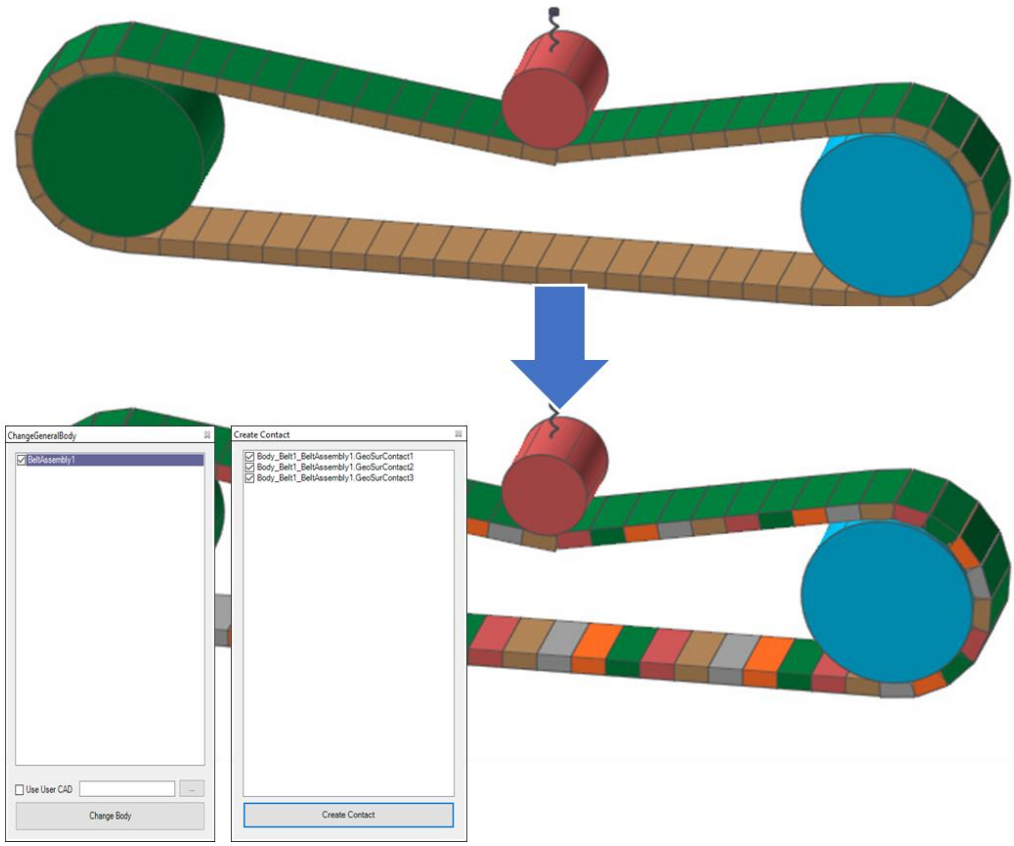
RECURDYN

Simple Belt System (ProcessNet VSTA)



FunctionBay

Edition Note

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

목차

Chapter 1

개요

목적

RecurDyn/Belt 에서 제공하는 Belt Assembly 의 Link Body 는 Clone Body 로써 RecurDyn/Professional 에서 지원하는 Contact, Force 등의 general entity 들을 사용할 수 없으며 직접 RecurDyn/Mesh 을 이용하여 FFlex Body 을 만들 수도 없습니다. 이를 이용하려면 기존 Clone Body 들을 General Body 로 변환하는 것이 좋은 방법중의 하나입니다. 본 튜토리얼 에서는 ProcessNet 을 사용해서 Belt Assembly 의 Clone Link Body 를 General Body 로 변환하여 Body 사이에 Force 와 Contact 을 생성할 것입니다.

사전 학습 내용

RecurDyn/Belt Toolkit 에 대한 튜토리얼과 기존의 ProcessNet 튜토리얼을 전부 학습하고 오는 것을 권장합니다.

필요 요건

ProcessNet 튜토리얼을 익혔거나 이에 상용하는 작업을 해 본 것을 전제로 하며, 물리학에 대한 기본 지식이 있어야 합니다.

과정

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

과정	시간(분)
ProcessNet 시작	5
Clone Body Convert 코드	20
Create Contact 코드	20
Register DLL	5
모델 해석	10
총합	60

예상 소요 시간

60 분

Chapter 2

ProcessNet 시작

목적

ProcessNet 을 사용하기 위해 RecurDyn 에서 ProcessNet 을 어떻게 시작하는지를 살펴봅시다.

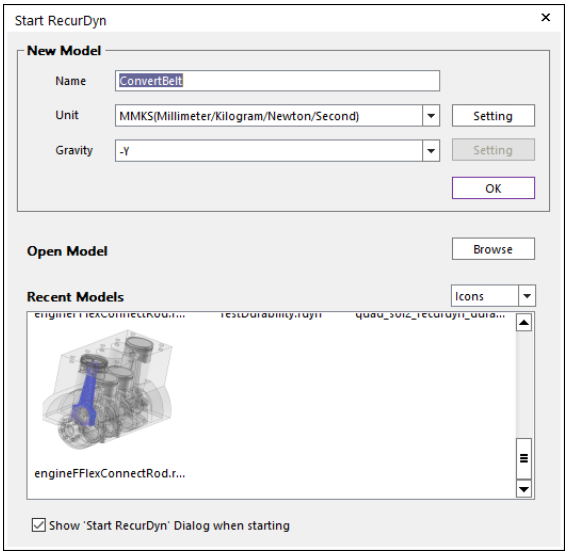
예상 소요 시간

5 분

RecurDyn 시작하기

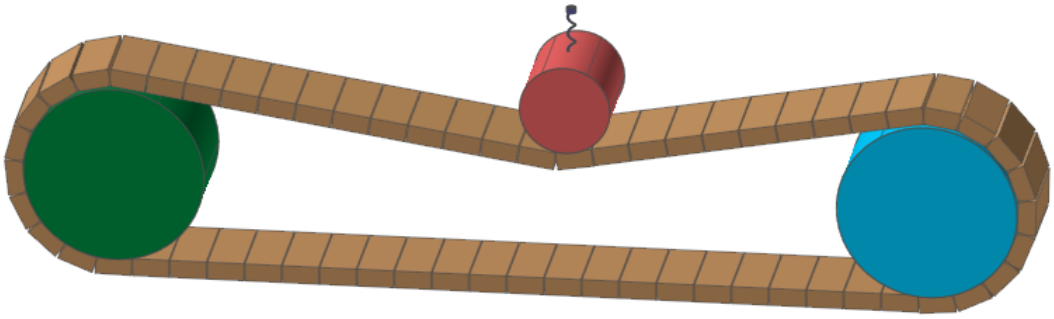
RecurDyn 을 시작하기

1. **RecurDyn** 을 실행합니다.
2. **Start RecurDyn** 다이얼로그 박스가 나타나면, 새 모델이 아닌 기존 모델을 사용해야 하므로, 닫아 줍니다.



3. **Quick Access Toolbar** 에서, **Open** 을 클릭합니다.

4. **SimpleBeltAssembly.rdyn** 파일을 선택합니다. (파일 경로: **<InstallDir>/Help/Tutorial/ProcessNet/VSTA/SimpleBeltSystem**)
5. **Open** 을 클릭합니다.



모델 저장하기**:**

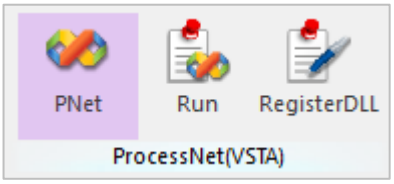
1. **File** 메뉴에서, **Save As** 를 클릭합니다.
   - 파일 이름은 **SimpleBeltAssembly_01.rdyn** 로 저장합니다.

   (튜토리얼 경로에서는 직접 시뮬레이션 실행이 불가하므로 다른 경로에 본 모델을 다시 저장해야 합니다.)

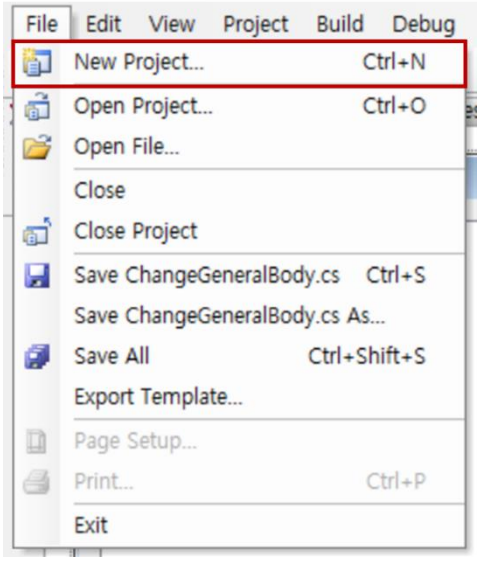
ProcessNet 시작하기

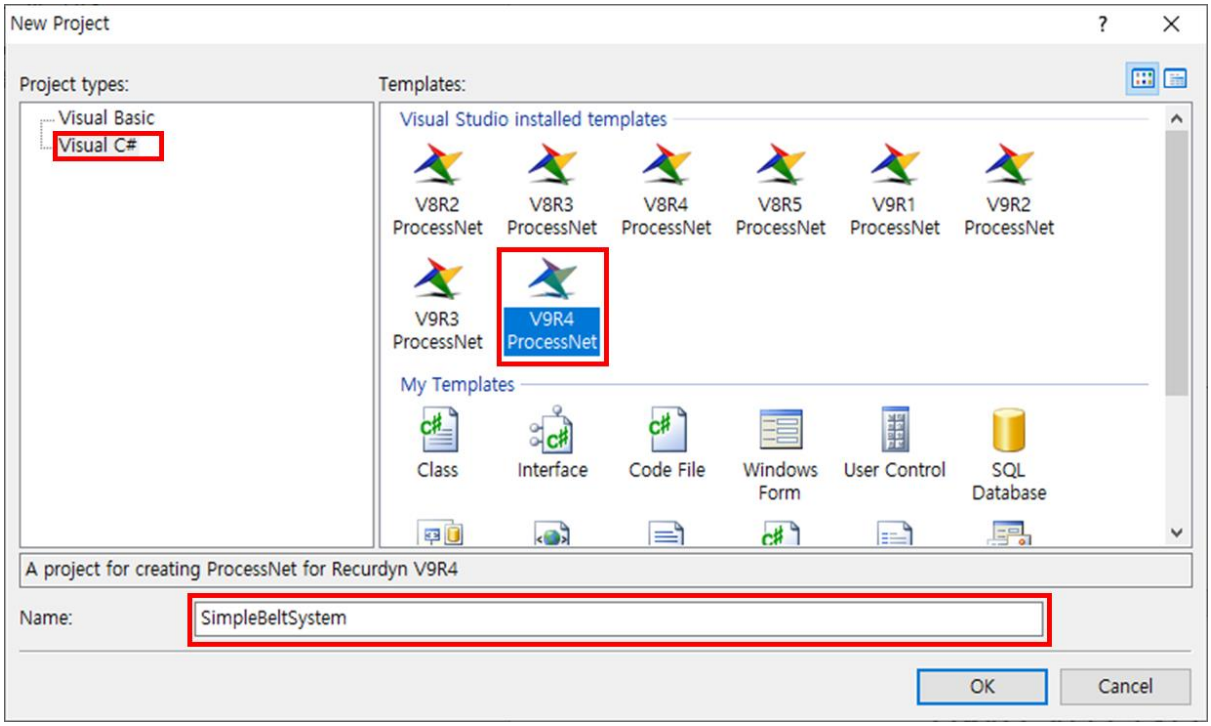
ProcessNet 시작한 후 초기화 하기

1. **ProcessNet** 의 통합 개발환경(IDE)로 들어가기 위해서 **Customize** 탭의 **ProcessNet(VSTA)** 그룹의 **PNet** 버튼을 클릭합니다.



2. **ProcessNet IDE** 가 실행되면 **File** 메뉴에서 **New Project** 를 클릭합니다.





3. **New Project** 다이얼로그가 발생합니다 여기서 **RecurDyn** 의 버전과 맞는 **Template** 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. **Project Types** 는 **Visual C#**을 선택하고 **Name** 에 **SimpleBeltSystem** 를 입력하고 **OK** 를 누릅니다.
5. **SimpleBeltSystem Project** 가 생성이 되면 **File - Save SimpleBeltSystem** 를 클릭하고 원하는 위치에 **ProcessNet** 프로젝트를 저장합니다.
6. 이제 **ProcessNet** 애플리케이션을 개발하기 위한 준비가 완료되었습니다.

Chapter 3

Change General Body 코드

이 장에서는 Clone Link Body 를 General Body 로 변환하는 다이얼로그 윈도우를 생성하고 그 디자인을 구성한 다음 다이얼로그 윈도우 창과 코드 사이에 연결관계를 설정할 것입니다.

목적

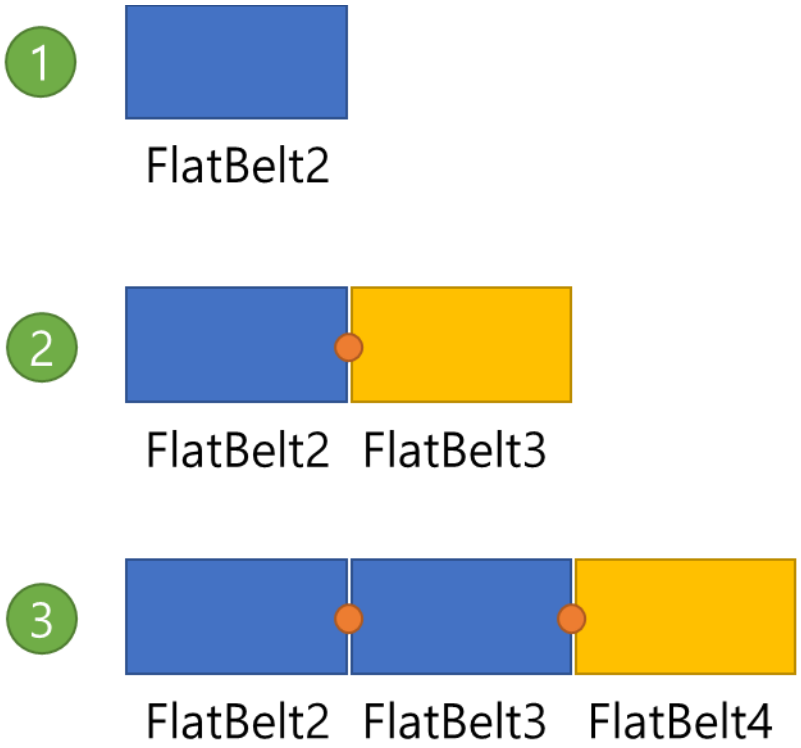
ProcessNet 에서 다이얼로그 윈도우를 생성하는 방법과 ProcessNet 에서 다이얼로그 윈도우를 호출하는 함수를 생성하는 방법, 그리고 Clone Link Body 를 General Body 로 변환하는 코드를 살펴볼 것입니다.

예상 소요 시간

20 분

알고리즘 구성하기

Assembly 에 있는 Clone Link Body 를 General Body 로 변환하고 Body 간 Connector 를 생성하는 알고리즘은 아래와 같습니다.

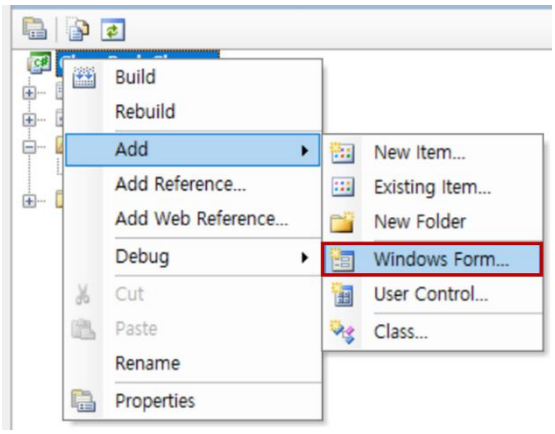


1. 첫번째 Clone Body 를 Convert 해줍니다.
2. 두번째 Clone Body 를 Convert 해주고 1 번에서 Convert 한 Body 사이에 Connector 를 생성합니다.
3. 세번째 Clone Body 를 Convert 해주고 2 번에서 Convert 한 Body 사이에 Connector 를 생성합니다.

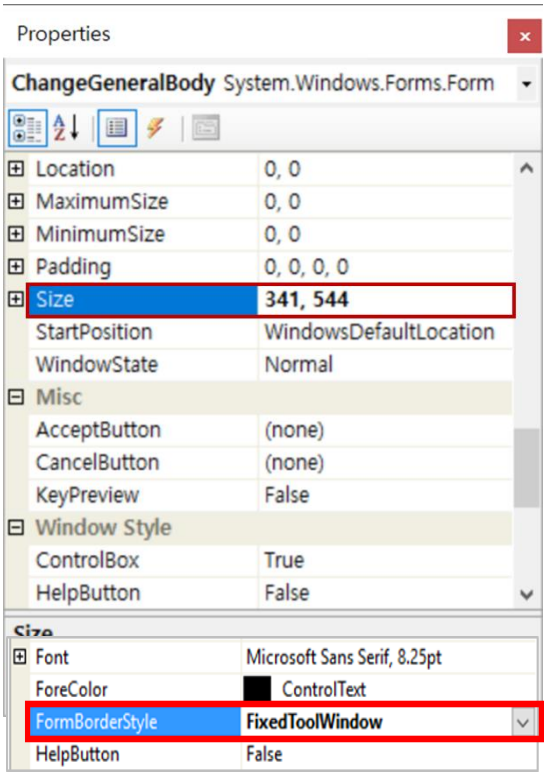
다이얼로그 생성하기

위 알고리즘 구현한 것을 dialog 을 통해서 사용할 수 **ChangeGeneralBody** 다이얼로그를 생성하는 법을 배울 것입니다.

ChangeGeneralBody 다이얼로그 윈도우 생성하기



1. **ProcessNet IDE** 의 **Project Explorer** 창에서 **SimpleBeltSystem** 를 마우스로 클릭한후 마우스의 오른쪽을 클릭합니다.
2. **Add – Windows Form** 을 클릭합니다.
3. 다음과 같이 **Add New Item** 다이얼 로그 창이 나타나면 **Windows Form** 아이콘을 클릭하고 Name 을 **ChangeGeneralBody** 로 입력합니다.
4. **Add** 버튼을 클릭합니다.
5. **Windows Form** 을 위한 설계 윈도우인 **ChangeGeneralBody.cs[Design]**이 **IDE Project Editor** 윈도우에 나타납니다.
6. 화면 왼쪽상단에 있는 **ChangeGeneralBody** 다이얼로그를 클릭합니다.
7. 화면 오른쪽 하단의 **Properties Windows** 에 **ChangeGeneralBody** 의 정보가 나타나는데 여기서 **size** 를 341, 544 로 수정합니다.
8. 마찬가지로 **FormBorderStype 을 FixedToolWindow 로** 수정합니다.



9. 화면 왼쪽 상단에서 **ToolBox** 

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

Note: ToolBox 가 보이지 않는 경우 View – ToolBox 를 클릭하거나 단축키 Ctrl + Alt + X 를 누릅니다.

---

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

11. **Properies** 창에서 **Loation** 는 12,12 로 변경하고, **Size** 는 299, 409 으로 입력합니다.
12. **Name** 은 **lbAssemblyList** 를 입력합니다.
13. **Button1, Button2, TextBox1, CheckBox1** 에 대해서도 앞의 과정과 동일한 방법으로 다음 표를 참조하여 **Text** 와 **Name** 의 값을 수정합니다.

Dialog Element	Text	Name	Location	Size
Button1	...	btCADPath	270, 423	41, 23
Button2	Change Body	btChagneBody	12, 451	299, 37
CheckBox1	Use User CAD	cbUseCAD	12, 428	
TextBox1		tbCADPath	113, 425	151, 20

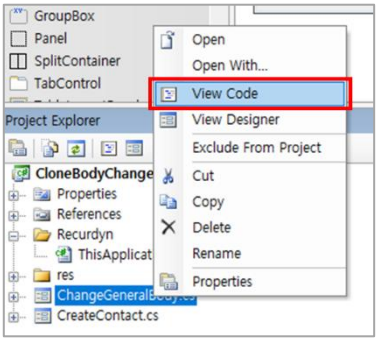
14. 위의 과정을 수행하면 오른쪽의 그림과 같이 다이얼로그가 완성된 것을 확인할 수 있습니다.
15. **File – Save ChangeGeneralBody.cs** 버튼을 클릭해서 **Save** 를 수행합니다.

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

다이얼로그의 초기환경 정의하기

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

ChangeGeneralBody 다이얼로그 윈도우의 초기환경 정의하기

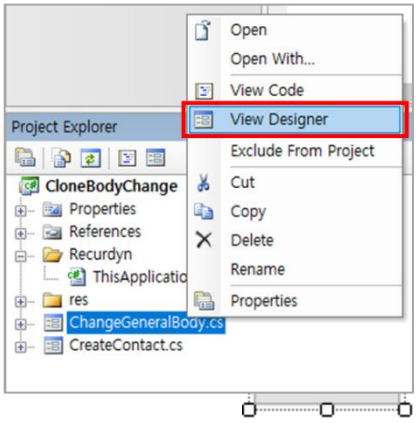


1. **Project Explorer** 에서 **ChangeGeneralBody.cs** 을 선택한 후, 오른쪽 버튼을 클릭합니다**. View Code** 를 선택하면 **ChangeGeneralBody.cs** 의 소스 코드가 **IDE Project** 편집창에 나타납니다.
2. 편집창에 다이얼로그 윈도우에 사용할 변수를 아래와 같이 코드를 삽입합니다.

```
using System.IO;

using FunctionBay.RecurDyn.ProcessNet;
using FunctionBay.RecurDyn.ProcessNet.BNP;

namespace SimpleBeltSystem
{
        public partial class ChangeGeneralBody : Form
        {
                string path = System.Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        + @"\RecurDyn\V9R4";
                bool closeAssembly = true;

                IModelDocument modelDocument;
                ISubSystem sub;
                public ChangeGeneralBody(IModelDocument modelDoc)
                {
                        InitializeComponent();
                        modelDocument = modelDoc;
                        sub = modelDocument.GetDataStorage() as ISubSystem;
                }
        }
}
```

- **FunctionBay.RecurDyn.ProcessNet**, **FunctionBay.RecurDyn.ProcessNet.BNP** 는 ProcessNet 의 함수를 사용하기 위한 Reference 입니다.
- **path** 은 Export 한 CAD 파일을 저장할 위치입니다. 본 튜토리얼 에서는 **Document 의 RecurDyn 폴더**에 CAD 파일을 저장할 것입니다.

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



4. 다이얼로그 윈도우 상에서 **...** 버튼을 더블 클릭하여 **btCADPath_Click()** 함수를 생성합니다.

5. 자동으로 생성된 새로운 함수 안에 아래와 같이 코드를 삽 입합니다.

```
private void btCADPath_Click(object sender, EventArgs e)
{
        OpenFileDialog openFileDlg = new OpenFileDialog();
        openFileDlg.DefaultExt = "db";
        openFileDlg.Filter = "ParaSolid File Files(*.x_t;*.x_b)|*.x_t;*.x_b";
        if (openFileDlg.ShowDialog() == DialogResult.OK)
        {
                this.tbCADPath.Text = openFileDlg.FileName.ToString();
        }
}
```

- 사용자가 가지고 있는 CAD 파일을 사용하고 싶을 때는 CAD 파일을 선택하는 **Dialog** 창을 발생합니다.
- ... 버튼을 클릭하면 **btCADPath_Click()** 함수가 실행되면서 **Folder Dialog** 창이 발생합니다.

6. **Project Explorer** 의 **ChangeGeneralBody.cs** 파일을 클릭하고 오른쪽 버튼을 클릭 후, **View Designer** 버튼을 클릭한 후, 다이얼로그 윈도우 상의 **Change Body** 버튼을 더블 클릭하여 **btChagneBody_Click()** 함수를 생성합니다.

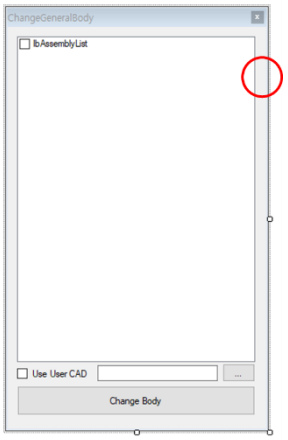
7. **btChagneBody_Click()** 함수에 아래의 코드를 입력합니다.

```
private void btChagneBody_Click(object sender, EventArgs e)
{
        bool useUserCAD = this.cbUseCAD.Checked;
        string useUserCADFilePath = this.tbCADPath.Text;

        if (useUserCAD)
        {
                if (File.Exists(useUserCADFilePath) == false)
                {
```

```
                    MessageBox.Show("The file does not exist");
                    return;
                }
            }

            if (this.lbAssemblyList.CheckedItems.Count != 0)
            {
                for (int count = 0; count <= this.lbAssemblyList.Items.Count - 1;
count++)
                {
                    if (this.lbAssemblyList.GetItemCheckState(count) ==
CheckState.Checked)
                    {
                        string assemblylistName =
this.lbAssemblyList.Items[count].ToString();
                        changeBNPGeneralBody(assemblylistName, useUserCAD,
useUserCADFilePath);
                        this.lbAssemblyList.SetItemChecked(count, false);
                    }
                }
            }
            modelDocument.UpdateDatabaseWindow();
            modelDocument.Redraw();
}
```

- **lbAssemblyList** 에서 선택 되어 있는 Assembly 의 이름과 User CAD 사용 여부를 **changeBNPGeneralBody()** 함수에 전달해서 Clone Link Body 를 General Body 로 변환해 줍니다. **changeBNPGeneralBody()** 함수는 다음 장에 설명을 할 것입니다.

8. 다시한번 **Project Explorer** 의 **ChangeGeneralBody.cs** 파일을 클릭하고 오른쪽 버튼을 클릭 후, **View Designer** 를 클릭합니다.



9. 다이얼로그 윈도우 빈 곳을 더블 클릭해서 **ChangeGeneralBody_Load()** 함수를 생성합니다.

10. **ChangeGeneralBody_Load()** 함수 안에 아래의 코드를 입력합니다.

```
private void ChangeGeneralBody_Load(object sender, EventArgs e)
{
        IBNPSubSystem bnpSub = sub.BNPSubSystem as IBNPSubSystem;
        if (bnpSub == null)
                return;

        IBNPAssemblyCollection bnpAssemblies = bnpSub.AssemblyCollection;
        foreach (IBNPAssembly assembly in bnpAssemblies)
        {
                this.lbAssemblyList.Items.Add(assembly.Name);
        }
        IBNPAssembly2DCollection bnp2DAssemies = bnpSub.Assembly2DCollection;
        foreach (IBNPAssembly2D assembly in bnp2DAssemies)
        {
                this.lbAssemblyList.Items.Add(assembly.Name);
        }
}
```

- 다이얼로그 윈도우가 실행될 때 현재 Subsystem 에서의 Belt Assembly 를 찾아오는 함수입니다.
- Belt 는 2D Belt, 3D Belt 로 2 종류가 있기 때문에 따로 Collection 을 수행합니다.

Assembly 의 정보를 가지고 Body 와 Connector 생성

ProcessNet 을 사용해서 Belt Assembly 의 Clone Link Body 와 Connector 정보를 가지고 올 수 있습니다. 이 정보를 가지고 General Body 와 Force 를 생성하는 방법을 배울 것입니다.

Body Convert 하기

Clone Link Body 의 정보를 가지고 오고 가지고 온 정보를 바탕으로 Clone Link Body 를 General Body 로 변환을 할 것입니다.

1. **changeBNPGeneralBody(), copyBeltBodyInfor(), convertCloneBody()** 함수를 생성합니다. Clone Body 를 CAD 파일로 Export 한 후 Import 하는 작업과 Clone Link Body 의 정보를 Import 한 General Body 에 입력하는 작업은 반복적으로 진행되기 때문에 Clone Link 의 Body 정보를 General Body 로 복사하는 함수 **copyBeltBodyInfor()** 와 Clone Link Body 를 CAD 파일을 사용해서 General Body 로 변환해주는 **convertCloneBody()** 함수를 생성합니다.

```
public void changeBNPGeneralBody(string assemblyName,bool useUserCAD,string useUserCADFilePath)
{
}
private IBody convertCloneBody(IReferenceFrame refFrame1,bool useUserCAD,string
useUserCADFilePath ,IBNPBodyBelt linkBody, string generalBodyName)
{
}
private void copyBeltBodyInfor(IBNPBodyBelt BeforeBody, IBody AfterBody)
{
}
```

2. **changeBNPGeneralBody()** 함수를 생성합니다. 다음 코드를 복사해서 삽입합니다.

```
public void changeBNPGeneralBody(string assemblyName,bool useUserCAD,string useUserCADFilePath)
{
	IReferenceFrame refFrame1 = modelDocument.CreateReferenceFrame();
	IBNPAssembly assembly = sub.GetEntity(assemblyName) as IBNPAssembly;
	IBNPBodyBeltCollection linkBodies = null;
	IBNPAssembly2D assembly2D = null;


	IGroupGeneral genearlBodyGroup = sub.CreateGroupGeneral("Body_" + sub.Name + "_" +
assemblyName, new object[] { });
	IGroupGeneral genearlForceGroup = sub.CreateGroupGeneral("Connector_" + sub.Name + "_" +
assemblyName, new object[] { });
	IGroupGeneral genearlJointGroup = null;

	if (assembly != null)
	{
		linkBodies = assembly.BNPBodyBeltCollection as IBNPBodyBeltCollection;
	}
	else
	{
		assembly2D = sub.GetEntity(assemblyName) as IBNPAssembly2D;
		linkBodies = assembly2D.BNPBodyBeltCollection as IBNPBodyBeltCollection;
		genearlJointGroup = sub.CreateGroupGeneral("Joint_" + sub.Name + "_" + assemblyName,
new object[] { });
	}

	int linkCount = linkBodies.Count;


	IBody generalBodyfirst = convertCloneBody(refFrame1,useUserCAD, useUserCADFilePath,linkBodies[0],
"ImportedBody1");
	genearlBodyGroup.AddEntities(new object[] { generalBodyfirst });

	Dictionary<string, string> generalLinkNames = new Dictionary<string, string>();
	generalLinkNames.Add(linkBodies[0].Name, generalBodyfirst.Name);

	for (int iCount = 1; iCount < linkBodies.Count; iCount++)
	{
		IBody generalBody01 = convertCloneBody(refFrame1, useUserCAD, useUserCADFilePath,
linkBodies[iCount], "ImportedBody" + (iCount + 1).ToString());
		IMarker generalBody01Marker = generalBody01.GetEntity("CMMarker1") as IMarker;
		IBody generalBody02 = sub.GetEntity("ImportedBody" + iCount.ToString()) as IBody;
		IMarker generalBody02Marker = null;
		if (generalBody02.Name == "ImportedBody1")
		{
			generalBody02Marker = generalBody02.GetEntity("CMMarker1") as IMarker;
		}
		else
		{
			generalBody02Marker = generalBody02.GetEntity("CMMarker2") as IMarker;
		}
		if (assembly2D == null)
		{
			IForceMatrix matrixForce = createMatrixForce(assembly.Name + "_Matrix_" + "_"
+ linkBodies[iCount].Name.ToString(), assembly, generalBody02, generalBody01,
generalBody01Marker.RefFrame, generalBody02Marker, generalBody01Marker);
		}
		else
		{
			IForceBushing busingForce = create2DBeltBusingForce(assembly.Name +
"_Busing_" + "_" + linkBodies[iCount].Name.ToString(), assembly2D, generalBody02, generalBody01,
generalBody01Marker.RefFrame, generalBody02Marker, generalBody01Marker);
		}
		if (iCount == linkCount - 1 && closeAssembly)
		{
```

```
                                IMarker generalBodyFirstMarker = generalBodyfirst.GetEntity("CMMarker2") as
IMarker;
                                IMarker generalBody01Marker02 = generalBody01.GetEntity("CMMarker2") as
IMarker;
                                if (assembly2D == null)
                                {
                                        IForceMatrix matrixForceFinal = createMatrixForce(assembly.Name +
"_Matrix_" + "_" + linkBodies[0].Name.ToString(), assembly, generalBody01, generalBodyfirst,
generalBody01Marker02.RefFrame, generalBody01Marker02, generalBodyFirstMarker);
                                }
                                else
                                {
                                        IForceBushing busingForceFinal =
create2DBeltBusingForce(assembly.Name + "_Busing_" + "_" + linkBodies[0].Name.ToString(), assembly2D,
generalBody01, generalBodyfirst, generalBody01Marker02.RefFrame, generalBody01Marker02,
generalBodyFirstMarker);
                                }

                        }
                        genearlBodyGroup.AddEntities(new object[] { generalBody01 });
                        generalLinkNames.Add(linkBodies[iCount].Name, generalBody01.Name);
                }

                modelDocument.SetUndoHistory("BNP Body Change");
                modelDocument.DeleteEntity(linkBodies[0]);

                foreach (KeyValuePair<string, string> generalLinkName in generalLinkNames)
                {
                        IBody linkBody = sub.GetEntity(generalLinkName.Value) as IBody;
                        linkBody.Name = generalLinkName.Key;

                }
                modelDocument.SetUndoHistory("Delete Assembly");

}
```

changeBNPGeneralBody() 함수에 대한 설명은 아래에 있습니다.

```
IReferenceFrame refFrame1 = modelDocument.CreateReferenceFrame();
IBNPAssembly assembly = sub.GetEntity(assemblyName) as IBNPAssembly;
IBNPBodyBeltCollection linkBodies = null;
IBNPAssembly2D assebly2D = null;
```

위의 코드는 **changeBNPGeneralBody()** 함수에서 사용할 변수를 선언해줍니다.

- **assembly**: GetEntity 함수를 사용해서 sub Subsystem 안에 있는 belt Assembly 를 casting 합니다. as 연산자를 사용해서 belt Assembly 가 Subsystem 에 없는 경우 Null 이 반환되서 입력됩니다.
- **linkBodies**: Belt Assembly 에 포함된 Link Body 들의 정보를 저장합니다.
- **assembly2D**: Belt Assembly 가 2D 일 때를 위해서 선언해줍니다

```
IGroupGeneral genearlBodyGroup = sub.CreateGroupGeneral("Body_" + sub.Name + "_" + assemblyName, new
object[] { });
IGroupGeneral genearlForceGroup = sub.CreateGroupGeneral("Connector_" + sub.Name + "_" +
assemblyName, new object[] { });
IGroupGeneral genearlJointGroup = null;
```

위의 코드는 assembly 에 포함된 Body 와 Connector 의 정보를 가지고 Body, Force 를 생성하는데 생성된 Body 와 Force 를 하나로 묶어줄 General Group 을 생성합니다.

```
if (assembly != null)
{
	linkBodies = assembly.BNPBodyBeltCollection as IBNPBodyBeltCollection;
}
else
{
	assebly2D = sub.GetEntity(assemblyName) as IBNPAssembly2D;
	linkBodies = assebly2D.BNPBodyBeltCollection as IBNPBodyBeltCollection;
	genearlJointGroup = sub.CreateGroupGeneral("Joint_" + sub.Name + "_" + assemblyName, new object[]
{ });
}
```

위의 코드는 Belt Assembly 가 2D 인지 3D 인지 판별하는 조건문입니다. assembly 를 선언할 때 as 키워드를 사용했기 때문에 assembly 가 3D 가 아닌 경우 Null 이 반환되는 점을 이용해서 assembly 가 2D 인지 3D 인지 판단합니다. 2D Assembly 인 경우에는 Joint 을 하나로 묶어줄 General Group 을 추가로 생성합니다.

```
IBody generalBodyfirst =convertCloneBody(refFrame1, useUserCAD, useUserCADFilePath, linkBodies[0],
"ImportedBody1");
genearlBodyGroup.AddEntities(new object[] { generalBodyfirst });

Dictionary<string, string> generalLinkNames = new Dictionary<string, string>();
generalLinkNames.Add(linkBodies[0].Name, generalBodyfirst.Name);

for (int iCount = 1; iCount < linkBodies.Count; iCount++)
{
}
```

위의 코드는 Belt Assembly 에 속한 Clone Link Body 를 General Body 로 Convert 하는 코드입니다.

- 첫번째 Clone Link Body 를 **convertCloneBody()** 함수를 사용해서 General Body 로 변환합니다. 변환된 Body 는 General Group 에 추가합니다. 그리고 그 외 Clone Link Body 를 변환하기 위해서 반복문을 작성합니다. 처음 Clone Body 를 변경했기 때문에 1 부터 Link Body 의 개수까지 함수가 실행되어야 합니다.
- **generalLinkNames**: Import 한 CAD 파일의 이름을 Link Body 이름과 맞춰주기 위해서 Dictionary 에 CAD Body 이름과 Link Body 이름을 저장합니다

```
IBody generalBody01 = convertCloneBody(refFrame1, useUserCAD, useUserCADFilePath, linkBodies[iCount],
"ImportedBody" + (iCount + 1).ToString());
IMarker generalBody01Marker = generalBody01.GetEntity("CMMarker1") as IMarker;
IBody generalBody02 = sub.GetEntity("ImportedBody" + iCount.ToString()) as IBody;
IMarker generalBody02Marker = null;
if (generalBody02.Name == "ImportedBody1")
{
        generalBody02Marker = generalBody02.GetEntity("CMMarker1") as IMarker;
}
else
{
        generalBody02Marker = generalBody02.GetEntity("CMMarker2") as IMarker;
}
//Create Connector
if (assebly2D == null)
{
        IForceMatrix matrixForce = createMatrixForce(assembly.Name + "_Matrix_" + "_" +
linkBodies[iCount].Name.ToString(), assembly, generalBody02, generalBody01, generalBody01Marker.RefFrame,
generalBody02Marker, generalBody01Marker);
}
else
{
        IForceBushing busingForce = create2DBeltBusingForce(assembly.Name + "_Busing_" + "_" +
linkBodies[iCount].Name.ToString(), assebly2D, generalBody02, generalBody01,
generalBody01Marker.RefFrame, generalBody02Marker, generalBody01Marker);
}
if (iCount == linkCount - 1 && closeAssembly)
{
        IMarker generalBodyFirstMarker = generalBodyfirst.GetEntity("CMMarker2") as IMarker;
        IMarker generalBody01Marker02 = generalBody01.GetEntity("CMMarker2") as IMarker;
        if (assebly2D == null)
        {
                IForceMatrix matrixForceFinal = createMatrixForce(assembly.Name + "_Matrix_" + "_" +
linkBodies[0].Name.ToString(), assembly, generalBody01, generalBodyfirst, generalBody01Marker02.RefFrame,
generalBody01Marker02, generalBodyFirstMarker);
        }
        else
        {
                IForceBushing busingForceFinal = create2DBeltBusingForce(assembly.Name + "_Busing_" +
"_" + linkBodies[0].Name.ToString(), assebly2D, generalBody01, generalBodyfirst,
generalBody01Marker02.RefFrame, generalBody01Marker02, generalBodyFirstMarker);
        }

}
genearlBodyGroup.AddEntities(new object[] { generalBody01 });
generalLinkNames.Add(linkBodies[iCount].Name, generalBody01.Name);
```

위의 코드는 반복문 안에 입력되는 내용입니다.

- 두번째 Link Body 부터는 General Body 을 생성하면서 Force 도 같이 생성합니다.
- Link Body 를 General Body 로 Convert 한 Body, 그리고 Body 안에 생성한 Marker 와 이전에 Convert 한 General Body, 그리고 그 안에 생성한 Marker 를 generalBody01, generalBody02, generalBody01Marker, generalBody02Marker 로 선언합니다. 만약 General Body 의 이름이 "ImportedBody1" 이면 TMarker 의 위치가 다른 Body 와 다르기 때문에 반대로 선언해야 합니다.
- BeltAssembly 가 2D 인 경우는 Bushing Force 을, 3D 인 경우는 Matrix Force 를 사용해서 Connector 를 생성합니다.

- Belt System 이 Close Loop 인 경우에는, 마지막 Link Body 가 첫번째로 변환한 General Body 와 Connector 를 생성해야 합니다.

```
modelDocument.SetUndoHistory("BNP Body Change");
modelDocument.DeleteEntity(linkBodies[0]);
```

위의 코드는 반복문 다음에 Undo History 를 생성하고 Assembly 를 삭제하는 코드를 추가합니다. 이는 RecurDyn UI 에서 Undo 시 비정상 종료되는 부분을 줄일 수 있습니다.

```
foreach (KeyValuePair<string, string> generalLinkName in generalLinkNames)
{
        IBody linkBody = sub.GetEntity(generalLinkName.Value) as IBody;
        linkBody.Name = generalLinkName.Key;
}
modelDocument.SetUndoHistory("Delete Assembly");
```

위의 코드는 생성한 CAD Body 의 이름을 Link Body 로 변경주는 코드를 추가합니다.

3. **convertCloneBody**() 함수를 생성합니다. 다음 코드를 복사해서 삽입합니다.

```
private IBody convertCloneBody(IReferenceFrame refFrame1,bool useUserCAD,string
useUserCADFilePath ,IBNPBodyBelt linkBody, string generalBodyName)
{
        string cadFileName = "CloneBody1.x_t";
        if (useUserCAD)
        {
                cadFileName = useUserCADFilePath;
        }
        else
        {
                linkBody.GeneralBody.FileExport(path + @"\" + cadFileName, true);
        }
        IBody generalBody = sub.CreateBodyGeneral(generalBodyName);
        generalBody.FileImport(path + @"\" + cadFileName);
        copyBeltBodyInfor(linkBody, generalBody);

        IMarkerCollection dummyMarkers = linkBody.GeneralBody.MarkerCollection;
        List<IMarker> linkMarkers = new List<IMarker>();
        foreach (IMarker marker in dummyMarkers)
        {
                if (marker.Name.Contains("TMarker"))
                {
                        linkMarkers.Add(marker);
                }
        }
        if (linkMarkers.Count == 1 && generalBodyName == "ImportedBody1")
        {
                closeAssembly = false;
        }
        int markerCount = 0;
        IMarker[] generalLinkMarkers = new IMarker[2];
        foreach (IMarker linkMarker in linkMarkers)
        {
                EulerAngle eulerType;
                double[] eulerAngles = { 0, 0, 0 };
                linkMarker.RefFrame.GetEulerAngleDegree(out eulerType, out eulerAngles[0], out
eulerAngles[1], out eulerAngles[2]);
```

```
                refFrame1.SetOrigin(linkMarker.RefFrame.Origin.x, linkMarker.RefFrame.Origin.y,
linkMarker.RefFrame.Origin.z);
                refFrame1.SetEulerAngleDegree(eulerType, eulerAngles[0], eulerAngles[1], eulerAngles[2]);
                generalLinkMarkers[markerCount] = generalBody.CreateMarker("CMMarker" + (markerCount
+ 1).ToString(), refFrame1);
                markerCount++;
        }

        return generalBody;
}
```

- Clone Link Body 를 CAD File 로 Export 하고 Export 한 CAD File 을 Import 하는 과정을 거처서 Clone Body 를 General Body 로 변환합니다.

convertCloneBody() 함수에 대한 설명은 아래에 있습니다.

```
string cadFileName = "CloneBody1.x_t";
if (useUserCAD)
{
        cadFileName = useUserCADFilePath;
}
else
{
        linkBody.GeneralBody.FileExport(path + @"\" + cadFileName, true);
}
IBody generalBody = sub.CreateBodyGeneral(generalBodyName);
generalBody.FileImport(path + @"\" + cadFileName);
copyBeltBodyInfor(linkBody, generalBody);
```

위 코드는 사용자가 **User CAD** 옵션을 선택하면 CAD 파일을 사용하고 옵션을 선택하지 않았으면 입력된 Clone Link Body 를 Export 해주고 다시 Import 해줍니다. 그리고 Clone Link Body 의 정보를 **copyBeltBodyInfor()** 함수를 사용해서 복사해줍니다.

```
IMarkerCollection dummyMarkers = linkBody.GeneralBody.MarkerCollection;
List<IMarker> linkMarkers = new List<IMarker>();
foreach (IMarker marker in dummyMarkers)
{
        if (marker.Name.Contains("TMarker"))
        {
                linkMarkers.Add(marker);
        }
}
if (linkMarkers.Count == 1 && generalBodyName == "ImportedBody1")
{
        closeAssembly = false;
}
```

위 코드는 Clone Link Body 의 TMarker 정보를 저장하는 과정입니다. TMarker 는 Clone Link Body 간의 Connector 의 위치를 나타내는 Marker 입니다. 만약 Link 의 TMarker 의 개수가 1 개인 경우 Belt Assembly 가 Close Loop 가 아닌 Open Loop 입니다.

```
int markerCount = 0;
IMarker[] generalLinkMarkers = new IMarker[2];
foreach (IMarker linkMarker in linkMarkers)
{
	EulerAngle eulerType;
	double[] eulerAngles = { 0, 0, 0 };
	linkMarker.RefFrame.GetEulerAngleDegree(out eulerType, out eulerAngles[0], out eulerAngles[1], out
eulerAngles[2]);
	refFrame1.SetOrigin(linkMarker.RefFrame.Origin.x, linkMarker.RefFrame.Origin.y,
linkMarker.RefFrame.Origin.z);
	refFrame1.SetEulerAngleDegree(eulerType, eulerAngles[0], eulerAngles[1], eulerAngles[2]);
	generalLinkMarkers[markerCount] = generalBody.CreateMarker("CMMarker" + (markerCount +
1).ToString(), refFrame1);
		markerCount++;
}
```

위 코드는 General Body 에 Marker 를 생성해주는 코드입니다.

4. Clone Link Body 의 정보를 General Body 에 복사해 줄 **copyBeltBodyInfor()** 함수를 생성합니다. 아래 코드를 입력하십시오.

```
private void copyBeltBodyInfor(IBNPBodyBelt BeforeBody, IBody AfterBody)
{
	AfterBody.RefFrame.Origin.x = BeforeBody.GeneralBody.RefFrame.Origin.x;
	AfterBody.RefFrame.Origin.y = BeforeBody.GeneralBody.RefFrame.Origin.y;
	AfterBody.RefFrame.Origin.z = BeforeBody.GeneralBody.RefFrame.Origin.z;

	AfterBody.RefFrame.EulerAngle.Type = BeforeBody.GeneralBody.RefFrame.EulerAngle.Type;
	AfterBody.RefFrame.EulerAngle.Alpha.Value =
BeforeBody.GeneralBody.RefFrame.EulerAngle.Alpha.Value;
	AfterBody.RefFrame.EulerAngle.Beta.Value = BeforeBody.GeneralBody.RefFrame.EulerAngle.Beta.Value;
	AfterBody.RefFrame.EulerAngle.Gamma.Value =
BeforeBody.GeneralBody.RefFrame.EulerAngle.Gamma.Value;
	string strMaterialType = BeforeBody.GeneralBody.MaterialInput.ToString();
	if (strMaterialType == "Density")
	{
		AfterBody.MaterialInput = BeforeBody.GeneralBody.MaterialInput;
		if (BeforeBody.GeneralBody.Density2.ParametricValue == null)
		{
			AfterBody.Density2.Value = BeforeBody.GeneralBody.Density2.Value;
		}
		else
		{
			AfterBody.Density2.ParametricValue =
BeforeBody.GeneralBody.Density2.ParametricValue;
		}
	}
	else if (strMaterialType == "UserInput")
	{
		AfterBody.MaterialInput = BeforeBody.GeneralBody.MaterialInput;
		if (BeforeBody.GeneralBody.Mass.ParametricValue == null)
		{
			AfterBody.Mass.Value = BeforeBody.GeneralBody.Mass.Value;
		}
		else
		{
			AfterBody.Mass.ParametricValue = BeforeBody.GeneralBody.Mass.ParametricValue;
		}

		if (BeforeBody.GeneralBody.Ixx.ParametricValue == null)
		{
			AfterBody.Ixx.Value = BeforeBody.GeneralBody.Ixx.Value;
		}
```

```
                else
                {
                        AfterBody.Ixx.ParametricValue = BeforeBody.GeneralBody.Ixx.ParametricValue;
                }
                AfterBody.Mass.ParametricValue = BeforeBody.GeneralBody.Mass.ParametricValue;

        if (BeforeBody.GeneralBody.Ixx.ParametricValue == null)
        {
                AfterBody.Ixx.Value = BeforeBody.GeneralBody.Ixx.Value;
        }
        else
        {
                AfterBody.Ixx.ParametricValue = BeforeBody.GeneralBody.Ixx.ParametricValue;
        }

        if (BeforeBody.GeneralBody.Ixy.ParametricValue == null)
        {
                AfterBody.Ixy.Value = BeforeBody.GeneralBody.Ixy.Value;
        }
        else
        {
                AfterBody.Ixy.ParametricValue = BeforeBody.GeneralBody.Ixy.ParametricValue;
        }

        if (BeforeBody.GeneralBody.Iyy.ParametricValue == null)
        {
                AfterBody.Iyy.Value = BeforeBody.GeneralBody.Iyy.Value;
        }
        else
        {
                AfterBody.Iyy.ParametricValue = BeforeBody.GeneralBody.Iyy.ParametricValue;
        }

        if (BeforeBody.GeneralBody.Iyz.ParametricValue == null)
        {
                AfterBody.Iyz.Value = BeforeBody.GeneralBody.Iyz.Value;
        }
        else
        {
                AfterBody.Iyz.ParametricValue = BeforeBody.GeneralBody.Iyz.ParametricValue;
        }

                if (BeforeBody.GeneralBody.Izx.ParametricValue == null)
                {
                        AfterBody.Izx.Value = BeforeBody.GeneralBody.Izx.Value;
                }
                else
                {
                        AfterBody.Izx.ParametricValue = BeforeBody.GeneralBody.Izx.ParametricValue;
                }

                if (BeforeBody.GeneralBody.Izz.ParametricValue == null)
                {
                        AfterBody.Izz.Value = BeforeBody.GeneralBody.Izz.Value;
                }
                else
                {
                        AfterBody.Izz.ParametricValue = BeforeBody.GeneralBody.Izz.ParametricValue;
                }

        }
        else if (strMaterialType == "DefaultMaterial")
        {
                AfterBody.MaterialInput = BeforeBody.GeneralBody.MaterialInput;
                AfterBody.Material = BeforeBody.GeneralBody.Material;
        }
        else if (strMaterialType == "UserMaterial")
        {
```

```
                AfterBody.MaterialInput = BeforeBody.GeneralBody.MaterialInput;
                AfterBody.Material = BeforeBody.GeneralBody.Material;
        }

}
```

- Clone Link 의 Body 정보를 General Body 에 입력합니다.

Connector 생성하기

1. **createMatrixForce()** 함수를 생성합니다. **Belt Assembly 3D** 는 **Matrix Force** 를 Connector 로 사용하기 때문에 Matrix Force 를 사용해서 Body 간 연결을 해줍니다.

```
private IForceMatrix createMatrixForce(string matrixName, IBNPAssembly assembly, IBody ActionBody, IBody
BaseBody, IReferenceFrame refFrame, IMarker ActionMarker, IMarker BaseMarker)
{
        double BusingForceMarkerX;
        double BusingForceMarkerY;
        double BusingForceMarkerZ;

        EulerAngle eulerType;
        double[] eulerAngles = { 0, 0, 0 };

        IBNPAssemblyConnectingForceParameter assConForce = assembly.ConnectingForceParameter;
        IForceMatrix matrixForce = sub.CreateForceMatrix(matrixName, BaseBody, ActionBody, refFrame,
refFrame);
        BaseMarker.RefFrame.GetOrigin(out BusingForceMarkerX, out BusingForceMarkerY, out
BusingForceMarkerZ);
        BaseMarker.RefFrame.GetEulerAngleDegree(out eulerType, out eulerAngles[0], out eulerAngles[1], out
eulerAngles[2]);
        matrixForce.BaseMarker.RefFrame.SetOrigin(BusingForceMarkerX, BusingForceMarkerY,
BusingForceMarkerZ);
        matrixForce.BaseMarker.RefFrame.SetEulerAngleDegree(eulerType, eulerAngles[0], eulerAngles[1],
eulerAngles[2]);

        ActionMarker.RefFrame.GetOrigin(out BusingForceMarkerX, out BusingForceMarkerY, out
BusingForceMarkerZ);
        ActionMarker.RefFrame.GetEulerAngleDegree(out eulerType, out eulerAngles[0], out eulerAngles[1], out
eulerAngles[2]);
        matrixForce.ActionMarker.RefFrame.SetOrigin(BusingForceMarkerX, BusingForceMarkerY,
BusingForceMarkerZ);
        matrixForce.ActionMarker.RefFrame.SetEulerAngleDegree(eulerType, eulerAngles[0], eulerAngles[1],
eulerAngles[2]);
        for (int iCount = 0; iCount < 6; iCount++)
        {
                for (int jCount = 0; jCount < 6; jCount++)
                {
                        if (assConForce.StiffnessMatrix(iCount, jCount).ParametricValue == null)
                        {
                                matrixForce.StiffnessMatrix(iCount, jCount).Value =
assConForce.StiffnessMatrix(iCount, jCount).Value;
                        }
                        else
                        {
                                matrixForce.StiffnessMatrix(iCount, jCount).ParametricValue =
assConForce.StiffnessMatrix(iCount, jCount).ParametricValue;
                        }
                }
        } for (int iCount = 0; iCount < 6; iCount++)
        {
                if (assConForce.Preload(iCount).ParametricValue == null)
                {
                        matrixForce.Preload(iCount).Value = assConForce.Preload(iCount).Value;
                }
```

```
                else
                {
                        matrixForce.Preload(iCount).ParametricValue =
assConForce.Preload(iCount).ParametricValue;
                }
        }
        for (int iCount = 0; iCount < 6; iCount++)
        {
                if (assConForce.ReferenceLength(iCount).ParametricValue == null)
                {
                        matrixForce.ReferenceLength(iCount).Value =
assConForce.ReferenceLength(iCount).Value;
                }
                else
                {
                        matrixForce.ReferenceLength(iCount).ParametricValue =
assConForce.ReferenceLength(iCount).ParametricValue;
                }
        }

        return matrixForce;
}
```

2. **create2DBeltBusingForce()** 함수를 생성합니다. **Belt Assembly 2D** 는 **Bushing Force** 를 Connector 로 사용하기 때문에 Bushing Force 를 사용해서 Body 간 연결을 해줍니다.

```
private IForceBushing create2DBeltBusingForce(string BusingName, IBNPAssembly2D assembly, IBody ActionBody,
IBody BaseBody, IReferenceFrame refFrame, IMarker ActionMarker, IMarker BaseMarker)
{
        double BusingForceMarkerX;
        double BusingForceMarkerY;
        double BusingForceMarkerZ;

        EulerAngle eulerType;
        double[] eulerAngles = { 0, 0, 0 };

        IBNPAssembly2DBushingForceParameter assemblyBushingForce = assembly.BusingForceParameter;

        IForceBushing BusingForce = sub.CreateForceBushing(BusingName, BaseBody, ActionBody, refFrame);
        BaseMarker.RefFrame.GetOrigin(out BusingForceMarkerX, out BusingForceMarkerY, out
BusingForceMarkerZ);
        BaseMarker.RefFrame.GetEulerAngleDegree(out eulerType, out eulerAngles[0], out eulerAngles[1], out
eulerAngles[2]);
        BusingForce.BaseMarker.RefFrame.SetOrigin(BusingForceMarkerX, BusingForceMarkerY,
BusingForceMarkerZ);
        BusingForce.BaseMarker.RefFrame.SetEulerAngleDegree(eulerType, eulerAngles[0], eulerAngles[1],
eulerAngles[2]);

        ActionMarker.RefFrame.GetOrigin(out BusingForceMarkerX, out BusingForceMarkerY, out
BusingForceMarkerZ);
        ActionMarker.RefFrame.GetEulerAngleDegree(out eulerType, out eulerAngles[0], out eulerAngles[1], out
eulerAngles[2]);
        BusingForce.ActionMarker.RefFrame.SetOrigin(BusingForceMarkerX, BusingForceMarkerY,
BusingForceMarkerZ);
        BusingForce.ActionMarker.RefFrame.SetEulerAngleDegree(eulerType, eulerAngles[0], eulerAngles[1],
eulerAngles[2]);

        BusingForce.UseRadial = true;

        BusingForce.RotationalDampingZ.Coefficient.Value =
assemblyBushingForce.RotationalDampingZ.Coefficient.Value;
        BusingForce.RotationalStiffnessZ.Coefficient.Value =
assemblyBushingForce.RotationalStiffnessZ.Coefficient.Value;
```

```
	BusingForce.RotationalPreloadZ.Value = assemblyBushingForce.RotationalPreloadZ.Value;

	if (assemblyBushingForce.RotationalDampingZ.UseExponentValue)
	{
		BusingForce.RotationalDampingZ.UseExponentValue = true;
		BusingForce.RotationalDampingZ.ExponentValue.Value =
assemblyBushingForce.RotationalDampingZ.ExponentValue.Value;
	}

	if (assemblyBushingForce.RotationalStiffnessZ.UseExponentValue)
	{
		BusingForce.RotationalStiffnessZ.UseExponentValue = true;
		BusingForce.RotationalStiffnessZ.ExponentValue.Value =
assemblyBushingForce.RotationalStiffnessZ.ExponentValue.Value;
	}


	BusingForce.TranslationalDampingX.Coefficient.Value =
assemblyBushingForce.TranslationalDampingX.Coefficient.Value;
	BusingForce.TranslationalDampingY.Coefficient.Value =
assemblyBushingForce.TranslationalDampingY.Coefficient.Value;

	BusingForce.TranslationalStiffnessX.Coefficient.Value =
assemblyBushingForce.TranslationalStiffnessX.Coefficient.Value;
	BusingForce.TranslationalStiffnessY.Coefficient.Value =
assemblyBushingForce.TranslationalStiffnessY.Coefficient.Value;

	return BusingForce;
}
```

Chapter 4

Create Contact 코드

목적

ProcessNet 에서 다이얼로그 윈도우를 생성하는 방법과 ProcessNet 에서 다이얼로그 윈도우를 호출하는 함수를 생성하는 방법 그리고 Contact 을 생성하는 코드를 살펴볼 것입니다.

.

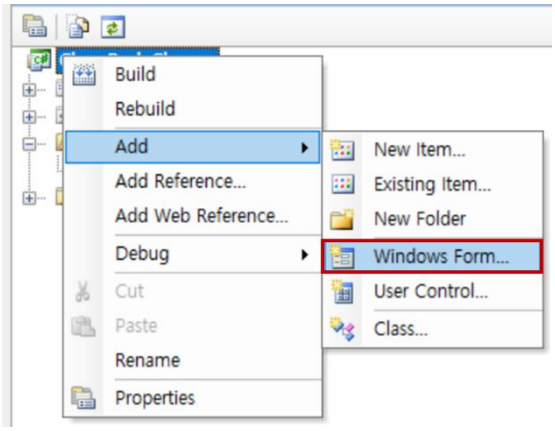
예상 소요 시간

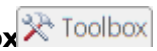
20 분

다이얼로그 생성하기

사용자가 하나의 Link Body 와 Pulley 간의 Contact 을 정의하면, 정의한 Contact 정보를 바탕으로 3 장에서 생성한 Body Group 과 Pulley 간의 Contact 을 생성하는 다이얼로그를 생성하는 법을 배울 것입니다.

CreateContact 다이얼로그 윈도우 생성하기



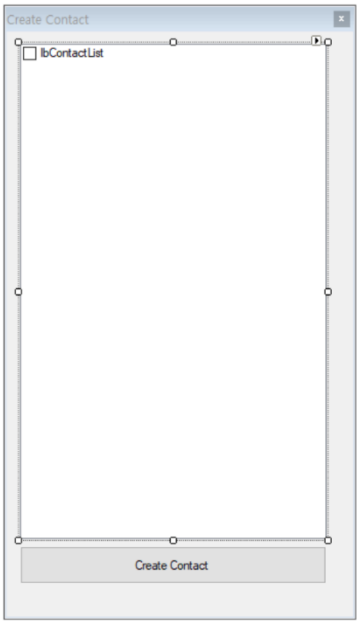
1. **ProcessNet IDE** 의 **Project Explorer** 창에서 **SimpleBeltSystem** 를 마우스로 클릭한후 마우스의 오른쪽을 클릭합니다.
2. **Add – New Item** 을 클릭합니다.
3. 다음과 같이 **Add New Item** 다이얼 로그 창이 나타나면 **Windows Form** 아이콘을 클릭하고 Name 을 **CreateContact** 로 입력합니다.
4. **Add** 버튼을 클릭합니다.
5. **Windows Form** 을 위한 설계 윈도우인 **CreateContact.cs[Design]**이 **IDE Project Editor** 윈도우에 나타납니다.
6. 화면 왼쪽상단에 있는 **CreateContact** 다이얼로그를 클릭합니다.
7. **화면 오른쪽 하단의 Properties Windows** 에 **ChangeGeneralBody** 의 정보가 나타나는데 여기서 **size** 를 341, 544 로 수정합니다.
8. 마찬가지로 **FormBorderStype** 을 **FixedToolWindow** 로 수정합니다.
9. 화면 왼쪽 상단에서 **ToolBox** 

위로 커서를 이동시킵니다. 그러면 다이얼로그 윈도우 및 버튼, 기타 비슷한 제어기능을 추가할 수 있는 메뉴가 보여집니다.
10. **CheckedListBox1**, **Button1**, **TextBox1** 는 다음 표에 있는 내용을 참조해서 Text 와 Name 의 값을 수정합니다.

Dialog Element	Text	Name	Location	Size
CheckedListBox1		lbContactList	12, 12	299, 424
Button1	Create Contact	btCreateContact	12, 451	299, 37

11. 위의 과정을 수행하면 오른쪽의 그림과 같이 다이얼로그가 완성된 것을 확인할 수 있습니다.

12. **File – Save CreateContact.cs** 버튼을 클릭해서 **Save** 를 수행합니다.



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.

CreateContact 다이얼로그 윈도우의 초기환경 정의하기

1. **Project Explorer** 에서 **CreateContact.cs** 을 선택한 후, 오른쪽 버튼을 클릭합니다. **View Code** 를 선택하면 **CreateContact.cs** 의 소스 코드가 **IDE Project** 편집창에 나타납니다.
2. 편집창에 다이얼로그 윈도우에 사용할 변수를 입력합니다.

```
using FunctionBay.RecurDyn.ProcessNet;
using FunctionBay.RecurDyn.ProcessNet.BNP;

namespace SimpleBeltSystem
{
	public partial class CreateContact: Form
	{
		IModelDocument modelDocument;
		ISubSystem sub;
		public CreateContact(IModelDocument modelDoc)
		{
			InitializeComponent();
			modelDocument = modelDoc;
			sub = modelDocument.GetDataStorage() as ISubSystem;
		}
	}
}
```

- **FunctionBay.RecurDyn.ProcessNet, FunctionBay.RecurDyn.ProcessNet.BNP** 는 **ProcessNet** 의 함수를 사용하기 위한 **Reference** 입니다.

3. **Project Explorer** 의 **CreateContact.cs** 파일을 클릭하고 오른쪽 버튼을 클릭 후, **View Designer** 버튼을 클릭한 후, 다이얼로그 윈도우 상의 **Create Contact** 버튼을 더블 클릭하여 **btCreateContact_Click()** 함수를 생성합니다.
4. **btCreateContact_Click()** 함수에 아래의 코드를 입력합니다.

```
private void btCreateContact_Click(object sender, EventArgs e)
{
	if (this.lbContactList.CheckedItems.Count != 0)
	{
		for (int assemblyCount = 0; assemblyCount <= this.lbContactList.Items.Count - 1;
assemblyCount++)
		{
			if (this.lbContactList.GetItemCheckState(assemblyCount) == CheckState.Checked)
			{
				string[] assembliesName =
this.lbContactList.Items[assemblyCount].ToString().Split('.');
				createContact(assembliesName[0], assembliesName[1]);
				this.lbContactList.SetItemChecked(assemblyCount, false);
			}
		}
	}
}
```

5. 다시한번 **Project Explorer** 의 **CreateContact.cs** 파일을 클릭하고 오른쪽 버튼을 클릭 후, **View Designer** 를 클릭합니다.
6. 다이얼로그 윈도우 빈 곳을 더블 클릭합니다.
7. 아래와 같이 **CreateContact_Load()** 함수가 생성되는데 함수 안에 아래의 코드를 입력합니다.

```
private void CreateContact_Load(object sender, EventArgs e)
{
	ISubSystem sub = modelDocument.GetDataStorage() as ISubSystem;
	IContactCollection contacts = sub.ContactCollection as IContactCollection;
	foreach (IContact contact in contacts)
	{
		string contactType = Microsoft.VisualBasic.Information.TypeName(contact);
		if (contactType == "IContactGeoSurface" || contactType == "IContactGeoCurve")
		{
			string groupName = findContact(contact);
			if (groupName != "")
			{
				this.lbContactList.Items.Add(groupName + "." + contact.Name, true);
			}
		}
	}
}
```

- 다이얼로그 윈도우가 실행될 때 Group Body 에 연결된 Contact 을 찾아오는 함수입니다.

8. **Group Body** 에 연결된 **Contact** 을 찾기 위해 **findContact()** 함수를 생성합니다.

```
public string findContact(IContact contact)
{
        string groupName = "";
        IGeometry actionGeometry = contact.ActionEntity;
        IGeometry baseGeometry = contact.BaseEntity;

        string[] actiondummy = actionGeometry.FullName.Split('.');
        string[] basedummy = baseGeometry.FullName.Split('.');

        string actionName = actiondummy[0];
        string baseName = basedummy[0];

        ISubSystem sub = modelDocument.GetDataStorage() as ISubSystem;
        IGroupGeneralCollection groupGenerals = sub.GroupGeneralCollection as IGroupGeneralCollection;
        if (groupGenerals.Count > 1)
        {
                foreach (IGroupGeneral groupGeneral in groupGenerals)
                {
                        IGenericCollection generics = groupGeneral.GroupEntities();
                        if (generics.Count > 1)
                        {
                                foreach (IGeneric generic in generics)
                                {
                                        if (generic.FullName.Contains(actionName) ||
                                generic.FullName.Contains(baseName))
                                        {
                                                groupName = groupGeneral.Name;
                                                break;
                                        }
                                }
                                if (groupName !="")
                                        break;
                        }
                }
        }
        return groupName;
}
```

- General Body 로 이루어진 Group 에 Geometry 가 Contact 의 Geometry 와 일치하면 Contact 이 있다고 판단합니다.

Contact 생성하기

Belt 의 Body, Connector 의 경우 Assembly 에서 ProcessNet 을 이용하여 정보를 가지고 올 수 있었지만 Contact 의 경우 정보를 가지고 올 수 없습니다. 그래서 유저가 직접 General Contact 을 1 개 정의하면 정의한 Contact 을 ProcessNet 을 사용해서 Pulley 와 Assembly 간의 Contact 을 생성해줍니다. 이 과정에서 Geo Surface Contact 와 Geo Curve Contact 을 사용해서 Contact 을 생성하는 법을 배울 것입니다.

Contact 생성하기

1. **createContact()** 함수를 생성합니다. 아래 코드를 복사해서 삽입합니다.

- General Body 와 Pulley 간 Contact 을 생성하는 함수입니다. Contact 은 Geo Surface Contact, Geo Curve Contact 함수를 사용합니다.

```
public void createContact(string generalGroupName, string contactName)
{
	IContact contact = sub.GetEntity(contactName) as IContact;
	IGroupGeneral groupGeneral = sub.GetEntity(generalGroupName) as IGroupGeneral;
	IGenericCollection generics = groupGeneral.GroupEntities();

	IContactGeoSurface geoSurface = contact as IContactGeoSurface;
	IContactGeoCurve geoCurve = contact as IContactGeoCurve;
	IGeometry actionGeometry = contact.ActionEntity as IGeometry;
	IGeometry baseGeometry = contact.BaseEntity as IGeometry;

	string actionBodyName = actionGeometry.FullName.Split('.')[0];
	string baseBodyName = baseGeometry.FullName.Split('.')[0];
	bool contactActionbody = false;
	foreach (IGeneric linkbody in generics)
	{
		if (linkbody.Name == actionBodyName)
		{
			contactActionbody = true;
			break;
		}
	}
	IGroupGeneral contactgroup = sub.CreateGroupGeneral("Contact_" + generalGroupName + "_" +
contactName, new object[] { });
	if (geoSurface != null)
	{
		foreach (IBody body in generics)
		{
			if (contactActionbody)
			{
				string[] dummy = actionGeometry.FullName.Split('.');
				dummy[0] = body.Name;
				string linkGeometryName = null;
				for (int count = 0; count < dummy.Length; count++)
				{
					if (count == dummy.Length - 1)
					{
						linkGeometryName += dummy[count];
					}
					else
					{
						linkGeometryName += dummy[count] + ".";
					}
				}
				actionGeometry =
modelDocument.GetEntityFromFullName(linkGeometryName) as IGeometry;

			}
			else
			{
				string[] dummy = baseGeometry.FullName.Split('.');
				dummy[0] = body.Name;
				string linkGeometryName = null;
				for (int count = 0; count < dummy.Length; count++)
				{
					if (count == dummy.Length - 1)
					{
						linkGeometryName += dummy[count];
					}
					else
					{
						linkGeometryName += dummy[count] + ".";
					}
				}
```

```
                                }
                                geoSurface = sub.CreateContactGeoSurface(contactName + "_" + body.Name,
baseGeometry, actionGeometry);
                                contactgroup.AddEntities(new object[] { geoSurface });
                        }
                }
                else if (geoCurve != null)
                {
                        foreach (IBody body in generics)
                        {
                                if (contactActionbody)
                                {
                                        string[] dummy = actionGeometry.FullName.Split('.');
                                        dummy[0] = body.Name;
                                        string linkGeometryName = null;
                                        for (int count = 0; count < dummy.Length; count++)
                                        {
                                                if (count == dummy.Length - 1)
                                                {
                                                        linkGeometryName += dummy[count];
                                                }
                                                else
                                                {
                                                        linkGeometryName += dummy[count] + ".";
                                                }
                                        }
                                        actionGeometry =
modelDocument.GetEntityFromFullName(linkGeometryName) as IGeometry;

                                }
                                else
                                {
                                        string[] dummy = baseGeometry.FullName.Split('.');
                                        dummy[0] = body.Name;
                                        string linkGeometryName = null;
                                        for (int count = 0; count < dummy.Length; count++)
                                        {
                                                if (count == dummy.Length - 1)
                                                {
                                                        linkGeometryName += dummy[count];
                                                }
                                                else
                                                {
                                                        linkGeometryName += dummy[count] + ".";
                                                }
                                        }
                                        baseGeometry =
modelDocument.GetEntityFromFullName(linkGeometryName) as IGeometry;

                                }
                                geoCurve = sub.CreateContactGeoCurve(contactName + "_" + body.Name,
baseGeometry, actionGeometry);
                                contactgroup.AddEntities(new object[] { geoCurve });
                        }

                }

                IGenericCollection contacts = contactgroup.GroupEntities();

                foreach (IContact cpoyContact in contacts)
                {
                        copyContactProf(cpoyContact, contact);
                }
                modelDocument.DeleteEntity(contact);
                modelDocument.SetUndoHistory("Create Contact");
        }
```

createContact() 함수에 대한 설명은 아래와 같습니다.

```
IContact contact = sub.GetEntity(contactName) as IContact;
IGroupGeneral groupGeneral = sub.GetEntity(generalGroupName) as IGroupGeneral;
IGenericCollection generics = groupGeneral.GroupEntities();

IContactGeoSurface geoSurface = contact as IContactGeoSurface;
IContactGeoCurve geoCurve = contact as IContactGeoCurve;
IGeometry actionGeometry = contact.ActionEntity as IGeometry;
IGeometry baseGeometry = contact.BaseEntity as IGeometry;

string actionBodyName = actionGeometry.FullName.Split('.')[0];
string baseBodyName = baseGeometry.FullName.Split('.')[0];
```

위의 코드는 **createContact()** 함수에서 사용할 변수를 선언해줍니다.

- **Contact**: General Body 와 Pulley 사이의 Contact
- **groupGeneral**: Clone Body 를 Convert 한 General Body 를 모아둔 Group General
- **actionBodyName**: Action Body 의 이름
- **baseBodyName**: Base Body 의 이름

```
bool contactActionbody = false;
foreach (IGeneric linkbody in generics)
{
	if (linkbody.Name == actionBodyName)
	{
		contactActionbody = true;
		break;
	}
}
```

위의 코드는 Contact 의 Action Entity 가 General Body 인지 Pulley 인지 확인하는 조건식을 추가합니다. Pulley 가 Action Entity 면 Base Entity 의 값을 반복문을 통해서 Contact 을 생성하고 반대인 경우 Action Entity 의 값을 반복문을 통해서 Contact 을 생성합니다.

```
IGroupGeneral contactgroup = sub.CreateGroupGeneral("Contact_" + generalGroupName + "_" + contactName,
new object[] { });
```

위의 코드는 생성한 Contact 을 Group General 에 추가하기 위해서 Group General 을 생성하는 코드를 추가합니다.

```
if (geoSurface != null)
{
	foreach (IBody body in generics)
	{
		if (contactActionbody)
		{
			string[] dummy = actionGeometry.FullName.Split('.');
			dummy[0] = body.Name;
			string linkGeometryName = null;
			for (int count = 0; count < dummy.Length; count++)
			{
				if (count == dummy.Length - 1)
				{
					linkGeometryName += dummy[count];
				}
				else
				{
					linkGeometryName += dummy[count] + ".";
				}
			}
			actionGeometry = modelDocument.GetEntityFromFullName(linkGeometryName)
		as IGeometry;

		}
		else
		{
			string[] dummy = baseGeometry.FullName.Split('.');
			dummy[0] = body.Name;
			string linkGeometryName = null;
			for (int count = 0; count < dummy.Length; count++)
			{
				if (count == dummy.Length - 1)
				{
					linkGeometryName += dummy[count];
				}
				else
				{
					linkGeometryName += dummy[count] + ".";
				}
			}
		}
		geoSurface = sub.CreateContactGeoSurface(contactName + "_" + body.Name,
baseGeometry, actionGeometry);
		contactgroup.AddEntities(new object[] { geoSurface });
	}
}
```

위의 코드는 Pulley 와 General Body 사이에 Geo Surface Contact 을 생성하는 코드입니다. Pulley 와 generics Collection 에 포함된 Body 사이의 Geo Surface Contact 을 생성합니다. Link Body 가 Action 이냐 Base 이냐를 구분해서 반복문을 수행합니다.

- FullName 매서드를 실행하면 Body 이름.Geometry 이름.Face 이름@Subsystem 이름의 형식으로 반환되기 때문에 Split 함수를 사용해서 맨 앞의 Body 이름을 분리해서 다른 Body 의 이름으로 변경해서 Contact 을 생성합니다.

```
else if (geoCurve != null)
{
	foreach (IBody body in generics)
	{
		if (contactActionbody)
		{
			string[] dummy = actionGeometry.FullName.Split('.');
			dummy[0] = body.Name;
			string linkGeometryName = null;
			for (int count = 0; count < dummy.Length; count++)
			{
				if (count == dummy.Length - 1)
				{
					linkGeometryName += dummy[count];
				}
				else
				{
					linkGeometryName += dummy[count] + ".";
				}
			}
			actionGeometry = modelDocument.GetEntityFromFullName(linkGeometryName)
		as IGeometry;

		}
		else
		{
			string[] dummy = baseGeometry.FullName.Split('.');
			dummy[0] = body.Name;
			string linkGeometryName = null;
			for (int count = 0; count < dummy.Length; count++)
			{
				if (count == dummy.Length - 1)
				{
					linkGeometryName += dummy[count];
				}
				else
				{
					linkGeometryName += dummy[count] + ".";
				}
			}
			baseGeometry = modelDocument.GetEntityFromFullName(linkGeometryName)
		as IGeometry;

		}
		geoCurve = sub.CreateContactGeoCurve(contactName + "_" + body.Name, baseGeometry,
	actionGeometry);
		contactgroup.AddEntities(new object[] { geoCurve });
		}

}
```

위의 코드는 Pulley 와 General Body 사이에 Geo Curve Contact 을 생성하는 코드입니다. Geo Surface Contact 과 동일한 과정을 진행합니다.

```
IGenericCollection contacts = contactgroup.GroupEntities();

foreach (IContact cpoyContact in contacts)
{
        copyContactProf(cpoyContact, contact);
}
modelDocument.DeleteEntity(contact);
modelDocument.SetUndoHistory("Create Contact");
```

위의 코드는 Contact 정보를 복사하는 코드와 Undo History 를 추가합니다.

2. Contact Property 값을 복사하는 **copyContactProf()** 함수를 생성합니다.

```
private void copyContactProf(IContact copyContact, IContact originalContact)
{
        string contactType = Microsoft.VisualBasic.Information.TypeName(originalContact);
        if (contactType == "IContactGeoSurface")
        {
                IContactGeoSurface copyGeoSurface = copyContact as IContactGeoSurface;
                IContactGeoSurface originalGeoSurface = originalContact as IContactGeoSurface;

                copyGeoSurface.ActionPatchOption.SurfaceType =
originalGeoSurface.ActionPatchOption.SurfaceType;
                if (originalGeoSurface.ActionPatchOption.BoundingBufferLength.ParametricValue == null)
                {
                        copyGeoSurface.ActionPatchOption.BoundingBufferLength.Value =
originalGeoSurface.ActionPatchOption.BoundingBufferLength.Value;

                }
                else
                {
                        copyGeoSurface.ActionPatchOption.BoundingBufferLength.ParametricValue =
originalGeoSurface.ActionPatchOption.BoundingBufferLength.ParametricValue;

                }
                copyGeoSurface.ActionPatchOption.UsePlaneToleranceFactor =
originalGeoSurface.ActionPatchOption.UsePlaneToleranceFactor;
                if (originalGeoSurface.ActionPatchOption.UsePlaneToleranceFactor)
                {
                        if (originalGeoSurface.ActionPatchOption.PlaneToleranceFactor.ParametricValue ==
null)
                        {
                                copyGeoSurface.ActionPatchOption.PlaneToleranceFactor.Value =
originalGeoSurface.ActionPatchOption.PlaneToleranceFactor.Value;

                        }
                        else
                        {

        copyGeoSurface.ActionPatchOption.PlaneToleranceFactor.ParametricValue =
originalGeoSurface.ActionPatchOption.PlaneToleranceFactor.ParametricValue;

                        }
                }

                copyGeoSurface.ActionPatchOption.UseMaxFacetSizeFactor =
originalGeoSurface.ActionPatchOption.UseMaxFacetSizeFactor;
                if (originalGeoSurface.ActionPatchOption.UseMaxFacetSizeFactor)
                {
                        if (originalGeoSurface.ActionPatchOption.MaxFacetSizeFactor.ParametricValue ==
null)
                        {
```

```
                                        copyGeoSurface.ActionPatchOption.MaxFacetSizeFactor.Value =
originalGeoSurface.ActionPatchOption.MaxFacetSizeFactor.Value;

                                }
                                else
                                {
                                        copyGeoSurface.ActionPatchOption.MaxFacetSizeFactor.ParametricValue
= originalGeoSurface.ActionPatchOption.MaxFacetSizeFactor.ParametricValue;

                                }
                        }
                        copyGeoSurface.ActionPatchOption.UseCubicCell =
originalGeoSurface.ActionPatchOption.UseCubicCell;
                        if (originalGeoSurface.ActionPatchOption.UseCubicCell)
                        {
                                copyGeoSurface.ActionPatchOption.CubicCell =
originalGeoSurface.ActionPatchOption.CubicCell;
                        }

                        copyGeoSurface.BasePatchOption.SurfaceType =
originalGeoSurface.BasePatchOption.SurfaceType;
                        if (originalGeoSurface.ActionPatchOption.BoundingBufferLength.ParametricValue == null)
                        {
                                copyGeoSurface.ActionPatchOption.BoundingBufferLength.Value =
originalGeoSurface.ActionPatchOption.BoundingBufferLength.Value;

                        }
                        else
                        {
                                copyGeoSurface.ActionPatchOption.BoundingBufferLength.ParametricValue =
originalGeoSurface.ActionPatchOption.BoundingBufferLength.ParametricValue;

                        }
                        copyGeoSurface.ActionPatchOption.UsePlaneToleranceFactor =
originalGeoSurface.ActionPatchOption.UsePlaneToleranceFactor;
                        if (originalGeoSurface.ActionPatchOption.UsePlaneToleranceFactor)
                        {
                                if (originalGeoSurface.ActionPatchOption.PlaneToleranceFactor.ParametricValue ==
null)
                                {
                                        copyGeoSurface.ActionPatchOption.PlaneToleranceFactor.Value =
originalGeoSurface.ActionPatchOption.PlaneToleranceFactor.Value;

                                }
                                else
                                {

            copyGeoSurface.ActionPatchOption.PlaneToleranceFactor.ParametricValue =
originalGeoSurface.ActionPatchOption.PlaneToleranceFactor.ParametricValue;

                                }
                        }

                        copyGeoSurface.ActionPatchOption.UseMaxFacetSizeFactor =
originalGeoSurface.ActionPatchOption.UseMaxFacetSizeFactor;
                        if (originalGeoSurface.ActionPatchOption.UseMaxFacetSizeFactor)
                        {
                                if (originalGeoSurface.ActionPatchOption.MaxFacetSizeFactor.ParametricValue ==
null)
                                {
                                        copyGeoSurface.ActionPatchOption.MaxFacetSizeFactor.Value =
originalGeoSurface.ActionPatchOption.MaxFacetSizeFactor.Value;

                                }
                                else
                                {
                                        copyGeoSurface.ActionPatchOption.MaxFacetSizeFactor.ParametricValue
= originalGeoSurface.ActionPatchOption.MaxFacetSizeFactor.ParametricValue;
```

```
                        }
                }
                copyGeoSurface.ActionPatchOption.UseCubicCell =
originalGeoSurface.ActionPatchOption.UseCubicCell;
                if (originalGeoSurface.ActionPatchOption.UseCubicCell)
                {
                        copyGeoSurface.ActionPatchOption.CubicCell =
originalGeoSurface.ActionPatchOption.CubicCell;
                }

                copyGeoSurface.BasePatchOption.SurfaceType =
originalGeoSurface.BasePatchOption.SurfaceType;
                if (originalGeoSurface.BasePatchOption.BoundingBufferLength.ParametricValue == null)
                {
                        copyGeoSurface.BasePatchOption.BoundingBufferLength.Value =
originalGeoSurface.BasePatchOption.BoundingBufferLength.Value;

                }
                else
                {
                        copyGeoSurface.BasePatchOption.BoundingBufferLength.ParametricValue =
originalGeoSurface.BasePatchOption.BoundingBufferLength.ParametricValue;

                }
                copyGeoSurface.BasePatchOption.UsePlaneToleranceFactor =
originalGeoSurface.BasePatchOption.UsePlaneToleranceFactor;
                if (originalGeoSurface.BasePatchOption.UsePlaneToleranceFactor)
                {
                        if (originalGeoSurface.BasePatchOption.PlaneToleranceFactor.ParametricValue ==
null)
                        {
                                copyGeoSurface.BasePatchOption.PlaneToleranceFactor.Value =
originalGeoSurface.BasePatchOption.PlaneToleranceFactor.Value;

                        }
                        else
                        {
                                copyGeoSurface.BasePatchOption.PlaneToleranceFactor.ParametricValue
= originalGeoSurface.BasePatchOption.PlaneToleranceFactor.ParametricValue;

                        }
                }

                copyGeoSurface.BasePatchOption.UseMaxFacetSizeFactor =
originalGeoSurface.BasePatchOption.UseMaxFacetSizeFactor;
                if (originalGeoSurface.BasePatchOption.UseMaxFacetSizeFactor)
                {
                        if (originalGeoSurface.BasePatchOption.MaxFacetSizeFactor.ParametricValue ==
null)
                        {
                                copyGeoSurface.BasePatchOption.MaxFacetSizeFactor.Value =
originalGeoSurface.BasePatchOption.MaxFacetSizeFactor.Value;

                        }
                        else
                        {
                                copyGeoSurface.BasePatchOption.MaxFacetSizeFactor.ParametricValue =
originalGeoSurface.BasePatchOption.MaxFacetSizeFactor.ParametricValue;

                        }
                }
                copyGeoSurface.BasePatchOption.UseCubicCell =
originalGeoSurface.BasePatchOption.UseCubicCell;
                if (originalGeoSurface.BasePatchOption.UseCubicCell)
                {
                        copyGeoSurface.BasePatchOption.CubicCell =
originalGeoSurface.BasePatchOption.CubicCell;
```

```
                }

                copyGeoSurface.BasePatchOption.SurfaceType =
originalGeoSurface.BasePatchOption.SurfaceType;
                if (originalGeoSurface.BasePatchOption.BoundingBufferLength.ParametricValue == null)
                {
                        copyGeoSurface.BasePatchOption.BoundingBufferLength.Value =
originalGeoSurface.BasePatchOption.BoundingBufferLength.Value;

                }
                else
                {
                        copyGeoSurface.BasePatchOption.BoundingBufferLength.ParametricValue =
originalGeoSurface.BasePatchOption.BoundingBufferLength.ParametricValue;

                }
                copyGeoSurface.BasePatchOption.UsePlaneToleranceFactor =
originalGeoSurface.BasePatchOption.UsePlaneToleranceFactor;
                if (originalGeoSurface.BasePatchOption.UsePlaneToleranceFactor)
                {
                        if (originalGeoSurface.BasePatchOption.PlaneToleranceFactor.ParametricValue ==
null)
                        {
                                copyGeoSurface.BasePatchOption.PlaneToleranceFactor.Value =
originalGeoSurface.BasePatchOption.PlaneToleranceFactor.Value;

                        }
                        else
                        {
                                copyGeoSurface.BasePatchOption.PlaneToleranceFactor.ParametricValue
= originalGeoSurface.BasePatchOption.PlaneToleranceFactor.ParametricValue;

                        }
                }

                copyGeoSurface.BasePatchOption.UseMaxFacetSizeFactor =
originalGeoSurface.BasePatchOption.UseMaxFacetSizeFactor;
                if (originalGeoSurface.BasePatchOption.UseMaxFacetSizeFactor)
                {
                        if (originalGeoSurface.BasePatchOption.MaxFacetSizeFactor.ParametricValue ==
null)
                        {
                                copyGeoSurface.BasePatchOption.MaxFacetSizeFactor.Value =
originalGeoSurface.BasePatchOption.MaxFacetSizeFactor.Value;

                        }
                        else
                        {
                                copyGeoSurface.BasePatchOption.MaxFacetSizeFactor.ParametricValue =
originalGeoSurface.BasePatchOption.MaxFacetSizeFactor.ParametricValue;

                        }
                }
                copyGeoSurface.BasePatchOption.UseCubicCell =
originalGeoSurface.BasePatchOption.UseCubicCell;
                if (originalGeoSurface.BasePatchOption.UseCubicCell)
                {
                        copyGeoSurface.BasePatchOption.CubicCell =
originalGeoSurface.BasePatchOption.CubicCell;
                }
                copyGeoSurface.ActionUpDirection = originalGeoSurface.ActionUpDirection;
                copyGeoSurface.ActionNodeContact = originalGeoSurface.ActionNodeContact;

                copyGeoSurface.BaseUpDirection = originalGeoSurface.BaseUpDirection;
                copyGeoSurface.BaseNodeContact = originalGeoSurface.BaseNodeContact;

                copyGeoSurface.EdgeContact = originalGeoSurface.EdgeContact;
```

```
                copyGeoSurface.UseCPM = originalGeoSurface.UseCPM;
                copyGeoSurface.SmoothNodeContact = originalGeoSurface.SmoothNodeContact;

                if (originalGeoSurface.ContactProperty.UseStiffnessSpline)
                {
                        copyGeoSurface.ContactProperty.StiffnessSpline =
originalGeoSurface.ContactProperty.StiffnessSpline;
                }
                else
                {
                        if (originalGeoSurface.ContactProperty.StiffnessCoefficient.ParametricValue == null)
                        {
                                copyGeoSurface.ContactProperty.StiffnessCoefficient.Value =
originalGeoSurface.ContactProperty.StiffnessCoefficient.Value;
                        }
                        else
                        {
                                copyGeoSurface.ContactProperty.StiffnessCoefficient.ParametricValue =
originalGeoSurface.ContactProperty.StiffnessCoefficient.ParametricValue;
                        }
                }

                if (originalGeoSurface.ContactProperty.UseDampingSpline)
                {
                        copyGeoSurface.ContactProperty.DampingSpline =
originalGeoSurface.ContactProperty.DampingSpline;
                }
                else
                {
                        if (originalGeoSurface.ContactProperty.DampingCoefficient.ParametricValue ==
null)
                        {
                                copyGeoSurface.ContactProperty.DampingCoefficient.Value =
originalGeoSurface.ContactProperty.DampingCoefficient.Value;
                        }
                        else
                        {
                                copyGeoSurface.ContactProperty.DampingCoefficient.ParametricValue =
originalGeoSurface.ContactProperty.DampingCoefficient.ParametricValue;
                        }
                }

                copyGeoSurface.ContactProperty.Friction.ContactFrictionType =
originalGeoSurface.ContactProperty.Friction.ContactFrictionType;
                if (copyGeoSurface.ContactProperty.Friction.ContactFrictionType ==
ContactFrictionType.CoefficientSpline)
                {
                        copyGeoSurface.ContactProperty.Friction.Spline =
originalGeoSurface.ContactProperty.Friction.Spline;
                }
                else if (copyGeoSurface.ContactProperty.Friction.ContactFrictionType ==
ContactFrictionType.ForceSpline)
                {
                        copyGeoSurface.ContactProperty.Friction.Spline =
originalGeoSurface.ContactProperty.Friction.Spline;
                }
                else
                {
                        if (originalGeoSurface.ContactProperty.Friction.Coefficient.ParametricValue == null)
                        {
                                copyGeoSurface.ContactProperty.Friction.Coefficient.Value =
originalGeoSurface.ContactProperty.Friction.Coefficient.Value;
                        }
                        else
                        {
                                copyGeoSurface.ContactProperty.Friction.Coefficient.ParametricValue =
originalGeoSurface.ContactProperty.Friction.Coefficient.ParametricValue;
                        }
```

```
                                    if
(originalGeoSurface.ContactProperty.Friction.StaticThresholdVelocity.ParametricValue == null)
                                    {
                                                copyGeoSurface.ContactProperty.Friction.StaticThresholdVelocity.Value =
originalGeoSurface.ContactProperty.Friction.StaticThresholdVelocity.Value;
                                    }
                                    else
                                    {

            copyGeoSurface.ContactProperty.Friction.StaticThresholdVelocity.ParametricValue =
originalGeoSurface.ContactProperty.Friction.StaticThresholdVelocity.ParametricValue;

                                    }

                                    if
(originalGeoSurface.ContactProperty.Friction.DynamicThresholdVelocity.ParametricValue == null)
                                    {

            copyGeoSurface.ContactProperty.Friction.DynamicThresholdVelocity.Value =
originalGeoSurface.ContactProperty.Friction.DynamicThresholdVelocity.Value;
                                    }
                                    else
                                    {

            copyGeoSurface.ContactProperty.Friction.DynamicThresholdVelocity.ParametricValue =
originalGeoSurface.ContactProperty.Friction.DynamicThresholdVelocity.ParametricValue;

                                    }
                                    if (originalGeoSurface.ContactProperty.Friction.StaticCoefficient.ParametricValue ==
null)
                                    {
                                                copyGeoSurface.ContactProperty.Friction.StaticCoefficient.Value =
originalGeoSurface.ContactProperty.Friction.StaticCoefficient.Value;
                                    }
                                    else
                                    {

            copyGeoSurface.ContactProperty.Friction.StaticCoefficient.ParametricValue =
originalGeoSurface.ContactProperty.Friction.StaticCoefficient.ParametricValue;
                                    }
                                    if (originalGeoSurface.ContactProperty.Friction.MaximumForce.ParametricValue ==
null)
                                    {
                                                copyGeoSurface.ContactProperty.Friction.MaximumForce.Value =
originalGeoSurface.ContactProperty.Friction.MaximumForce.Value;
                                    }
                                    else
                                    {

            copyGeoSurface.ContactProperty.Friction.MaximumForce.ParametricValue =
originalGeoSurface.ContactProperty.Friction.MaximumForce.ParametricValue;
                                    }
                                    copyGeoSurface.ContactProperty.Friction.UseMaximumForce =
originalGeoSurface.ContactProperty.Friction.UseMaximumForce;


                        }

                        if (originalGeoSurface.ContactProperty.StiffnessExponent.ParametricValue == null)
                        {
                                    copyGeoSurface.ContactProperty.StiffnessExponent.Value =
originalGeoSurface.ContactProperty.StiffnessExponent.Value;
                        }
                        else
                        {
                                    copyGeoSurface.ContactProperty.StiffnessExponent.ParametricValue =
originalGeoSurface.ContactProperty.StiffnessExponent.ParametricValue;
```

```
                }


                if (originalGeoSurface.ContactPropertyAdditional.ReboundDampingFactor.ParametricValue ==
null)
                {
                        copyGeoSurface.ContactPropertyAdditional.ReboundDampingFactor.Value =
originalGeoSurface.ContactPropertyAdditional.ReboundDampingFactor.Value;
                }
                else
                {

        copyGeoSurface.ContactPropertyAdditional.ReboundDampingFactor.ParametricValue =
originalGeoSurface.ContactPropertyAdditional.ReboundDampingFactor.ParametricValue;
                }

                if (originalGeoSurface.ContactPropertyAdditional.GlobalMaxPenetration.ParametricValue ==
null)
                {
                        copyGeoSurface.ContactPropertyAdditional.GlobalMaxPenetration.Value =
originalGeoSurface.ContactPropertyAdditional.GlobalMaxPenetration.Value;
                }
                else
                {
                        copyGeoSurface.ContactPropertyAdditional.GlobalMaxPenetration.ParametricValue
= originalGeoSurface.ContactPropertyAdditional.GlobalMaxPenetration.ParametricValue;
                }


                if (copyGeoSurface.ContactPropertyAdditional.ContactForceType ==
ContactForceType.BoundaryPenetration)
                {
                        if
(originalGeoSurface.ContactPropertyAdditional.BoundaryPenetration.ParametricValue == null)
                        {
                                copyGeoSurface.ContactPropertyAdditional.BoundaryPenetration.Value =
originalGeoSurface.ContactPropertyAdditional.BoundaryPenetration.Value;
                        }
                        else
                        {

        copyGeoSurface.ContactPropertyAdditional.BoundaryPenetration.ParametricValue =
originalGeoSurface.ContactPropertyAdditional.BoundaryPenetration.ParametricValue;
                        }
                }
                else
                {
                        if (originalGeoSurface.ContactProperty.UseDampingExponent)
                        {
                                if
(originalGeoSurface.ContactProperty.DampingExponent.ParametricValue == null)
                                {
                                        copyGeoSurface.ContactProperty.DampingExponent.Value =
originalGeoSurface.ContactProperty.DampingExponent.Value;
                                }
                                else
                                {

        copyGeoSurface.ContactProperty.DampingExponent.ParametricValue =
originalGeoSurface.ContactProperty.DampingExponent.ParametricValue;
                                }
                        }
                        if (originalGeoSurface.ContactProperty.UseIndentationExponent)
                        {
                                if
(originalGeoSurface.ContactProperty.IndentationExponent.ParametricValue == null)
                                {
```

```
                                                copyGeoSurface.ContactProperty.IndentationExponent.Value =
originalGeoSurface.ContactProperty.IndentationExponent.Value;
                                        }
                                        else
                                        {

            copyGeoSurface.ContactProperty.IndentationExponent.ParametricValue =
originalGeoSurface.ContactProperty.IndentationExponent.ParametricValue;
                                        }
                                }
                        }

                        if (originalGeoSurface.MaxStepSizeFactor.ParametricValue == null)
                        {
                                copyGeoSurface.MaxStepSizeFactor.Value =
originalGeoSurface.MaxStepSizeFactor.Value;
                        }
                        else
                        {
                                copyGeoSurface.MaxStepSizeFactor.ParametricValue =
originalGeoSurface.MaxStepSizeFactor.ParametricValue;
                        }
            }

            else if (contactType == "IContactGeoCurve")
            {

                        IContactGeoCurve copyGeoCurve = copyContact as IContactGeoCurve;
                        IContactGeoCurve originalGeoCurve = originalContact as IContactGeoCurve;

                        copyGeoCurve.ActionCurveSegmentOption.Segment =
originalGeoCurve.ActionCurveSegmentOption.Segment;
                        copyGeoCurve.ActionCurveSegmentOption.UseTotalSegment =
originalGeoCurve.ActionCurveSegmentOption.UseTotalSegment;

                        if (originalGeoCurve.ActionCurveSegmentOption.BoundingBufferLength.ParametricValue ==
null)
                        {
                                copyGeoCurve.ActionCurveSegmentOption.BoundingBufferLength.Value =
originalGeoCurve.ActionCurveSegmentOption.BoundingBufferLength.Value;

                        }
                        else
                        {
                                copyGeoCurve.ActionCurveSegmentOption.BoundingBufferLength.ParametricValue
= originalGeoCurve.ActionCurveSegmentOption.BoundingBufferLength.ParametricValue;

                        }

                        copyGeoCurve.ActionCurveSegmentOption.UseCubicCell =
originalGeoCurve.ActionCurveSegmentOption.UseCubicCell;
                        if (originalGeoCurve.ActionCurveSegmentOption.UseCubicCell)
                        {
                                copyGeoCurve.ActionCurveSegmentOption.CubicCell =
originalGeoCurve.ActionCurveSegmentOption.CubicCell;
                        }

                        copyGeoCurve.BaseCurveSegmentOption.Segment =
originalGeoCurve.BaseCurveSegmentOption.Segment;
                        copyGeoCurve.BaseCurveSegmentOption.UseTotalSegment =
originalGeoCurve.BaseCurveSegmentOption.UseTotalSegment;

                        if (originalGeoCurve.BaseCurveSegmentOption.BoundingBufferLength.ParametricValue ==
null)
                        {
                                copyGeoCurve.BaseCurveSegmentOption.BoundingBufferLength.Value =
originalGeoCurve.BaseCurveSegmentOption.BoundingBufferLength.Value;
```

```
                }
                else
                {
                        copyGeoCurve.BaseCurveSegmentOption.BoundingBufferLength.ParametricValue =
originalGeoCurve.BaseCurveSegmentOption.BoundingBufferLength.ParametricValue;

                }

                copyGeoCurve.BaseCurveSegmentOption.UseCubicCell =
originalGeoCurve.BaseCurveSegmentOption.UseCubicCell;
                if (originalGeoCurve.BaseCurveSegmentOption.UseCubicCell)
                {
                        copyGeoCurve.BaseCurveSegmentOption.CubicCell =
originalGeoCurve.BaseCurveSegmentOption.CubicCell;
                }

                //contact plane normal ÀÌ ¾ø´Ù.

                copyGeoCurve.ActionUpDirection = originalGeoCurve.ActionUpDirection;
                copyGeoCurve.ActionNodeContact = originalGeoCurve.ActionNodeContact;

                copyGeoCurve.BaseUpDirection = originalGeoCurve.BaseUpDirection;
                copyGeoCurve.BaseNodeContact = originalGeoCurve.BaseNodeContact;

                copyGeoCurve.UseCPM = originalGeoCurve.UseCPM;
                copyGeoCurve.SmoothNodeContact = originalGeoCurve.SmoothNodeContact;

                if (originalGeoCurve.ContactProperty.UseStiffnessSpline)
                {
                        copyGeoCurve.ContactProperty.StiffnessSpline =
originalGeoCurve.ContactProperty.StiffnessSpline;
                }
                else
                {
                        if (originalGeoCurve.ContactProperty.StiffnessCoefficient.ParametricValue == null)
                        {
                                copyGeoCurve.ContactProperty.StiffnessCoefficient.ParametricValue =
originalGeoCurve.ContactProperty.StiffnessCoefficient.ParametricValue;
                        }
                        else
                        {
                                copyGeoCurve.ContactProperty.StiffnessCoefficient.Value =
originalGeoCurve.ContactProperty.StiffnessCoefficient.Value;
                        }
                }

                if (originalGeoCurve.ContactProperty.UseDampingSpline)
                {
                        copyGeoCurve.ContactProperty.DampingSpline =
originalGeoCurve.ContactProperty.DampingSpline;
                }
                else
                {
                        if (originalGeoCurve.ContactProperty.DampingCoefficient.ParametricValue == null)
                        {
                                copyGeoCurve.ContactProperty.DampingCoefficient.ParametricValue =
originalGeoCurve.ContactProperty.DampingCoefficient.ParametricValue;
                        }
                        else
                        {
                                copyGeoCurve.ContactProperty.DampingCoefficient.Value =
originalGeoCurve.ContactProperty.DampingCoefficient.Value;
                        }
                }

                copyGeoCurve.ContactProperty.Friction.ContactFrictionType =
originalGeoCurve.ContactProperty.Friction.ContactFrictionType;
```

```
                    if (copyGeoCurve.ContactProperty.Friction.ContactFrictionType ==
ContactFrictionType.CoefficientSpline)
                    {
                              copyGeoCurve.ContactProperty.Friction.Spline =
originalGeoCurve.ContactProperty.Friction.Spline;
                    }
                    else if (copyGeoCurve.ContactProperty.Friction.ContactFrictionType ==
ContactFrictionType.ForceSpline)
                    {
                              copyGeoCurve.ContactProperty.Friction.Spline =
originalGeoCurve.ContactProperty.Friction.Spline;
                    }
                    else
                    {
                              if (originalGeoCurve.ContactProperty.Friction.Coefficient.ParametricValue == null)
                              {
                                        copyGeoCurve.ContactProperty.Friction.Coefficient.Value =
originalGeoCurve.ContactProperty.Friction.Coefficient.Value;
                              }
                              else
                              {
                                        copyGeoCurve.ContactProperty.Friction.Coefficient.ParametricValue =
originalGeoCurve.ContactProperty.Friction.Coefficient.ParametricValue;
                              }

                              if
(originalGeoCurve.ContactProperty.Friction.StaticThresholdVelocity.ParametricValue == null)
                              {
                                        copyGeoCurve.ContactProperty.Friction.StaticThresholdVelocity.Value =
originalGeoCurve.ContactProperty.Friction.StaticThresholdVelocity.Value;
                              }
                              else
                              {

          copyGeoCurve.ContactProperty.Friction.StaticThresholdVelocity.ParametricValue =
originalGeoCurve.ContactProperty.Friction.StaticThresholdVelocity.ParametricValue;

                              }

                              if
(originalGeoCurve.ContactProperty.Friction.DynamicThresholdVelocity.ParametricValue == null)
                              {
                                        copyGeoCurve.ContactProperty.Friction.DynamicThresholdVelocity.Value
= originalGeoCurve.ContactProperty.Friction.DynamicThresholdVelocity.Value;
                              }
                              else
                              {

          copyGeoCurve.ContactProperty.Friction.DynamicThresholdVelocity.ParametricValue =
originalGeoCurve.ContactProperty.Friction.DynamicThresholdVelocity.ParametricValue;

                              }
                              if (originalGeoCurve.ContactProperty.Friction.StaticCoefficient.ParametricValue ==
null)
                              {
                                        copyGeoCurve.ContactProperty.Friction.StaticCoefficient.Value =
originalGeoCurve.ContactProperty.Friction.StaticCoefficient.Value;
                              }
                              else
                              {
                                        copyGeoCurve.ContactProperty.Friction.StaticCoefficient.ParametricValue
= originalGeoCurve.ContactProperty.Friction.StaticCoefficient.ParametricValue;
                              }
                              if (originalGeoCurve.ContactProperty.Friction.MaximumForce.ParametricValue ==
null)
                              {
                                        copyGeoCurve.ContactProperty.Friction.MaximumForce.Value =
originalGeoCurve.ContactProperty.Friction.MaximumForce.Value;
```

```
                }
                else
                {
                        copyGeoCurve.ContactProperty.Friction.MaximumForce.ParametricValue
= originalGeoCurve.ContactProperty.Friction.MaximumForce.ParametricValue;
                }
                copyGeoCurve.ContactProperty.Friction.UseMaximumForce =
originalGeoCurve.ContactProperty.Friction.UseMaximumForce;


        }

        if (originalGeoCurve.ContactProperty.StiffnessExponent.ParametricValue == null)
        {
                copyGeoCurve.ContactProperty.StiffnessExponent.Value =
originalGeoCurve.ContactProperty.StiffnessExponent.Value;
        }
        else
        {
                copyGeoCurve.ContactProperty.StiffnessExponent.ParametricValue =
originalGeoCurve.ContactProperty.StiffnessExponent.ParametricValue;
        }


        if (originalGeoCurve.ContactPropertyAdditional.ReboundDampingFactor.ParametricValue ==
null)
        {
                copyGeoCurve.ContactPropertyAdditional.ReboundDampingFactor.Value =
originalGeoCurve.ContactPropertyAdditional.ReboundDampingFactor.Value;
        }
        else
        {
                copyGeoCurve.ContactPropertyAdditional.ReboundDampingFactor.ParametricValue
= originalGeoCurve.ContactPropertyAdditional.ReboundDampingFactor.ParametricValue;
        }

        if (originalGeoCurve.ContactPropertyAdditional.GlobalMaxPenetration.ParametricValue ==
null)
        {
                copyGeoCurve.ContactPropertyAdditional.GlobalMaxPenetration.Value =
originalGeoCurve.ContactPropertyAdditional.GlobalMaxPenetration.Value;
        }
        else
        {
                copyGeoCurve.ContactPropertyAdditional.GlobalMaxPenetration.ParametricValue =
originalGeoCurve.ContactPropertyAdditional.GlobalMaxPenetration.ParametricValue;
        }


        if (copyGeoCurve.ContactPropertyAdditional.ContactForceType ==
ContactForceType.BoundaryPenetration)
        {
                if
(originalGeoCurve.ContactPropertyAdditional.BoundaryPenetration.ParametricValue == null)
                {
                        copyGeoCurve.ContactPropertyAdditional.BoundaryPenetration.Value =
originalGeoCurve.ContactPropertyAdditional.BoundaryPenetration.Value;
                }
                else
                {

            copyGeoCurve.ContactPropertyAdditional.BoundaryPenetration.ParametricValue =
originalGeoCurve.ContactPropertyAdditional.BoundaryPenetration.ParametricValue;
                }
        }
        else
        {
                if (originalGeoCurve.ContactProperty.UseDampingExponent)
```

```
                                {
                                        if (originalGeoCurve.ContactProperty.DampingExponent.ParametricValue
== null)
                                        {
                                                copyGeoCurve.ContactProperty.DampingExponent.Value =
originalGeoCurve.ContactProperty.DampingExponent.Value;
                                        }
                                        else
                                        {

            copyGeoCurve.ContactProperty.DampingExponent.ParametricValue =
originalGeoCurve.ContactProperty.DampingExponent.ParametricValue;
                                        }
                                }
                                if (originalGeoCurve.ContactProperty.UseIndentationExponent)
                                {
                                        if
(originalGeoCurve.ContactProperty.IndentationExponent.ParametricValue == null)
                                        {
                                                copyGeoCurve.ContactProperty.IndentationExponent.Value =
originalGeoCurve.ContactProperty.IndentationExponent.Value;
                                        }
                                        else
                                        {

            copyGeoCurve.ContactProperty.IndentationExponent.ParametricValue =
originalGeoCurve.ContactProperty.IndentationExponent.ParametricValue;
                                        }
                                }
                        }

                        if (originalGeoCurve.MaxStepSizeFactor.ParametricValue == null)
                        {
                                copyGeoCurve.MaxStepSizeFactor.Value =
originalGeoCurve.MaxStepSizeFactor.Value;
                        }
                        else
                        {
                                copyGeoCurve.MaxStepSizeFactor.ParametricValue =
originalGeoCurve.MaxStepSizeFactor.ParametricValue;
                        }

                }
}
```

Chapter 5

Register DLL

목적

이 과정에서는 **ConvertCloneBody** 와 **CreateContact** 다이얼로그를 **RecurDyn** 리본 메뉴에 등록하기 위해서 **Register DLL** 을 사용하는 법을 배울 것입니다.

예상 소요 시간

5 분

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 자동생성 된 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 **RunChangeBody()**, **RunCreateContact()** 함수를 작성합니다.
   - MainWindow 의 값을 **ChangeGeneralBody** 클래스와 **CreateCotact** 클래스에 전달해줍니다.

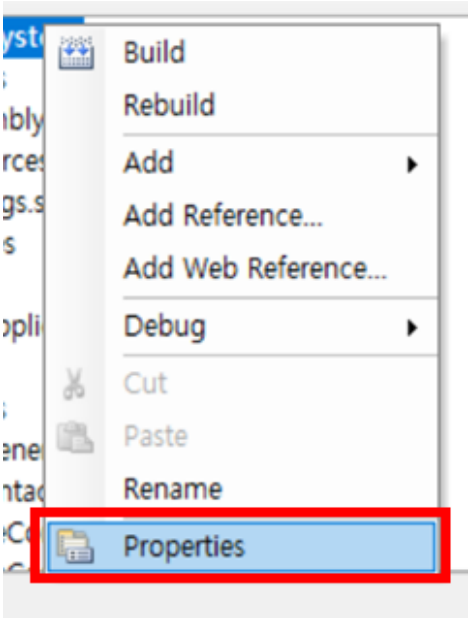
Note: MainWindow 를 전달해야 Winform 에서 ProcessNet 매서드를 사용할 수 있습니다.

```
public void RunChangeBody()
{
	ChangeGeneralBody Dialog = new ChangeGeneralBody(modelDocument);
	Dialog.Show(MainWindow);
}

public void RunCreateContact()
{
	CreateContact Dialog = new CreateContact(modelDocument);
	Dialog.Show(MainWindow);
}
```

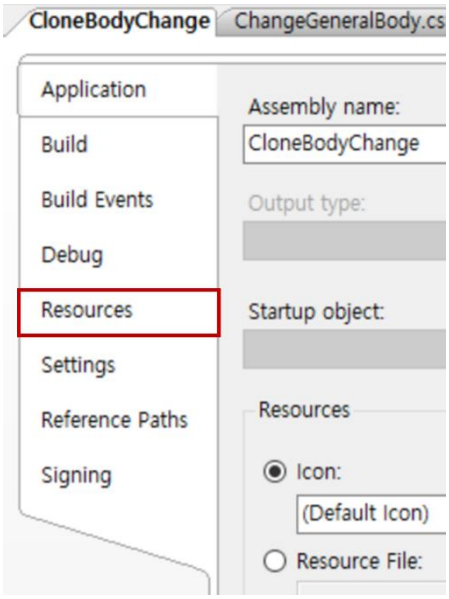
4. **File** 메뉴에서 **Save ThisApplication.cs** 를 클릭해서 **Save** 를 수행합니다.

Icon 등록하기

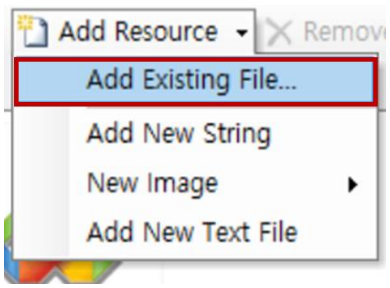
ChangeGeneralBody 와 **CreateContact** 다이얼로그를 리본 메뉴에 등록할 때 리본 메뉴에 표시될 Icon 을 추가할 것입니다.



1. **ProcessNet IDE** 의 **Project Explorer** 창에서 **SimpleBeltSystem** 을 마우스로 클릭한후 마우스의 오른쪽을 클릭합니다.
2. **Properties** 버튼을 클릭합니다.
3. **SimpleBeltSystem Property** 창이 생기면 **Resource** 버튼을 클릭합니다.



4. **Resources** 창이 열리면 **Add Resource-Add Existing File** 버튼을 클릭합니다.



5. **ConvertGeneral.bmp** 를 등록합니다. (파일 경로: **<InstallDir>/Help/Tutorial/ProcessNet/VSTA/SimpleBeltSystem**).

6. 4-5 번 과정을 다시 거쳐서 **CreateContact.bmp** 를 등록합니다.

Register DLL 을 위한 함수 생성하기

1. **RegisterFunction()** 함수를 생성합니다. 함수이름이 **RegisterFunction** 이 아닌 경우 **Register DLL** 이 동작하지 않으니 주의해야 합니다.

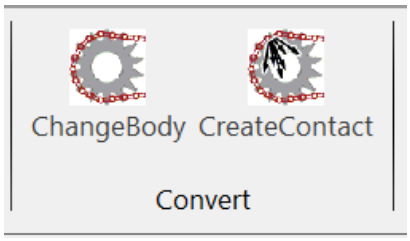
```
public void RegisterFunction()
{
            IRibbonManager ribbonManager = application.RibbonManager;
             IRibbonTab ribbonTab = ribbonManager.FindRibbonTab("Customize");
             IRibbonGroup ribbonGroup = ribbonTab.AddRibbonGroup("Convert");

             //ID for user created processNet function must be between 8000 - 8999 .
             IMenuControl menuControl01 =
ribbonGroup.AddMenuControl(MenuControlType.MenuControlType_Button, 8011);

             // Set example control information.
             IntPtr iIcon01 = SimpleBeltSystem.Properties.Resources.ConvertGeneral.GetHicon();
             menuControl01.SetIcon(iIcon01);
             menuControl01.UseBigIcon = true;

             menuControl01.Caption = "ChangeBody";
             menuControl01.Tooltip = "ChangeBody";
             //menuControl.Description = "MyDescription";
             menuControl01.UseProcessNetFunction = true; //if ID is not between 8000 - 8999, this will set
ID to 8000

             // Get current ProcessNet dll fullpath. 'ProcessNetDllPath' must be absolute path.
             string assemblyname =
System.Reflection.Assembly.GetExecutingAssembly().GetName().Name;
             string ProcessNetDllPath = AppDomain.CurrentDomain.BaseDirectory + @"\" + assemblyname
+ ".dll";

             menuControl01.ProcessNetDllPath = ProcessNetDllPath;
             menuControl01.ProcessNetFunctionName = "RunChangeBody";

             //ID for user created processNet function must be between 8000 - 8999 .
             IMenuControl menuControl02 =
ribbonGroup.AddMenuControl(MenuControlType.MenuControlType_Button, 8012);

             // Set example control information.
             IntPtr iIcon02 = SimpleBeltSystem.Properties.Resources.CreateContact.GetHicon();
             menuControl02.SetIcon(iIcon02);
             menuControl02.UseBigIcon = true;

             menuControl02.Caption = "CreateContact";
             menuControl02.Tooltip = "CreateContact";
             //menuControl.Description = "MyDescription";
             menuControl02.UseProcessNetFunction = true; //if ID is not between 8000 - 8999, this will set
ID to 8000

             menuControl02.ProcessNetDllPath = ProcessNetDllPath;
             menuControl02.ProcessNetFunctionName = "RunCreateContact";
}
```

생성한 애플리케이션의 테스트

생성한 애플리케이션이 정상적으로 작동하는지 테스트합니다.

애플리케이션 실행하기

1. **Build** 메뉴에서 **Build SimpleBeltSystem** 을 선택합니다. **IDE** 창 하단의 **Error List** 창에서 에러나 경고가 없는지를 확인합니다. 에러나 경고가 있다면, 목록을 확인하여 수정합니다.
2. **RecurDyn** 프로그램의 **Customize** 탭의 **ProcessNet(VSTA)**그룹에서 **Run** 을 선택합니다.
3. **Run ProcessNet** 다이얼로그 하단의 트리컨트롤에서 **SimpleBeltSystem** 하위의 **RegisterFunction** 을 클릭합니다.
4. **Run ProcessNet** 다이얼로그에 있는 **Run** 버튼을 클릭합니다.
5. 리본 메뉴에 Icon 이 등록됩니다.



Chapter 6

모델 해석

목적

이번 장에서는 그동안 작성한 PorcessNet 코드를 사용해서 Clone Link Body 를 General Body 로 변환하고 General Contact 도 정의해볼 것입니다.

예상 소요 시간

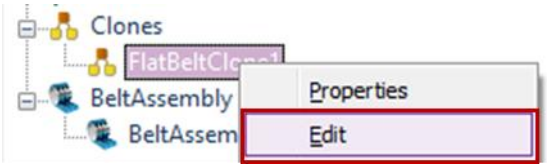
5 분

Clone Link Body 를 General Body 로 Convert 하기

Clone Link Body 에 FaceSurface 생성하기

Contact 을 생성할 위치의 Face 에 FaceSurface 를 생성합니다. Face 에도 Contact 을 생성할 수 있기 때문에 FaceSurface 는 1 개만 생성합니다.

1. **Belt Subsystem Edit Mode** 로 들어갑니다.
2. 다음의 방법을 이용하여 **FlatBeltClone1** Body 의 Edit 모드로 진입합니다.

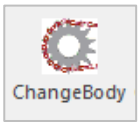


   - Database 에서 **FlatBeltClone1** 를 마우스 오른쪽으로 클릭합니다.
   - Pup-up 메뉴가 나타나면 **Edit** 를 클릭합니다.

3. **Geometry** 탭의 **Surface** 그룹에서 **Face** 를 클릭합니다.
4. Creation Method 를 **Face** 로 설정합니다.
5. Input bar 에 **FlatBelt1.Face2** 를 입력합니다.
6. Working Window 에서 오른쪽 마우스 버튼을 클릭한 후 **Exit** 를 클릭하여 Body 의 Edit 모드를 나옵니다.

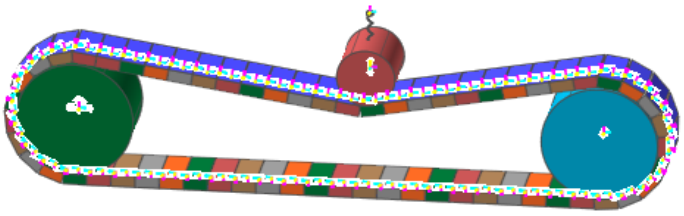
Clone Link Body 변환하기

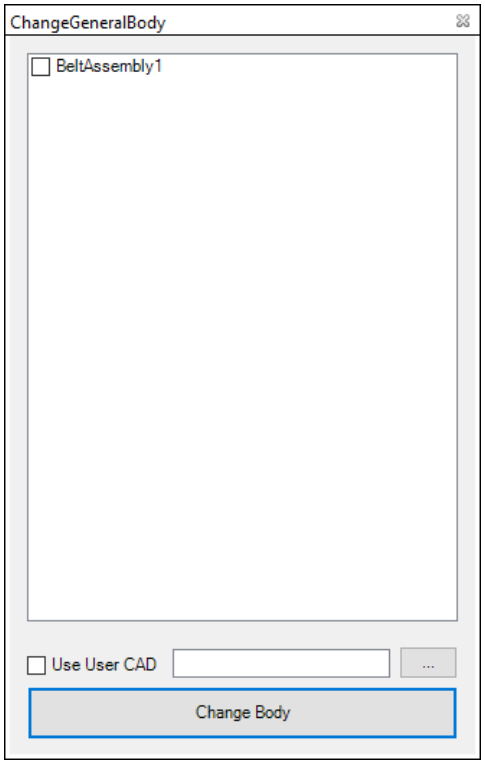
그동안 작성한 ProcessNet 함수를 사용해서 Open 한 Belt 모델의 Clone Link 를 General 로 변환할 것입니다.

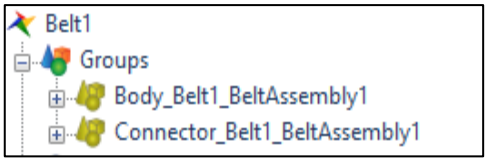


1. **Customize** 탭의 **Convert** 그룹에서 **ChangeBody** 를 클릭합니다.

2. **ChangeGeneralBody** 다이얼로그 창이 열리면 **Belt Assembly1** 을 선택하고 **ChangeBody** 버튼을 클릭합니다.
3. 아래 그림처럼 Clone Link Body 가 General Body 로 변환된 것을 확인할 수 있습니다.
4. Database 창을 보면 **Body_Belt1_BeltAssembly1**, **Connector_Belt1_BeltAssembly1** 이 생성된 것을 확인할수 있고 각 그룹을 선택하면 생성된 것을 확인할수 있습니다.



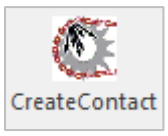




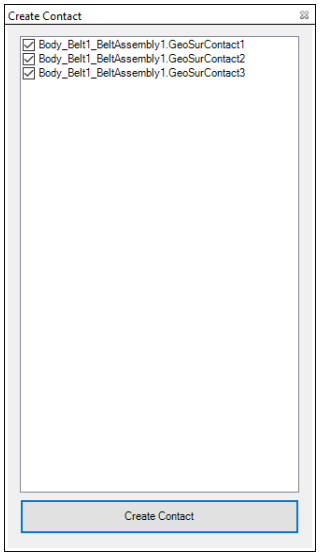
Contact 생성하기

1. **Professional** 탭의 **Contact** 그룹에서 **GeoSur** 를 클릭합니다.
2. Creation Method 를 **Surface(PatchSet), Surface(PatchSet)**으로 설정합니다.
3. 아래 표에 있는 데로 입력해서 Geo Surface Contact 을 생성합니다.

Contact	Action	Base
GeoSurContact1	FlatBelt50.FaceSurface1	Roller3.Roller1.Face3
GeoSurContact2	FlatBelt16.FlatBelt1.Face4	Roller1.Roller1.Face3
GeoSurContact3	FlatBelt17.FlatBelt1.Face4	Roller2.Roller1.Face3



4. **Customize** 탭의 **Convert** 그룹에서 CreateContact 를 클릭합니다.
5. 생성한 Contact 이 Check 되어 있는지 확인하고 Create Contact 버튼을 클릭합니다.
6. Contact 이 생성된 것을 확인합니다.



Dynamic/Kinematic 해석의 실행

생성된 모델의 **Dynamic/Kinematic** 해석을 실행합니다.

1. **Anaylsis** 탭의 **Simulation Type** 그룹에서 **Dyn/Kin** 을 클릭하여 **Dynamic/Kinematic** 대화상자를 엽니다.
2. 시뮬레이션의 종료시간과 Step 의 수를 정합니다.
   - **End Time**: 1
   - **Step**: 100
   - **Plot Multiplier Step Factor**: 1
3. **Simulate** 를 클릭하면 해석이 진행됩니다.
4. 기존 **Belt Toolkit model** 과 결과를 비교해 봅니다.

Thanks for participating in this tutorial