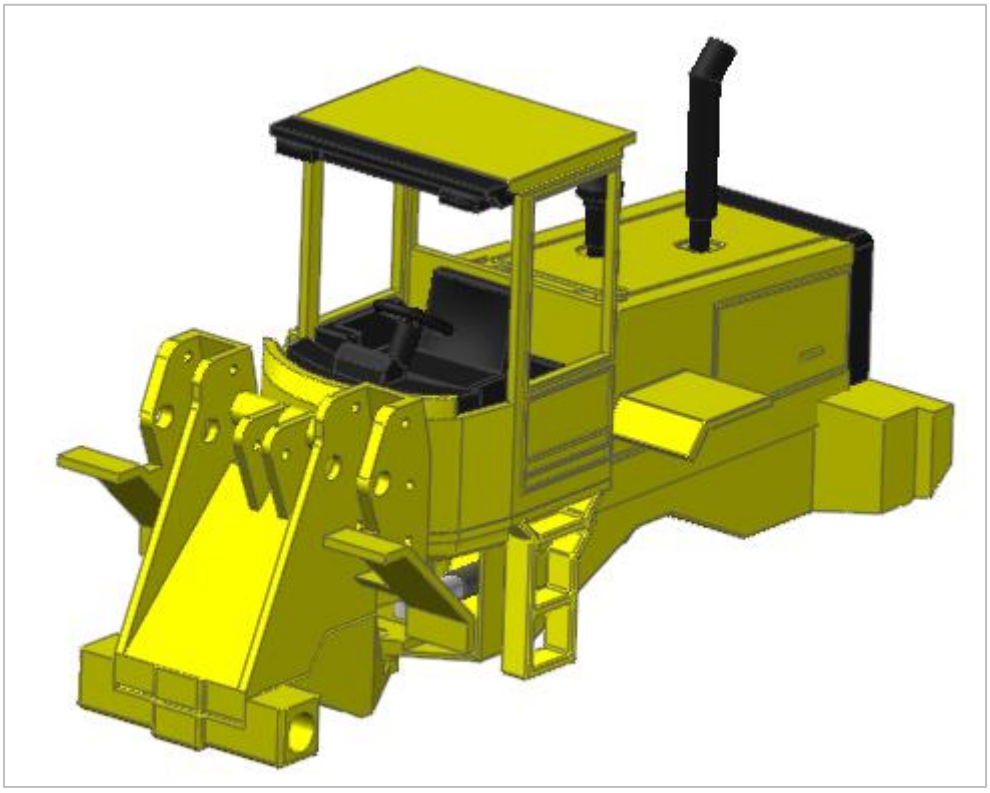
RECURDYN

4WD Loader Tutorial (ProcessNet VSTA)



FunctionBay

Edition Note

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

목차

Chapter 1

개요

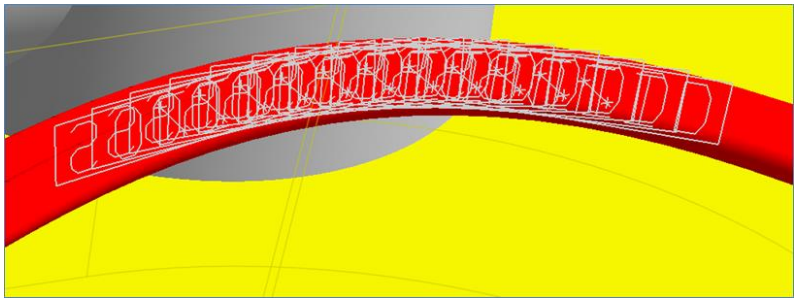
목적

ProcessNet 은 모델링, 해석, 플로팅하는 과정을 자동적으로 사용을 할 수 있게 합니다. 이번 튜토리얼에서는 사용자가 **Process Net** 을 사용해서 **RecurDyn** 의 3 가지 과정를 자동화 하는것을 배울것입니다. 3 가지 과정의 자동화 세부내용은 다음과 같습니다.

- 일련의 Contact 생성의 자동화
- Contact 의 사용자 제어가 가능한 사용자 정의 다이얼로그 박스 생성
- Contact 의 결과값을 자동으로 확인하여 제로 이외의 결과값을 가진 Contact 만을 Plotting

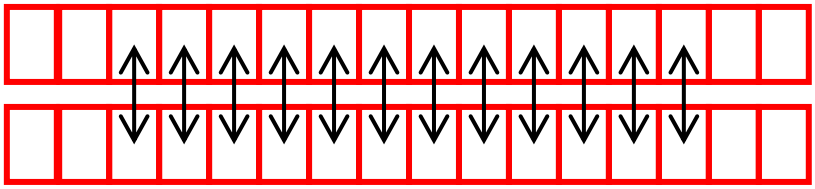
이 튜토리얼에서는 사륜 구동 로더에 있는 한 쌍의 호스를 시뮬레이션 하게 될 것입니다. 호스는 차량의 앞쪽에 있는 로더 연결 장치의 실린더에 의해 차량의 뒤쪽의 유압 펌프와 연결되며, 로더는 기계관절에 의해 조정됩니다. 호스는 관절이 있는 프레임의 두 부분과 연결되어 구부러지며 서로 Contact 하게 될 것입니다.

이 과정에서는 어떠한 결합 또는 마모 문제를 예측하는 Contact 를 이해하는 것은 중요합니다. 사용자는 컨택이 발생하는 곳과 그 크기의 규모를 알고싶어합니다. 하지만 실제 호스의 움직임과 Contact 의 위치를 예상하는 것은 어렵기 때문에, 따라서 사용자는 많은 세그먼트로 잘려진 호스(아래 그림 참조)사이에 컨텍을 정의할 필요가 있습니다. 그것은 수동으로 수행하는 지루한 작업이 될 것이며 사용자는은 몇 가지 실수를 할수 있습니다. 그러나, ProcessNet 을 이용한 자동화 과정은 실수를 줄여주고 더욱 효율적으로 모델링 작업을 할 수 있게 만들어 줍니다.



접근

모든 메카니컬 요소와, 호스와 모션 컨트롤이 포함이 되어있는 사륜 구동 로더가 주어집니다. 호스는 2 개의 포인트와 50 개의 세그먼트이 사용된 RFlex Beam Gruop 으로 정의되 있으며, 호스의 앞부분의 두 점에 모션이 입력된 후 시뮬레이션 0.18 초까지 뒤쪽으로 이동하면서 호스는 적절하게 늘어지게 됩니다. 입력된 다른 모션은 조정장치의 실린더를 늘여서 프레임의 거동을 표현합니다.
초기의 Contact 은 위의 그림처럼 두 개의 호스에 해당하는 세그먼트 사이에서 정의됩니다.



독자

이 튜토리얼은 RecuDyn 을 이전에 배워 본 경험이 있고, Geometry, Joint, Force 를 생성할 수 있는 중급 수준의 사용자를 대상으로 하며, 모든 작업은 자세히 설명되어 있습니다.

필요 요건

- 3D Crank-Slider 와 Engine with Propeller 튜토리얼을 익혔거나 이에 상응하는 작업을 해 본 것을 전제로 하며, 물리학에 대한 기본 지식이 있어야 합니다.
- RFlex Beam Group 을 사용하여 호스를 정의해야 하기 때문에 이 모델을 가지고 작업을 하기 위해서 RFlex 모듈의 라이선스가 필요합니다.

과정

이 튜토리얼은 다음의 과정으로 구성되며, 각 과정을 완성하기까지 걸리는 시간은 다음의 표와 같습니다.

과정	시간(분)
모델 열기 및 ProcessNet 시작	10

자동 Contact 정의	35
다이얼로그 박스와 출력 메시지 추가	20
자동 Plot 생성	20
총 합	**85**

예상 소요 시간

약 85 분

Chapter 2

모델 열기 및 ProcessNet 시작

목적

ProcessNet 을 사용하기 위해 모델을 어떻게 Import 하는 지와 ProcessNet 을 어떻게 시작하는지를 배워 봅시다.

예상 소요 시간

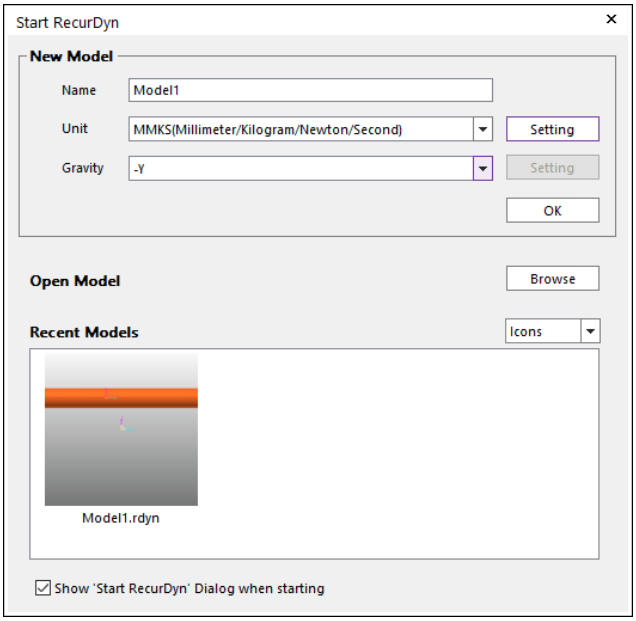
10 분

RecurDyn 시작하기

RecurDyn 을 시작한 후 초기 모델 열기**:**

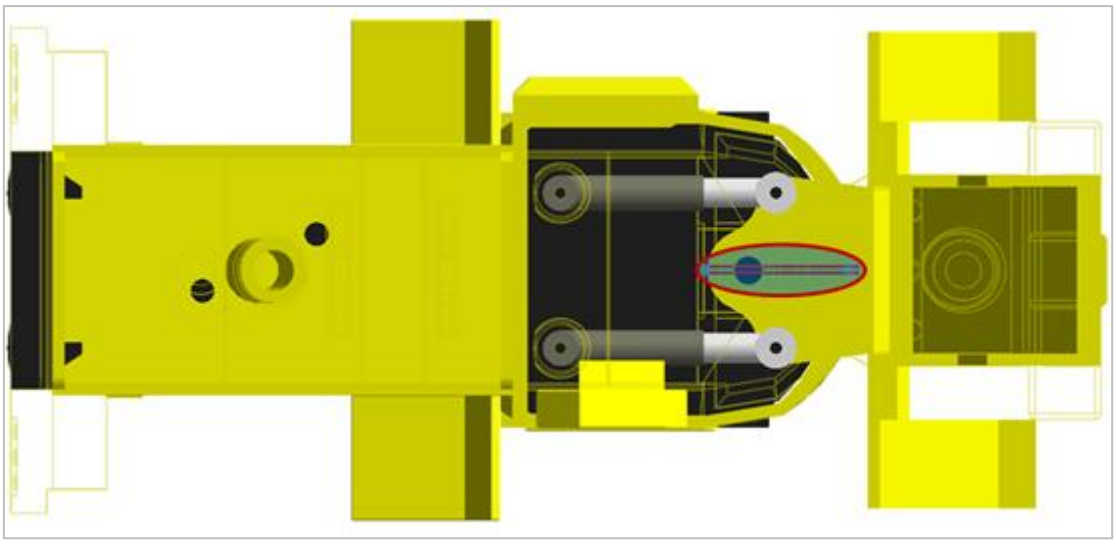
1. **바탕 화면**에서, **RecurDyn** 아이콘을 더블 클릭합니다.
2. **Start RecurDyn** 다이얼로그 박스가 나타나면, 새 모델이 아닌 기존 모델을

사용해야 하므로, 닫아 줍니다.



3. **Quick Access Toolbar** 에서, **Open** 을 클릭합니다.
4. **4WD_Loader_Start.rdyn** 파일을 선택합니다. (파일 경로: **<InstallDir>/Help/Tutorial/ProcessNet/VSTA/4WDLoader**).
5. Open 을 클릭합니다.

그리하면, 모델은 다음과 같이 보일 것입니다.



빨간 선은 이 튜토리얼에서 다루게 될 유압 호스입니다.

Tip: 호스의 모습이 차량의 하단에 가리기 때문 (**Render Each Object**) 보기 모드를 선택해야만 위의 그림과 같이 모습이 보여 집니다. (만약 호스의 모습을 보기가 어렵다면, 현재 **Shaded** 보기 모드인지 확인하여 **Render Each Object** 보기 모드로 모드를 변경해야 합니다.)

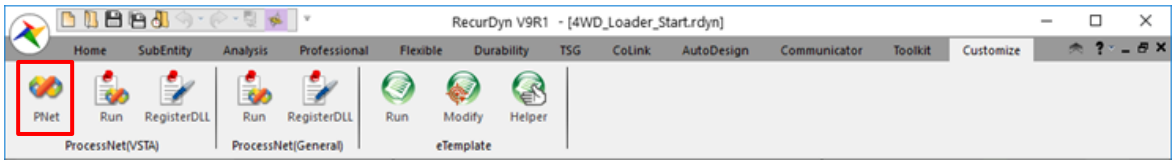
모델 저장하기**:**

1. **File** 메뉴에서, **Save As** 를 클릭합니다.
2. Tutorial 디렉토리에서는 시뮬레이션을 할 수 없기 때문에 다른 디렉토리에 다시 저장합니다.

ProcessNet 시작하기

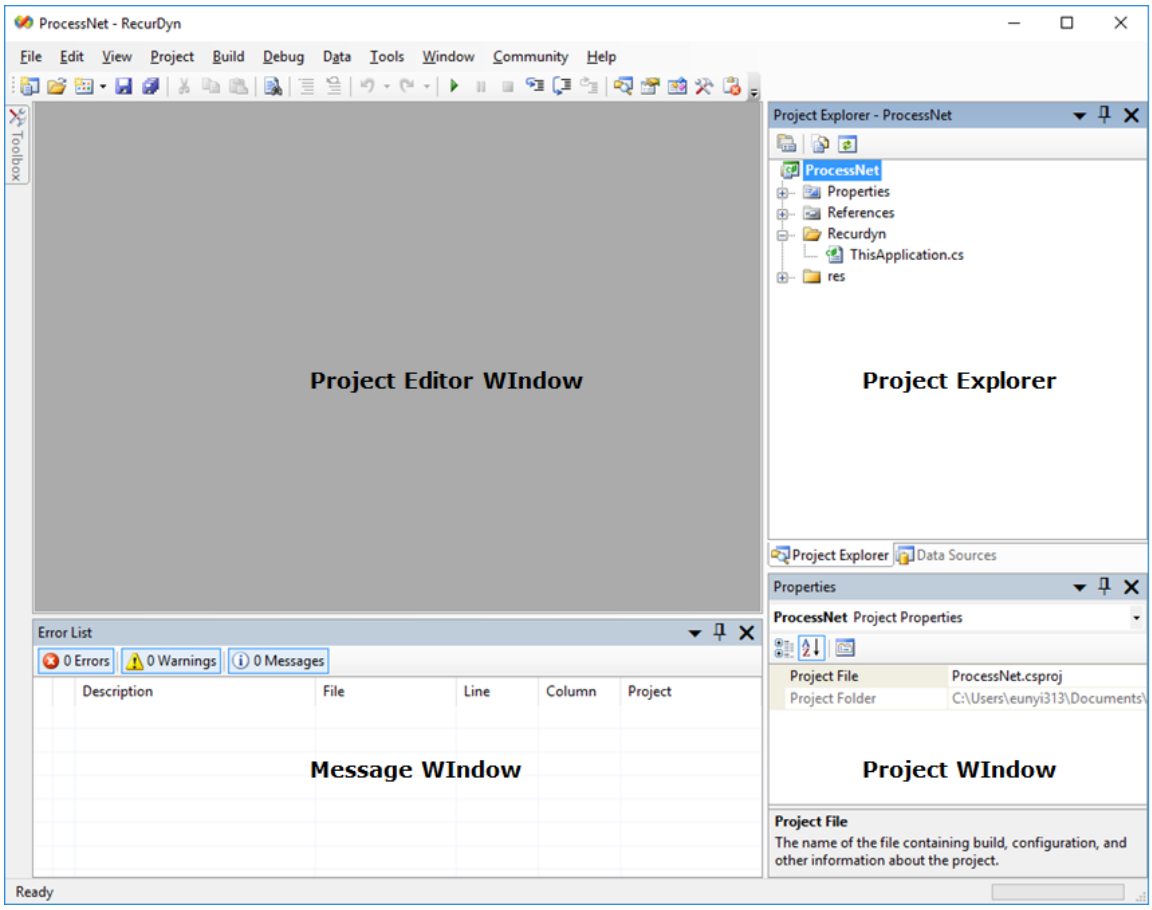
ProcessNet 시작한 후 초기화 하기

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



RecurDyn 을 설치한 후 **ProcessNet IDE** 를 처음으로 실행할 경우, 초기화를 완료하기까지 몇 분의 시간이 걸릴 수가 있습니다. 그 이후 실행부터는 10 초에서 15 초 정도 걸립니다.

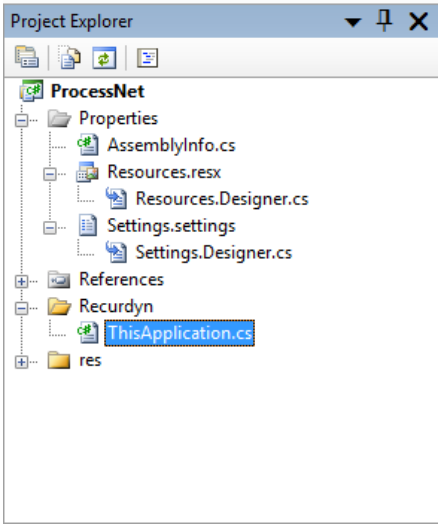
초기화가 완료되면 **ProcessNet – RecurDyn** 이라는 이름의 새로운 창이 다음과 같이 나타납니다.



ProcessNet – RecurDyn 은 4 가지 구역으로 구성되어 있습니다.

- **Project Editor** 창– 코드와 Design Interface Object 를 입력하여 편집할 수 있습니다.
- **Project Explorer** 창 – 프로젝트의 구성요소 또는 그에 대한 파일들을 얻을 수 있습니다.
- **Properties** 창 – Project Editor 창 또는 Project Explorer 에서 선택된 object 의 속성을 보거나 편집할 수 있습니다.
- **Message** 창 – 코드에 있는 에러 등과 같은 생성한 코드 및 실행한 코드에 대한 정보를 볼 수 있습니다.

2. **Project Explorer** 는 오른쪽 그림처럼 **ThisApplication.cs** 와 함께 **Recurdyn** 폴더에 포함되어 있어야 합니다.



   - 오른쪽 그림과 같이 **Project Explorer** 창이 보이면 아래의 Step 3 를 진행하시면 됩니다.
   - **Project Explorer** 창의 내용이 그림과 다르다면, 그 Project directory 는 이전에 사용된 설정이므로 Project 를 기본 설정으로 다시 생성해야 합니다.

Note: 기본 **Project directory** 의 위치는 **RecurDyn installation directory** 의 **ProcessNetProject directory** 에 있으며, 기본 **Project directory** 의 이름은 **ProcessNet** 입니다.

---

a. **ProcessNet** directory 를 복사하여 컴퓨터의 원하는 위치에 붙여 넣은 후 원하는 이름으로 그 directory 의 이름을 지정합니다.

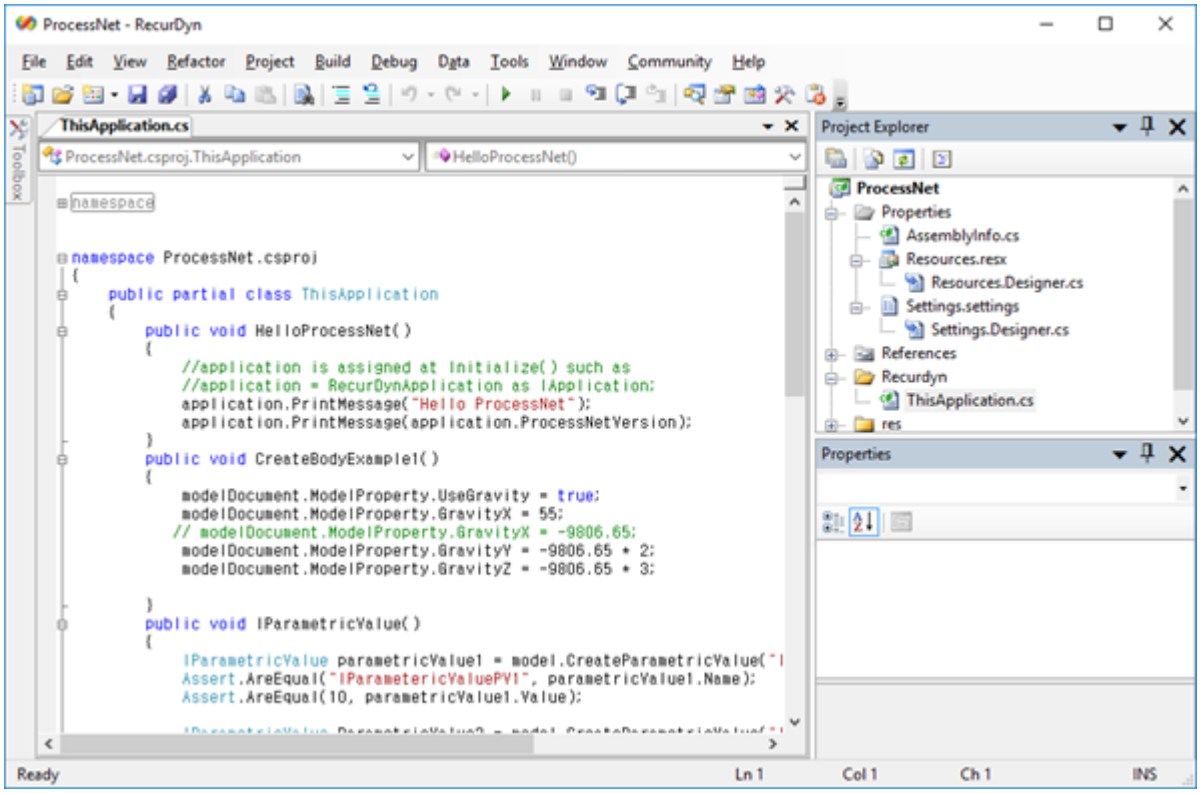
b. ProcessNet IDE 어플리케이션으로 다시 돌아옵니다.

c. **File** 메뉴에서 **Close Project** 를 선택합니다.

d. Open Project 를 선택한 후에 Step a 에서 생성했던 새 directory 에서 ProcessNet.csproj 파일을 선택합니다.

e. Project Explorer 창에서 위의 그림에 없는 item 은 지웁니다.

3. **Project Explorer** 창에서 **ThisApplication.cs** 을 더블 클릭하여 해당 파일을 엽니다.



파일의 내용이 Project 편집기 창에 보여집니다.

이제 ProcessNet 어플리케이션을 개발하기 위한 준비가 완료되었습니다.

4. **File** 메뉴의 Save All 을 선택합니다. **4WD_Loader** 로 이름을 변경하고 저장합니다.

Chapter

3

자동 **Contact** 정의

목적

두 단계를 거쳐서, 호스의 세그먼트 사이에서 Contact 를 생성하는 ProcessNet 어플리케이션을 생성하게 될 것입니다.

- RecurDyn 의 엔티티를 생성하기 위한 정보 획득
- RecurDyn 에 있는 ProcessNet Help 이용
- 어플리케이션을 개발하기 위한 IDE 의 작업

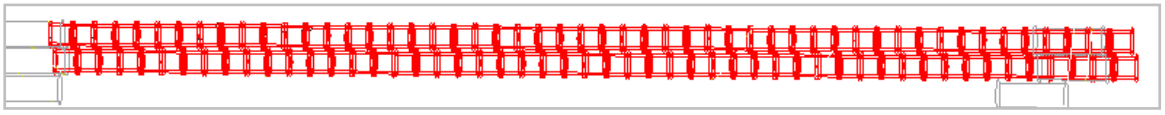
시뮬레이션을 실행하여 결과를 본 후, 어플리케이션을 업데이트 할 것이며, 향상된 결과를 관찰하기 위해서 시뮬레이션을 다시 실행하게 될 것입니다.

예상 소요 시간

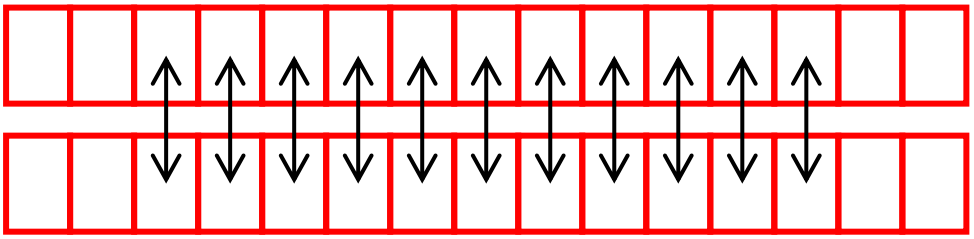
35 분

생성해야 할 **Contact** 에 대한 이해

1 장에서는 아래의 그림과 같이 50 개의 세그먼트를 가지고 있는 두 개의 호스에 대해서 설명하였습니다. 호스는 로더와 프레임을 호스 양 끝 지점으로 연결되어 있으며, 로더의 회전 중심축이 호스 양 끝지점 사이에서 적용됩니다. 한 호스에서 관절이 더 잘 굽혀지는 현상이 나타나고 다른 호스는 상대적으로 덜 굽혀지는 현상이 나타나게 됩니다. 두 호스의 굴곡이 다름에 따라 Contact 현상이 호스 중간부분에서 발생하므로 호스의 중간 위치에서 Contact 을 생성합니다.

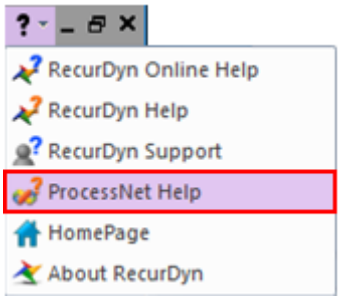


ProcessNet 어플리케이션을 생성하기 위해 아래의 그림처럼 호스의 중간 위치에서 11 개의 contact 을 생성하게 될 것입니다.



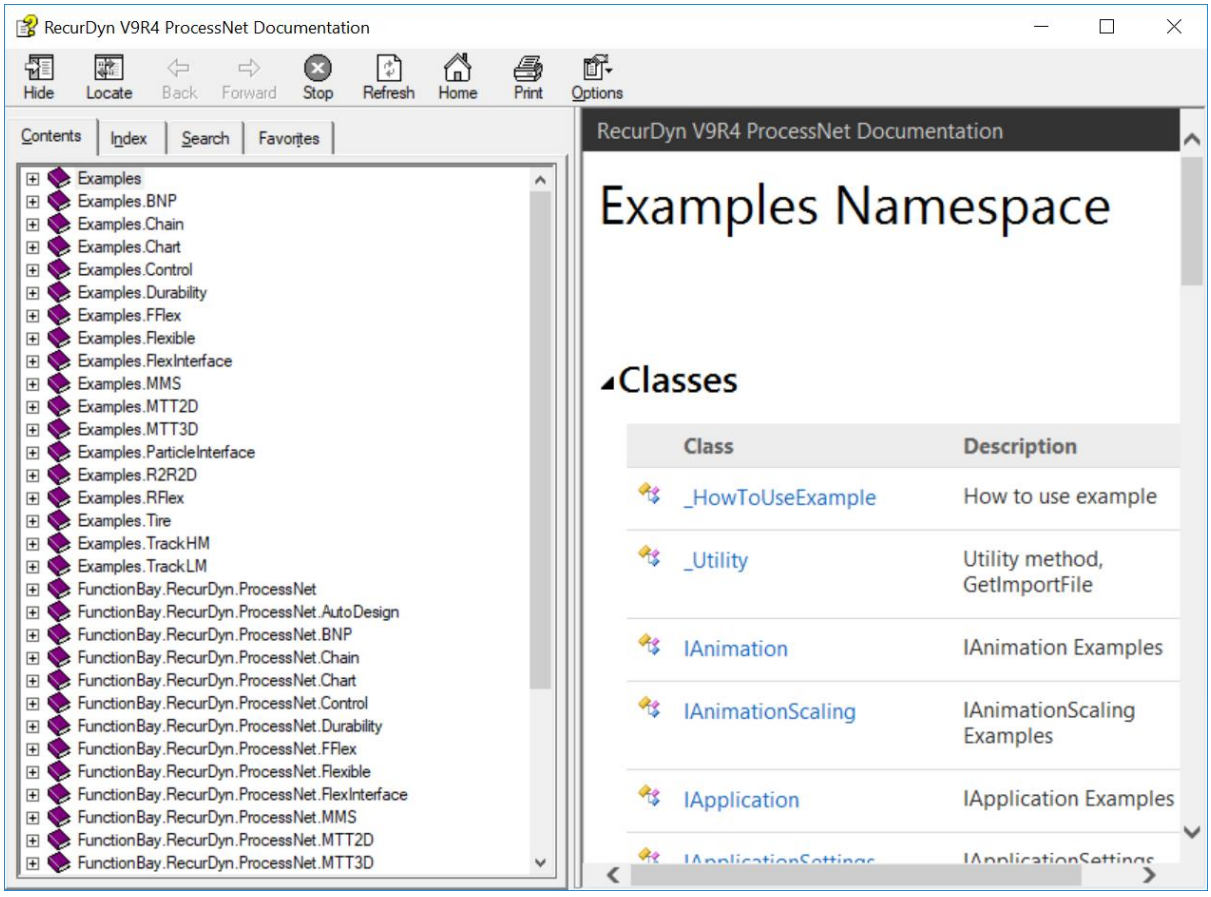
ProcessNet 헬프 예제 활용

이 부분에서는 ProcessNet 어플리케이션을 생성하기 위해 도움을 줄 수 있는 정보를 배우게 될 것입니다.



ProcessNet 헬프 창 열기**:**

1. 메인 탭의 끝에 있는 **Help** 를 클릭합니다.
2. **ProcessNet Help** 를 클릭하면 다음과 같이 Help 창이 나타납니다.



이 창에서 다양한 코드 예제를 찾아볼 수 있습니다. 특성화된 Toolkit 을 사용하지 않고 기본적인 RecurDyn 매크로를 개발하려면, 다음의 namespace 가 유용할 것입니다.

- **Examples Namespace** – 기본적인 RecurDyn 엔티티 및 파라미터 생성 등 기초적인 ProcessNet 어플리케이션의 실제적인 예제들을 다루고 있습니다.
- **Examples Plot Namespace –** RecurDyn Plot 생성에 대한 예제들을 다루고 있습니다.

3. 예제를 보기 위해 다음의 순서대로 펼쳐보겠습니다.
   Examples Namespace→IBody Class→IBodyExamples Method (**Tip**: 예제를 보기 위해 오른쪽에 있는 동일한 이름의 링크를 클릭해도 됩니다.)

4. 이 특정한 방법은 sphere, box, cylinder 와 같은 기초적인 body 를 어떻게 생성하고 그에 해당하는 파라미터들을 어떻게 수정하는지에 대해서 다루고 있습니다.

Tip: Macro 를 개발할 때 예제 코드의 화면 오른쪽 위에 있는 버튼 Copy 을 마우스 버튼으로 클릭하면 코드가 전체 복사가 됩니다.

---

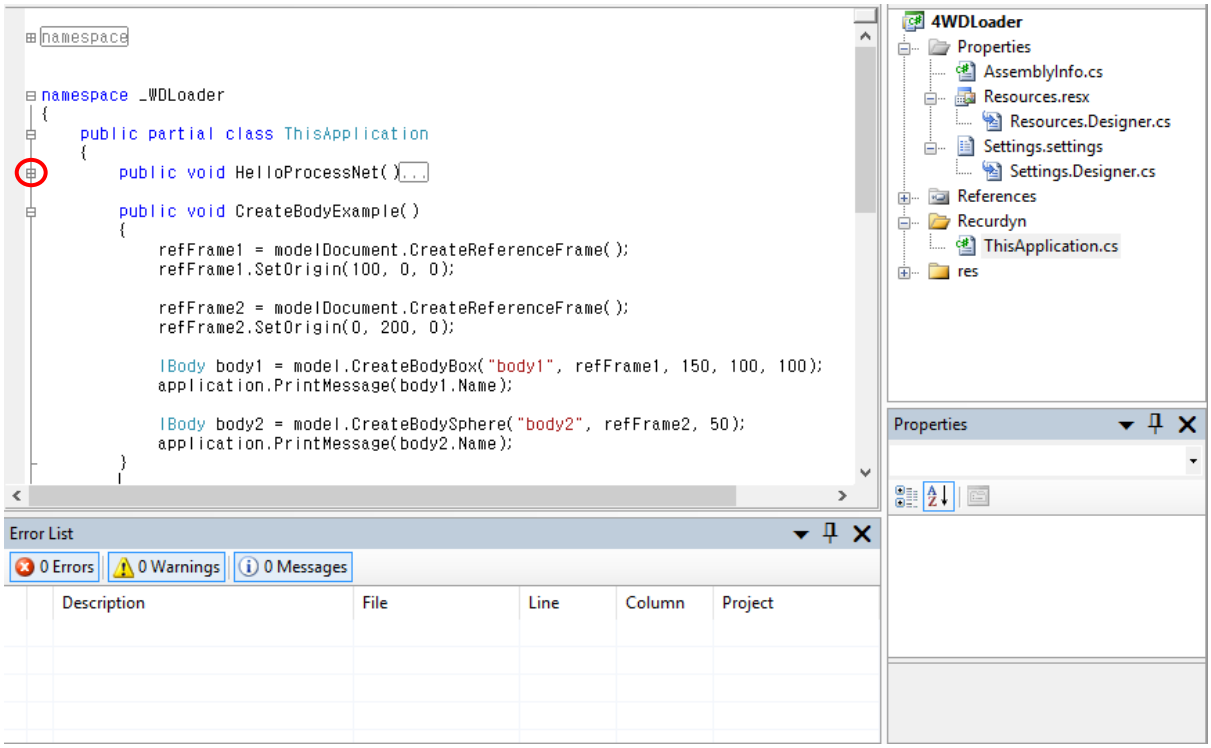
5. ProcessNet 헬프 창을 닫습니다.

ProcessNet 템플릿의 이용

코딩을 시작하기 전에 Visual C#용 RecurDyn ProcessNet 템플릿을 참고하기 바랍니다. 이것은 **HelloProcessNet()**이라고 불리는 예제 매크로이며, RecurDyn 메시지 창에 Hello ProcessNet 이라는 output 을 보여주는 간단한 프로그램입니다.

화면 옆에 있는 + 표시를 클릭하여 닫혀진 부분을 펼칩니다. RecurDyn generated code 를 펼치기 위해서 이 방법을 사용하며, **HelloProcessNet()** 매크로에서도 이 방법을 사용합니다.

선언된 몇 가지 일반적인 변수들을 볼 수 있으며, 변수는 **Initialize()** method 에 의해 초기화됩니다. 이 코드는 변수를 선언한 후 초기화 하지 않고 코드상에서 변수를 쉽게 사용할 수 있도록 하여 편리합니다.



Base 어플리케이션의 생성

이제 호스 Contact 을 생성하기 위한 어플리케이션을 개발해 보겠습니다.

Base 어플리케이션 생성하기**:**

1. 아래의 그림과 같이 **HelloProcessNet()** 매크로 다음 위치에 아래의 코드를 복사하여

```
public void ProcessNetTutorialCreateSolidContact()
{
}
```

삽입합니다.

2. 지금까지 매크로를 위해 **stub** 를 생성하였습니다. 지금 코드를 컴파일 하였다면, 어떤 기능을 실행하지 않았더라도 메소드는 RecurDyn 에서 실행 가능한 매크로 리스트를 보여줄 것입니다.
3. 다음으로, 방금 생성한 매크로 **stub** 에 다음의 변수 선언들을 삽입합니다. (여는 중괄호 아래에 닫는 중괄호 위에 다음과 같이 코드를 삽입합니다.)

```
{
   int BodyNumStart = 20;  // Start creating contacts with body 20
   int BodyNumEnd = 30;  // Continue until body 30
   int BodyInterval = 51;  // Interval between body number on hose 1
                     // and corresponding body's number on
                     // hose 2 is 51
}
```

// 뒤에는 각각의 변수에 대해 설명하고 있습니다. C#에서는 // 이후에 오는 어떠한 글자도 컴파일러에 의해서 무시되며, 코드에서는 설명을 위한 주석으로 사용될 수 있습니다.

Note: C#에서 // 이후에 오는 글자는 주석입니다.

4. 방금 추가한 변수 선언 다음에 loop 를 위해 다음의 코드를 추가합니다.

```
public void ProcessNetTutorialCreateSolidContact()
{
            int BodyNumStart = 20;  // Start creating contacts with body 20
            int BodyNumEnd = 30;  // Continue until body 30
            int BodyInterval = 51;  // Interval between body number on hose 1
                                    // and corresponding body's number on
                                    // hose 2 is 51

            for (int i = BodyNumStart; i <= BodyNumEnd; i++)
            {
            }
}
```

- 이 코드는 for 문 안에서 반복될 것입니다. 처음에 index **i** 는 **BodyNumStart** 와 동일하며. **i** 는 1 씩 증가하면서 다음 loop 가 실행될 것입니다.
- 이것은 **i ≤ BodyNumEnd** 이 참이 될 때까지 반복될 것입니다.
- for 문안에서 선언된 i 는 **for** 문 안에서만 유효합니다. 또한, C#에서는 **i++**는 코드가 실행된후 **i** 가 1 씩 증가한다는 것을 뜻합니다. **i = i + 1** 이나 **i += 1** 과 동일한값을 얻을수 있습니다.

5. 방금 추가한 **for** 문안에, 다음 코드를 삽입합니다.

```
public void ProcessNetTutorialCreateSolidContact()
{
        int BodyNumStart = 20;  // Start creating contacts with body 20
        int BodyNumEnd = 30;  // Continue until body 30
        int BodyInterval = 51;  // Interval between body number on hose 1
                                // and corresponding body's number on
                                // hose 2 is 51

        for (int i = BodyNumStart; i <= BodyNumEnd; i++)
        {
        int j = i + BodyInterval;  // j is the index for the
                                // corresponding bodies on hose #2
                                // Do the contact for corresponding bodies

        IBody baseBody = model.GetEntity("BeamBody" + i.ToString()) as IBody;
        IGeometry baseGeom = baseBody.GetEntity("HollowCircularBeam1") as IGeometry;
        IBody actionBody = model.GetEntity("BeamBody" + j.ToString()) as IBody;
        IGeometry actionGeom= actionBody.GetEntity("HollowCircularBeam1") as IGeometry;
        IContactSolidContact solidContact= model.CreateContactSolidContact("solidContact"
      + i.ToString(), baseGeom, actionGeom);
        }
}
```

이 코드 블록에 대한 설명은 다음과 같습니다.

```
int j = i + BodyInterval;  // j is the index for the
                        // corresponding bodies on hose #2
```

위의 코드에서:

- i 는 호스 1 의 Body 에 대한 index 이며, j 는 호스 2 의 Body 에 대한 index 입니다.
- 위 조건에 부합하는 Body 를 생성하기 위해서는 j 의 값이 **i + bodyInterval** 이 되도록 합니다.
- 또한, j 는 for 문 안에서 선언되어야 하며, for 문 밖에서는 유효하지 않습니다.

```
IBody baseBody
        = model.GetEntity("BeamBody" + i.ToString()) as IBody;
```

위의 코드는 우리가 사용하기를 원하는 base body 를 body 의 이름을 사용해서 찾을수 있게 해줍니다.

- **+**기호는 두 개의 string 을 함께 사용합니다.
- **ToString()** 메소드는 i 의 정수 값을 한 개의 string 으로 변환합니다.
- 그러므로, **GetEntity()** 메소드는 "BeamBody20", "BeamBody21"등 이름을 사용해서 body 를 검색합니다.

- **GetEntity()** 메소드는 어떤 값이 반환될지 모르기 때문에 기본적으로 **Igeneric** 로 반환되며 여기서는 **Ibody** 로 사용해야 하기 때문에 **as Ibody** 를 사용해서 형변환을

```
IGeometry baseGeom
        = baseBody.GetEntity("HollowCircularBeam1") as IGeometry;
```

해줍니다.

위의 코드는 base body 의 geometry 를"HollowCircularBeam1"라는 이름으로 검색합니다. 이 geometry 는 solid contact 을 생성하기 위해 사용됩니다.

action body 에 대해서도 앞에서 실행한 두 단계를 다시 실행합니다.

```
IContactSolidContact solidContact
        = model.CreateContactSolidContact("solidContact"
        + i.ToString(), baseGeom, actionGeom);
```

마침내, 위의 커맨드가 "solidContact20" (또는 "solidContact21", "solidContact22", 기타)라는 이름의 solid contact 을 생성하였습니다. 이 커맨드는 우리가 이전 단계에서 정의한 base body 와 action body 에 사용됩니다.

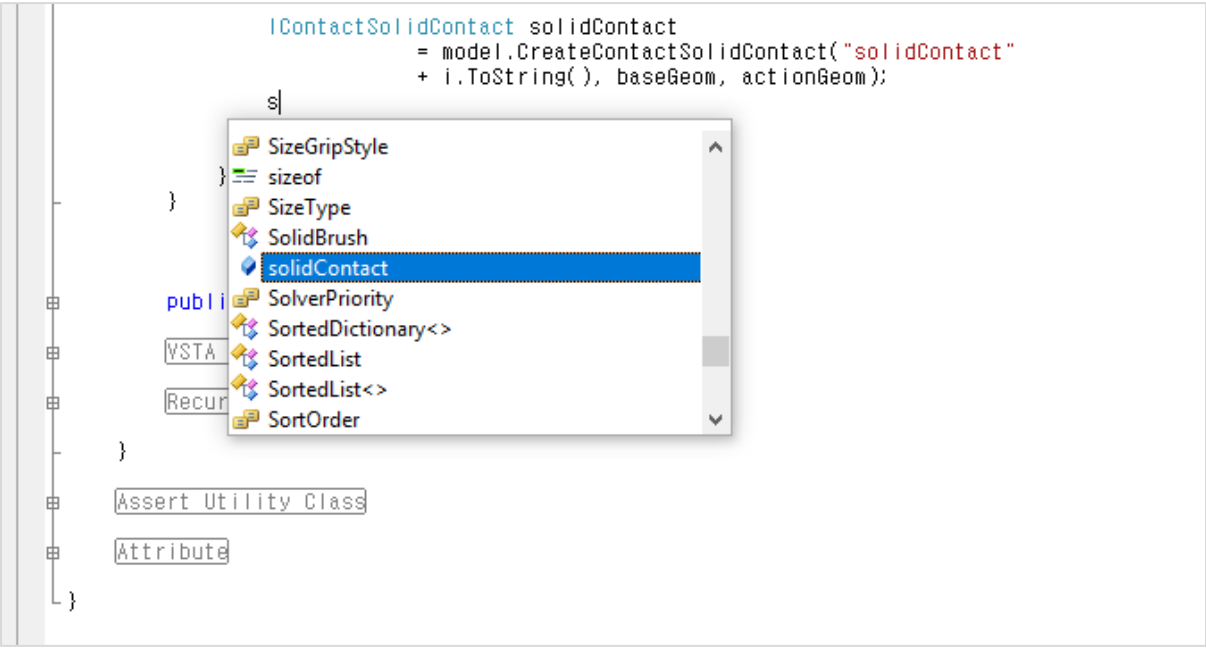
IntelliSense 를 이용한 코딩

지금까지는 튜토리얼에있는 코드를 복사 불어넣기를 사용해서 코딩을 하였습니다. 사용자가 자신의 매크로를 작성할 때 코드를 자동적으로 타이핑하는 방법이 있습니다. ProcessNet IDE 의 특징중에 하나는 사용자가 IntelliSense 를 이용해서 코드를 더 빠르게 작업하게 할 수 있습니다. 다음의 작업들을 수행하면서 이 특징들을 알게 될 것입니다.

컴파일 후에 매크로는 기본값을 가진 Contact 이 생성이 됩니다. 그러나, Solid Contact 의 기본적인 stiffness 와 damping 값들이 너무 높기 때문에 Contact 의 parameter 값을 낮춰야 합니다.

IntelliSense 를 이용하여 Contact 파라미터 수정하기

1. 이전 단계에서 입력한 코드의 마지막 줄 다음에 **'s'**를 입력하면 아래의 그림처럼 팝업 목록이 보입니다.



이 목록에서는 이 코드 및 호출할 수 있는 메소드 등 안에서 접근 가능한 변수의 이름을 포함하여 다음에 입력이 가능한 것을 보여줍니다.

2. IntelliSense 목록에서 **solidContact** 에 마우스를 위치 시킵니다.
3. **solidContact** 을 마우스 버튼을 더블 클릭하거나 키보드의 Enter 키를 눌러서 선택합니다.
4. 마침표를 입력합니다.
5. IntelliSense 목록에서 **ContactProperty** 를 선택합니다.
6. Value 값을 입력할때까지 과정을 반복해서 실행합니다.

```
solidContact.ContactProperty.StiffnessCoefficient.Value =1000;
```

7. 이전 과정을 반복해서 다음코드를 입력합니다.

```
solidContact.ContactProperty.DampingCoefficient.Value = 0.1;
```

위 과정을 마치면, 아래와 같이 메소드가 나타납니다.

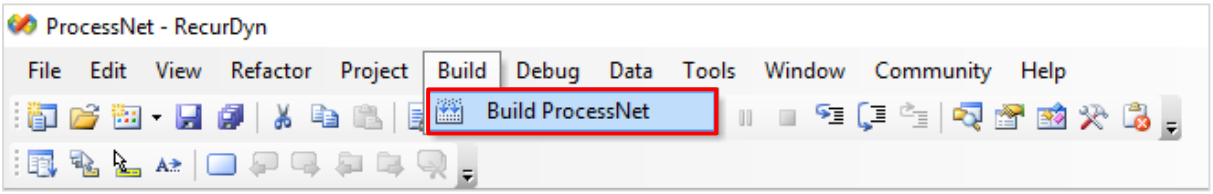
```
public void ProcessNetTutorialCreateSolidContact()
{
        int BodyNumStart = 20;  // Start creating contacts with body 20
        int BodyNumEnd = 30;  // Continue until body 30
        int BodyInterval = 51;  // Interval between body number on hose 1
                                // and corresponding body's number on
                               // hose 2 is 51

        for (int i = BodyNumStart; i <= BodyNumEnd; i++)
        {
                int j = i + BodyInterval;  // j is the index for the
                                        // corresponding bodies on hose #2
                                        //Do the contact for corresponding bodies

                IBody baseBody = model.GetEntity("BeamBody" + i.ToString()) as IBody;
                IGeometry baseGeom = baseBody.GetEntity("HollowCircularBeam1") as IGeometry;
                IBody actionBody = model.GetEntity("BeamBody" + j.ToString()) as IBody;
                IGeometry actionGeom = actionBody.GetEntity("HollowCircularBeam1") as IGeometry;
                IContactSolidContact solidContact = model.CreateContactSolidContact("solidContact"
                + i.ToString(), baseGeom, actionGeom);

                solidContact.ContactProperty.StiffnessCoefficient.Value =1000;
                solidContact.ContactProperty.DampingCoefficient.Value = 0.1;
        }
}
```

IDE 창 아래에 Error 목록 창에서 에러와 경고가 보이지 않아야 합니다.

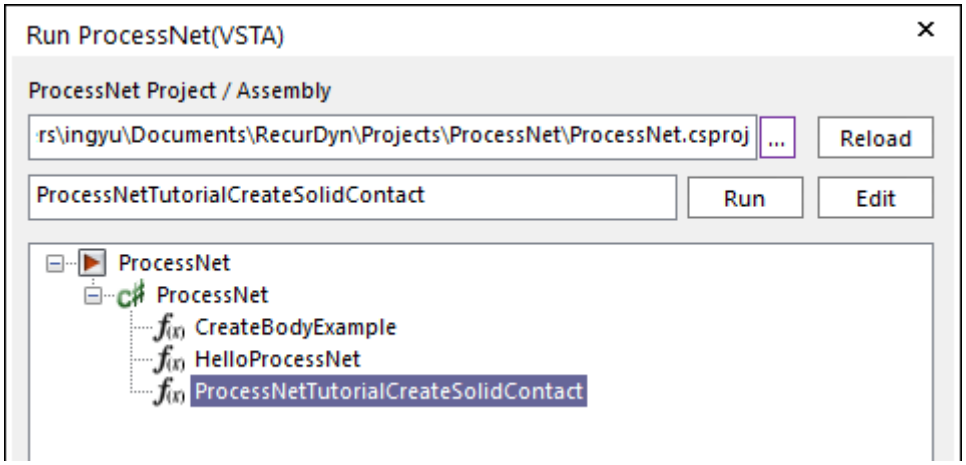
매크로의 빌드와 실행

매크로를 빌드하여 실행하기**:**

1. 만약 에러나 경고가 생겼다면 목록을 확인하여 수정합니다.
2. **Build** 메뉴에서 **Build ProcessNet** 을 클릭합니다.



3. RecurDyn 으로 돌아와서 **Customize** 메뉴에서 **Processnet(VSTA)** 그룹의 **Run** 을 선택합니다.
4. 매크로 목록에서 **ProcessNetTutorialCreateSolidContact** 을 선택합니다. **Run** 버튼 다음 필드에 그 이름이 보여집니다.
5. **Run** 을 클릭합니다.

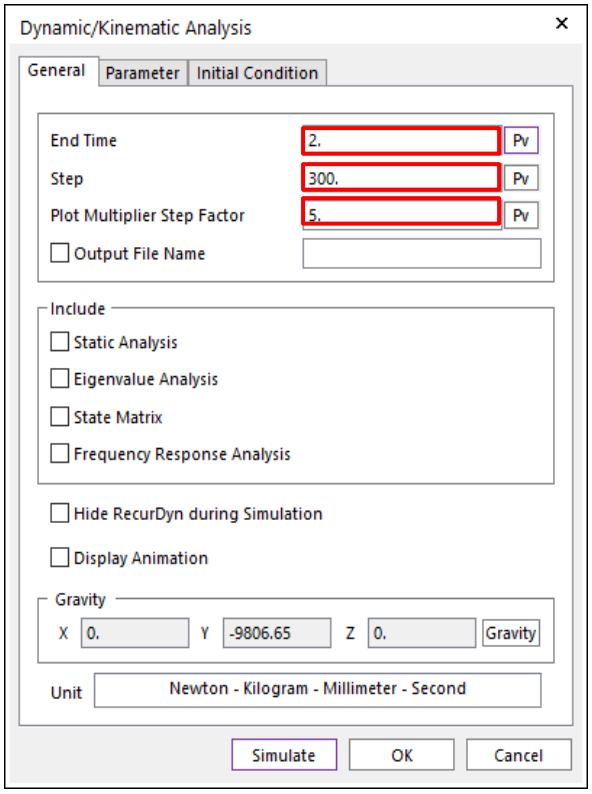


이제, 모델에 11 개의 Contact 이 추가되었습니다.

시뮬레이션의 실행

시뮬레이션 실행하기:

1. **Analysis** 탭에서 **Simulation Type** 그룹의, **Dynamic/Kinematic** 을 선택합니다.
2. 시뮬레이션을 실행하기 위해 End Time 을 **2.0**, Step 을 **300** 으로 설정하고, **Plot Multiplier Step Factor** 는 **5** 로 설정합니다.
3. **Simulate** 를 클릭합니다. 컴퓨터의 속도에 따라 시뮬레이션이 3 - 4 분간 실행될 수도 있습니다.

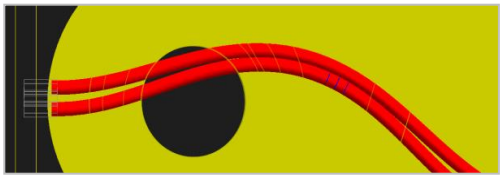


결과보기

결과보기:

1. 차량의 아래쪽을 살펴보기 위해 Render Each 모드로 설정합니다.

2. Simulation toolbar 의 Animation Control 에서 **Play** 버튼을 클릭합니다.

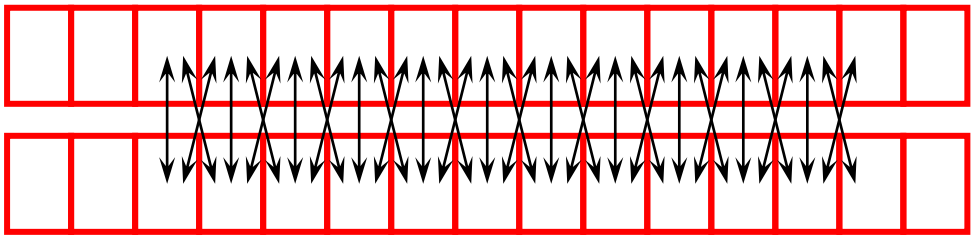


시뮬레이션의 끝부분에서 호스는 오른쪽 그림과 같이 변형됩니다.

3. 다음과 같이 모든 Contact 의 Action Force Display 를 켭니다.
   - Database 창에서 첫 번째 Solid contact 을 선택합니다.
   - **Shift** 키를 누른 채로, Database 창에서 마지막 Solid contact 을 선택합니다.
   - 오른쪽 마우스 버튼을 클릭하여 **Property** 를 클릭합니다.
   - 다이얼로그박스가 나타나면, Solid 탭을 클릭합니다. 다이얼로그박스 하단의 풀다운 메뉴에서 **Force Display** 를 **Action** 으로 설정합니다.

- OK 를 클릭합니다.

4. 애니메이션을 다시 실행합니다. 그러면, Force 가 약간 불규칙적이라는 것을 알 수 있을 것입니다. 움직임을 관찰해보면, 호스가 서로 미끄러지고 있음을 알 수 있는데, 이것은 해당 세그먼트 사이에서 Contact 이 충분하지 않기 때문입니다. 그렇기 때문에 이웃하는 세그먼트에 Contact 을 추가하는 작업이 필요합니다. 이것은 아래의 그림처럼 보여집니다.



추가적으로 필요한 **Contact** 설정

추가적으로 필요한 **Contact** 설정하기**:**

1. 아래 보여지는 것처럼 코드를 수정합니다.

```
            IContactSolidContact solidContact = model.CreateContactSolidContact("solidContact"
            + i.ToString(), baseGeom, actionGeom);
            solidContact.ContactProperty.StiffnessCoefficient.Value = 1000;
            solidContact.ContactProperty.DampingCoefficient.Value = 0.1;

            // Do the contact for body i+1 and body j
            solidContact = model.CreateContactSolidContact("solidContact" + i.ToString() + "a",
            (model.GetEntity("BeamBody" + Convert.ToString(i + 1)) as IBody).
            GetEntity("HollowCircularBeam1") as IGeometry,
             (model.GetEntity("BeamBody" + j.ToString()) as IBody).GetEntity("HollowCircularBeam1")
            as IGeometry);
            solidContact.ContactProperty.StiffnessCoefficient.Value = 1000;
            solidContact.ContactProperty.DampingCoefficient.Value = 0.1;

            // Do the contact for body i and body j+1
            solidContact = model.CreateContactSolidContact("solidContact" + Convert.ToString(i) + "b",
        (model.GetEntity("BeamBody" + i.ToString()) as IBody).GetEntity("HollowCircularBeam1")
            as IGeometry, (model.GetEntity("BeamBody" + Convert.ToString(j + 1))
            as IBody).GetEntity("HollowCircularBeam1") as IGeometry);
            solidContact.ContactProperty.StiffnessCoefficient.Value = 1000;
            solidContact.ContactProperty.DampingCoefficient.Value = 0.1;
            }
}
```

여기서, 코드를 통해 각 loop 에 대해서 2 개 이상의 Contact 이 생성됩니다. 호스 1 의 $i+1^{th}$ segment 와 hose 2 의 j^{th} segment 사이에 생성된 Contact 에 "a"를 붙여서 이름을 지정합니다. (예를 들어, "solidContact20a"). 이와 유사하게, 호스 1 의 i^{th}segment 와 호스 2 의 $j+1^{th}$ segment 사이에서 생성된 Contact 에 "b"를 붙여서 이름을 지정합니다. (예를 들어, "solidContact20b")

Note: 이미 존재하는 이름과 같은 이름으로 Contact 을 지정하면 Contact 생성이 되지 않습니다.

- 새로 추가된 CONTACT 을 만들기 위한 코드는 더 효율적이고 컴팩트한 형태입니다. 예를 들어, 이전에는 Contact 을 생성하기 위한 Action Body 와 Geometry 의 값을 저장하기 위해 임시 변수를 선언하고 할당을 한 반면에 지금은 **CreateContactSolidContact ()** 메서드 내에서 Action Body 와 Geometry 의 값을 바로 지정해 줍니다. 이러한 구문은 코드를 보다 효율적으로 만들어줍니다. 하지만 당신은 해당 매소드의 정확한 반환 값을 as 를 사용해서 Casting 해줘야 합니다. (i.e. **as IBody** or **as IGeometry**)

빌드, 시뮬레이션, 프로세스 보기의 반복 실행

방금 수정한 모델을 확정하기 위해 이전에 했던 단계를 반복하여 실행합니다.

프로세스를 반복실행하기:

1. IDE 창 하단의 Error 목록 창에서 에러나 경고가 없는지 확인합니다. 만약 에러나 경고가 있다면, 확인하여 수정합니다. **Build** 메뉴에서 **Build ProcessNet** 를 선택합니다.
2. RecurDyn 으로 돌아와서 이전에 실행하여 정의한 모든 Contact 을 지웁니다.
3. **Customize** 탭에 있는 **ProcessNet(VSTA)** 그룹의 **Run** 을 선택합니다.
4. Run 을 클릭합니다. 모델에 33 개의 Contact 이 추가되었습니다.
5. 이전에 실행했을 때와 동일한 파라미터 값으로 시뮬레이션을 다시 실행합니다.
6. 추가한 Contact 을 포함한 시뮬레이션은 4- 5 분간 진행됩니다.
7. 모든 Contact 에 대해서 이전에 했던 것과 동일한 방식으로 **Action Force Display** 를 켭니다
8. 차량의 아래쪽을 살펴보기 위해 Render Each shading 모드로 모델을 다시 설정하여, Simulation toolbar 의 Animation Control 에서 Play 버튼을 클릭합니다.

Contact Force 가 더욱 매끄러워진 것을 볼 수 있을 것입니다.

Chapter 4

다이얼로그와 출력 메세지의 추가

이 장에서는, Contact 을 생성하기 위한 세그먼트를 조절하기 위한 다이얼로그 윈도우를 생성해 볼 것입니다. 이 작업은 다이얼로그의 레이아웃의 설계 및 새 다이얼로그 호출을 위해 존재하는 Subroutine 에 대한 코드를 추가하는 작업을 포함합니다.

목적

호스와 호스 사이에 생성되는 Contact 을 조절하기 위한 다이얼로그 윈도우를 생성하는 과정을 통해 어플리케이션의 활용도를 어떻게 증가시킬 수 있을지 배워 봅니다. 또한, RecurDyn 메시지 창에서 사용자에게 메시지가 어떻게 보여지는지 배워 봅니다.

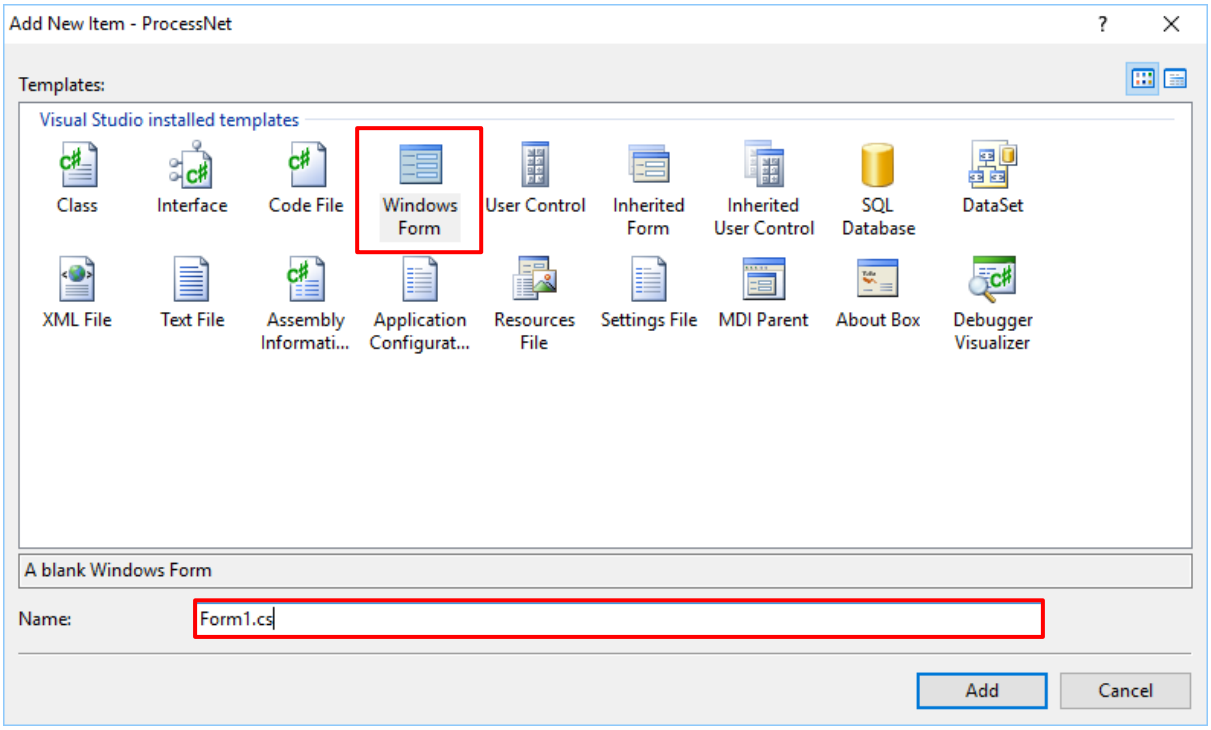
예상 소요 시간

20 분

다이얼로그의 설계

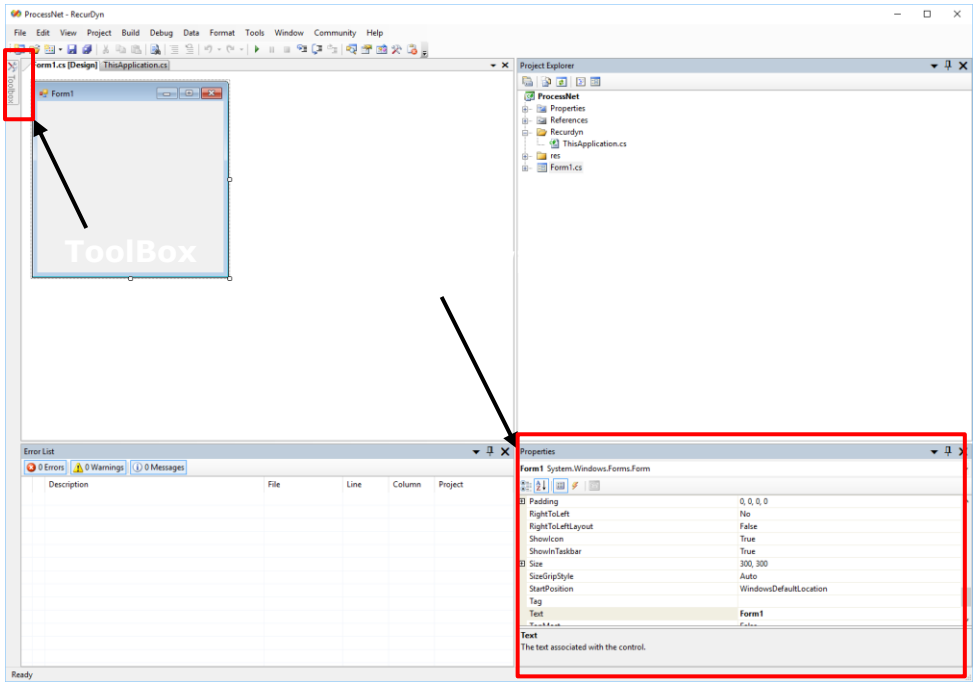
새 다이얼로그 윈도우 설계하기**:**

1. ProcessNet IDE 로 돌아옵니다.
2. **Project** 메뉴에서, **Add Windows Form** 을 선택합니다.
3. Add New Item 다이얼로그박스에서, 아래의 그림처럼 **Windows Form** 을 선택합니다.



4. 기본 설정 이름으로 **Form1.cs** 을 수락합니다.
5. **Add** 를 클릭합니다.

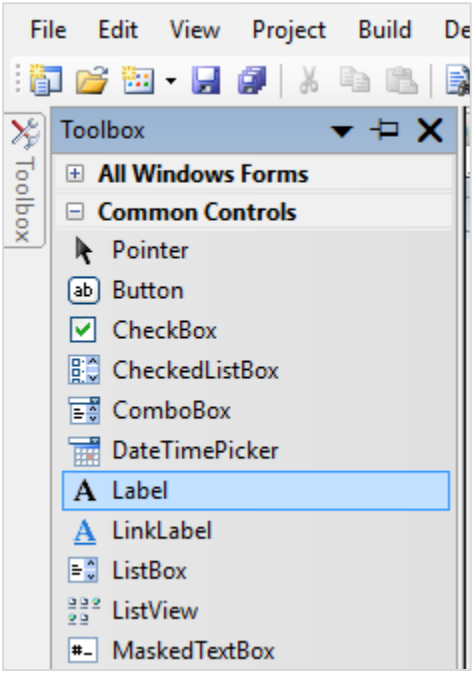
Form1 을 위한 설계 윈도우인 **Form1.cs [Design]**가 IDE Project Editor 윈도우가 나타납니다.



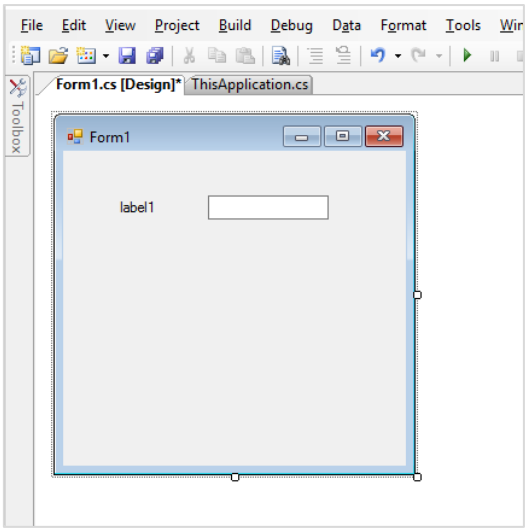
6. 화면 왼쪽상단에서 **Toolbox** 

위로 커서를 이동시킵니다. 그러면, 다이얼로그 윈도우 및 기타 비슷한 제어 기능을 추가할 수 있는 다른 요소가 있는 메뉴가 보여집니다.

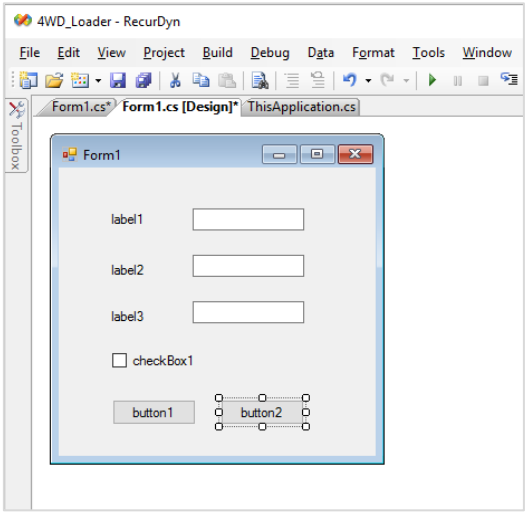
7. **Common Controls** 목록에서, **Label** 을 클릭한 후 설계하고자 하는 다이얼로그의 왼쪽 상단 구역으로 **Label** 을 드래그합니다.

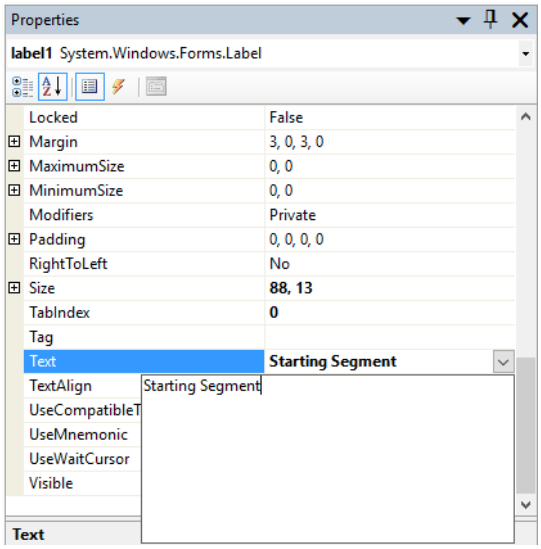


8. 위와 동일한 과정을 반복 실행하되, 오른쪽에 보여지는 것처럼 다이얼로그 안 Label 위치의 오른편으로 TextBox 를 드래그합니다.



9. 앞에서 실행했던 과정을 반복하여 오른쪽 그림과 같이 3 줄의 라벨과 텍스트박스를 생성합니다.
10. **Toolbox** 에서 **Checkbox** 를 선택하여 추가합니다.
11. **Toolbox** 에서 **Button** 을 선택하여 두 개의 버튼을 다이얼로그에 추가하면, 오른쪽 그림과 같이 다이얼로그가 보여집니다.
12. **label1** 을 클릭합니다.
13. 오른쪽 모서리 하단의 Properties 창에서 **Text** 의 값을 **Starting Segment** 로 수정합니다.





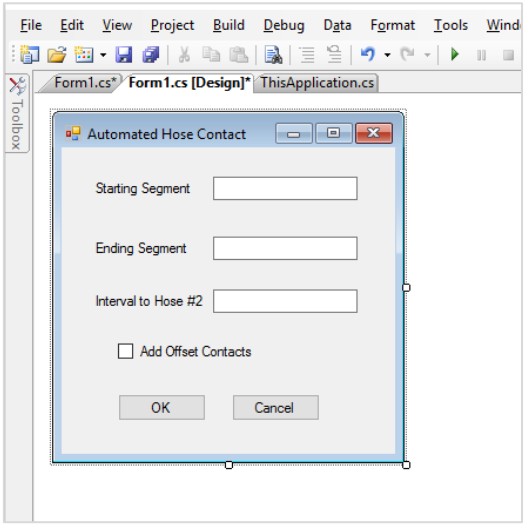
Tip: 라벨의 편집 필드는 작지만 오른쪽에 보이는 것처럼 드랍다운 화살표를 클릭하여 쉽게 이름을 수정할 수 있습니다.

14. **Labels 2** 와 **3, Checkbox, button1, button2, Dialog** 에 대해서도 앞의 과정과 동일한 방법으로 다음 표를 참고하여 **Text** 의 값을 수정합니다.

Dialog Element	Text
label1	Starting Segment
label2	Ending Segment
label3	Interval to Hose #2
checkBox1	Add Offset Contacts
button1	OK
button2	Cancel
Form1	Automated Hose Contact

Tip: 구성요소가 없는 위치 중 어느 한 곳을 클릭을 하면 다이얼로그가 선택됩니다.

15. 다이얼로그의 구성요소의 크기와 위치를 오른쪽 그림과 같이 재조정합니다.



Tip: ProcessNet IDE 는 이 단계를 도와주기 위한 alignment 가이드를 제공합니다.

16. **File** 메뉴에 **Save All** 을 선택합니다. 이것은 프로젝트 구성뿐 만 아니라 **Form1.cs** 을 저장해줍니다.

다이얼로그의 초기 환경 정의하기

지금까지 다이얼로그의 외관을 구성해보았습니다. 이제, 체크박스, 텍스트박스에 대해 사용자가 값을 입력할 수 있도록 변수를 추가하여 다이얼로그의 초기환경을 정의할 것입니다. 모든 변수를 초기화하며, OK 버튼을 클릭했을 때의 기능을 정의해 줄 코드를 추가하는 작업을 해보겠습니다.

다이얼로그 윈도우의 초기환경 정의하기**:**

1. 설계된 다이얼로그 윈도우에서 구성요소가 없는 부분을 더블 클릭합니다. 그러면, Form1.cs 의 코드가 IDE Project 편집 창에 보이며, 호출된 **Form1_Load** Subroutine 에 대한 stub 가 생성됩니다. 이 subroutine 은 다이얼로그 박스의 새 복사본이 생성되고 로드 될 때마다 호출될 것이며, 변수를 초기화할 코드를 위한 이상적인 곳입니다. 다이얼로그 윈도우에 사용될 변수를

```
public partial class Form1 : Form
{
        public int BNumStart;
        public int BNumEnd;
        public int BodyInterval;
        public bool AddOffsetFlag;

        public Form1()
        {
                InitializeComponent();
        }
```

정의하기 위해 코드에 다음 부분을 삽입합니다.

2. 다이얼로그 윈도우에 보여지는 초기 변수를 구성하기 위한 다음 코드를 삽입합니다.

```
private void Form1_Load(object sender, EventArgs e)
        {
        textBox1.Text = "20";
        BNumStart = 20;
        textBox2.Text = "30";
        BNumEnd = 30;
        textBox3.Text = "51";
        BodyInterval = 51;
        checkBox1.Checked = false;
        }
```

3. 설계된 다이얼로그 윈도우로 돌아와서 OK 버튼을 더블 클릭합니다.

4. 자동으로 생성된 새로운 메소드 안에 아래와 같이 코드를 삽입합니다.

```
private void button1_Click(object sender, EventArgs e)
        {
        BNumStart = Convert.ToInt32(textBox1.Text);
        BNumEnd = Convert.ToInt32(textBox2.Text);
        BodyInterval = Convert.ToInt32(textBox3.Text);
        AddOffsetFlag = checkBox1.Checked;
        DialogResult = DialogResult.OK;
        Close();
        }
```

5. 설계된 다이얼로그 윈도우로 다시 돌아와서 **Cancel** 버튼을 더블 클릭합니다.

6. 자동으로 생성된 새로운 Method 안에 아래와 같이 코드를 삽입합니다.

```
private void button2_Click(object sender, EventArgs e)
        {
        Close();
        }
```

7. **File** 메뉴에서 **Save Form1.cs** 를 선택합니다.

매크로를 실행했을 때 다이얼로그 윈도우 나타내기

이제, 매크로의 Subroutine 안에 Subroutine 의 새로운 Instance 를 생성하여, 매크로를 실행했을 때 다이얼로그가 보이도록 정의할 것입니다.

매크로를 이용하여 다이얼로그 윈도우 나타내기**:**

1. 생성된 **ProcessNetTutorialCreateSolidContact** subroutine 의 모든 코드를 복사합니다.
2. ThisApplication.cs 에 복사한 코드를 붙여 넣은 후, **ProcessNetTutorialCreateSolidContact_WithDialog** 으로 subroutine 의 이름을 재정의 합니다.
3. 다음과 같이 줄이 그어진 코드는 지워서 subroutine 를 수정합니다. 각각에 대한 설명은 밑에 나와있습니다.

```
public void ProcessNetTutorialCreateSolidContact_WithDialog()
{
        int BodyNumStart = 20;  // Start creating contacts with body 20
        int BodyNumEnd = 30;  // Continue until body 30
        int BodyInterval = 51;  // Interval between body number on hose 1
                        // and corresponding body's number on
                        // hose 2 is 51

        // Create a Form
        Form1 MyForm = new Form1();

        // Open the Dialog
        MyForm.ShowDialog();

        if (MyForm.DialogResult == System.Windows.Forms.DialogResult.OK)
                {
                int NumContacts = 0;
                int BodyNumStart = MyForm.BNumStart;
                int BodyNumEnd = MyForm.BNumEnd;
                int BodyInterval = MyForm.BodyInterval;

                for (int i = BodyNumStart; i <= BodyNumEnd; i++)
                        {
                        int j = i + BodyInterval;  // j is the index for the
                                        // corresponding bodies on
                                        // hose #2
                                        // Do the contact for corresponding bodies


                        solidContact.ContactProperty.DampingCoefficient.Value = 0.1;

                        // Increment the number of contacts
                        NumContacts = NumContacts + 1;

                        if (MyForm.AddOffsetFlag)
                                {
                                //Do the contact for body i+1 and body j
                                //Do the contact for body i and body j+1


                                solidContact.ContactProperty.DampingCoefficient.Value = 0.1;
                                // Increment the number of contacts
                                NumContacts = NumContacts + 2;
                                }
                        }
                application.PrintMessage(NumContacts.ToString() +
                " contacts were created between the two hoses.");
                }
}
```

- 수정 1:

```
        int BodyNumStart = 20;  // Start creating contacts with body 20
        int BodyNumEnd = 30;  // Continue until body 30
        int BodyInterval = 51;  // Interval between body number on hose 1
                        // and corresponding body's number on
                        // hose 2 is 51
```

여기서 이제 다이얼로그 윈도우 상자에서 사용자가 값을 입력하게 되므로 하드코딩된 값을 삭제해줍니다.

- 수정 2:

```
// Create a Form
Form1 MyForm = new Form1();

// Open the Dialog
MyForm.ShowDialog();

if (MyForm.DialogResult == System.Windows.Forms.DialogResult.OK)
{
        int NumContacts = 0;
```

여기서는 Form1 의 새로운 인스턴스를 생성하고 그것을 보이도록 하였습니다. if 문은 사용자의 응답을 테스트할 것입니다. 사용자가 OK 를 클릭할 경우, if 문 은 실행될 것입니다. 또한, NumContacts 라 불리는 변수를 선언 할것입니다. 이 변수는 생성된 contact 의 개수를 유지하며. if 문은 수정 5 에서 닫힙니다.

- 수정 3:

```
        int BodyNumStart = MyForm.BNumStart;
        int BodyNumEnd = MyForm.BNumEnd;
        int BodyInterval = MyForm.BodyInterval;
```

여기에서는 Segment 수에 대한 변수를 포함하는 코드를 교체하였습니다. 새로운 코드는 다이얼로그에 사용자가 입력한 값과 함께 변수들을 할당할 것입니다.

- 수정 4:

```
        // Increment the number of contacts
        NumContacts = NumContacts + 1;

        if (MyForm.AddOffsetFlag)
        {
                // Increment the number of contacts
                NumContacts = NumContacts + 2;
```

NumContacts 이 업데이트 됩니다. 또한, if 선언문은 사용자가 **Add Offset Contacts** 체크박스를 체크하는 지 안 하는지 여부를 체크합니다. 사용자가 체크 박스를 체크했다면, if 선언문이 실행됩니다.

- 수정 5:

```
        application.PrintMessage(NumContacts.ToString() +
                " contacts were created between the two hoses.");
}
```

몇 개의 Contact 이 생성되었는지를 확인하는 출력 메시지가 RecurDyn 메시지 창에서 사용자에게 보여집니다. 또한, 가장 바깥쪽의 if 문이 끝납니다. (수정 1 을 참고하세요.)

4. **File** 메뉴에서 **Save ThisApplication.cs** 를 클릭합니다.

다이얼로그 윈도우의 테스트

매크로를 빌드 하기 위해 앞에서 실행한 과정들을 반복하고 수정된 코드를 테스트합니다.

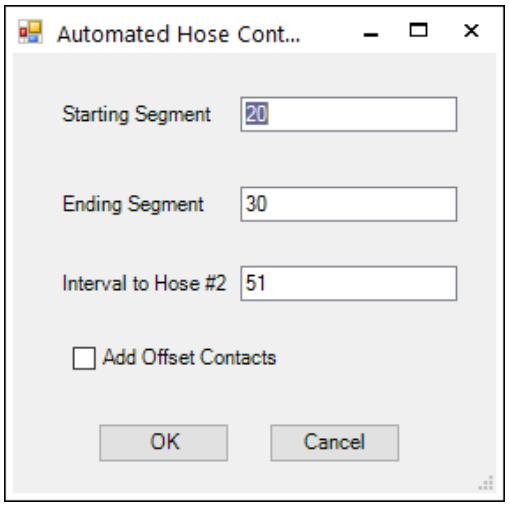
실행 과정 반복하기:

1. IDE 창 하단의 Error List창에서 에러나 경고가 없는지 확인합니다. 에러나 경고가 있다면, 목록을 확인하여 수정합니다. Build메뉴에서 Build ProcessNet을 선택합니다.

2. **Customize** 탭의 **ProcessNet(VSTA)** 그룹에서 **Run** 을 선택합니다.

   리스트에서 ProcessNetDialogCreateSolidContact_WithDailog 라는 새로운 아이템이 보여지면 ProcessNetDialogCreateSolidContact_WithDailog 를 선택합니다. Run 버튼 옆에 ProcessNetDialogCreateSolidContact 이 로드 됩니다.

3. **Run** 을 클릭합니다. 그러면 오른쪽 그림과 같이 다이얼로그박스가 보여집니다.



4. 기본 값들을 사용하기 위해 **OK** 를 클릭합니다.

   Database 창에서 생성된 11 개의 Contact 을 확인하고, 11 개의 Contact 이 생성되었다는 확인 메세지를 **RecurDyn** 메시지 출력 창에서 확인합니다.

5. **RecurDyn** 에서 **Undo** 화살표를 클릭하여 생성한 Contact 들을 지우고 매크로를 다시 실행시킵니다. 단, 이때 **AddOffset Contacts** 체크박스를 체크합니다.

6. **Database** 창에서 모델에 추가된 33 개의 Contact 을 확인하고, 33 개의 Contact 이 생성되었다는 확인 메시지를 **RecurDyn** 메시지 출력 창에서 확인합니다.

7. 이전에 실행한 시뮬레이션의 결과를 복원합니다. **File** 메뉴에서 **Import** 를 선택합니다. 풀다운 메뉴를 이용하여 파일 형식을 **RecurDyn Animation Data File (*.rad)**으로 변경합니다. **4WD_Loader.rad** 파일을 선택한 후 Open 을 클릭합니다.

8. **Run ProcessNet** 창을 닫습니다.

Chapter 5

Plot 자동생성

이 장에서는 여러 개의 Plot 창에서 두 개의 Plot 을 만들 것입니다.

- 첫 번째로 그리게 되는 Plot 은 호스의 개별적인 segment-to-segment contact 에 대한 결과를 보여줄 것입니다. Force 가 발생하지 않는 Contact 은 배제됨으로 Plot 은 오직 그리고자 하는 데이터만을 보여줍니다. 첫 번째 Plot 은 호스 Contact Force 에 대하여 X 축을 시간으로 두고 label 과 scaling 포멧을 개선하여 보일 것입니다.
- 두 번째로 그리게 되는 Plot 은 X, Y, Z 축의 구성요소를 분리하여 전체적인 hose-to-hose contact 에 대한 결과를 보여줄 것입니다.

목적

자동으로 Plot 을 그리기 위해서 **ProcessNet** 명령을 어떻게 사용해야 되는지 배웁니다. 이 과정에서는 다음과 같은 내용을 배웁니다.

- Plot Data 파일의 임포트
- 효과적으로 데이터를 보여주기 위한 기능
- 여러 Plot 창의 데이터를 그리기 위한 과정
- 데이터를 기초로 한 차트의 서식 설정

예상 소요 시간

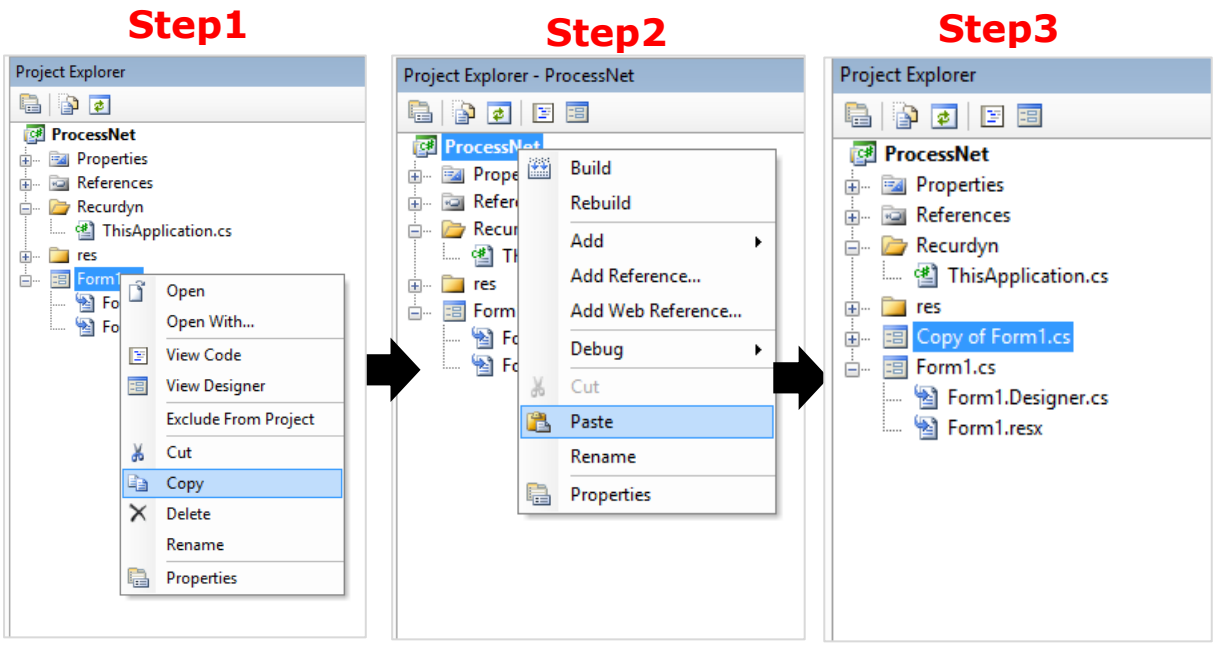
20 분

다이얼로그의 생성

Plot 의 데이터를 설정하기 위해, 사용자 정의 값을 불러올 다른 다이얼로그 윈도우가 필요합니다. 이 다이얼로그는 생성된 첫 번째 다이얼로그와 매우 비슷하므로, 첫 번째 다이얼로그를 복사하여 생성할 것입니다.

새로운 다이얼로그 윈도우 생성하기**:**

1. 오른 편에 있는 Project Explorer 창의 **Form1.cs** 위에서 오른쪽 마우스 버튼을 클릭하여 **Copy** 를 선택합니다. (다음 단계들은 아래의 그림을 참고하세요.)
2. **ProcessNet** 위에서 오른쪽 마우스 버튼을 클릭한 후 **Paste** 를 선택합니다.
3. **Copy of Form1.cs** 를 선택하고 이름을 **edit** 합니다**.**
4. **Copy of Form1.cs** 의 이름을 **Form2.cs** 로 변경합니다.



새로운 다이얼로그 윈도우 크기 조절하기**:**

1. **Form2.cs** 를 클릭하여 파일을 엽니다.

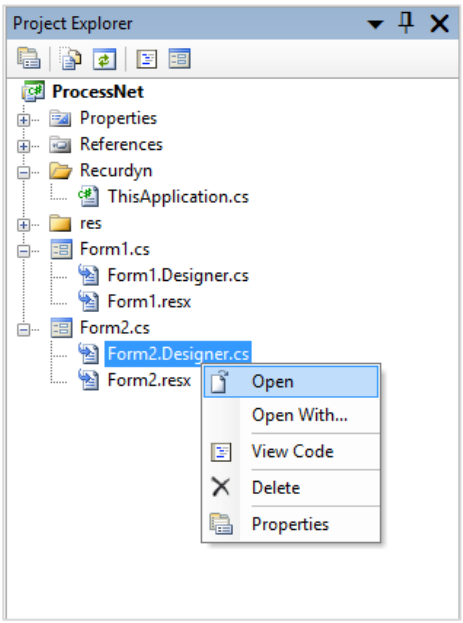
   다이얼로그 설계 윈도우가 나타납니다.

2. View 메뉴에서 코드 윈도우를 보기 위해 Code 를 선택합니다.

3. 코드에서 "Form1"을 찾아서 "Form2"로 모든 인스턴스를 교체합니다. 여기서는 아래에서 보이는 것처럼 3 개의 인스턴스를 수정합니다.

```
namespace ProcessNet.csproj
{
        public partial class ~~Form1~~ Form2 : Form
        {
                public int BNumStart;
                public int BNumEnd;
                public int BodyInterval;
                public bool AddOffsetFlag;

                public ~~Form1~~ Form2()
                {
                        InitializeComponent();
                }

                private void ~~Form1_Load~~ Form2_Load(object sender, EventArgs e)
                {
                        textBox1.Text = "20";
                        BNumStart = 20;
```

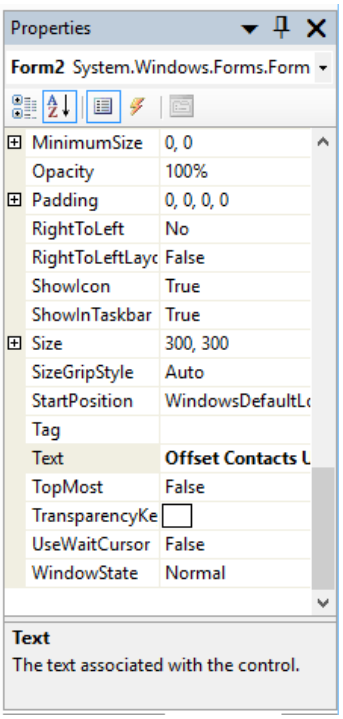
4. 오른쪽 그림에서 보이는 것처럼 **Project Explorer** 창의 **Form2.Designer.cs** 위에서 오른쪽 마우스 버튼을 클릭한 후 Open 을 선택합니다.



5. "**Form1**"을 다시 찾아서 "**Form2**"로 모든 인스턴스를 교체합니다. 여기서는 아래에 보이는 것처럼 2 개의 인스턴스를 수정해야 합니다.

```
partial class ~~Form1~~ Form2
{
        private void InitializeComponent()
        {
                this.Load += new System.EventHandler(this.~~Form1_Load~~ Form2_Load);
                this.ResumeLayout(false);
                this.PerformLayout();
        }
```

6. IDE Project Editor 창 위에서 **Form2.cs [Design]** 탭을 클릭합니다.
7. 구성요소가 없는 다이얼로그의 아무 곳이나 클릭하여 다이얼로그 윈도우를 선택합니다.
8. 화면의 오른쪽 하단의 Properties 윈도우에서 Text 를 **Automated Hose Contact Plotting** 로 수정합니다.
9. IDE Project Editor 윈도우에서 다이얼로그 윈도우의 오른 편을 클릭한 후 드래그하여 다이얼로그의 새로운 이름이 모두 보이도록 다이얼로그 윈도우의 크기를 다시 조정합니다.
10. Add Offset Contacts 라벨을 클릭합니다.
11. Properties 윈도우에서 **Offset Contacts Used** 를 수정합니다.
12. **File** 메뉴에서 **Save All** 을 선택합니다.



Contact Force 의 Plot 생성

이제, ThisApplication class 에서 새로운 Subroutine 을 생성해보겠습니다.

새로운 **subroutine** 생성하기**:**

1. 다음 코드 부분을 복사한 후, ThisApplication class 에 복사한 코드를 삽입합니다. (**Tip**: 다른 subroutine 이 아닌 class 안에 이 코드를 삽입해야 합니다.)

```
public void ProcessNetTutorialPlotData()
{
	// Create an auto contact plotting dialog
	Form2 MyForm = new Form2();

	// Open the dialog
	MyForm.ShowDialog();

	// If the user clicked OK:
	if (MyForm.DialogResult == System.Windows.Forms.DialogResult.OK)
	{
		// Get the TIME data
		double[] TIME = plotDocument.GetPlotData("4WD_Loader/TIME");

		// |||||||| Plot individual contact forces ||||||||

		// Active upper-left plot window
		plotDocument.ActivateView(0, 0);

		for (int bodyIndex = MyForm.BNumStart;
		bodyIndex <= MyForm.BNumEnd; bodyIndex++)
		{
			// Load up the contact name number array

			String[] contNum = {bodyIndex.ToString(), "", ""};

			if (MyForm.AddOffsetFlag)
			{
				contNum[1] = bodyIndex.ToString() + "a";
				contNum[2] = bodyIndex.ToString() + "b";
			}

			for (int contNumIndex = 0; contNumIndex < contNum.Length;
				contNumIndex++)
			{
				if (String.Compare(contNum[contNumIndex], "") != 0)
				{
					// Get the contact data for this segment
					double[] contact = plotDocument.GetPlotData(
						"4WD_Loader/Contact/Solid Contact/solidContact"
						+ contNum[contNumIndex] +
						"/FM_SolidContact");

					// Plot vs. TIME
					plotDocument.DrawPlot("Contact"
						+ contNum[contNumIndex], TIME, contact);
				}
			}
		}
	}
}
```

첫 부분의 두 개의 Command 는 사용자에게 새로운 Form2 윈도우를 생성하여 보여줍니다.

```
// Create an auto contact plotting dialog
Form2 MyForm = new Form2();

// Open the dialog
MyForm.ShowDialog();
```

If 선언문은 사용자가 OK 버튼을 클릭할 것인지 말 것인지에 따라서 코드를 실행할지 안 할지를 결정합니다.

```
// If the user clicked OK:
if (MyForm.DialogResult == System.Windows.Forms.DialogResult.OK
{
```

다음 코드는 임폴트 된 파일에서 TIME 에 대한 Plot 데이터를 얻게 되며, 새로운 Array 에 그 데이터를 저장합니다.

```
// Get the TIME data
double[] TIME = plotDocument.GetPlotData("4WD_Loader/TIME");
```

다음 명령은 왼쪽 상단 Plotting 윈도우를 활성화시킵니다.

```
// Active upper-left plot window
plotDocument.ActivateView(0, 0);
```

다음 코드는 Plot 에 대해서 데이터의 부분적인 이름을 가지고 있는 String 의 Array 를 로드 합니다. 그 Array 는 다이얼로그에 입력한 값에 따라서 로드 되며, 예를 들어 사용자가 Starting Contact 을 20 으로 설정하고 Offset Contacts Used 체크박스를 선택했다면, 첫 번째 Loop 동안

```
for (int bodyIndex = MyForm.BNumStart;
        bodyIndex <= MyForm.BNumEnd; bodyIndex++)
{
        // Load up the contact name number array

        String[] contNum = {bodyIndex.ToString(), "", ""};

        if (MyForm.AddOffsetFlag)
        {
                contNum[1] = bodyIndex.ToString() + "a";
                contNum[2] = bodyIndex.ToString() + "b";
        }

        for (int contNumIndex = 0; contNumIndex < contNum.Length; contNumIndex++)
        {
                if (String.Compare(contNum[contNumIndex], "") != 0)
                {
```

그 Array 는 {“20”, “20a”, “20b”}이라는 String 을 포함하게 될 것입니다.

다음의 이러한 명령은 데이터의 이름에 의해서 Plot 데이터를 검색합니다. 그 이름은 Plotting Database window 에 보여지는 것이며, separator 문자인 "/"을 사용하여 나타냅니다.

마지막으로, 다음 명령은 Plot 윈도우에서 데이터를 그리게 합니다.

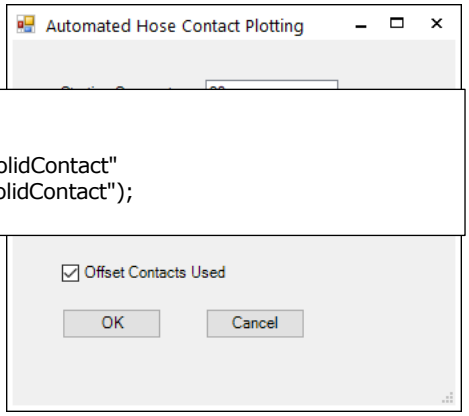
```
// Plot vs. TIME
plotDocument.DrawPlot("Contact"
+ contNum[contNumIndex], TIME, contact);
```

2. 파일을 저장합니다.
3. **Build** 메뉴에서 **Build ProcessNet** 을 선택합니다.

플로팅 하면서 수정된 **Subroutine** 을 테스트하기**:**

1. 현재의 로더 모델을 로드하면서 RecurDyn 모델링

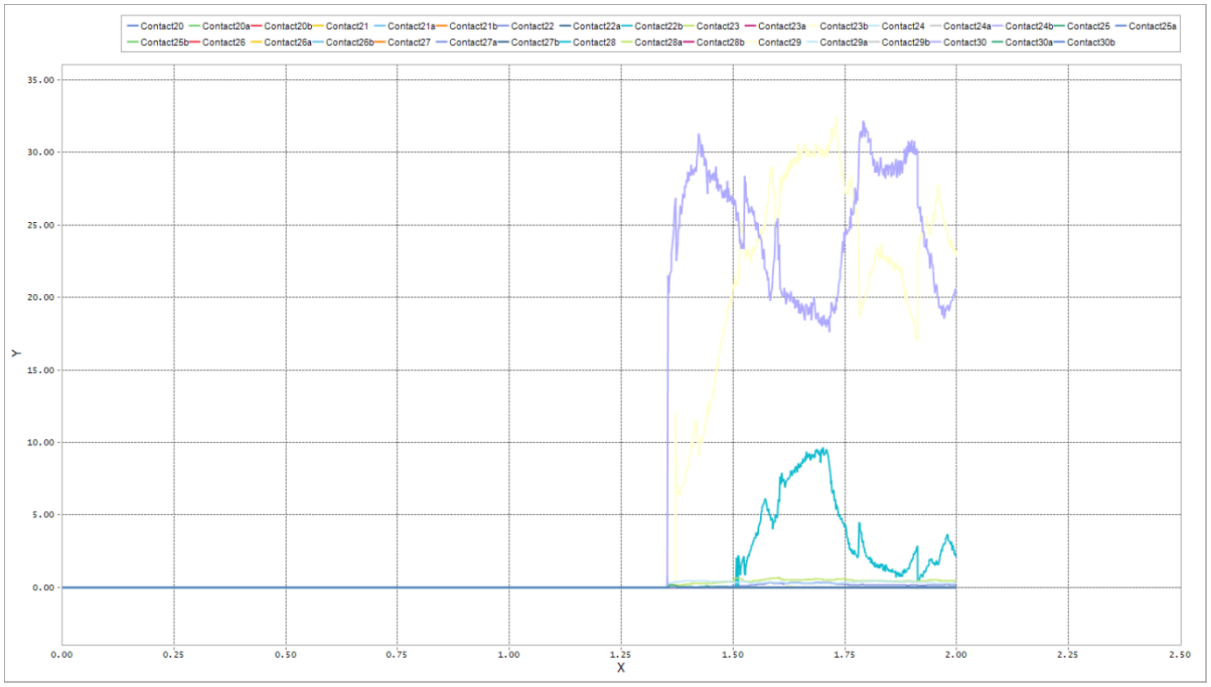
```
// Get the contact data for this segment
double[] contact = plotDocument.GetPlotData(
        "4WD_Loader/Contact/Solid Contact/solidContact"
        + contNum[contNumIndex] + "/FM_SolidContact");
```



윈도우로 돌아옵니다. 그리고, Plotting 창을 엽니다.

2. Customize 탭의 ProcessNet 그룹에서 Run 을 클릭합니다.
3. 해당 목록에서 ProcessNetTutorialPlotData 를 선택합니다.
4. **Run** 을 선택합니다.
5. Dialog2 다이얼로그 윈도우에서 기본적인 세그먼트 수들을 수락하고, Offset Contacts Used 를 체크합니다.

   다음과 같은 Plot 이 보여야 합니다. (RecurDyn Plot 창의 사이즈를 조절하십시오.)



Note: Home 탭 Draw 그룹 Draw Setting 에서 Auto Fit after drawing curve 를 True 로 설정해 줘야 위와 같은 결과가 나옵니다.

위에서 33 개의 모든 Contact 은 Plot 으로 그려지지만, 호스들 사이의 Contact 이 각 호스의 몇 개의 세그먼트들을 제한하기 때문에 곡선들의 일부는 0 이 아닌 값을 가지고 있습니다. 또한, 마지막 3 번째 시뮬레이션에서 모든 Contact 이 생기지만, 그 Plot 은 시뮬레이션의 전체 타임라인을 보여줍니다**.** (이것은 제거해야 합니다.)

Contact Force Plot 의 개선

0 이 아닌 값을 가진 Contact 의 Plot 에만 초점을 두어 Plot 을 개선해 보겠습니다. X 축 범위에 있는 타임라인을 줄이려면 0 이 아닌 값을 가진 Contact 만을 포함시켜야 합니다.

이제, 플로팅 된 Contact 중에 0 이 아닌 값으로 된 Contact 들만 가지고 작업을 시작할 것입니다. 타임라인을 줄이기 위한 몇 가지 로직을 먼저 소개를 하고서, X 축의 서식 설정에 대해서 설명하겠습니다.

0 이 아닌 값을 가진 **Contact Force** 들에 대해서만 **Plot** 으로 그리기**:**

1. 코드를 다음과 같이 변경합니다.

```
public void ProcessNetTutorialPlotData()
{
	// Create an auto contact plotting dialog
	Form2 MyForm = new Form2();

	// Open the dialog
	MyForm.ShowDialog();

	// If the user clicked OK:
	if (MyForm.DialogResult == System.Windows.Forms.DialogResult.OK
	{
		// Get the TIME data
		double[] TIME = plotDocument.GetPlotData("4WD_Loader/TIME");

		// Initialize variables for X-axis limits
		double timeAtFirstContact = TIME[TIME.Length - 1];
		double timeAtLastContact = 0;

		// |||||||| Plot individual contact forces ||||||||

		// Active upper-left plot window
		plotDocument.ActivateView(0, 0);

		for (int bodyIndex = MyForm.BNumStart;
		bodyIndex <= MyForm.BNumEnd; bodyIndex++)
		{
			// Load up the contact name number array
			String[] contNum = {bodyIndex.ToString(), "", ""};

			if (MyForm.AddOffsetFlag)
			{
				contNum[1] = bodyIndex.ToString() + "a";
				contNum[2] = bodyIndex.ToString() + "b";
			}

			for (int contNumIndex = 0; contNumIndex < contNum.Length;
				contNumIndex++)
```

```
                        {
                                if (String.Compare(contNum[contNumIndex], "") != 0)
                                {
                                        // Get the contact data for this segment
                                        double[] contact = plotDocument.GetPlotData(
                                        "4WD_Loader/Contact/Solid Contact/solidContact"
                                        + contNum[contNumIndex] + "/FM_SolidContact");

                                        // Plot vs. TIME
                                        plotDocument.DrawPlot("Contact"
                                        + contNum[contNumIndex], TIME, contact);

                                        // Check for non-zero contact data, and determine
                                        // time at first contact
                                        int j;
                                        bool madeContact = false;
                                        for (j = 0; j < contact.Length; j++)
                                        {
                                                if (contact[j] > 0)
                                                {
                                                        madeContact = true;
                                                        if (TIME[j] < timeAtFirstContact)
                                                        timeAtFirstContact = TIME[j];
                                                        break;
                                                }
                                        }

                                // Determine time at last contact
                                j = contact.Length - 1;
                                while (j > 0)
                                {
                                        if (contact[j] > 0)
                                        {
                                                if (TIME[j] > timeAtLastContact)
                                                timeAtLastContact = TIME[j];
                                                break;
                                        }
                                        j--;
                                }

                                // If non-zero contact data found, then plot vs. TIME
                                if (madeContact)
                                      plotDocument.DrawPlot("Contact"
                                        + contNum[contNumIndex], TIME, contact);
                        }
                }
        }
}
}
```

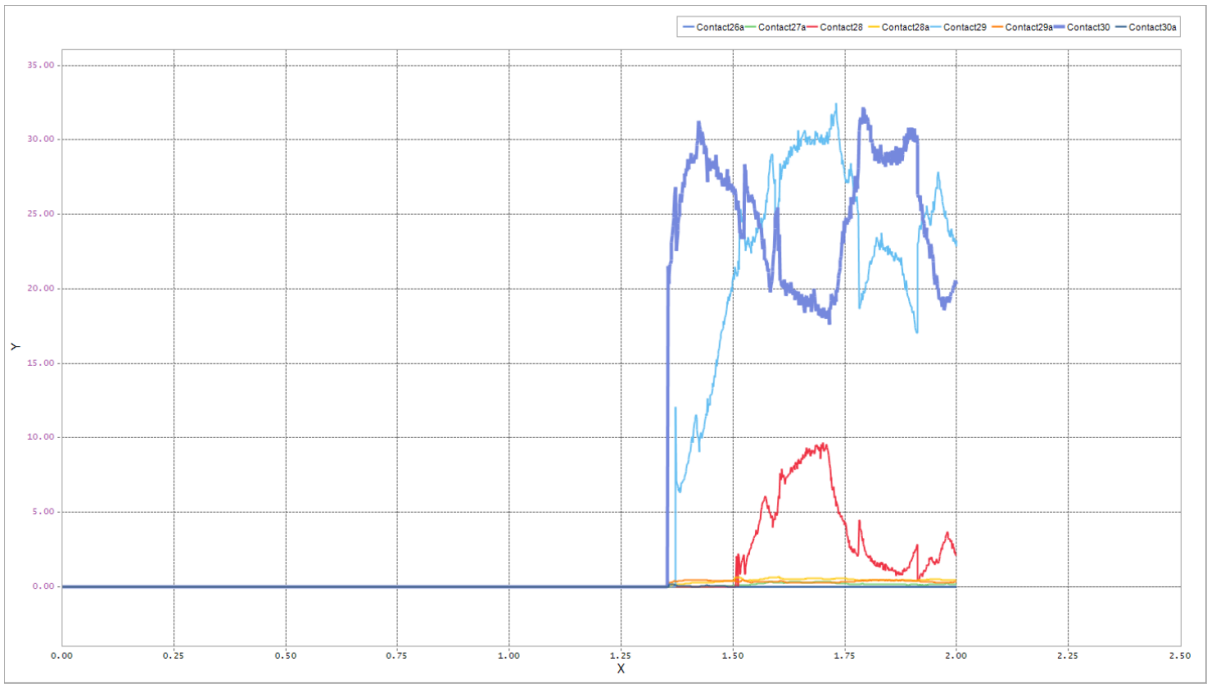
이제, 코드는 하나의 Array 의 형태로 된 데이터에 의한 로직을 포함하게 되어, 0 이 아닌 값을 찾게 됩니다. 또한, 이 코드는 0 이 아닌 값을 가진 Contact 에 대해 처음과 마지막 시간을 기록합니다.

3. 파일을 저장합니다.
4. **Build** 메뉴에서 **Build ProcessNet** 을 선택합니다.

0 이 아닌 값을 가진 **Contact Force** 의 **Plot** 을 테스트하기**:**

- 새로운 Plot 윈도우에서 새로운 ProcessNet 매크로를 테스트하기 위해서 이전과 같은 실행 단계를 따릅니다. (새로운 Plot 윈도우가 나타날 것입니다.)

다음과 같은 Plot 이 보여져야 합니다.



이제. 시뮬레이션을 실행하는 동안 0 이 아닌 값을 가진 Contact 을 볼 수 있습니다. 33 개의 Contact 중에 8 개만이 0 이 아닌 값을 가지고 있습니다.

Plot 서식의 개선

실행이 완료된 마지막 Plot 은 이전 버전보다 더욱 명확하게 보여집니다. 그러나, 여전히 개선해야 할 여지가 있습니다. Plot 의 타이틀과 축 라벨, 축 타이틀은 지정되지 않았으며, 시뮬레이션 시점은 여전히 알고 싶은 부분에 초점이 맞춰져 있지 않습니다. 이제, **ProcessNet** 기능들을 이용하여 몇 개의 Plot 에 대해서 그 서식을 수정해보겠습니다.

Plot 의 서식 개선하기**:**

1. 다음과 같이 코드를 변경합니다.

```
#region namespace
using System;
using Microsoft.VisualBasic;
using System.Windows.Forms; //IWin32Window
using System.IO;

using FunctionBay.RecurDyn.ProcessNet;
//For C#
using FunctionBay.RecurDyn.ProcessNet.Chart;
//using FunctionBay.RecurDyn.ProcessNet.MTT2D;
//using FunctionBay.RecurDyn.ProcessNet.FFlex;
//using FunctionBay.RecurDyn.ProcessNet.RFlex;
//using FunctionBay.RecurDyn.ProcessNet.Tire;
```

```
                                        // If non-zero contact data found, then plot vs.
                                        // TIME
                                        if (madeContact) plotDocument.DrawPlot("Contact"
                                + contNum[contNumIndex], TIME, contact);
                                 }
                        }
                }

                // Get Chart object and format

                IChart Chart1 = plotDocument.ActiveChartControl;
                Chart1.Title.Text = "Hose Contact Force";
                Chart1.AxisX.Title.Text = "Time (sec)";
                Chart1.AxisX.Min = 0.1 * Math.Truncate(timeAtFirstContact * 10);
                Chart1.AxisX.Max = 0.1 * Math.Ceiling(timeAtLastContact * 10);
                Chart1.AxisX.LabelsFormat.Decimals = 1;
                Chart1.AxisY.LabelsFormat.Decimals = 1;
                Chart1.AxisY.Title.Text = "Contact Force (N)";
                Chart1.LegendBox.Alignment = LegendBoxAlignment.LegendBoxAlignment_Far;
}
}
```

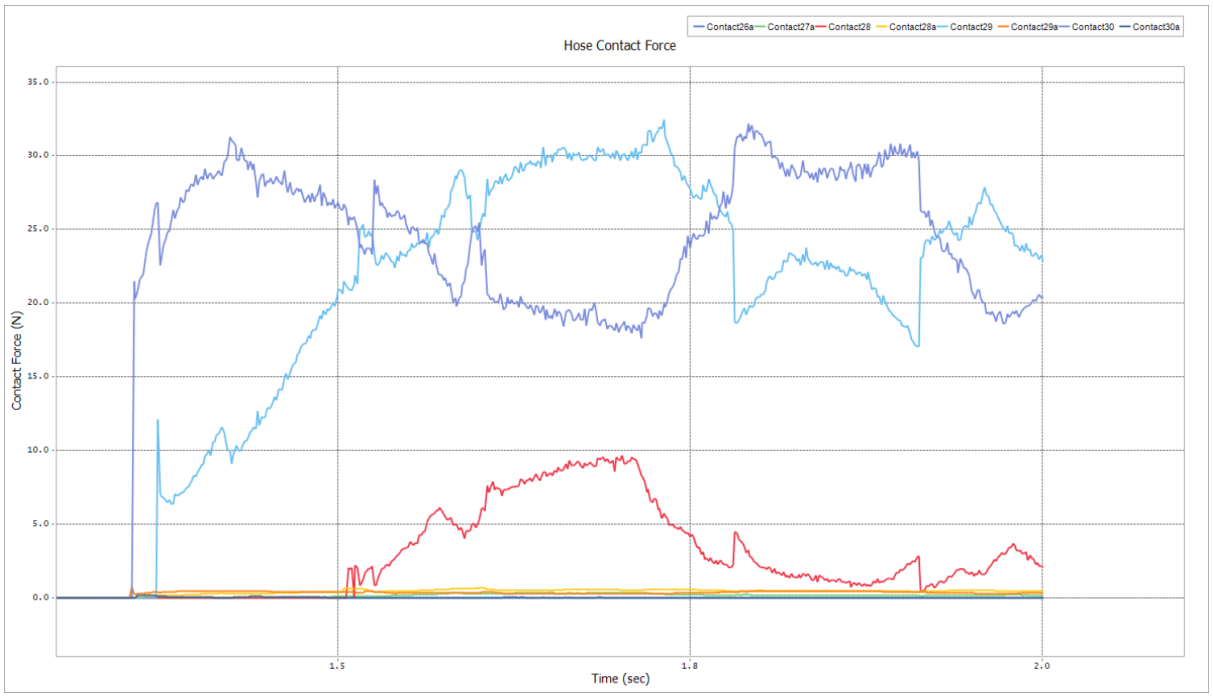
X 축 정의는 이전 섹션에서 계산된 **timeAtFirstContact** 과 **timeAtLastContact** 변수들을 이용하여 **Chart1.AxisX.Min 과 Chart1.AxisX.Max** 로 제한해야 한다는 것에 유의하십시오.

2. 파일을 저장합니다.
3. **Build** 메뉴에서 **Build ProcessNet** 을 선택합니다.

Plot 서식 설정에 대해 테스트하기**:**

- 새로운 Plot 윈도우에서 새로운 ProcessNet 매크로를 테스트하기 위해서 이전과 같은 실행 단계를 따릅니다. (새로운 Plot 윈도우가 나타날 것입니다.)

다음과 같은 Plot 이 보여야 합니다.



이제, Plot 에 대한 서식 설정이 완료 되어 X 축에 범위가 지정되었고 0 이 아닌 값을 가진 Contact 에 대한 세부 사항을 볼 수 있습니다.

모든 X, Y 와 Z Contact Force 에 대한 플로팅

X, Y 와 Z 구성요소로 나누어진 두 호스 사이에서 모든 Contact Force 를 보기 원한다고 가정해봅시다. 이 Plot 을 위한 데이터는 Plot 파일에 존재하는 데이터를 사용하여 계산되어야 할 것입니다. **ProcessNet** 에서 Plot Data Array 에 저장될 수 있으며, 그것에 대해 수리적인 연산들을 수행할 수 있습니다. 이러한 다음 Plot 에 대하여, 모든 Contact 들의 X, Y 와 Z 구성 요소의 합을 생성하기 위하여 Array 데이터를 같이 추가할 것입니다.

새로운 매크로를 생성하기 위한 대부분의 명령들은 이전 Plot 과정에서 이미 배웠기 때문에 코드의 하나의 큰 블록을 복사하고 튜토리얼의 이 부분에 해당되는 코드에 삽입할 것입니다. 그러나, 추가적인 데이터 처리 과정은 Array 와 함께 합해진다는 것에 유의하십시오.

두 번째 **Plot** 추가하기**:**

1. 있는 코드를 Subroutine 의 끝 근처 If 선언문 안에 삽입합니다.

```
// Get Chart object and format
.
.
.
Chart1.AxisY.Title.Text = "Contact Force (N)";
Chart1.LegendBox.DockedPosition = DockedPositionType.DockedPositionType_Right;

// |||||||| Plot sums of X, Y, and Z components ||||||||

// Activate the lower-left plot window
plotDocument.ActivateView(1, 0);

double[] xSum = new double[TIME.Length];
double[] ySum = new double[TIME.Length];
double[] zSum = new double[TIME.Length];

for (int bodyIndex = MyForm.BNumStart;  bodyIndex <= MyForm.BNumEnd; bodyIndex++)
{
        // Load up the contact name number array

        String[] contNum = {bodyIndex.ToString(), "", ""};
        if (MyForm.AddOffsetFlag)
        {
                contNum[1] = bodyIndex.ToString() + "a";
                contNum[2] = bodyIndex.ToString() + "b";
      }
        for (int contNumIndex = 0; contNumIndex < contNum.Length; contNumIndex++)
        {
                if (String.Compare(contNum[contNumIndex], "") != 0)
                {
                        // Get the contact data for this segment
                        double[] contactX = plotDocument.GetPlotData(
                        "4WD_Loader/Contact/Solid Contact/solidContact"
                        + contNum[contNumIndex] + "/FX_SolidContact");
                        double[] contactY = plotDocument.GetPlotData(
                        "4WD_Loader/Contact/Solid Contact/solidContact"
                        + contNum[contNumIndex] + "/FY_SolidContact");
                        double[] contactZ = plotDocument.GetPlotData(
                        "4WD_Loader/Contact/Solid Contact/solidContact"
                        + contNum[contNumIndex] + "/FZ_SolidContact");

                        if (contNumIndex == MyForm.BNumStart)
                        {
                                // If looping through the first contact,
                                // initialize the sum arrays
                                xSum = contactX;
                                ySum = contactY;
                                zSum = contactZ;
                        }
```

```
                                else
                                {
                                        // Else, add this contact's array data to the
                                        // running total
                                        for (int j = 0; j < TIME.Length; j++)
                                        {
                                                xSum[j] = xSum[j] + contactX[j];
                                                ySum[j] = ySum[j] + contactY[j];
                                                zSum[j] = zSum[j] + contactZ[j];
                                        }
                                }
                        }
                }
        }

// Plot vs. TIME
plotDocument.DrawPlot("X Sum", TIME, xSum);
plotDocument.DrawPlot("Y Sum", TIME, ySum);
plotDocument.DrawPlot("Z Sum", TIME, zSum);

// Get Chart object and format
IChart Chart2 = plotDocument.ActiveChartControl;
Chart2.Title.Text = "Total Hose-to-Hose Contact Force";
Chart2.AxisX.Title.Text = "Time (sec)";
Chart2.AxisX.Min = 0.1 * Math.Truncate(timeAtFirstContact * 10);
Chart2.AxisX.Max = 0.1 * Math.Ceiling(timeAtLastContact * 10);
Chart2.AxisX.LabelsFormat.Decimals = 1;
Chart2.AxisY.LabelsFormat.Decimals = 1;
Chart2.AxisY.Title.Text = "Contact Force (N)";
Chart2.LegendBox.Alignment = LegendBoxAlignment.LegendBoxAlignment_Far;

}
}
```

2. 파일을 저장합니다.

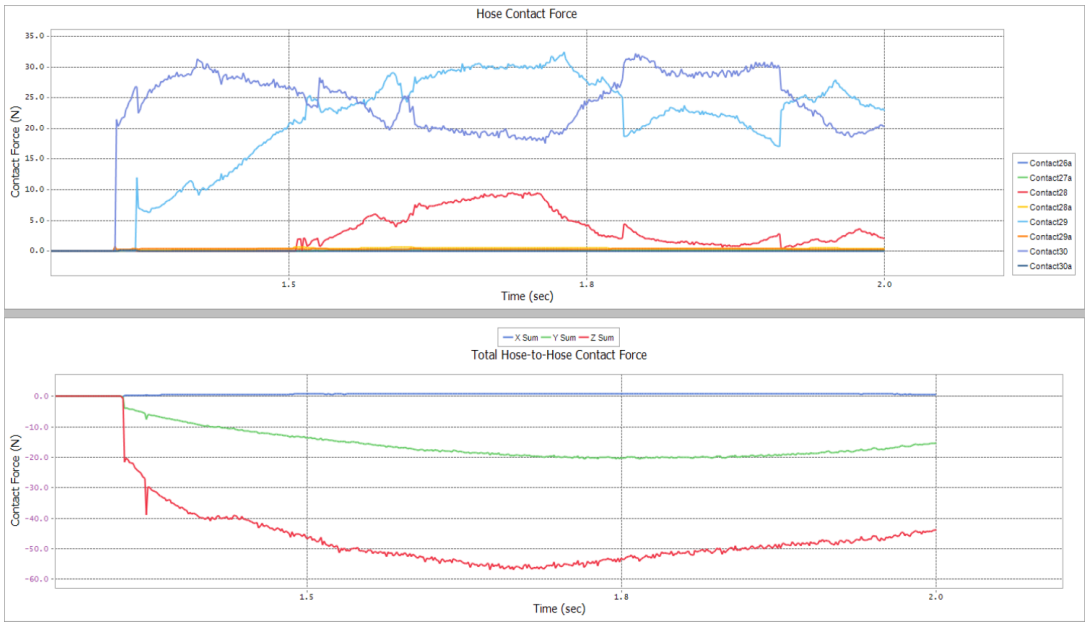
3. **Build** 메뉴에서, **Build ProcessNet** 을 선택합니다.

두 번째 **Plot** 테스트하기**:**

두 번째 Plot 은 왼쪽 하단 윈도우에 보일 것입니다. (이와 반대로, 첫 번째 Plot 은 오른쪽 상단 윈도우에 보였습니다.) 그래서, 두 개의 윈도우 모두 에러가 없이 보여야 합니다.

- Home 탭의 Windows 메뉴에서 Show Left Windows 를 선택한다음에 새로운 Plot 윈도우에서 새로운 ProcessNet 매크로를 테스트 합니다.

다음과 같이 두 개의 Plot 이 보여야 합니다.



Thank you for participating in this tutorial!