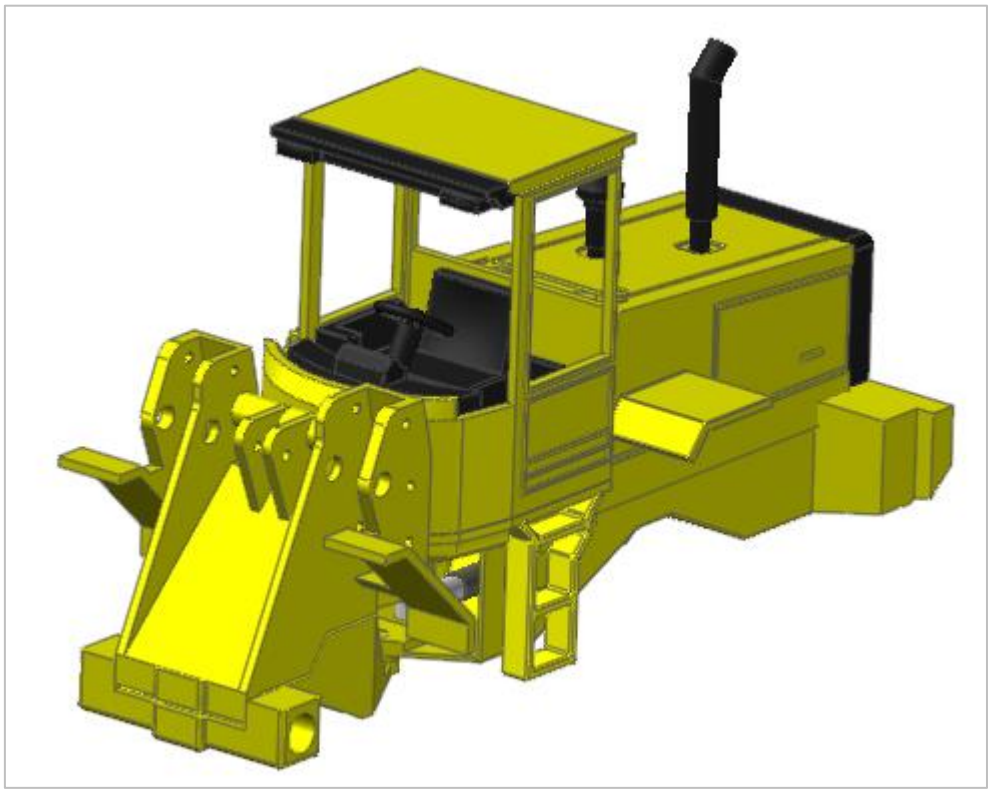
RECURDYN

ProcessNet 四驱装载机教程(General)



FunctionBay

Registered Trademarks of FunctionBay, Inc. or Subsidiary

RecurDyn™ is a registered trademark of FunctionBay, Inc.

RecurDyn™/SOLVER, *RecurDyn*™/MODELER, *RecurDyn*™/PROCESSNET, *RecurDyn*™/AUTODESIGN, *RecurDyn*™/COLINK, *RecurDyn*™/DURABILITY, *RecurDyn*™/FFLEX, *RecurDyn*™/RFLEX, *RecurDyn*™/RFLEXGEN, *RecurDyn*™/LINEAR, *RecurDyn*™/EHD(Styer), *RecurDyn*™/ECFD_EHD, *RecurDyn*™/CONTROL, *RecurDyn*™/MESHINTERFACE, *RecurDyn*™/PARTICLES, *RecurDyn*™/PARTICLEWORKS, *RecurDyn*™/ETEMPLATE, *RecurDyn*™/BEARING, *RecurDyn*™/SPRING, *RecurDyn*™/TIRE, *RecurDyn*™/TRACK_HM, *RecurDyn*™/TRACK_LM, *RecurDyn*™/CHAIN, *RecurDyn*™/MTT2D, *RecurDyn*™/MTT3D, *RecurDyn*™/BELT, *RecurDyn*™/R2R2D, *RecurDyn*™/HAT, *RecurDyn*™/曲柄, *RecurDyn*™/PISTON, *RecurDyn*™/VALVE, *RecurDyn*™/TIMINGCHAIN, *RecurDyn*™/ENGINE, *RecurDyn*™/GEAR are trademarks of FunctionBay, Inc.

Third-Party Trademarks

Windows and Windows NT are registered trademarks of Microsoft Corporation.

ProENGINEER and ProMECHANICA are registered trademarks of PTC Corp. Unigraphics and I-DEAS are registered trademark of UGS Corp. SolidWorks is a registered trademark of SolidWorks Corp. AutoCAD is a registered trademark of Autodesk, Inc.

CADAM and CATIA are registered trademark of Dassault Systems. FLEX*lm* is a registered trademark of GLOBEtrotter Software, Inc. All other brand or product names are trademarks or registered trademarks of their respective holders.

Edition Note

These documents describe the release information of ***RecurDyn™*** V9R1.

目录

Chapter

1

预备工作

目的

本教程学习如何通过 **Microsoft Visual Studio**，使用 **ProcessNet**。本教程会略过教程所用模型的解释，因为学习者应该已经使用 **VSTA** 完成了 **ProcessNet** 教程。这四个处理是：

- 使用 **Microsoft Visual Studio** 中的 **ProcessNet**。
- 完成一系列接触的自动化创建。
- 创建一个自定义对话框，使用户可以控制接触。
- 自动检查接触力输出，并只为有非零输出的接触力作图。

读者

本教程适用于 **RecurDyn** 的中级用户（已经学习过如何创建图形、运动副和力元素）。所有新的任务都将会做仔细说明。

预备知识

在学习本教程前，应先完成 **RecurDyn** 教程中的 **ProcessNet** 四驱装载机教程(**VSTA**)。

步骤

本教程包含了以下步骤，完成每个步骤需要的时间如下表所示。

步骤	时间(分钟)
打开模型并初始化 **ProcessNet**	10
自动化定义接触	35
增加对话框和信息输出	20
自动化创建绘图	20
总计	85

预计完成的时间

本教程大约需 85 分钟来完成。

Chapter

2

打开模型并初始化 ProcessNet

任务目标

通过学习如何打开一个机械模型，为使用 **ProcessNet** 做准备，同时学习如何在 **Visual Studio** 中初始化 **ProcessNet**。

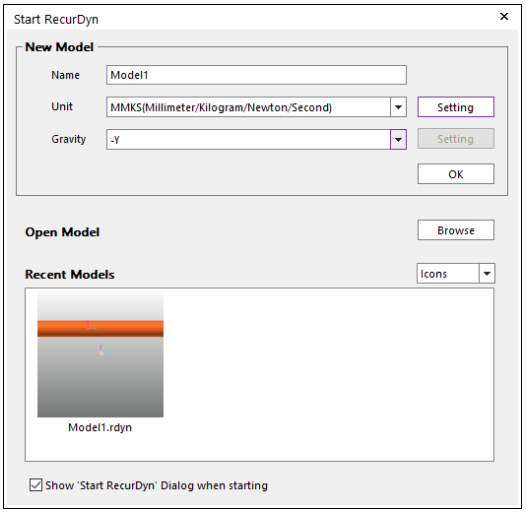
预计完成的时间

10 分钟

启动 RecurDyn

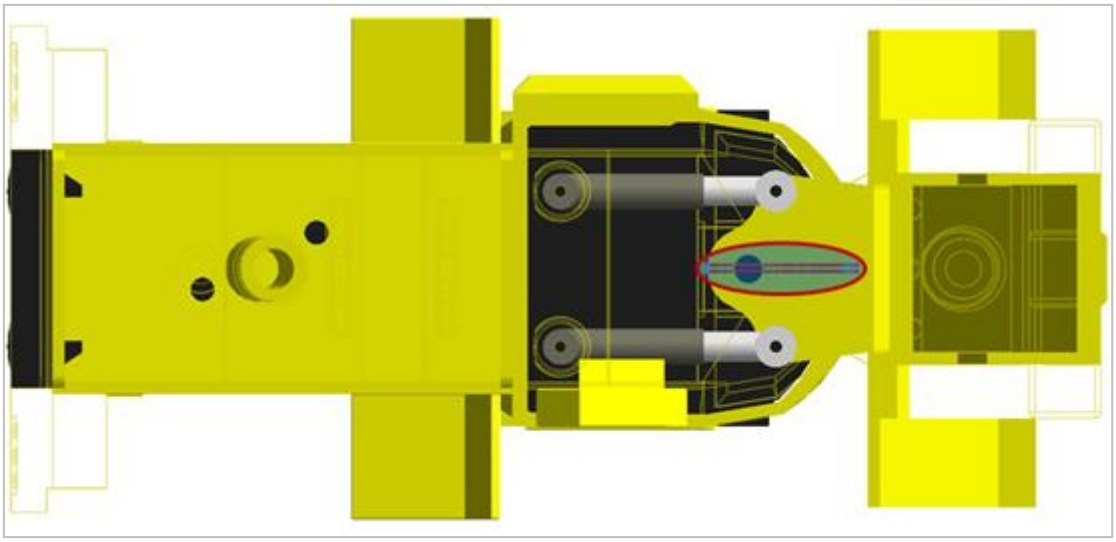
启动 RecurDyn 并打开初始模型：

1. 双击桌面上的 **RecurDyn** 图标。
2. **Start RecurDyn** 对话框弹出后，将其关闭。因为将使用一个已有的模型，而不是创建新模型。



3. 在 **Quick Access** 工具栏，点击 **Open**。
4. 选择文件 **4WD_Loader_Start.rdyn**。(文件路径：**<InstallDir>/Help/Tutorial/ProcessNet/General/4WDLoader**)。
5. 点击 **Open**。

模型会显示如下。



红线标注的部分就是将学习的液压软管。

提示：这个视图是在（  **Render Each Object**)查看模式下显示的，可以透过车辆底部可能会阻挡视线的部件看到软管。如果看不到软管，检查是否在 **Shaded** 查看模式下。如果是的话，返回 **Render Each Object** 查看模式。

保存初始模型：

1. 在 **File** 菜单中，点击 **Save As**。
2. 将模型保存在不同的目录下，因为无法在教程目录中运行仿真。

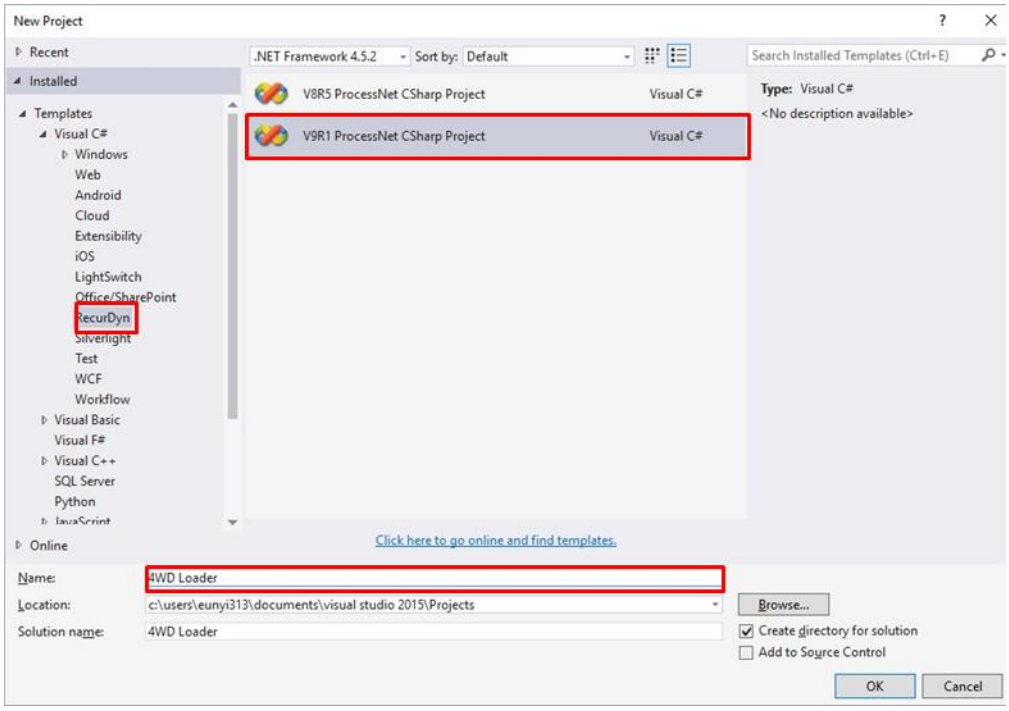
启动 ProcessNet

启动 ProcessNet:

ProcessNet General 并不提供集成开发环境(**IDE**)。因此需要使用 **IDE** 程序。如果没有这个程序，我们推荐使用 **Microsoft** 支持的 **Visual Studio Expression** 版本。

1. 执行 **Visual Studio** 程序。
   - 本教程中使用的是 **Visual Studio** 2015**Expression** 版本。
   - **RecurDyn** 不必像 **ProcessNet VSTA** 中一样初始化 **ProcessNet**。
2. 在 **File** 菜单中，选择 New，选择 **Project**。
3. **New Project** 对话窗口弹出后，选择与用户的 **RecurDyn** 版本相对应的模板。
   - 本教程以 **V**9R1 为基础，因此选择 **V**9R1。

注意：必须使用与用户的 **RecurDyn** 版本相兼容的 **ProcessNet** 项目。如果版本不正确，**ProcessNet** 可能无法正确执行操作。**New Project** 对话窗口中显示了与所安装 **RecurDyn** 版本相兼容的所有模板。



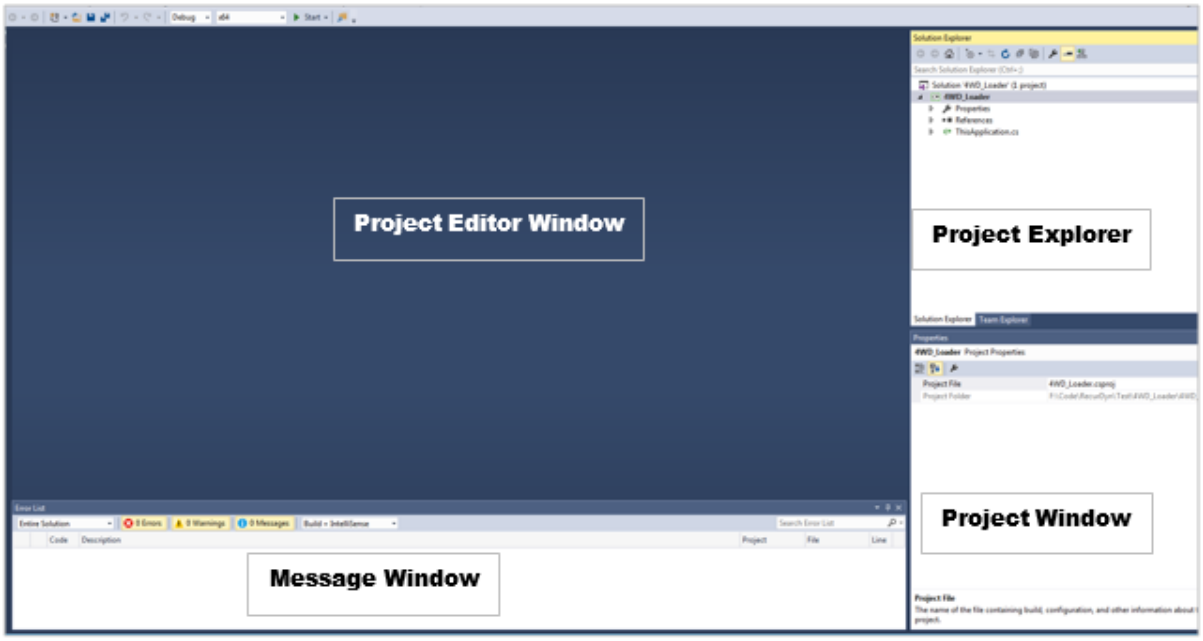
- 如果 **ProcessNet** 模板显示如上图，继续下面的步骤 4。
- 如果窗口中显示的内容不同，可能是因为模板未正确安装。如果是这样的话，在下述操作后再次启动 **Visual Studio**。

a. 复制“**<InstallDir>/Bin/Addin/ProcessNetManager**”目录下的 **V9R1ProcessNetCSharp Project.zip** 文件，并粘贴到“**C:\Users\<User Name>\Documents\Visual Studio** 2015**\Templates\ProjectTemplates**”目录下。

b. 返回 **Visual Studio** 应用。

c. 重复步骤 2。

4. 选择 **Templates->Visual C#->RecurDyn**。然后点击 **V9R1 ProcessNetCSharp Project** 图标。
5. **4D_Loader Project** 会按上图所示的 **Name**、**Location** 以及 **Solution name** 的设置创建。



它包含了四个区域：

- **Project Editor Window**–编写、编辑代码并设计界面对象。
- **Project Explorer**–提供项目和其文件的结构化视图。
- **Properties window**–查看并编辑在项目编辑器窗口或项目管理器中所选项目的属性。
- **Message window**–查看编写及运行代码时产生的信息，如代码中的错误。

现在可以开发的第一个 **ProcessNet** 应用了。

Chapter

3

自动化定义接触

任务目标

通过两个步骤创建出一个可以在两软管对应部位之间创建一系列接触的 **ProcessNet** 应用。将学习如何：

- 获得创建 **RecurDyn** 元素所必需的信息。
- 使用 **RecurDyn** 中的 **ProcessNet Help**。
- 在 **IDE** 中开发一个应用。

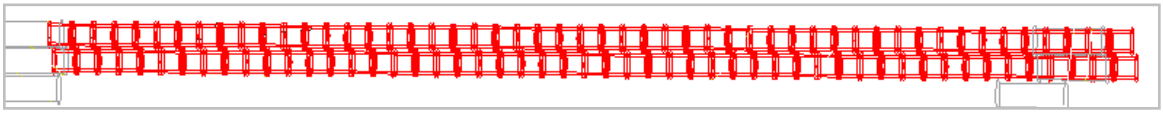
然后运行一次仿真，查看结果，更新应用，并再次运行仿真来观察改进后的结果。

预计完成的时间

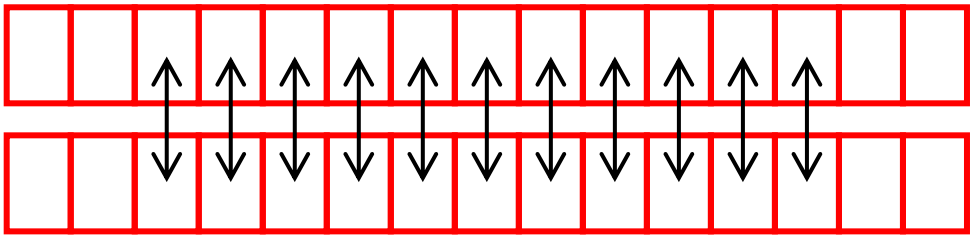
35 分钟

理解将要创建的接触

第一章解释过每个软管都包含了 51 个部分，如下图所示。两个软管在各自末端的安装位置分离。车辆的中心线在两根软管之间。当接合发生时，一根软管的拉伸程度比另一根软管更大。因为两根软管弯曲的程度不同，所以会在软管的中间发生接触。



应用第一个 **ProcessNet** 程序，将在软管的中间创建 11 个接触，如下图所示。



创建基础应用

下面开发创建软管接触的程序。

创建基础应用:

1. 在 Project Explorer 窗口，双击 ThisApplication.cs 打开它。文件里的内容出现在 Project 编辑窗口。
2. 复制下列代码，并将其插入到 **HelloProcessNet**()宏后，如下图所示。

```
publicvoid ProcessNetTutorialCreateSolidContact()
{
}
```

现在已经为宏创建了一个存根。如果现在需要编译代码，这个方法会显示出 **RecurDyn** 中可运行的宏列表，虽然可能不执行任何操作。

3. 接下来，将下述变量声明插入到刚创建的宏存根中 (在开始和结束的大括号之间)：

```
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
}
```

注意每个变量后面都跟了一条注释，前面有两条斜线//。在 **C#**中，所有在两条斜线后的字符都会被编译器忽略，可以用作代码的注释。

注意： C# 中的注释可以写在两个斜线后。

---

4. 在刚增加的变量声明后，加入如下循环：

```
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++)
    {
    }
}
```

- **for** 循环内部的代码会重复。第一次执行循环时，索引 **i** 等于 **BodyNumStart**。之后 **i** 会加 1 并执行下一次循环。这个步骤会一直重复直到条件 **i** ≤ **BodyNumEnd** 不成立。
- 注意 **i** 是在 **for** 循环语句内声明的，意味着它只对 **for** 循环内部的代码有效。

- 同时注意在 C#中，语句 i++表示 i 加 1，即在每次循环后 i = i+1。

5. 在刚加入的 for 循环语句中，插入如下代码：

```
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);
}
}
```

这段代码块的解释如下：

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

在上述代码中：

- i 是软管 1 部件的索引，j 是软管 2 部件的索引。
- 为确保部件一一对应，j 的值永远等于 i + body Interval 的值。
- 此外，注意 j 是在 for 循环语句内声明的，意味着它只对 for 循环内部的代码有效。

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

上述代码通过部件名称得到我们想要作为基础部件的部件：

- +字符用于连接两个字符串。
- ToString()方法用于将 i 的整数值转为字符串。
- 因此，GetEntity()方法通过搜索名称“BeamBody20”、“BeamBody21”等查找部件。

- 因为 GetEntity()方法可以返回任何类属元素，所以调用操作必须明确为 IBody，告诉编译器要返回什么类型的结果。

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

上述代码会通过部件名称“HollowCircularBeam1”得到基础部件的图形。这个图形会用于创建实体接触。

前面的两个步骤会对 action 部件再次重复。

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

最后，上述指令将创建实体接触，并命名为“solidContact20”（或“solidContact21”、“solidContact22”等）。该指令使用的是我们在前面步骤中所定义的 base 和 action 图形。

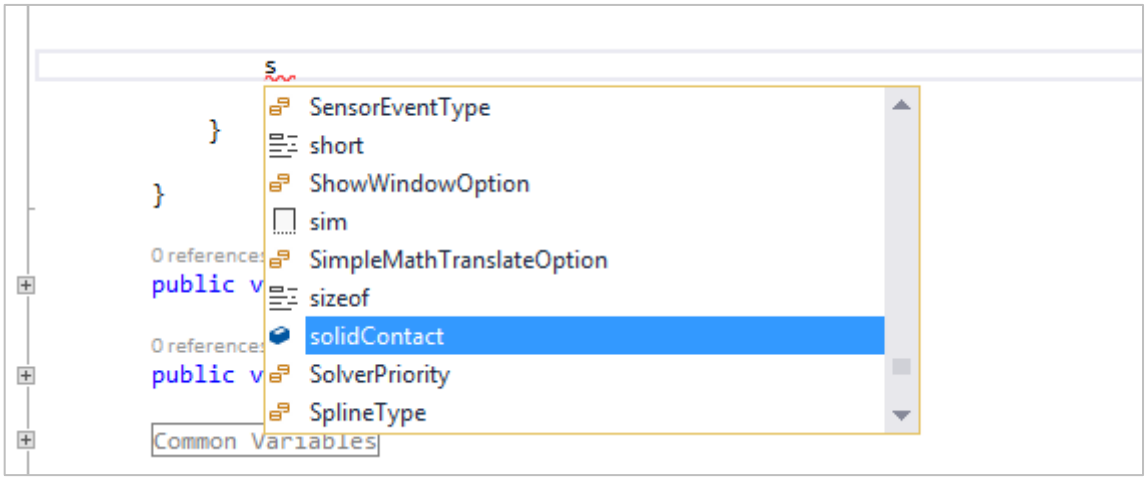
使用 IntelliSense 进行编码

到目前为止，可能一直都在直接复制粘贴本教程中的代码。然而，一旦开始编写自己的宏，就要手动输入代码了。ProcessNet IDE 中一个叫 IntelliSense 的功能可以帮助更快地编码，现在将学习如何在完成下述任务的同时，使用这个功能。

编译后，宏会创建一个默认属性的接触。但是实体接触的默认刚度值和阻尼值太高了。现在需要降低这些接触参数。

用 IntelliSense 修改接触的参数:

1. 在前面步骤中输入的最后一行代码后面，输入字符‘s’。会看到一个弹出列表，如下图所示：



这个列表包含了可能输入的所有有效字符，包括这段代码中可用的变量名、可以调用的方法等等。

2. 在 IntelliSense 列表中，定位 solidContact。

3. 通过双击或按 **Enter** 键来选中 **solidContact**。
4. 接下来，输入小数分隔符(.)。
5. 在 **IntelliSense** 列表中选择 **ContactProperty**。
6. 继续这个步骤，直到输入：

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

7. 重复这个步骤，输入下一行：

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

最后，整个代码会显示如下：

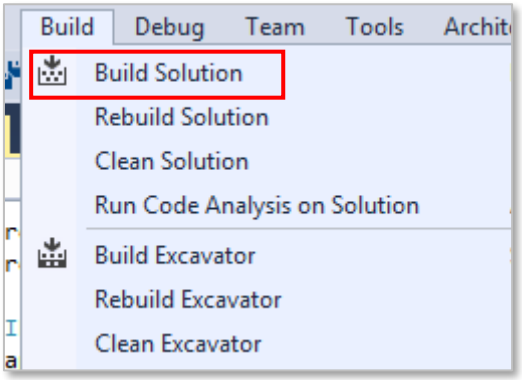
```
Public void ProcessNetTutorialCreateSolidContact()
{
int BodyNumStart = 20;  // Start creating contacts with body 20
int BodyNumEnd = 30;  // Continue until body 30
int BodyInterval = 51;  // Interval between body number on hose 1
// and corresponding body's number on
// hose 2 is 51

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 窗口底部的 **Error List** 窗口中，应该没有任何错误或警告。

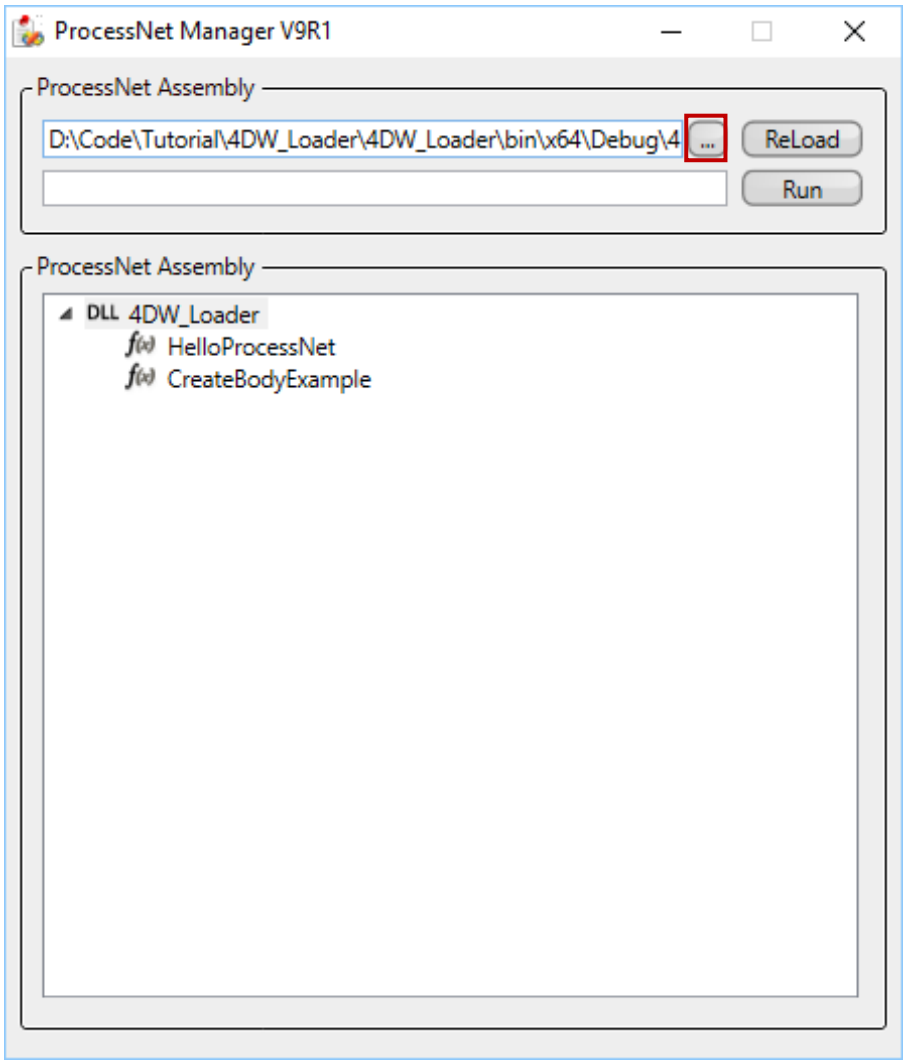
建立并运行宏

建立并运行宏:

1. 如果有任何错误或警告，检查列表并做出相应修正。
2. 在 **Build** 菜单中选择 **Build Solution**。



3. 返回 **RecurDyn**，并在 **Customize** 标签的 **ProcessNet(General)**组中，点击 **Run**。。
4. 载入之前在 **ProcessNet Manager** 窗口中建立的 **DLL** 文件。



5. 在宏列表中，选择 **ProcessNetTutorialCreateSolidContact**。

   宏的名称会显示在 **Run** 按钮旁边的栏中。

6. 点击 **Run**。

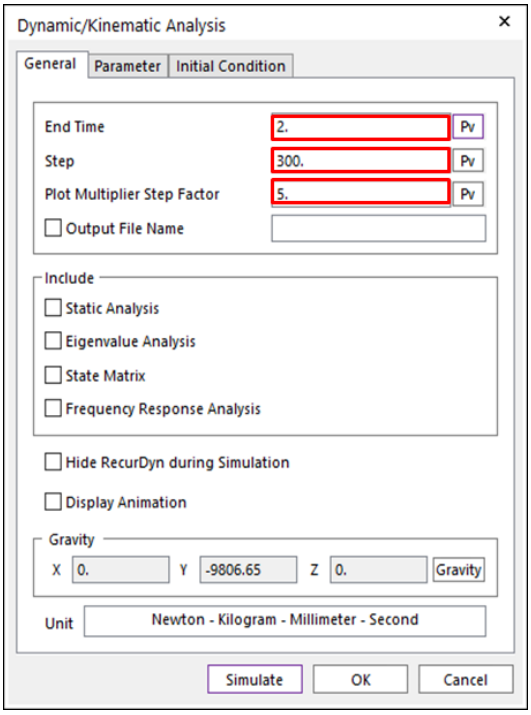
此时，模型中增加了 11 个接触。

运行仿真

现在可以运行一次仿真。

运行一次仿真：

1. 在 **Analysis** 标签的 **SimulationType** 组中，点击 **Dynamic/Kinematic**。
2. 设置仿真运行 300 步，2.0 秒。并将 **Plot Multiplier Step Factor** 设为 5，如右图所示。
3. 点击 **Simulate**。仿真会运行 3-4 分钟，具体视电脑的速度而定。



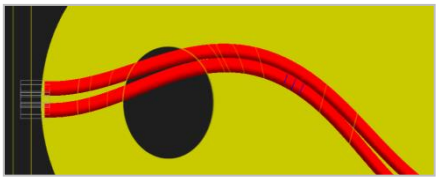
查看结果

查看结果：

1. 将模型视角调整为从车辆底部观察，同时打开 **Render Each shading** 模式。

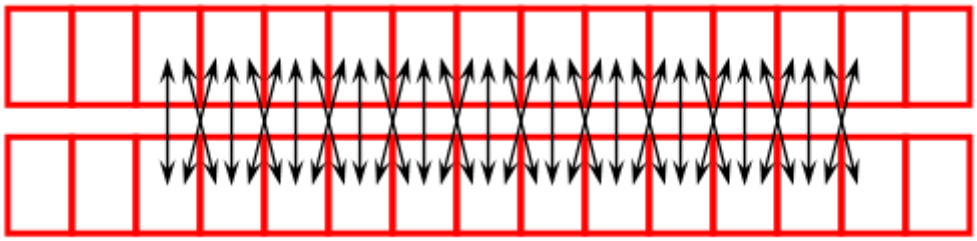
2. 在 **Simulation** 工具栏的 **Animation Control** 中，点击 **Play** 按钮。

   在仿真的最后两根软管都变形，如右图所示。



3. 打开所有接触的作用力显示：
   - 在数据库窗口选择第一个实体接触。
   - 长按 **Shift** 键，同时在数据库窗口选择最后一个实体接触。
   - 右键点击，并选择 **Property**。
   - 在弹出的对话框中点击 **Solid** 标签。在对话框底部，用下拉菜单将 **Force Display** 设为 **Action**。
   - 点击 **OK**。
4. 再次播放动画，将看到力出现了轻微失真。

   在查看运动时，可以看到软管相对滑动。对应部位间的接触可能还不够，可能需要相邻部位间的接触，如下图所示。



添加额外接触

添加额外接触:

1. 做代码修改。

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

2. 将代码修改为如下所示：

```
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;
    }
}
```

这里，代码在每次循环中都多创建了两个接触。软管 1 上第 **i**+1 个部位与软管 2 上第 **j** 个部位之间的接触以后缀“**a**”来命名(如“**solidContact**20**a**”)。类似的，软管 1 上第 **i** 个部位与软管 2 上第 **j**+1 个部位之间的接触以后缀“**b**”来命名(如“**solidContact**20**b**”)。

注意：如果已经存在相同名称的接触，接触不会创建成功。

- 可能已经注意到，创建这些额外接触的语句更加简洁和高效。例如，得到 **action** 部件和图形的操作都是在 **CreateContactSolidContact**()方法内完成的，但是用于存储各数值的临时变量在先前分别被声明。本语句提高了编程的效率，但是需要确保方法的返回对象用 **as** 语句规定 (如 **as IBody** 或 **as IGeometry**)。

重复建立、仿真和查看过程

重复前面的步骤，查看所做的更改对模型的影响。

重复处理过程:

1. 确保 **IDE** 窗口底部的 **Error List** 窗口中没有任何错误或警告。如果有的话，检查列表并作出相应修正。在 **Build** 菜单中，选择 **Build Solution**。
2. 返回 **RecurDyn**，删除所有在之前仿真中定义的接触。
3. 

在 **Customize** 菜单的 **ProcessNet(General)**组中，选择 **Run**。
4. 点击 **Run**。

   模型中增加了 33 个接触。
5. 用与之前运行相同的参数再次运行仿真，增加了接触后仿真大约需要运行 4-5 分钟。
6. 用与前面步骤相同的操作打开所有接触的 **action** 作用力显示。
7. 在 **Simulation** 工具栏中的动画控制中，点击 **Play**  按钮，并再次在 **Render Each shading**  模式开启的状态下从车辆底部视角观看。

   此时，接触力变得更平滑了。

Chapter

4

增加对话和信息输出

本章创建一个对话窗口，允许用户控制创建软管分段间的接触。包括设计对话框的布局，以及在现有子程序中增加代码来调用新对话。

任务目标

学习如何通过创建允许用户控制软管间接触的对话窗口，提高应用的灵活度。同时，学习如何在 **RecurDyn** 信息窗口中，对用户显示信息。

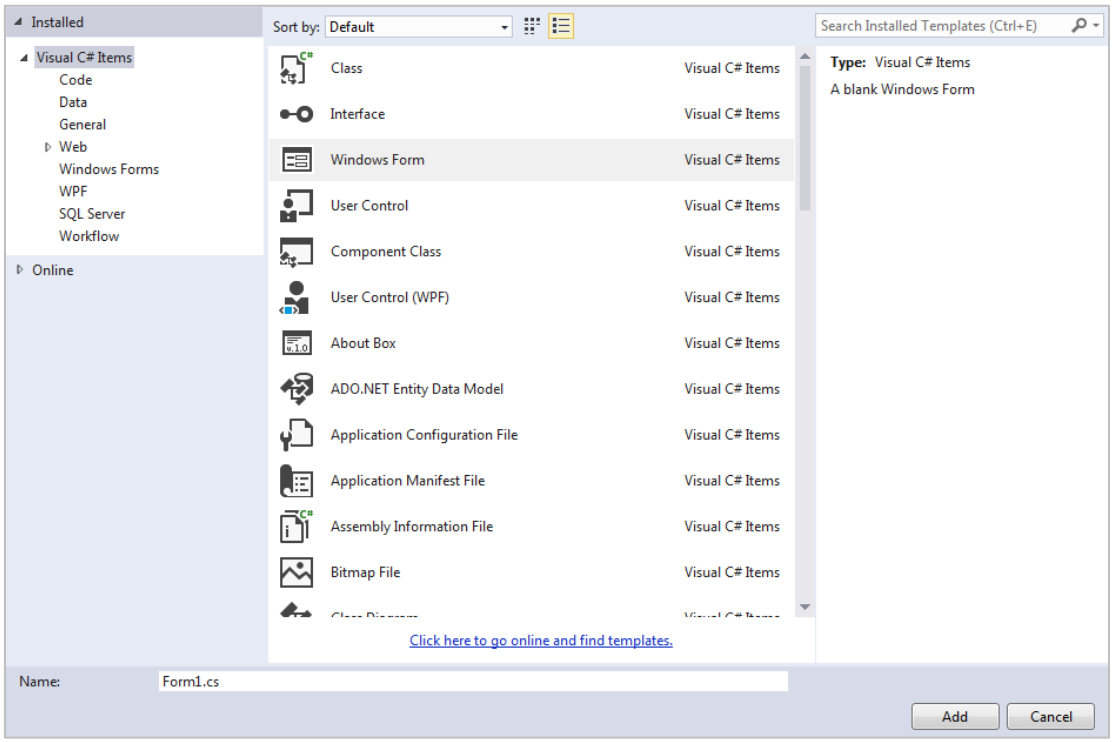
预计完成的时间

20 分钟

设计一个对话

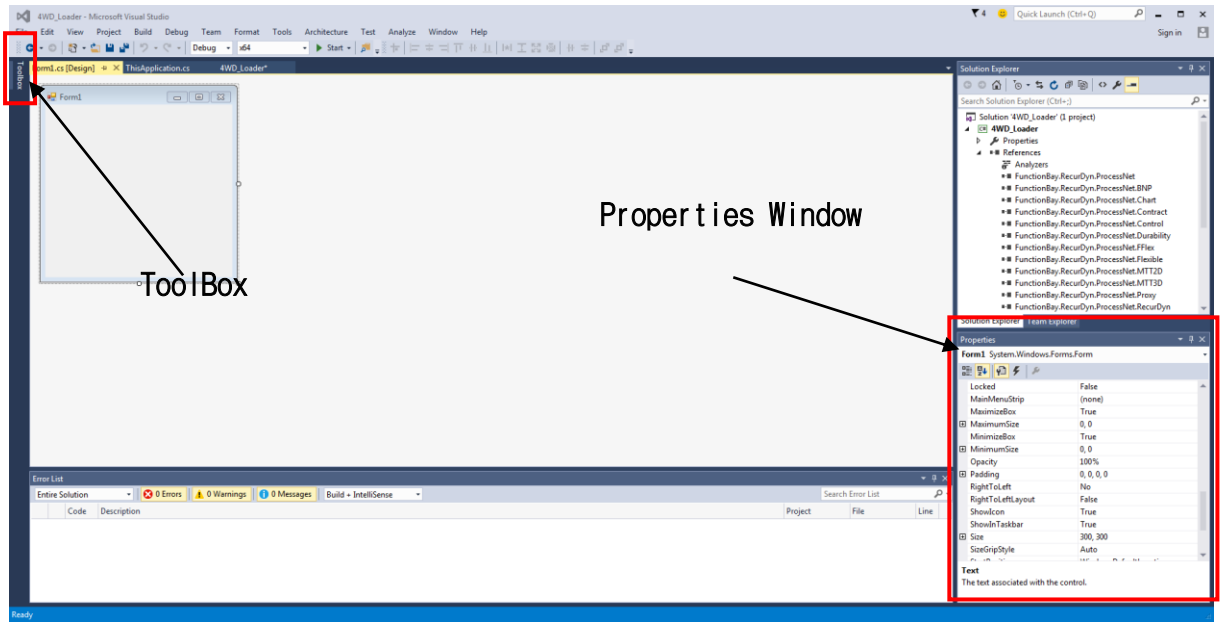
设计一个新对话窗口:

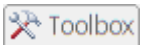
1. 返回 **Visual Studio**。
2. 在 **Project** 菜单中，选择 **AddWindows Form**。
3. 在 **Add New Item** 对话框中，选择 **Windows Form**，如下图所示。



4. 默认名称为 **Form1.cs**。
5. 点击 **Add**。

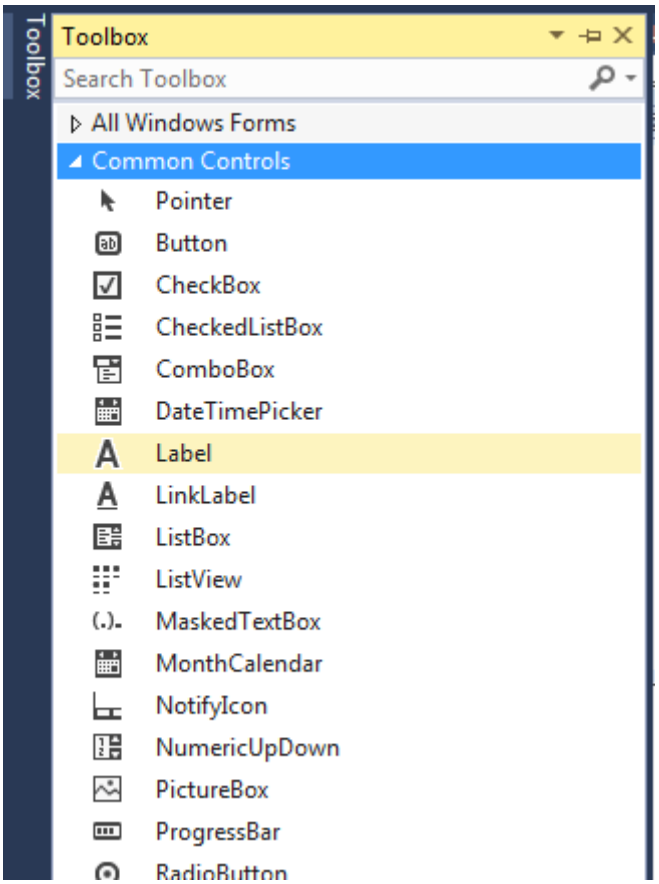
   Form1 的设计窗口 **Form1.cs [Design]**，会显示在 **IDE Project Editor** 窗口中。



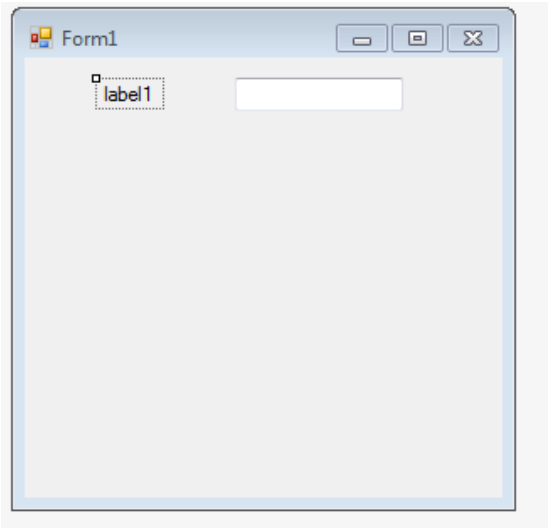
6. 在屏幕的左上角，将光标移动到 **Toolbox** 工具上。 

   随后会弹出一个菜单，包含了可以增加到对话窗口及其他相似控件上的不同单元。

7. 在 **Common Controls** 列表中，点击 **Label**，并将其拖拽到所设计的对话框的左上方区域。

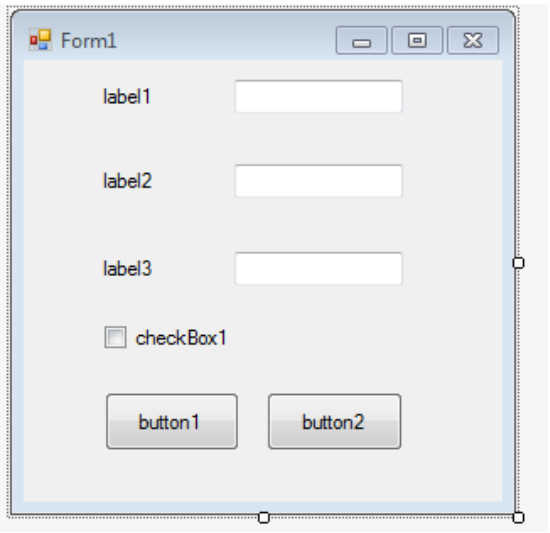


8. 重复上述步骤，这次将文本框拖拽到对话框标签的右侧，如右图所示。



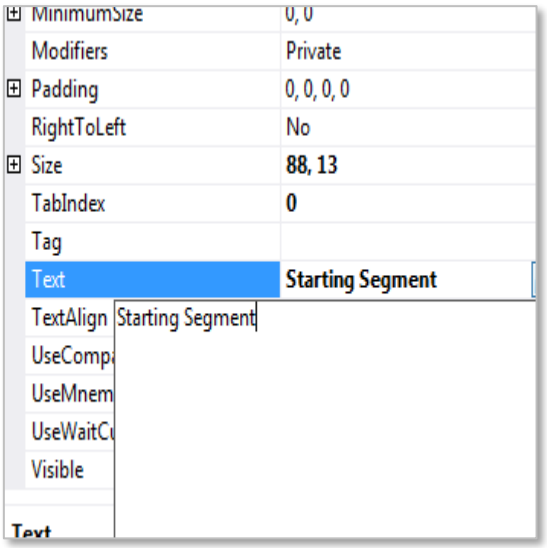
9. 重复步骤 7 和 8 两次，共创建三行标签和文本框。
10. 从 **Toolbox** 中，增加复选框到其它单元下方。
11. 从 **Toolbox** 中，增加两个按钮到其它单元下方。

    这时，对话应如右图所示。



12. 点击 **label1**，选中它。
13. 在右下角的 **Properties** 窗口中，将 **Text** 的值改为 **Starting Segment**。

提示：标签的编辑区域很小，可以通过点击区域右侧的下拉箭头更方便地编辑名称，如右图所示。这样可以为提供较大的区域。

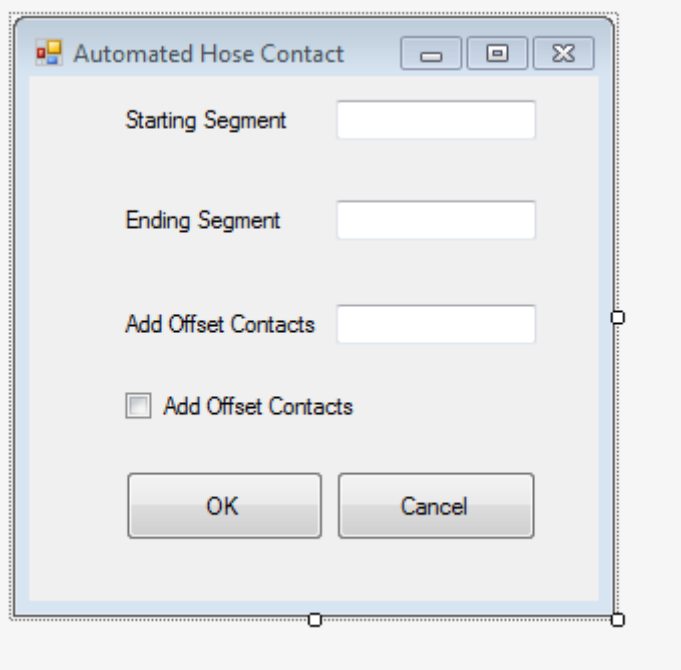


14. 对 **Labels** 2 和 3，**Checkbox**，**button**1 和 **button**2 以及 **Dialog** 重复步骤 13，名称根据下表更改：

对话单元	文本
label1	**Starting Segment**
label2	**Ending Segment**
label3	**Interval to Hose #2**
checkBox1	**Add Offset Contacts**
button1	**OK**
button2	**Cancel**
Form1	**Automated Hose Contact**

提示：如果没有组件的话，点击任意位置就可以选中对话。

15. 调整并移动对话单元，使对话显示如右图。



提示：**Visual Studio** 提供了校准指南来协助这一步骤。

16. 在 **File** 菜单中，选择 **Save All**。同时保存 **Form**1.**cs** 和项目设置。

定义对话的行为

现在，对话框的外表已经设置完成，需要通过增加变量，定义它的行为，这些变量会保存用户在文本框输入的值和复选框的状态。将增加初始化这些变量的代码，同时定义当点击 **OK** 按钮时会发生什么。

定义对话窗口行为：

1. 在对话设计窗口中，双击对话中任意没有组件的位置。

   IDE 项目编辑器窗口中会显示 **Form1.cs** 的代码，并为一个名为 **Form1_Load** 的子程序创建存根。这个子程序会在对话框被复制及加载时调用，因此这里是编写初始化变量代码的理想位置。

2. 插入如下代码块，它定义了对话框中要用到的变量：

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

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

3. 插入如下代码块，它设置了对话框中会显示的初始值：

```
    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;
    }
```

4. 返回对话设计窗口，双击 **OK** 按钮。

5. 在刚才自动创建的新方法内，插入如下所示的代码：

```
    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();
    }
```

6. 再次返回对话设计窗口，双击 **Cancel** 按钮。
7. 在刚才自动创建的新方法内，插入如下所示的代码：

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

8. 在 **File** 菜单中，选择 **Save Form1.cs**。

在运行宏时显示对话

现在通过在宏子程序内，创建一个新实例，定义在运行宏的时候，什么时候显示对话。

从宏内部显示对话窗口：

1. 复制创建的整个 **ProcessNetTutorialCreateSuToSuContact** 子程序。
2. 粘贴到 **ThisApplication.cs** 中，重命名为 **ProcessNetTutorialCreateSuToSuContact_WithDialog**。
3. 对子程序做出如下更改 (删除用~~删除线~~标记的代码)。每个改动的解释后文会详述。

```
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 = newForm1();

// 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.");
    }
}
```

- 改动 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
```

这里删除了赋予变量数值的代码，这些值现在将取决于在对话中输入的值。

- 改动 2：

```
// Create a Form
Form1 MyForm = newForm1();

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

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

这里创建了 Form1 的一个新实例，并展示了出来。If 语句用于检测用户的回应，如果用户点击 OK，那么 if 语句内的代码就会被执行。同时，这里声明了一个叫 NumContacts 的变量，它记录了所创建接触的数量。If 语句会在改动 5 结束。

- 改动 3：

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

这里您替换了赋予分段数目变量数值的代码，新代码将在对话中输入的值赋予这些变量。

- 改动 4：

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

if (MyForm.AddOffsetFlag)
            {
.
.
.

// Increment the number of contacts
                NumContacts = NumContacts + 2;
```

这里正确的更新了 NumContacts。同时，if 语句检测了用户是否勾选了 Add Offset Contacts 复选框。如果是的话，if 内的语句就会被执行。

- 改动 5：

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

在 RecurDyn 信息窗口中会对用户显示一条输出信息，确认创建了多少个接触。另外，最外层的 if 语句结束 (见改动 1)。

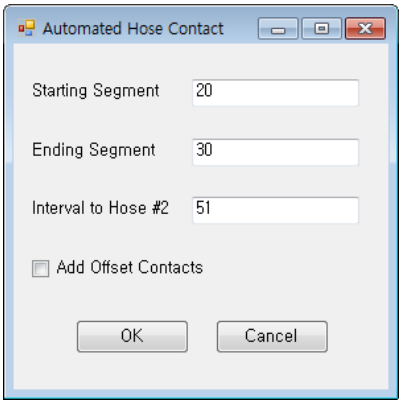
4. 在 File 菜单中，选择 Save ThisApplication.cs。

测试对话框

重复前面的步骤，建立宏，并测试刚刚所做的改动。

重复流程：

1. 确保 **IDE** 窗口底部的 **Error List** 窗口中，没有任何错误或警告。如果有的话，检查列表并作出相应修正。在 **Build** 菜单中，选择 **Build Solution**。
2. 返回 **RecurDyn**，删除所有在之前仿真中定义的接触。
3. 在 **Customize** 菜单的 **ProcessNet(General)**组中，选择 **Run**。
4. 会在列表中看到一个新项 **ProcessNetDialogCreateSolidContact_WithDailog**，选中它。
   它的名称会显示在 **Run** 按钮旁边的栏中。
5. 点击 **Run**。
   对话框会显示如右图。



6. 点击 **OK**，使用默认值。
   现在可以在 **RecurDyn** 数据库窗口中看到已经创建 11 个接触。同时 **RecurDyn** 信息输出窗口中也显示出一条确认信息，提示创建了 11 个接触。
7. 通过点击 **RecurDyn** 中的 **Undo** 箭头删除刚刚创建的接触，然后再次运行宏，这次勾选 **AddOffset Contacts** 复选框。
   会看到模型中增加了 33 个接触，同时 **RecurDyn** 信息输出窗口中再次显示了一条确认信息。
8. 恢复前一次仿真的结果。点击 **File** 菜单，并选择 **Import** 指令。通过下拉菜单，将文件类型改为 **RecurDyn Animation Data File** (*.**rad**)。选择 **4WD_Loader.rad** 文件，并点击 **Open**。
9. 关闭 **Run ProcessNet** 窗口。

Chapter

5

自动化创建绘图

本章在多窗口绘图中，绘制两个图：

- 第一幅图会显示单个部位间接触力的大小。未受力的接触会被排除在外，因此绘图只包含了感兴趣的数据。还将通过改进格式、增加标签以及缩放 **X** 轴关注软管接触时间等方法，改进该绘图。
- 第二幅图显示了整个软管间接触的力，分成 **X**，**Y** 和 **Z** 部分。两个绘图都会被标记以及设置格式。

任务目标

学习如何使用 **ProcessNet** 指令完成自动绘图，包括：

- 导入一个绘图数据文件。
- 智能判断要显示的数据。
- 在多窗口绘图中，处理并根据数据绘图。
- 根据数据，设置图表的格式。

预计完成的时间

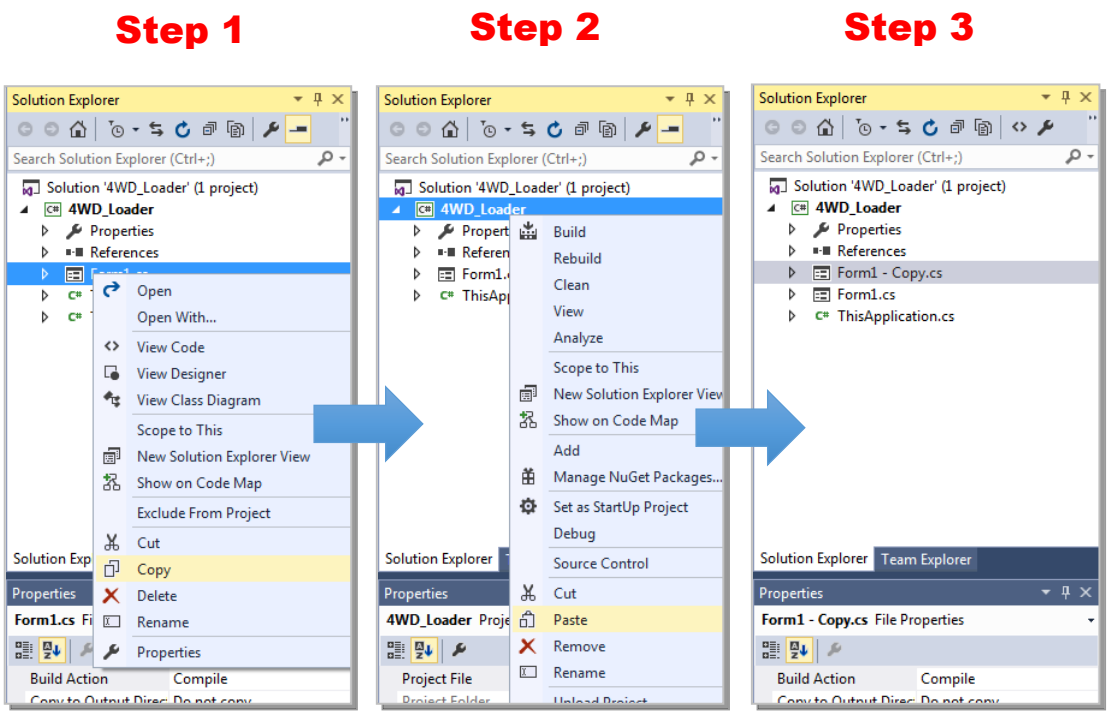
20 分钟

创建对话

需要另外一个会读入输入的对话窗口，确定将哪些数据绘图。这个对话与创建的第一个对话很相似，所以先复制该对话。

创建新对话窗口：

1. 在右侧的项目管理器窗口中，右键点击 **Form1.cs** 并选择 **Copy** (后面的步骤参考下图)。
2. 右键点击 **ProcessNet** 并选择 **Paste**。
3. 右键点击 **Copy of Form1.cs**，然后点击 Rename 来编辑其名称。
4. 重命名为 **Form2.cs**。

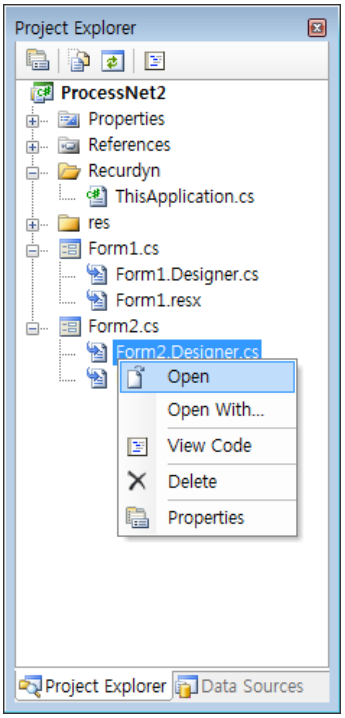


调整新的对话窗口：

1. 双击打开 Form2.cs。
   弹出对话设计窗口。
2. 在 View 菜单中选择 Code 来显示代码窗口。
3. 在代码中搜索“Form1”，并全部替换为“Form2”。一共会替换三个，如下图所示：

```
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;
```

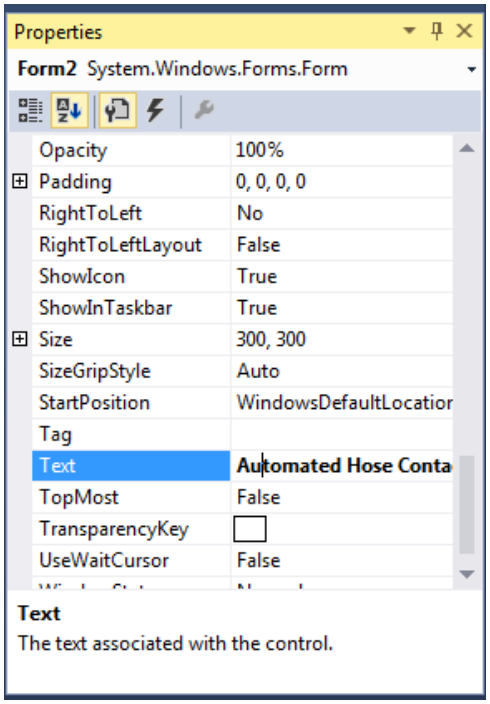
4. 在 **Project Explorer** 窗口中，右键点击 Form2.Designer.cs，并选择 Open，如右图所示。



5. 再次在代码中搜索“Form1”，并全部替换为“Form2”。这次会替换两个，如下图所示：

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

6. 在 **IDE Project Editor** 窗口上方，点击 **Form2.cs [Design]**标签。
7. 点击对话中任意没有组件的位置，选中对话窗口自身(而不是任何单独组件)。
8. 在 **Properties** 窗口中(屏幕右下角)，将 **Text** 改为 **Automated Hose Contact Plotting**。
9. 在 **IDE Project Edtior** 窗口中，点击并拖拽对话窗口的右侧，调整其大小，使得新标题可以完全显示出来。
10. 点击 **Add Offset Contacts** 标签。
11. 在属性窗口中，将文本改为 **Offset Contacts Used**。
12. 在 **File** 菜单中，选择 **Save All**。



绘制接触力

现在在 **ThisApplication** 类下，创建一个新的子程序。

创建一个新的子程序:

1. 复制下列代码块(包括下一页上的代码)，并插入到 **ThisApplication** 类中。(提示：确保代码被插入到类中，而不是另一个子程序中。)

```
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);
                }
            }
        }
    }
}
```

前两个指令创建了一个新的 **Form2** 窗口，并向用户显示：

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

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

后面的 **if** 语句检测用户是否点击了 **OK** 按钮，如果是的话，执行 **if** 内的语句：

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

下列代码从导入的文件中，获取 **TIME** 的绘图数据，并保存在一个新的数组中。

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

下列指令激活了左上方的绘图窗口：

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

下列代码存储了一组字符串，包含了作图数据的部分名称。数组是根据在对话中输入的值来存储的。例如，如果将 **Starting Contact** 设为 20，同时勾选了“**Offset Contacts Used**”复选框，那么在第一次循环中数组包含的字符串就是 {“20”、“20**a**”，“20**b**”}。

```
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)
                {
```

下一个指令根据数据名称，获得绘图数据。名称与从绘图数据库窗口中看到的一样，使用字符“/”作为分隔：

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

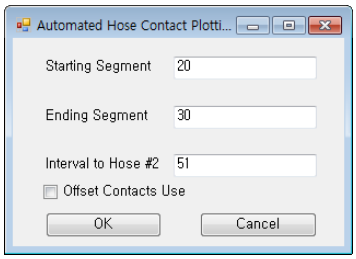
最后，下一个指令会在绘图窗口，根据数据绘图。

```
// Plot vs. TIME
                    plotDocument.DrawPlot("Force (N)", TIME, contact);
```

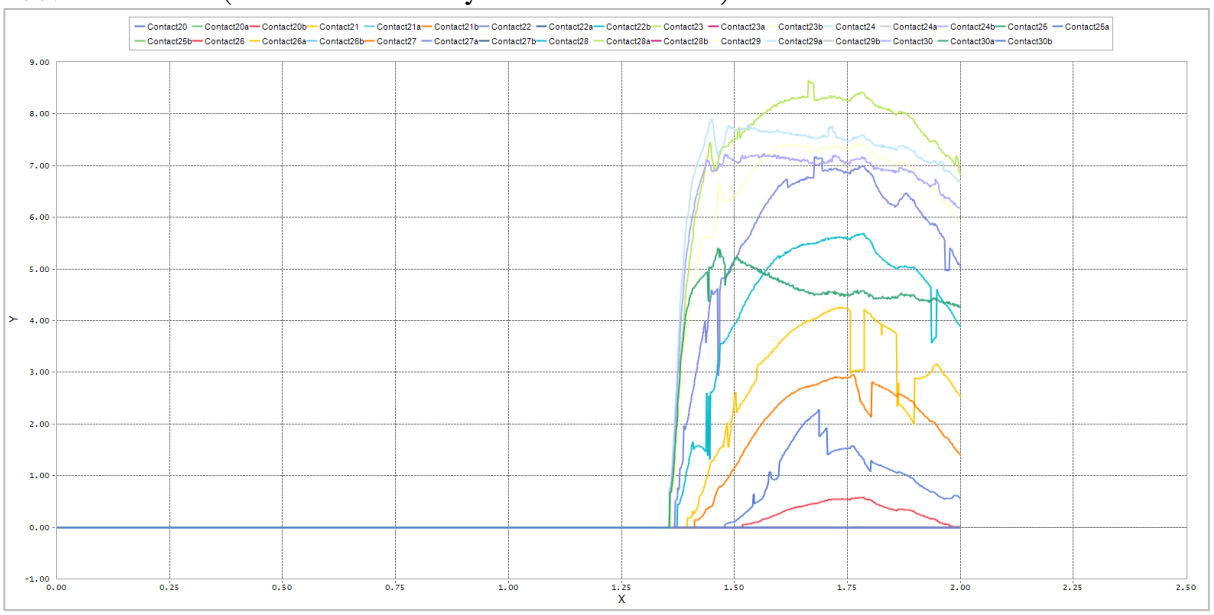
2. 保存文件。
3. 在 **Build** 菜单中选择 **Build Solution**。

测试编辑过的带绘图的子程序：

1. 返回加载了装载机模型的 **RecurDyn** 建模窗口，打开一个绘图窗口。
2. 在 **Customize** 标签下的 **ProcessNet** 组中，点击 **Run**。
3. 在列表中，选择 **ProcessNetTutorialPlotData**。
4. 选择 **Run**。
5. 在弹出的 **Dialog2** 对话窗口中，接受默认部位数目并勾选 **Offset Contacts Used**。



会看到如下绘图(已调整为 **RecurDyn** 绘图窗口的大小)：



上图包含了全部 33 个接触，但是只有一小部分曲线中包含非零值，因为软管间的接触局限在每个软管的几个部位中。同时，虽然所有的接触都是在仿真的最后 1/3 发生的，但是绘图显示了仿真的整个时间线。

改进接触力绘图

通过如下途径改进绘图，使其更关注有意义的区域：

- 只绘制非零接触。
- 缩短时间线(即 **X** 轴跨度)，使其只包含非零接触行为。

首先学习只绘制非零接触。这里也会介绍缩短时间线的一些逻辑，但如何设置 X 轴会在后文详述。

只绘制非零接触力：

1. 对代码做如下更改：

```
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);
                }
            }
        }
    }
}
```

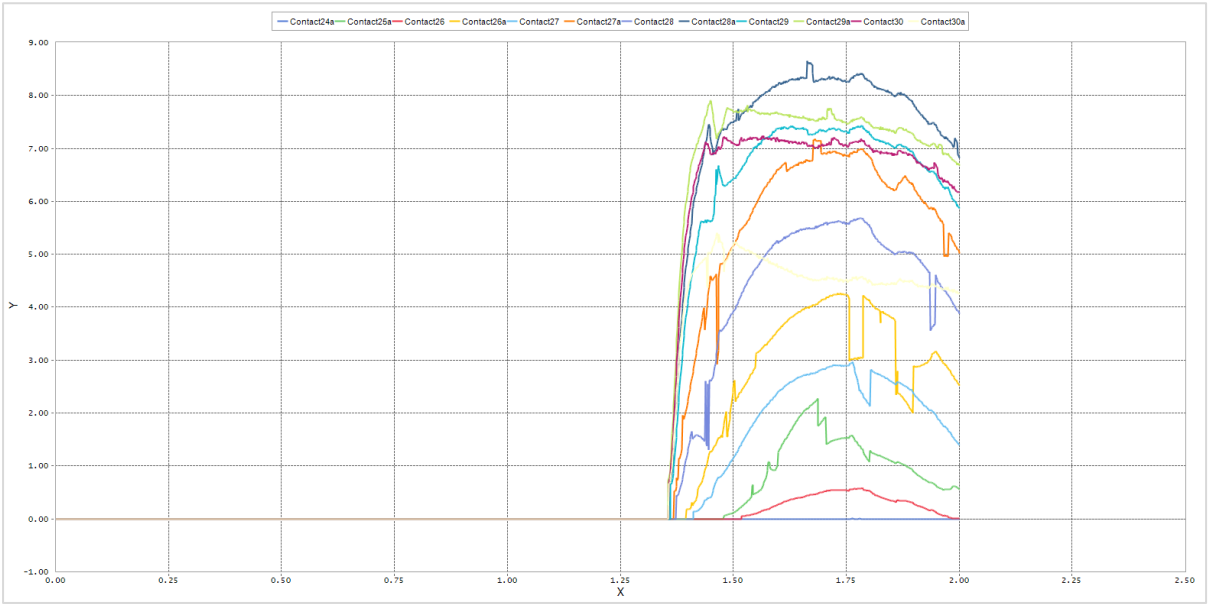
现在代码包含了对所有数据的逻辑，以数组的形式查找非零值。代码还记录了第一次和最后一次非零接触的时刻。

2. 保存文件。
3. 在 **Build** 菜单中，选择 **BuildSolution**。

测试非零接触力图:

遵循与之前相同的步骤，在新的绘图窗口中，测试新的 **ProcessNet** 宏(先打开一个新的绘图窗口)。

显示如下图形。



现在可以看到在仿真中哪些接触是非零的，33 个接触中只有 12 个是非零的。

改进绘图格式

上一幅图比之前的版本显示的更为清晰，但还是有改进的空间。绘图的标题、轴的标签和标题都不够具体，而且时间刻度也没有关注在仿真中最有意义的部分。现在将使用 **ProcessNet** 功能设置绘图的一些格式。

改进绘图格式:

1. 完成下述代码更改：

```
#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.LegBox.Alignment
            = LegendBoxAlignment.LegendBoxAlignment_Far;
    }
}
```

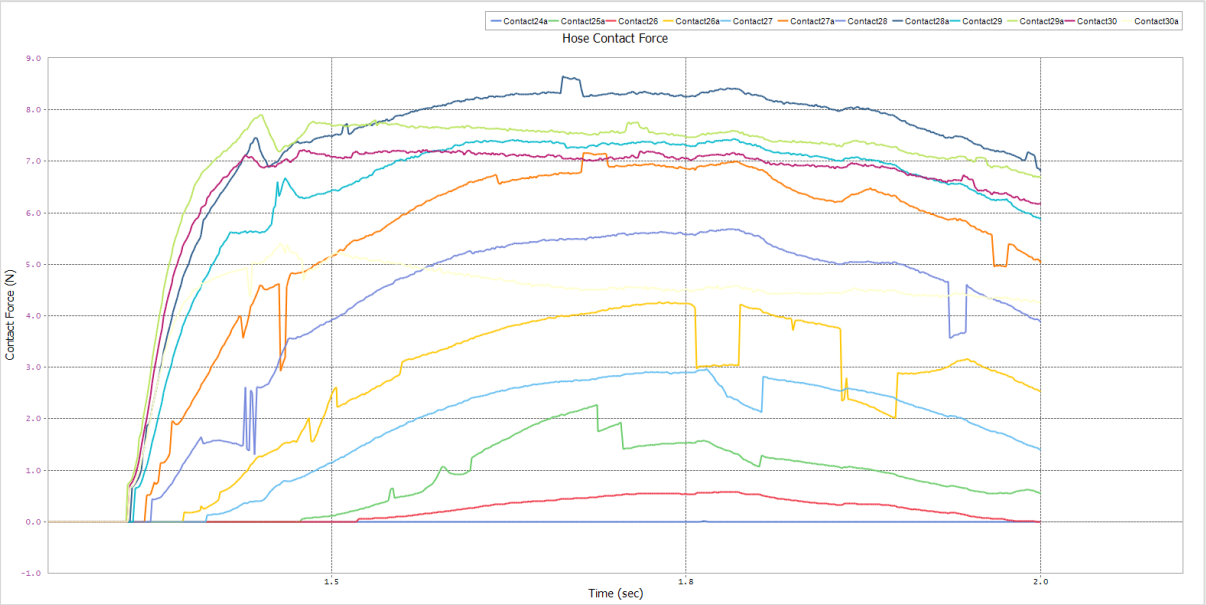
注意，**X** 轴的定义限制了 **Chart1.AxisX.Min** 和 **Chart1.AxisX.Max**（使用在前面部分计算过的变量 **timeAtFirstContact** 和 **timeAtLastContac**）。

2. 保存文件。
3. 在 **Build** 菜单中，选择 **BuildSolution**。

测试绘图的格式：

遵照之前相同的步骤，在新的绘图窗口中，测试新的 **ProcessNet** 宏(先打开一个新的绘图窗口)。

看到如下图形：



现在绘图已经全部设置完成，缩短 **X** 轴可以更具体地观察非零接触。

绘制总 X，Y 和 Z 接触力

假设希望看到两个软管间的总接触力，分为 **X**，**Y** 和 **Z** 方向三个分量。那么，所用数据需要通过现有绘图文件中的数据进行计算。在 **ProcessNet** 中，由于绘图数据可以存储在数组中，因此可以直接进行数学运算。对下一幅图来说，将把数组数据加总到一起，得到所有接触在 **X**，**Y** 和 **Z** 方向上的总和。

因为在前面的绘图中学习了创建新绘图需要的大部分指令，所以本教程的这个部分可以直接复制粘贴一段单独的代码块。但是注意需要添加总数组的处理过程。

增加第二幅图:

1. 将下述代码块插入到如下所示的位置中，接近子程序的末端但仍在 **if** 语句内：

```
        // 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. 保存文件。

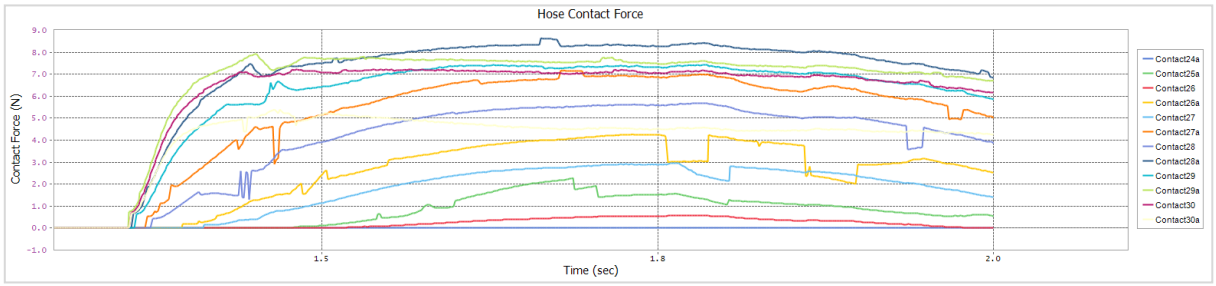
3. 在 **Build** 菜单中，选择 **BuildSolution**。

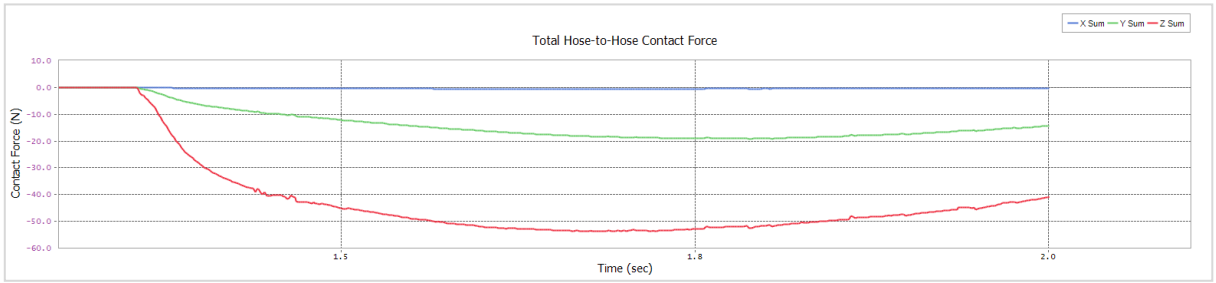
测试第二幅图:

第二幅图会显示在左下方的窗口中 (与第一幅图相反，其显示在右上方窗口中)，确保两个窗口都已显示，避免错误。

- 遵照与前文相同的步骤，在一个新的 **Plot Window** 中，测试新的 **ProcessNet** 宏，这次在 **Home** 标签的 **Windows** 组中，选择 **Show Left Windows**。

看到如下两幅图形：





将 VSTA 项目转为 General 项目

本章中，将学习如何将一个 **VSTA** 项目转为一个通用项目。

任务目标

学习如何通过转化步骤将在 **ProcessNet VSTA** 项目中使用的代码，用在一个 **ProceeNet** 通用项目中。

预计完成的时间

5 分钟

编辑 VSTA 代码

在 Visual Studio 中打开 VSTA 项目：

1. 打开“C:\ Users\ <User Name>\ Document\ Visual Studio 2015\ Project\ 4WD_Loader”目录。
2. 利用 **Text Edit application** 而不是 **Visual Studio**，打开子文件夹中的 4WD_Loader.csproj 文件。
3. 删除<**PropertyGroup**>中的<**ProjectTypeGuids**>，如下图所示。

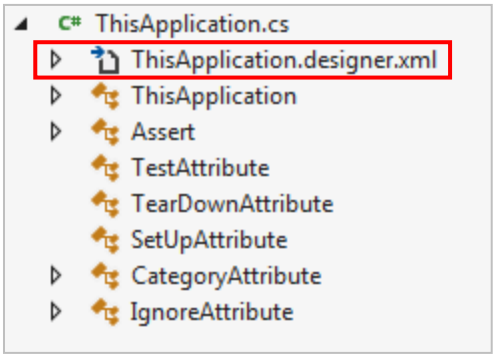
```
<PropertyGroup>
<ProjectTypeGuids>{A860303F-1F3F-4691-B57E-529FC101A107};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids>
<Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
<Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
<OutputType>Library</OutputType>
<NoStandardLibraries>false</NoStandardLibraries>
<RootNamespace>Loader</RootNamespace>
<AssemblyName>Loader</AssemblyName>
<ProjectGuid>{24C91D44-F891-47E7-B8A1-B422B35B5C7C}</ProjectGuid>
</PropertyGroup>
```

4. 保存文件。
5. 用 **Visual Studio** 打开上一级目录中的 4WD_Loader.sln 项目。

注意：如果编辑文件已经在 **VSTA** 中打开，那么 **Visual Studio** 可能无法转化项目。

编辑代码：

1. 删除项目管理器窗口中 **ThisApplication.cs** 下的 **ThisApplication.designer.xml** 文件。
   - **ThisApplication.Desinger.xml** 文件只对 **VSTA** 必要。



2. 删除 **ThisApplication.cs** 中名为 **VSTA generated code** 的区域。

```
#region VSTA generated code
private void ThisApplication_Startup(object sender, EventArgs e)
{
System.Threading.Thread.CurrentThread.CurrentCulture =
System.Globalization.CultureInfo.InvariantCulture;
MainWindow = new
WinWrapper(System.Diagnostics.Process.GetCurrentProcess().MainWindowHandle);
```

```
}

private void ThisApplication_Shutdown(object sender, EventArgs e)
{

}

private void InternalStartup()
{
this.Startup += new System.EventHandler(ThisApplication_Startup);
this.Shutdown += new System.EventHandler(ThisApplication_Shutdown);
}
```

3. 在 **Common Variables** 下加入如下代码。

```
        #region Common Variables

        FunctionBay.RecurDyn.ProcessNet.RecurDyn.IRecurDynApp app = new
FunctionBay.RecurDyn.ProcessNet.RecurDyn.RDApplication();

        static public IApplication application;
        public IModelDocument modelDocument = null;
        public IPlotDocument plotDocument = null;
        public ISubSystem model = null;

        public IReferenceFrame refFrame1 = null;
        public IReferenceFrame refFrame2 = null;
        #endregion
```

- 这段代码的目的是，不用 **VSTA** 进入 **RecurDyn** 应用。

4. 将 **Initiallize()**函数编辑如下。

```
        #region Common Variables

        FunctionBay.RecurDyn.ProcessNet.RecurDyn.IRecurDynApp app = new
FunctionBay.RecurDyn.ProcessNet.RecurDyn.RDApplication();

        public void Initialize() //Initialize() will be called automatically
before ProcessNet function call.
        {
            application = app.RecurDynApplication as IApplication;
            application = RecurDynApplication as IApplication;
            modelDocument = application.ActiveModelDocument;
            plotDocument = application.ActivePlotDocument;
            if (modelDocument == null & plotDocument == null)
            {
            application.PrintMessage("No model file");
            modelDocument = application.NewModelDocument("Examples");
            }
            if (modelDocument != null)
            {
                model = modelDocument.Model;
            }
```

```
        }
```

5. 在 **Build** 菜单中，选择 **Build Solution**。

感谢学习本教程!