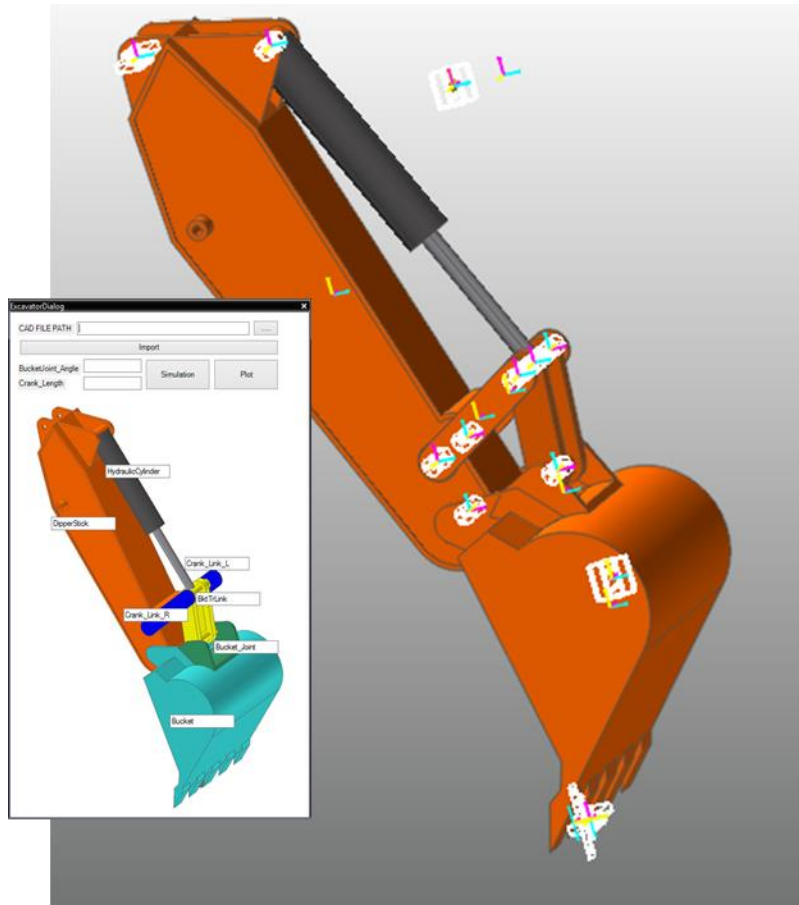




## ProcessNet 帶挖斗的斗杆教程 (VSTA)



Copyright © 2017 FunctionBay, Inc. All rights reserved

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

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

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

## **Registered Trademarks of FunctionBay, Inc. or Subsidiary**

*RecurDyn*<sup>™</sup> is a registered trademark of FunctionBay, Inc.

*RecurDyn*<sup>™</sup>/SOLVER, *RecurDyn*<sup>™</sup>/MODELER, *RecurDyn*<sup>™</sup>/PROCESSNET, *RecurDyn*<sup>™</sup>/AUTODESIGN, *RecurDyn*<sup>™</sup>/COLINK, *RecurDyn*<sup>™</sup>/DURABILITY, *RecurDyn*<sup>™</sup>/FFLEX, *RecurDyn*<sup>™</sup>/RFLEX, *RecurDyn*<sup>™</sup>/RFLEXGEN, *RecurDyn*<sup>™</sup>/LINEAR, *RecurDyn*<sup>™</sup>/EHD(Styer), *RecurDyn*<sup>™</sup>/ECFD\_EHD, *RecurDyn*<sup>™</sup>/CONTROL, *RecurDyn*<sup>™</sup>/MESHINTERFACE, *RecurDyn*<sup>™</sup>/PARTICLES, *RecurDyn*<sup>™</sup>/PARTICLEWORKS, *RecurDyn*<sup>™</sup>/ETEMPLATE, *RecurDyn*<sup>™</sup>/BEARING, *RecurDyn*<sup>™</sup>/SPRING, *RecurDyn*<sup>™</sup>/TIRE, *RecurDyn*<sup>™</sup>/TRACK\_HM, *RecurDyn*<sup>™</sup>/TRACK\_LM, *RecurDyn*<sup>™</sup>/CHAIN, *RecurDyn*<sup>™</sup>/MIT2D, *RecurDyn*<sup>™</sup>/MIT3D, *RecurDyn*<sup>™</sup>/BELT, *RecurDyn*<sup>™</sup>/R2R2D, *RecurDyn*<sup>™</sup>/HAT, *RecurDyn*<sup>™</sup>/曲柄, *RecurDyn*<sup>™</sup>/PISTON, *RecurDyn*<sup>™</sup>/VALVE, *RecurDyn*<sup>™</sup>/TIMINGCHAIN, *RecurDyn*<sup>™</sup>/ENGINE, *RecurDyn*<sup>™</sup>/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.

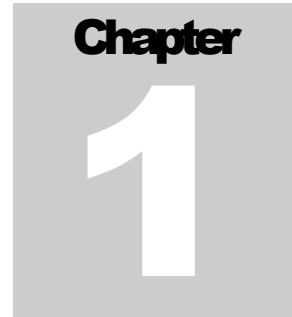
CADAM and CATIA are registered trademark of Dassault Systems. FLEX/m is a registered trademark of GLOBEtrouter Software, Inc. All other brand or product names are trademarks or registered trademarks of their respective holders.

## **Edition Note**

These documents describe the release information of *RecurDyn*<sup>™</sup> V9R1.

# 目录

目录 .....	4
概述 .....	5
任务目标 .....	5
学前要求 .....	5
预备知识 .....	5
任务 .....	5
完成本教程需要的时间 .....	6
启动 ProcessNet.....	7
任务目标 .....	7
预计完成本任务的时间 .....	7
启动 RecurDyn .....	8
启动 ProcessNet.....	8
创建对话框 .....	10
任务目标 .....	10
预计完成本任务的时间 .....	10
创建对话框 .....	11
配置对话框的初始设置 .....	15
运行应用时显示对话框 .....	18
测试对话框 .....	19
通过代码自动生成模型.....	21
任务目标 .....	21
预计完成本任务的时间 .....	21
创建一个新的类 .....	22
创建一个模型 .....	23
将一个函数关联到对话框 .....	33
测试对话框 .....	34
分析模型 .....	35
任务目标 .....	35
预计完成本任务的时间 .....	35
编辑对话框的布局 .....	36
模型分析功能 .....	37
自动创建绘图 .....	38
任务目标 .....	38
预计完成本任务的时间 .....	38
绘图函数 .....	39
测试所创建的应用 .....	40



## 概述

### 任务目标

ProcessNet 是一个基于 .NET framework 的编程工具，采用与 Microsoft Visual Studio 相同的方法调用类和变量。因此可以在 ProcessNet 中应用大量的编程技巧。在本教程中，基于 .NET framework 的界面，学习如何用 ProcessNet 来创建 Windows Forms (WinForms)。利用 WinForms 可以实现 RecurDyn 中建模、分析和绘图等过程的自动化。

- 用 WinForms 创建一个用户界面
- 用一个 CAD 文件实现模型创建的自动化
- 使用 ProcessNet 的类函数，而不是 ThisApplication
- 用对话框来实现建模、分析和绘图过程的自动化

### 学前要求

本教程中采用的模型以 Dipper Stick with Bucket 为基础，一个 RecurDyn 教程中的 DOE&Batch 仿真教程。因此，在开始本教程前，必须先完成 Dipper Stick with Bucket 教程的学习。

### 预备知识

必须熟悉 ProcessNet 和 Dipper Stick with Bucket 教程，或类似教程。同时，需要掌握基本的物理知识。

### 任务

本教程包含以下任务，下表列出了完成每个任务所需的时间。

任务	时长(分钟)
启动ProcessNet	5
创建对话框	15
通过代码自动生成模型	30
分析模型	10
自动创建绘图	10
总计	70



## 完成本教程需要的时间

本教程大约需要 70 分钟完成。

Chapter

2

## 启动 ProcessNet

### 任务目标

学习如何在 RecurDyn 中启动 ProcessNet。



预计完成本任务的时间

5 分钟

## 启动RecurDyn

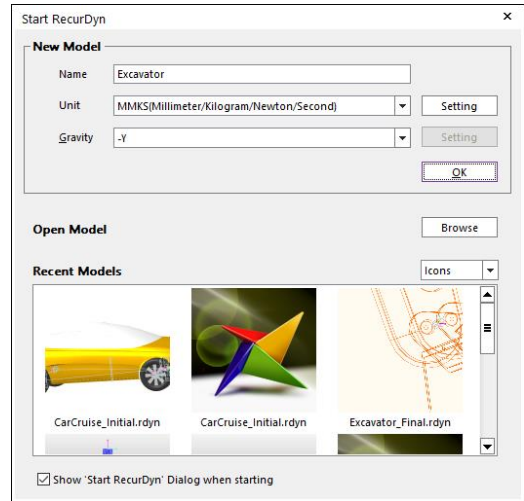


### 启动RecurDyn:

1. 运行 RecurDyn。

StartRecurDyn 对话框会弹出。

2. 在 New Model 的 Name 栏中, 输入 Excavator。
3. 将 Unit 设为 MMKS。
4. 点击 OK, 完成新模型的创建。



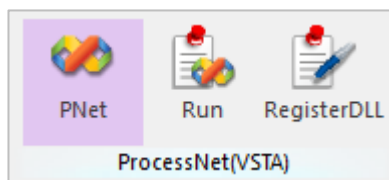
### 保存模型:

- 在 File 菜单中, 点击 Save As, 将模型 Excavator.rdyn 保存在希望保存的文件夹中。

## 启动ProcessNet

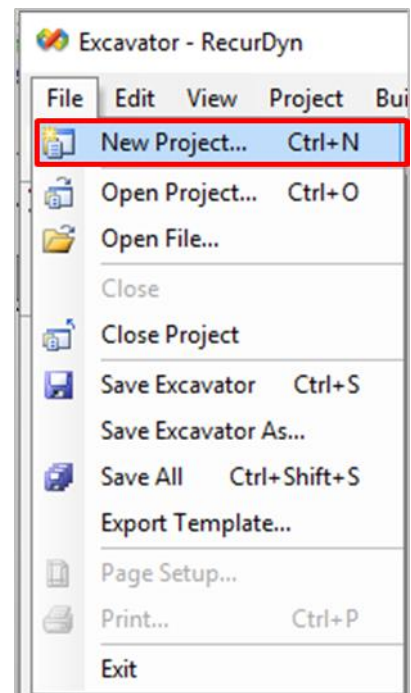
### 启动并初始化 ProcessNet:

1. 在 Customize 标签的 ProcessNet 组中, 点击 PNet, 打开 ProcessNet 的集成开发环境(IDE)。



2. ProcessNet IDE 启动后, 打开 File 菜单, 并点击 New Project。
3. New Project 对话框弹出后, 选择与的 RecurDyn 版本相对应的 Template。

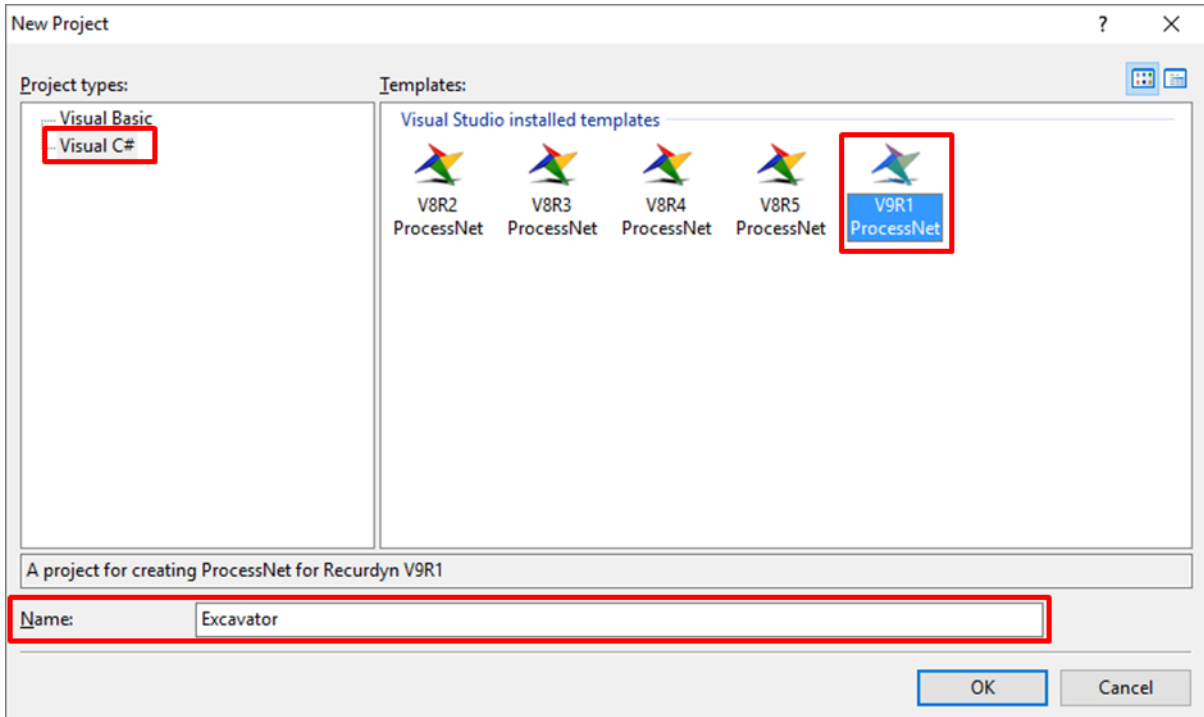
- 本教程以 V9R1 为基础, 因此选择 V9R1。



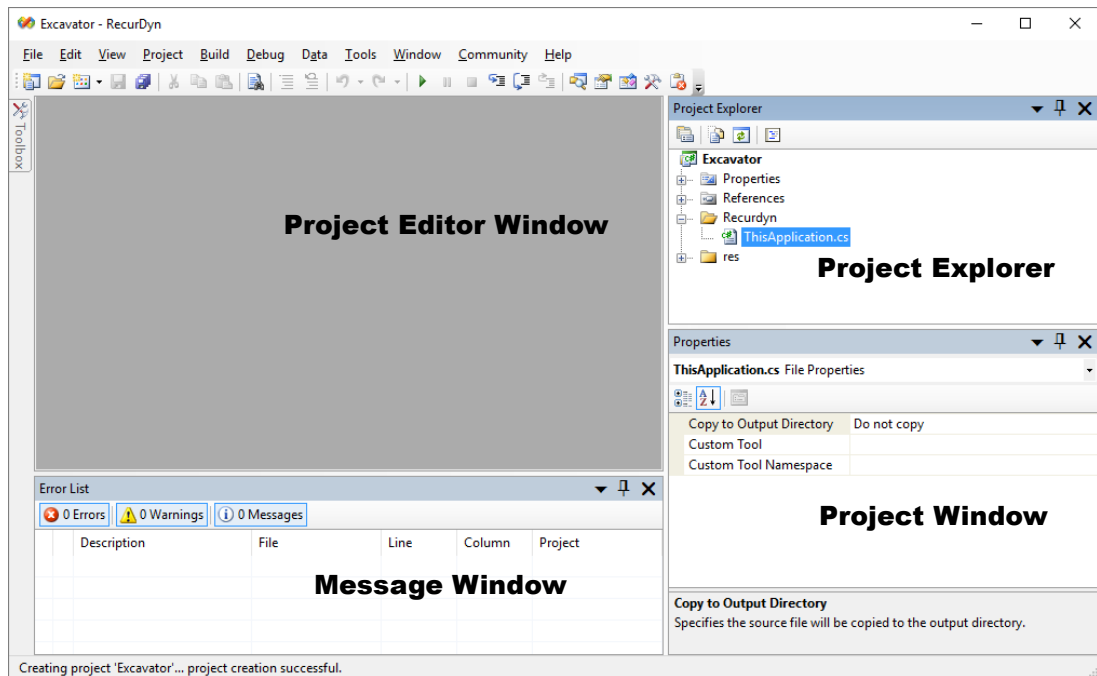
**注意:** 必须使用与 RecurDyn 版本相兼容的 ProcessNet 项目。如果版本不正确, ProcessNet 可能无法正确执行操作。New Project 对话框显示了所有与所安装 RecurDyn 版本相容的模板。



4. 在 **Project Type** 框中，选择 **Visual C#** 作为项目类型，在 **Name** 栏中，输入 **Excavator**。  
点击 **OK**。



5. 创建 **Excavator Project**，如下图所示。



6. 点击 **File**，点击 **Save Excavator**，并将 **ProcessNet** 项目保存在希望保存的文件夹中。  
现在，可以进行 **ProcessNet** 应用了。

## Chapter

## 3

## 创建对话框

在本章学习如何创建一个对话框，并设置其布局。包括设计对话框的布局，以及通过加入代码实现在 **ProcessNet** 中调用对话框。

### 任务目标

学习如何创建对话框和可以在 **ProcessNet** 中调用对话框的函数。



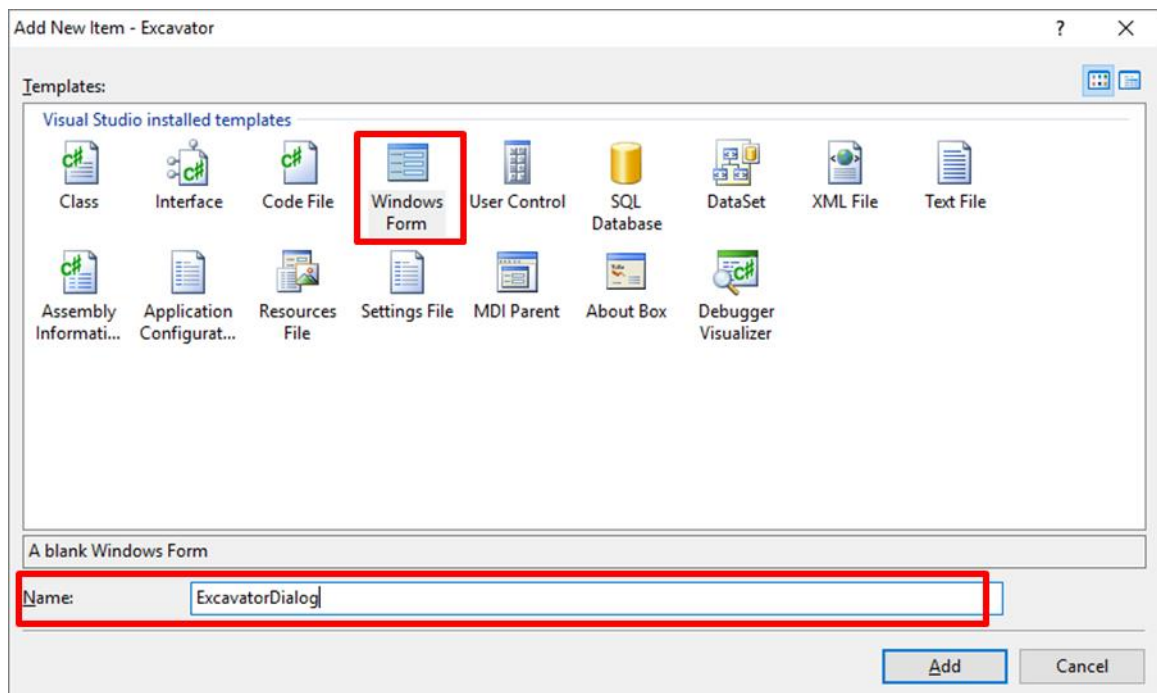
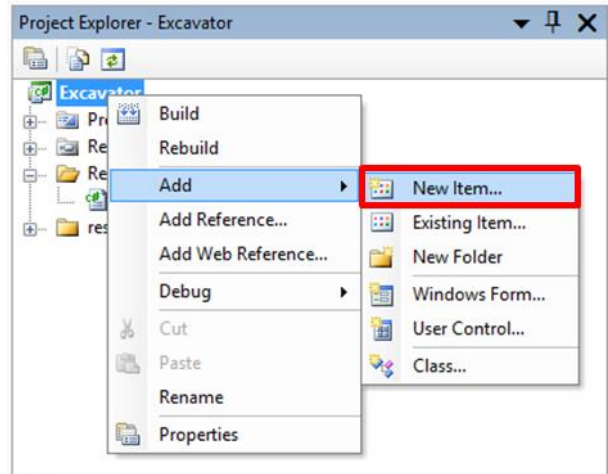
### 预计完成本任务的时间

15 分钟

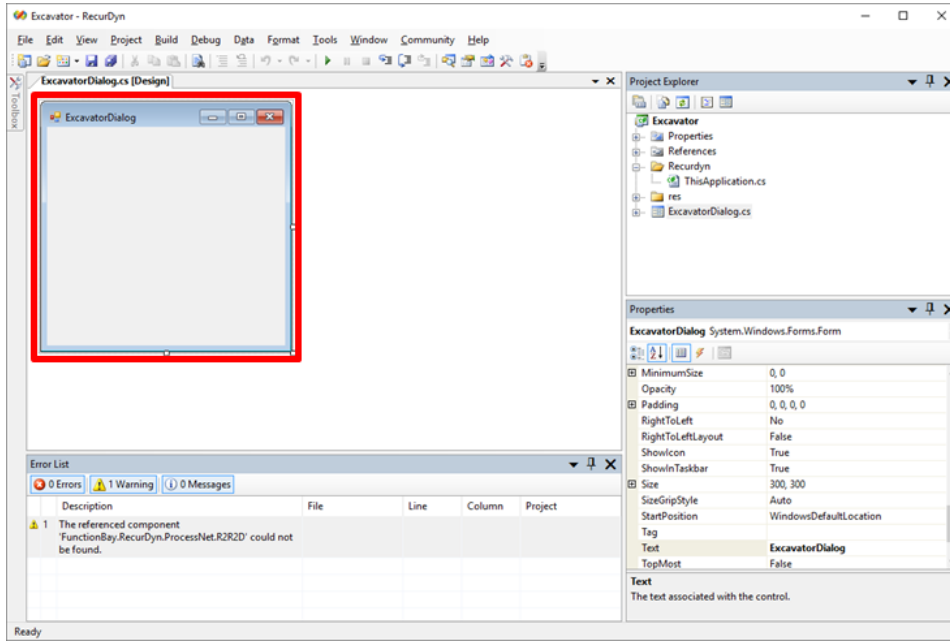
## 创建对话框

创建对话框：

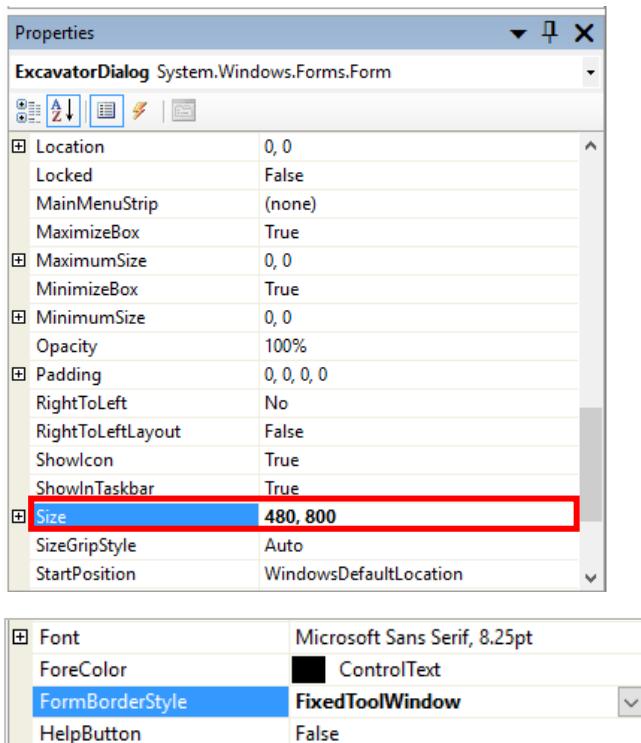
1. 在 **ProcessNet** 集成开发环境中，右键点击 **ProjectExplorer** 框。
2. 点击 **Add - New Item**。
3. **Add New Item** 对话框弹出后，点击 **Windows Form** 图标，并在名称栏中输入 **ExcavatorDialog**。
4. 点击 **Add**。




5. **IDE Project Editor** 框中会出现 **ExcavatorDialog.cs[Design]**，一个用于 **Windows Form** 的设计窗口。



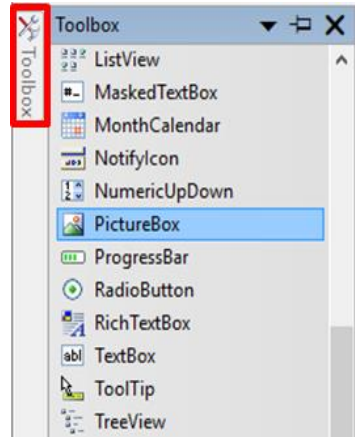
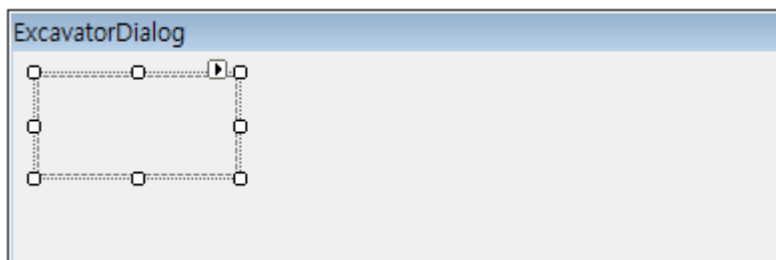
6. 点击屏幕左上角的 **ExcavatorDialog** 对话框。
7. **ExcavatorDialog** 的相关信息会显示在屏幕右下角的 **Properties** 框中。在 **Properties** 框中，将 **Size** 设为 480, 800。
8. 同时，将 **FormBorderStyle** 设为 **FixedToolWindow**。



9. 将光标移到屏幕左上角的 **ToolBox** 。可以看到用于增加对话框、按钮以及其他控制函数的菜单。

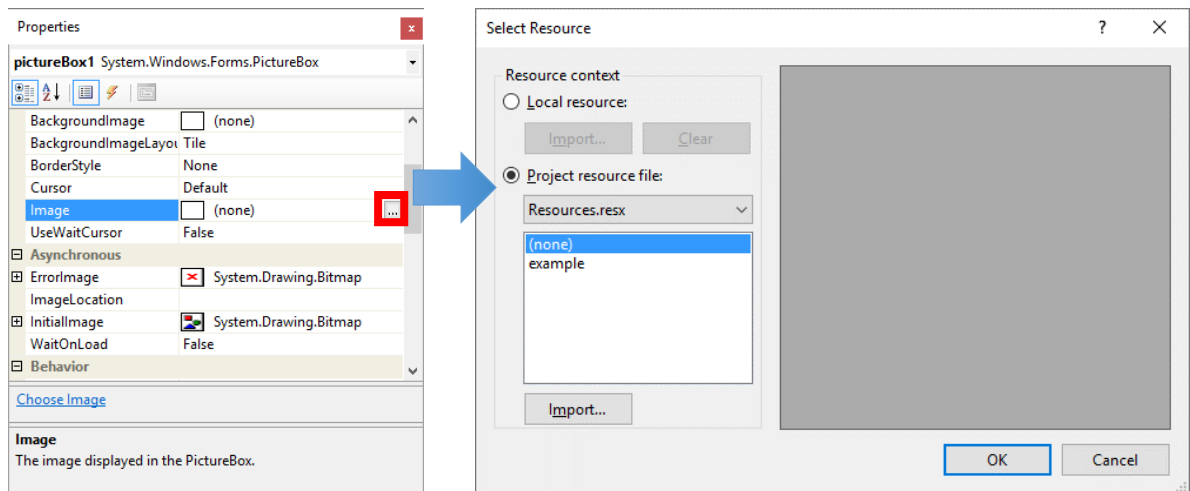
*注意: 如果看不到 **ToolBox**, 打开 **View** 菜单, 并点击 **ToolBox** 或同时按下 **Ctrl + Alt + X**。*

10. 在 **Common Controls** 列表中, 选择 **PictureBox**, 拖拽标签到所设计对话框的左上角。
11. 选择创建的 **PictureBox**。



12. 如下图所示, 在屏幕右下角的 **Properties** 框中, 点击 **Image** 行最右侧的...按钮。

**Select Resource** 对话框弹出。



13. 在 **Select Resource** 对话框中, 选择 **Local resource**, 并点击 **Import** 按钮。
14. 当对话框弹出后, 选择要使用的图像文件。(在本教程中, 使用位于 “<InstallDir>/Help/Tutorial/ProcessNet/VSTA/Excavator/Excavator” 目录下的 **Excavator\_1.png** 文件。)
15. 确认图像正确后, 点击 **OK** 按钮。
16. 在 **Properties** 框中, 将 **SizeMode** 设为 **AutoSize**, 并在 **Location** 栏中输入 0, 0。

Location	0, 0
Locked	False
Margin	3, 3, 3, 3
MaximumSize	0, 0
MinimumSize	0, 0
Modifiers	Private
Padding	0, 0, 0, 0
Size	460, 761
SizeMode	AutoSize

17. 选择 **ToolBox**。
18. 在 **Common Controls** 列表中，选择 **Label**，拖拽标签到对话框左上角。
19. 选择创建的 **Label**。然后，在 **Properties** 框中的 **Text** 栏中输入 **CAD File Patch**，**Location** 栏中输入 12，17。

Location	12, 17
Locked	False
Margin	3, 0, 3, 0
MaximumSize	0, 0
MinimumSize	0, 0
Modifiers	Private
Padding	0, 0, 0, 0
RightToLeft	No
Size	83, 12
TabIndex	1
Tag	
Text	CAD File Patch

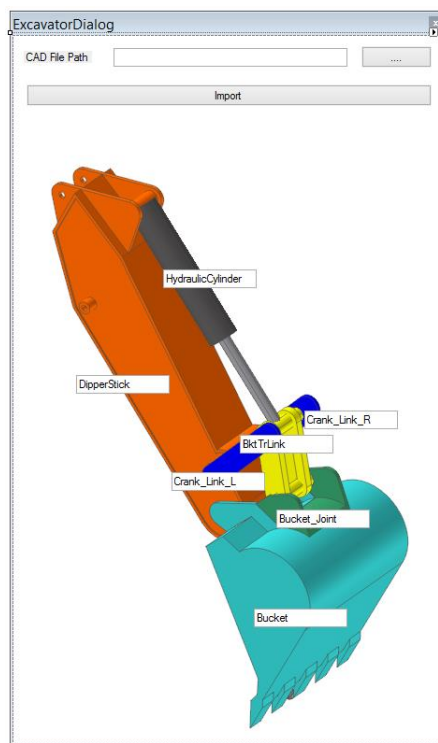
20. 再次选择 **ToolBox**。点击并拖拽 **Common Controls** 列表中的 **TextBox** 到 **Label** 的右侧。
21. 在窗口右上角的 **Textbox1** 的 **Properties** 框中，在 **Location** 栏中输入 108，14，在 **Name** 栏中输入 **tbPath**，**Size** 栏中输入 260，21。
22. 对于 **Button1**，**Button2**，**TextBox1**，**TextBox2**，**TextBox3**，**TextBox4**，**TextBox5** 和 **TextBox6**，重复上述步骤，根据下表更改 **Text** 和 **Name** 值。

对话框单元	文本	名称	位置	大小
<b>Button1</b>	...	<b>btSearchPath</b>	373, 12	75, 23
<b>Button2</b>	<b>Import</b>	<b>btImport</b>	15, 53	433, 23
<b>TextBox1</b>	<b>HydraulicCylinder</b>	<b>tbHydraulicCylinder</b>	161, 251	110, 20
<b>TextBox2</b>	<b>DipperStick</b>	<b>tbDipperStick</b>	68, 364	110, 20
<b>TextBox3</b>	<b>Crank_Link_L</b>	<b>tbCrank_Link_L</b>	312, 402	100, 20

TextBox4	Crank_Link_R	tbCrank_Link_R	170,468	100,20
TextBox5	BktTrLink	tbBktTrLink	243,428	100,20
TextBox6	Bucket_Joint	tbBucket_Joint	282,508	100,20
TextBox7	Bucket	tbBucket	258,614	100,20

23. 完成上述数值的设置后，对话框应如下图所示。

24. 打开 **File** 菜单，并点击 **Save ExcavatorDialog.cs** 来保存文件。

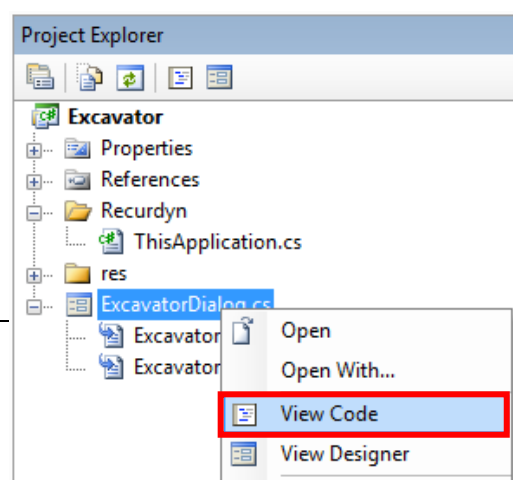


*注意：对话框的大小或位置可能会因 PC 环境的不同而存在差异。*

## 配置对话框的初始设置

前面的步骤设计了对话窗口的框架。现在必须为对话框增加变量来定义可以输入什么样的值，以及点击按钮后会触发的事件。此外，还将学习如何在对话框中使用 **ProcessNet** 的函数。

配置对话框初始设置：

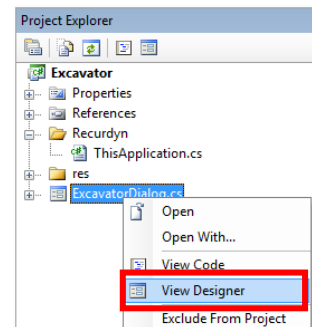
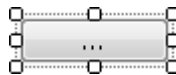


1. 在 **Project Explorer** 中，右键点击 **ExcavatorDialog.cs**，点击 **View Code**，以在 **Edit IDE Project** 窗口中显示 **ExcavatorDialog.cs** 的源代码。
2. 在 **Edit IDE Project** 窗口中，输入对话框中要使用的变量。
  - **FunctionBay.RecurDyn.ProcessNet** 提供了用于 **ProcessNet** 函数的参考信息。
  - **IApplication** 是用于识别 **RecurDyn** 的界面。
  - 当 **Excavator** 不存在时，**strFilePath** 和 **StrExcavatorPartName** 是用于显示 **CAD** 文件或子程序文件使用路径的字符串。

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

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

3. 在 **Project Explorer** 中，右键点击 **ExcavatorDialog.cs**，然后点击 **View Designer**，显示在上一步骤中所创建的对话框。
4. 在对话框中，双击...按钮，创建双击按钮时调用的函数。
5. 插入下列代码来自动创建函数。(输入粗体文字。)



- 此代码从 **Folder** 对话框导入文件路径。
- 点击...按钮来执行 **btSearchPatch\_Click()**函数并创建 **Folder** 对话框。

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



6. 在 **Project Explorer** 中，右键点击 **ExcavatorDialog.cs** 并点击 **View Designer**。在对话框口中，双击 **Import** 按钮来创建 **btImport\_Click()** 函数。
7. 在 **btImport\_Click()** 函数下，创建一个名为 **UpdateDB()** 的新函数并键入下列代码。
  - **UpdateDB()** 函数存储了对话框文本框中输入的变量。
  - **This.tbPath.text** 程序存储了 **tbPath** 文本框中输入的数值。
  - **strExcavatorPartName** 是一个存储了 CAD 文件或子系统文件的名称及路径的二维数组。

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

8. 插入下列代码，自动创建函数。

本代码将在点击对话框中的 **Import** 按钮后执行 **UpdateDB()** 函数。

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

9. 在 **File** 菜单中，点击 **SaveExcavatorDialog.cs** 来保存文件。

## 当运行程序时显示对话框

本节说明如何在运行一个 **RecurDyn** 中的 **ProcessNet** 应用时显示对话框，以及如何使对话框依赖于 **RecurDyn**。

当运行应用时显示对话框：

1. 在 **Project Explorer** 中，双击 **ThisApplication.cs**。
2. 在 **ThisApplication.cs** 文件中，删除用删除线标记的 **HelloProcessNet()** 和 **CreateBodyExample()** 函数，如下所示。(这些函数是作为范例被生成的。)

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

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

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

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

3. 将 **Run()** 函数编写如下。
  - 该函数创建了一个 **ExcavatorDialog** 的新实例。
  - 它将应用和主窗口的数值传递给 **ExcavatorDialog** 类。

---

**注意：**应用和主窗口的数值必须传递给这个类才能使用 **WinForms** 中的 **ProcessNet** 方法。

---

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

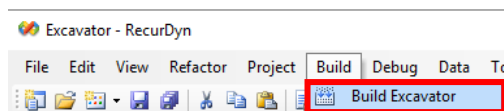
4. 在 **File** 菜单中点击 **Save This Application.cs** 来保存文件。

## 测试对话框

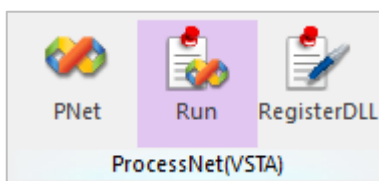
本节测试刚才创建的应用能否正确运行。

运行应用:

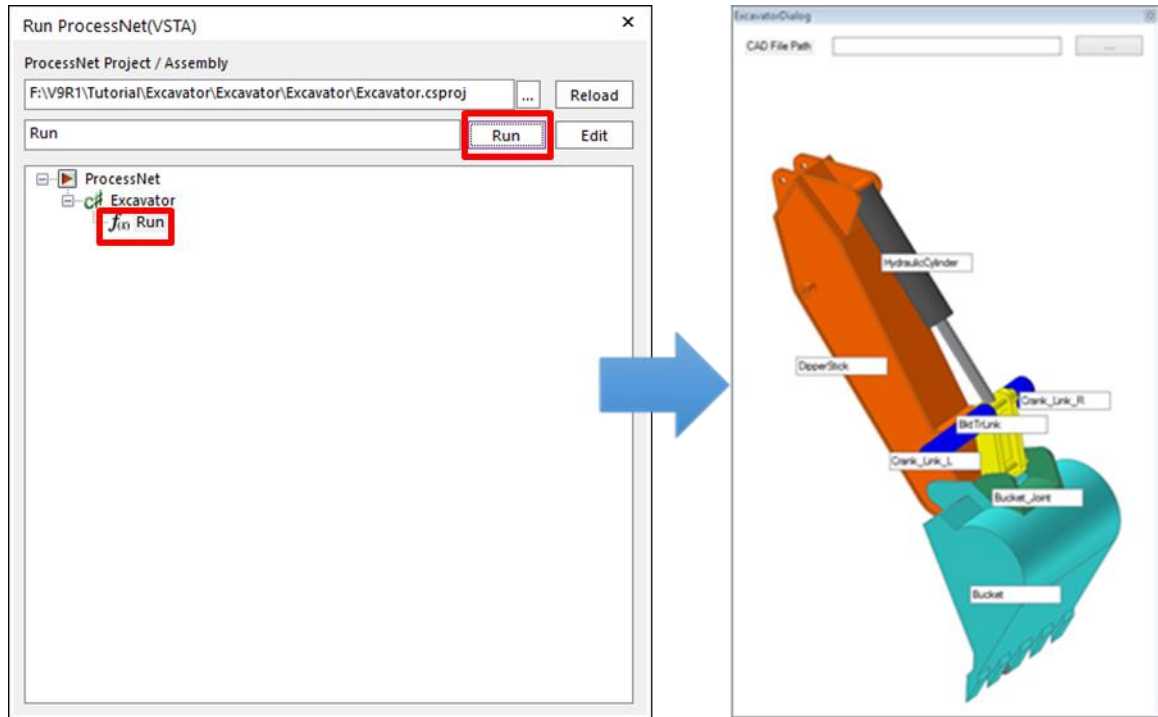
1. 检查 IDE 窗口底部的 **Error List** 框中是否显示了任何错误或警告。如果有的话，改正错误。在 **Build** 菜单中点击 **Build Excavator**。



2. 在 **RecurDyn** 的 **Customize** 标签中的 **ProcessNet** 组下，点击 **Run**。



3. 在 **Run ProcessNet** 对话框下半部的树状图中，点击 **Excavator** 下的 **Run**。
4. 在 **Run ProcessNet** 对话框中，点击 **Run** 按钮。



5. 创建的对话框会弹出。
6. 一旦确认应用正确运行，关闭对话框。
7. 关闭 **Run ProcessNet** 对话框。

## Chapter

## 4

## 通过代码自动生成模型

### 任务目标

本章创建一个新的类，并在该类下编写一个**ProcessNet**函数。随后，学习如何在前一章中创建的对话框中调用此函数。



### 预计完成本任务的时间

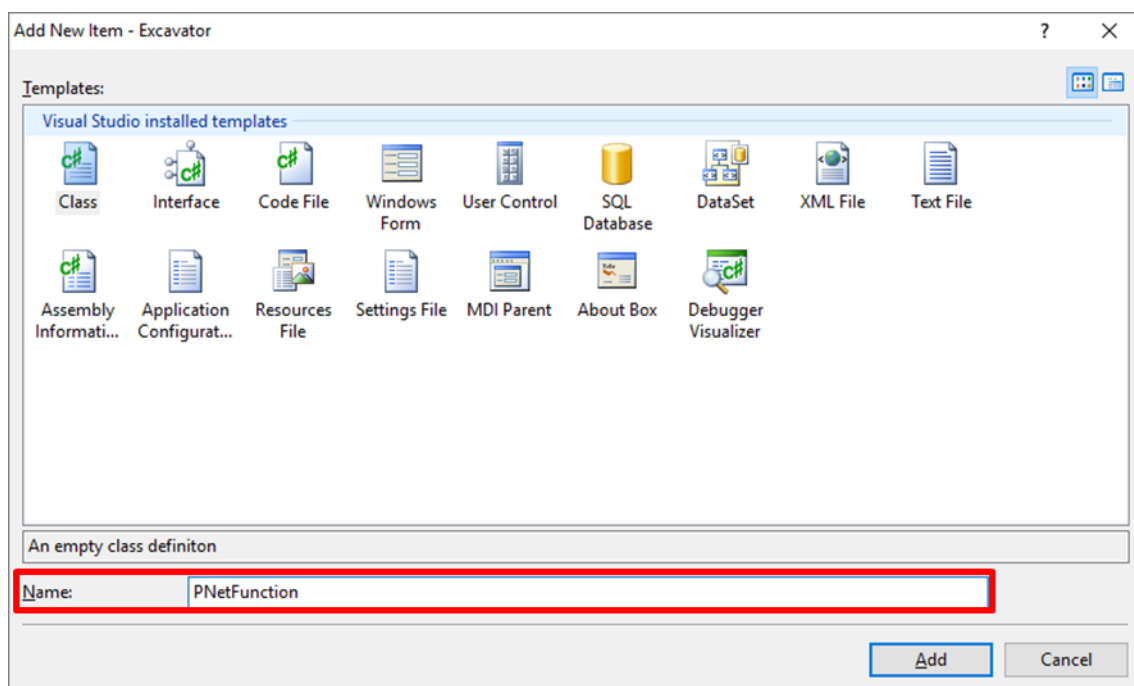
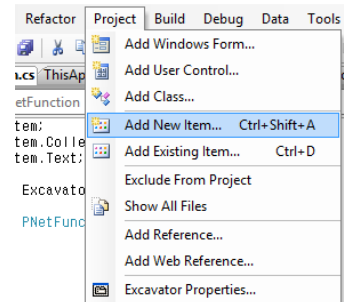
30 分钟

## 创建一个新的类

本节说明如何创建一个新的类，并在该类下输入一个 **ProcessNet** 函数。

### 创建一个新类：

1. 在 **ProcessNet IDE** 中，点击 **Project - Add New Item**。
2. **Add New Item** 对话框弹出后，在 **Templates** 框中，选择 **Class**，并在 **Name** 栏中输入 **PNetFunction**。点击 **Add** 按钮。



3. **Edit IDE Project** 框中出现 **PNetFunction.cs** 后，输入下列代码。(输入粗体文字。) 该代码会重置用于执行 **ProcessNet** 函数的基础变量。
  - **IApplication**: 用于识别 **RecurDyn**
  - **IModelDocument**: 在 **RecurDyn** 中使用的一个模型文件
  - **ISubsystem**: 模型文件中使用的一个子系统
  - **IReferenceFrame**: **RecurDyn** 的一个参考系
  - **IPlotDocument**: 一个 **RecurDyn** 绘图文件

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

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

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

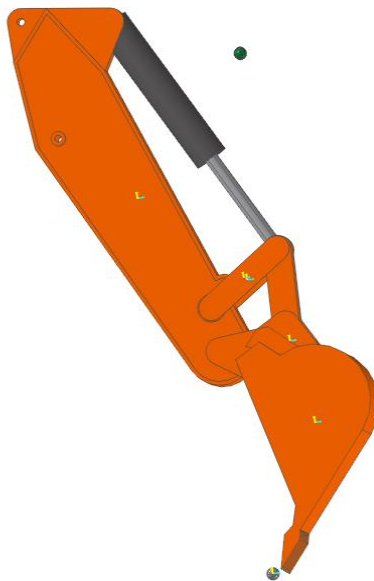
4. 在 **File** 菜单中，点击 **Save PNetFunction.cs** 来保存文件。

## 创建一个模型

本节学习如何自动创建一个挖掘机模型。

导入一个部件:

1. 编写代码，用一个 **CAD** 文件和子系统文件创建一个挖掘机模型。



2. 创建如下 `Import()` 函数。

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

3. 在 `Import()` 函数中，插入如下变量声明代码。

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

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

4. 声明变量后，加入如下循环来导入 `*.x_t` 和 `*.rdsp` 文件。`*.x_t` 文件是用于模型不同部件的 CAD 文件，`*.rdsp` 文件则是子系统文件。

- `FileImport()` 会导入 CAD 和子系统文件。
- `Crank Link R` 会创建一个标记，没有导入 CAD 文件。因此，连续语句会在下一个循环中使用。

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

5. 输入如下代码，利用 `SubSystemCollection` 的属性在模型中编写一串子系统，并将其声明为 `Sub01`，`Sub02`，以此类推。

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

6. 使用 `GetEntity()` 函数，查找导入的部件。

- `GetEntity()` 函数会查找 `RecurDyn` 中的实体。
- 通常函数会得到 `IGeneric` 的返回值。该函数还可以进行类型转换，生成用户希望得到的元素类型。



```

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

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

```

7. 输入如下代码，在名为 **Crank\_Link\_R** 的连接部件上生成一个标记，生成 **Bucket\_Joint** 部件。

- **Crank\_Link\_R** 会创建一个标记，而不用导入 CAD 文件，这样就可以通过在第二个变量中输入参数点控制模型的位置。

```

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

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

```

8. 将总成模型的地面声明为 **bodyground**。

```

IBodyBodyGround = model.Ground;

```

9. 创建一个 dummy body。

```

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

```

10. 在 **File** 菜单中，点击 **SavePNetFunction.cs**，保存文件。

## 创建一个 SubEntity:

1. 本节创建一个用于挖掘机模型的 SubEntity。
2. 在前一步骤中创建的 Import() 函数中，加入如下代码来创建参数值。

(所有用于创建 SubEntity 和创建变量方程的代码，都应依次连续加入到 Import() 函数中，并保持相同的缩进。)

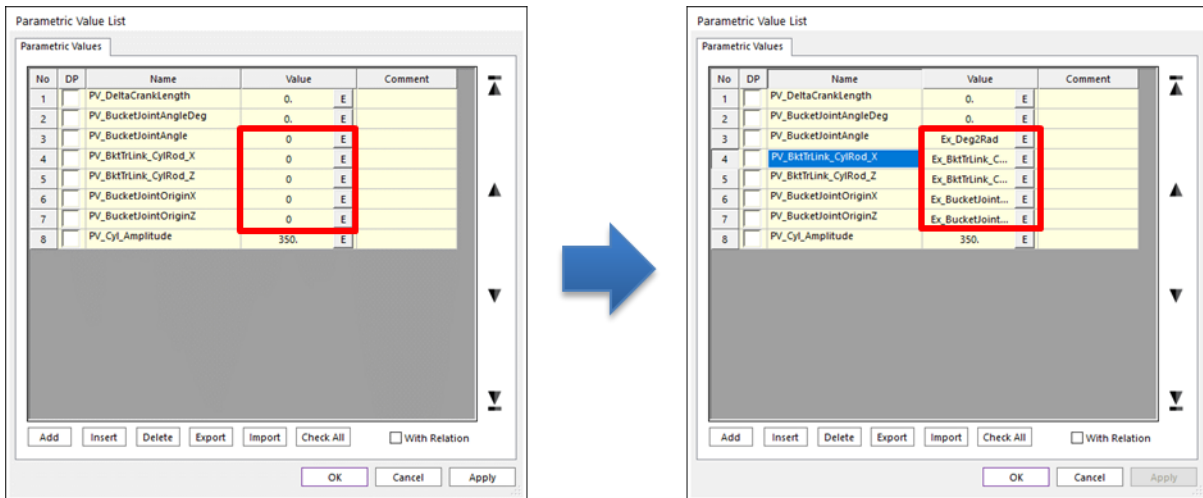
```
IParametricValue PV_DeltaCrankLength = model.CreateParametricValue("PV_DeltaCrankLength", 0);
IParametricValue PV_BucketJointAngleDeg = model.CreateParametricValue("PV_BucketJointAngleDeg", 0);
IParametricValue PV_BucketJointAngle = model.CreateParametricValue("PV_BucketJointAngle", 0);
IParametricValue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

3. 在创建参数值的代码下，添加创建如下表达式的代码。

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

4. 在创建每个表达式的代码下，添加生成参数值的代码。

- 创建完表达式后，输入**参数值**。当下列代码被执行后，**参数值**就会变为如下所示的表达式。



```

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

```

5. **参数值**确定后，添加如下代码创建**参数点**。

```

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

```

6. 在创建**参数点**的代码下，添加创建**PPC**和**PVC**的代码。

```

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

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

```

7. 在创建 PPCs 和 PVCs 的代码下，添加如下代码，更改 Crank\_Link\_R 的第二个点和法线方向。

```

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

```

8. 在 File 菜单中，点击 SavePNetFunction.cs 来保存文件。

## 创建一个运动副:

1. 本节创建用于挖掘机模型的运动副。

在前面步骤中创建的 `Import()`函数中, 加入如下代码, 创建固定运动副。

```

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

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

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

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

```

## 2. 添加如下代码，创建旋转运动副。

```

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

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

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

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

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

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

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

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

```

3. 在 **File** 菜单中, 点击 **SavePNetFunction.cs** 来保存文件。

创建一个力:

4. 本节创建用于挖掘机模型的力。
5. 在前面步骤中输入的代码下, 输入下面的代码来创建力。

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

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

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

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

6. 在 **File** 菜单中, 点击 **SavePNetFunction.cs**, 保存文件。

## 创建一个变量方程:

1. 本节创建用于挖掘机模型的变量方程和要求。
2. 在前面步骤中输入的代码下，输入下列代码来创建变量方程和要求。
  - `model.Redraw()`函数重新生成了工作框中的图形。

```

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

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

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

model.Redraw();

```

3. 在 **File** 菜单中，点击 **SavePNetFunction.cs** 来保存文件。
4. 在 **Build** 菜单中，点击 **Build Excavator** 来执行创建。检查 IDE 窗口底部的 **Error List** 框中是否显示了任何错误或警告。如果有的话，改正错误。



## 将函数关联到对话框

本节学习在点击对话框的 **Import** 按钮时，如何调用 **Import()**函数。

将函数关联到对话框：

1. 在 **Project Explorer** 中，右键点击 **ExcavatorDialog.cs**。
2. 在菜单中，点击 **View Code**。
3. 输入如下代码。(输入粗体文字。)
  - 创建一个调用 **Import** 功能的 **PNetFunction** 实例，。

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

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

4. 在 **btImport\_Click()**函数中，输入下列代码，用 **PNetFuction** 实例来调用 **Import()**函数。(输入粗体文字。)

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

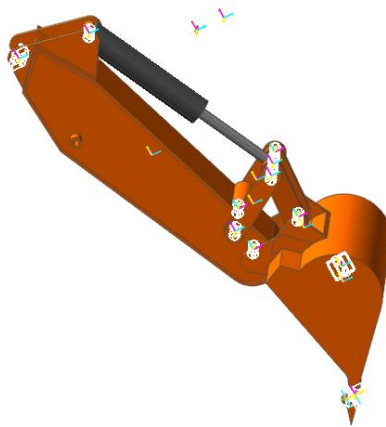
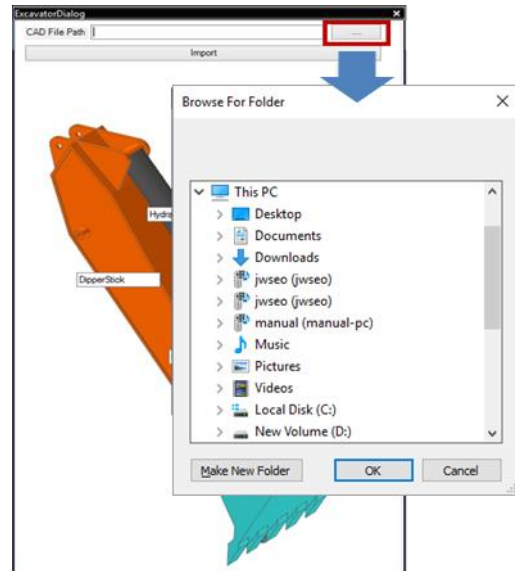
5. 在 **File** 菜单中，点击 **SaveExcavatorDialog.cs**，保存文件。

## 测试对话框

本节测试所创建的应用能否正确运行。

### 运行应用:

1. 在 **Build** 菜单中，点击 **Build Excavator**。检查 IDE 窗口底部的 **Error List** 框中是否显示了任何错误或警告。如果有的话，改正错误。
2. 在 **RecurDyn** 的 **Customize** 标签中的 **ProcessNet** 组下，点击 **Run**。
3. 在 **Run ProcessNet** 对话框下半部的树状图中，点击 **Excavator** 下的 **Run**。
4. 在 **Run ProcessNet** 对话框中，点击 **Run** 按钮。
5. 对话框会弹出，如右图所示。
6. 在对话框中点击...按钮。
7. **Browse For Folder** 对话框弹出后，指定文件导出的路径。(在本教程中，文件位于“<InstallDir>/Help/Tutorial/ProcessNet/VSTA/Excavator/Excavator”目录下。)
8. 确认文件路径已在 **CAD** 文件路径中输入后，点击 **Import** 按钮。
9. 挖掘机模型会自动出现，如下图所示。
10. 点击对话框右上角的×按钮，关闭挖掘机对话框。
11. 关闭 **Run ProcessNet** 对话框。



Chapter

5

## 分析模型

### 任务目标

本章创建一个函数，实现当在对话框中更改元素值时，将更改后的值应用于模型，同时还将学习如何在对话框中执行模型分析。



预计完成本任务的时间

10 分钟

## 编辑对话框的布局

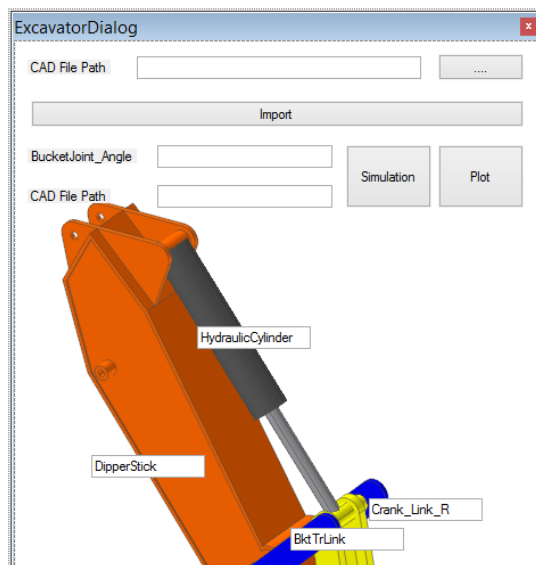
本节在对话框中增加一个文本框和一个按钮，使可以在对话框中执行模型分析和绘图。

编辑对话框的布局：

1. 在 **Project Explorer** 中，双击 **ExcavatorDialog.cs**。**ExcavatorDialog.cs** 对话框会在 **Edit IDE Project** 框中出现。
2. 选择 **ToolBox**，并将如下控件加入 **Common Controls** 列表。然后更改这些控件的值。

对话框元素	文本	名称	位置	大小
<b>Button1</b>	<b>Simulation</b>	<b>btSimulation</b>	292, 90	75, 57
<b>Button2</b>	<b>Plot</b>	<b>btPlot</b>	373, 90	75, 57
<b>TextBox1</b>		<b>tbBucketJointAngle</b>	126, 92	154, 20
<b>TextBox2</b>		<b>tbCrankLength</b>	126, 127	154, 20
<b>Label1</b>	<b>BucketJoint_Angle</b>	<b>lbBucketJointAngle</b>	12, 95	83, 12
<b>Label2</b>	<b>Crank_Length</b>	<b>lbCrankLength</b>	12, 130	83, 12

3. 在 **File** 菜单中，点击 **SaveExcavatorDialog.cs** 来保存文件。



## 模型分析功能

本节创建一个函数，实现当在对话框更改液压缸长度和挖斗角度时，将更改后的值应用于模型。当点击相应按钮时，代码会执行模型分析。

### 模型分析功能

1. 在 **Project Explorer** 中，双击 **PnetFunction.cs**。
2. 在前面章节中创建的 **Import()** 函数下，创建一个仿真函数。
  - 输入如下代码，用对话框中的 **Bucket Joint Angle** 和 **Crank Length** 值，更改 **PV\_DeltaCrankLength** 和 **PV\_BucketJointAngleDeg** 的参数值。

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

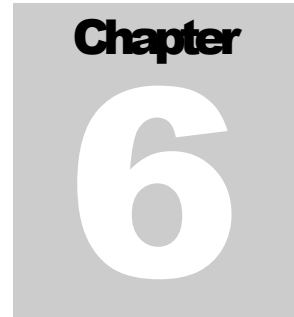
    IParametricValue PV_DeltaCrankLength =
    model.GetEntity("PV_DeltaCrankLength") as IParametricValue;
    IParametricValue PV_BucketJointAngleDeg =
    model.GetEntity("PV_BucketJointAngleDeg") as IParametricValue;
    PV_BucketJointAngleDeg.Value = dPVValue[0];
    PV_DeltaCrankLength.Value = dPVValue[1];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

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

3. 在 **Project Explorer** 中，右键点击 **ExcavatorDialog.cs**，并点击 **View Designer**。
4. 在双击 **Simulation** 按钮创建出的函数中，输入如下代码。

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

5. 在 **File** 菜单中，点击 **SaveExcavatorDialog.cs** 来保存文件。



## 自动创建绘图

### 任务目标

本章学习 ProcessNet 中用于绘图的命令。



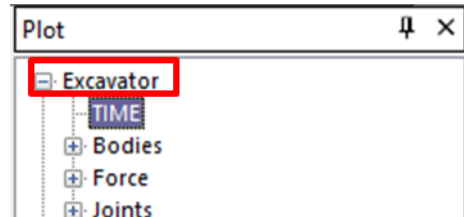
预计完成本任务的时间

10 分钟

## 绘图函数

使用绘图函数:

1. 在 **Project Explorer** 中, 双击 **PnetFunction.cs**。
2. 在前面章节中创建的 **Simulation()** 函数中, 创建如下绘图函数。



- 对 **GetPlotData** 来说, “EXCAVATOR”是绘图的数据来源, 这个数据来源可能会随 **RecurDyn** 版本的不同而有所差别。
- 用 **ActivateView** 功能来确定视图。

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

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

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

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

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

3. 在 **Project Explorer** 中, 右键点击 **ExcavatorDialog.cs**, 并点击 **View Designer**。
4. 双击 **Plot** 按钮。
5. 在所创建的函数下, 输入如下绘图函数。

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

6. 在 **File** 菜单中，点击 **SaveExcavatorDialog.cs** 来保存文件。

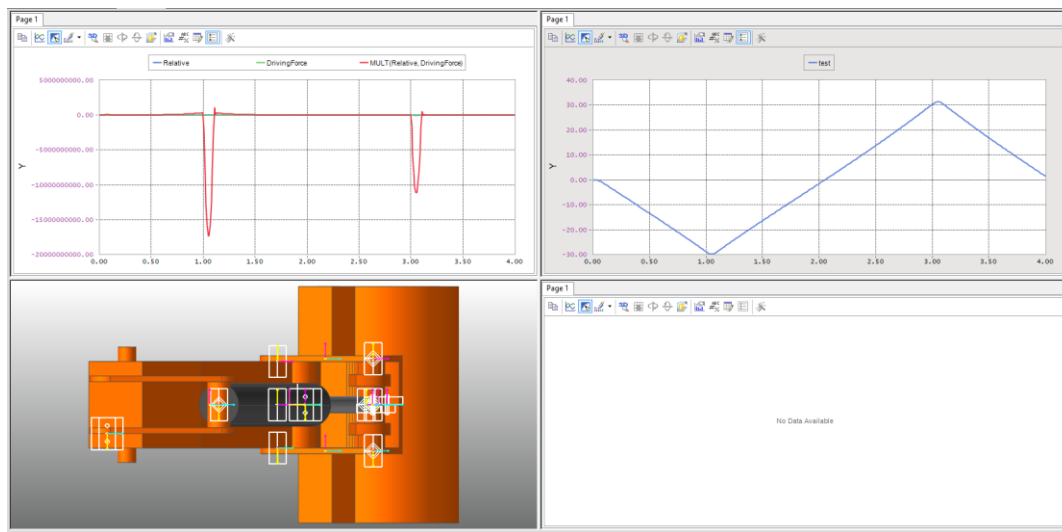
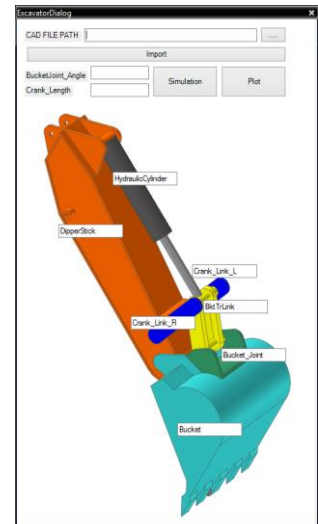
## 测试所创建的应用

本节测试所创建的应用能否正确运行。



## 运行应用:

1. 在 **Build** 菜单中, 点击 **BuildExcavator**。检查 IDE 窗口底部的 **ErrorList** 框中是否显示了任何错误或警告。如果有的话, 改正错误。
2. 在 **RecurDyn** 的 **Customize** 标签中的 **ProcessNet** 组下, 点击 **Run**。
3. 在 **RunProcessNet** 对话框窗口下半部的树状图中, 点击 **Excavator** 下的 **Run**。
4. 在 **RunProcessNet** 对话框窗口中, 点击 **Run** 按钮。
5. 对话框窗口会弹出, 如右图所示。
6. 在对话框窗口中, **BucketJoint\_Angle** 和 **Crank\_Length** 值都输入 0。
7. 点击仿真按钮, 确认 **PV\_DeltaCrankLength** 和 **PV\_BucketJointAngleDeg** 的值会随着输入的 **BucketJoint\_Angle** 和 **Crank\_Length** 值而改变, 同时根据新的值进行模型分析。
8. 分析完成后, 点击 **Plot** 按钮, 绘制如下所示的图。



9. 关闭 **Excavator** 对话框窗口。
10. 关闭 **Run ProcessNet** 对话框窗口。

感谢学习本教程!