생성한 **Import()** 아래에 **Simulation** 함수를 생성합니다.
   - 다이얼로그에서 수정된 BucketJoint Angle 과 Crank Length 의 값을 가지고 PV_DeltaCrankLength 와 PV_BucketJointAngleDeg 의 PV 값를 수정하는 코드를 입력합니다..

```
public void Simulation(double[] dPVValue)
{
        modelDocument = application.ActiveModelDocument;
        model = modelDocument.Model;

        IParametricValue PV_DeltaCrankLength = model.GetEntity("PV_DeltaCrankLength") as
        IParametricValue;
        IParametricValue PV_BucketJointAngleDeg = model.GetEntity("PV_BucketJointAngleDeg") as
        IParametricValue;
        PV_BucketJointAngleDeg.Value = dPVValue[0];
        PV_DeltaCrankLength.Value = dPVValue[1];
        model.Redraw();

        modelDocument.ModelProperty.DynamicAnalysisProperty.SimulationStep.Value = 400;
        modelDocument.ModelProperty.DynamicAnalysisProperty.SimulationTime.Value = 4;
        modelDocument.Analysis(AnalysisMode.Dynamic);
}
```

3. **Project Explorer** 에서 **ExcavatorDialog.cs** 을 마우스로 클릭하고 오른쪽 버튼을 눌러서 **View Disginer** 를 클릭합니다.
4. **Simulation** 버튼을 더블 클릭하여 생성된 함수에 아래의 **코드**를 입력합니다.

```
private void btSimulation_Click(object sender, EventArgs e)
{
        double dAngle = Convert.ToDouble(this.tbBucketJointAngle.Text);
        double dLength = Convert.ToDouble(this.tbCrankLength.Text);
        double[] dPVVAlue = new double[] { dAngle , dLength};
        Function.Simulation(dPVVAlue);
}
```

5. **File 메뉴에서 Save ExcavatorDialog.cs 를 클릭해서 Save 를 수행합니다**.

Chapter

6

Plot 자동 생성

목적

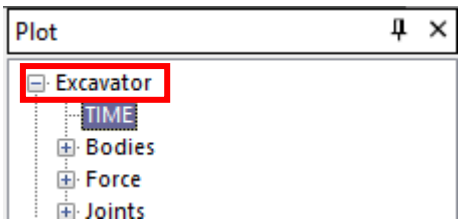
이번 장에서는 **ProcessNet** 에서 **Plot** 을 그리기 위한 명령어를 살펴볼 것입니다.

예상 소요 시간

10 분

Plot 함수

Plot 함수



1. Project Explorer 에서 PnetFunction.cs 를 더블 클릭합니다.
2. 앞서 생성했던 Simulation() 함수 아래에 **Plot** 함수를 생성합니다.
   - GetPlotData 에서 "EXCAVATOR"는 Plot 의 최상위 root 의 값입니다. 이 값은 RecurDyn 의 버전에 따라서 달라질 수 있습니다.

```
public void Plot()
{
        modelDocument = application.ActiveModelDocument;
        plotDocument = modelDocument.CreatePlotDocument(PlotDocType.WithRPLT);

        double[] Time = plotDocument.GetPlotData("EXCAVATOR/TIME");
        double[] dRelative = plotDocument.GetPlotData
        ("EXCAVATOR/Joints/TraJoint1@HydraulicCylinder/Pos1_Relative");
        double[] dDrivingForce = plotDocument.GetPlotData
        ("EXCAVATOR/Joints/TraJoint1@HydraulicCylinder/Driving_Force");
        double[] dPos1_Relative = plotDocument.GetPlotData
        ("EXCAVATOR/Joints/Rev_Dipper_Bucket/Pos1_Relative");

        plotDocument.PlotShowWindowType(ShowWindowOption.ShowAll);
        plotDocument.LoadAnimation(PlotWindowPosition.LeftLower);

        plotDocument.ActivateView(0, 0);
        plotDocument.DrawPlot("Relative", Time, dRelative);
        plotDocument.DrawPlot("DrivingForce", Time, dDrivingForce);
        plotDocument.SimpleMathMultiply(0, 1, false, true);

        plotDocument.ActivateView(0, 1);
        plotDocument.DrawPlot("Post Relative", Time, dPos1_Relative);
}
```

   - ActivateView 함수를 사용하면 원하는 View 가 활성화가 됩니다.

3. **Project Explorer** 에서 **ExcavatorDialog.cs** 을 마우스로 클릭하고, 오른쪽 버튼을 눌러서 View Designer 를 클릭합니다.
4. **Plot** 버튼을 더블 클릭합니다.
5. 생성된 함수 아래에 아래의 Plot 함수를 입력합니다.

```
private void btPlot_Click(object sender, EventArgs e)
{
        Function.Plot();
}
```

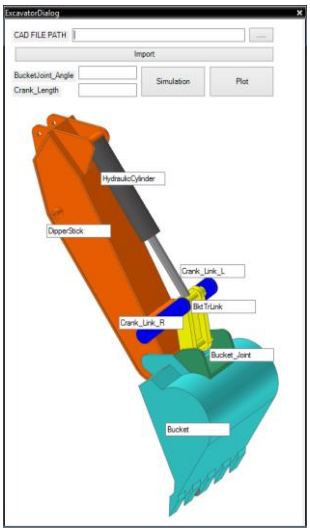
6. **File** 메뉴에서 **Save ExcavatorDialog.cs** 를 클릭해서 **Save** 를 수행합니다.

생성한 애플리케이션의 테스트

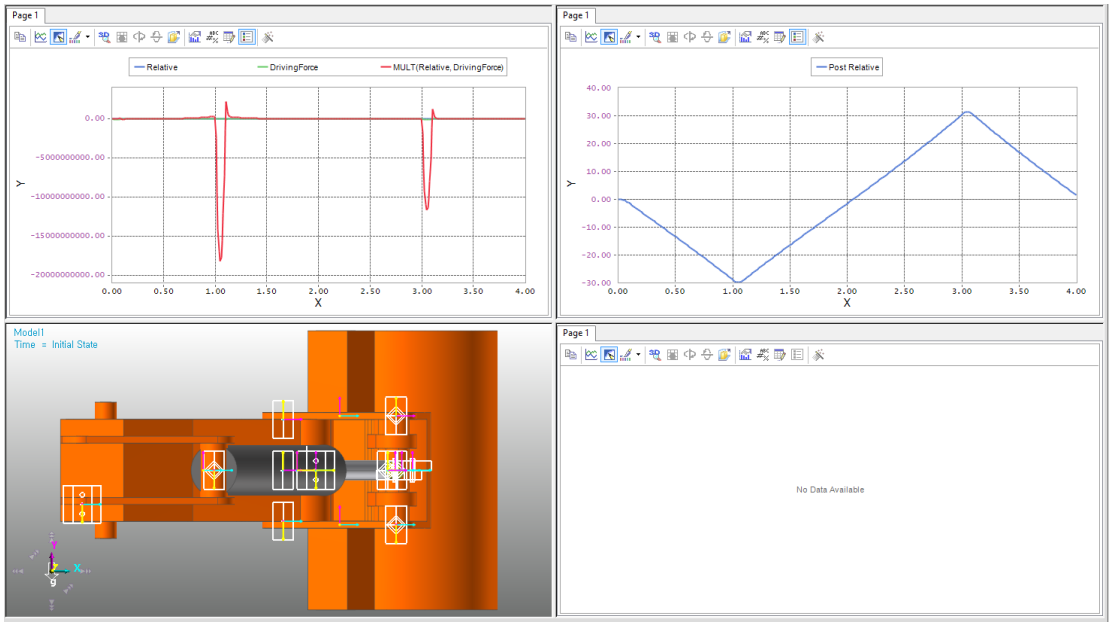
생성한 애플리케이션이 정상적으로 작동하는지 테스트합니다.

애플리케이션 실행하기

1. **Build** 메뉴에서 **Build Solution** 을 선택합니다. **IDE** 창 하단의 **Error List** 창에서 에러나 경고가 없는지를 확인합니다. 에러나 경고가 있다면, 목록을 확인하여 수정합니다.
2. **RecurDyn** 프로그램의 **Customize** 탭의 **ProcessNet(General)** 그룹에서 **Run** 을 선택합니다.
3. **ProcessNet Manager** 다이얼로그 하단의 트리컨트롤에서 **Excavator** 하위의 **Run** 을 클릭합니다.



4. **ProcessNet Manager** 다이얼로그에 있는 **Run** 버튼을 클릭합니다.
5. 옆의 그림과 같이 다이얼로그 박스가 나타납니다.
6. 나타난 다이얼로그에서 BucketJoint_Angle 의 값을 0 으로 입력하고 Crank_Length 의 값을 0 으로 입력합니다.
7. Simulation 버튼을 클릭하면 BucketJoint_Angle 와 Crank_Length 에 입력된 값에 맞춰서 PV_DeltaCrankLength 와 PV_BucketJointAngleDeg 의 값이 변하고 그에 맞춰서 모델이 해석이 되는 것을 확인할 수 있습니다..
8. 해석이 끝난 후, **Plot** 버튼을 누르면 아래 그림처럼 Plot 이 그려집니다.



9. **ExcavatorDialog** 창을 닫습니다.
10. **ProcessNet Manager** 다이얼로그를 닫습니다.

Thanks for participating in this tutorial