Tekla® Open API

Self Learning Exercises

Tekla Structures 2016

Contents

Preface

This material is targeted at developers, who have very little experience in Tekla Open API. This includes exercises, which have certain amount of guidance, but there's lot of room for self learning and discovering different things from the Tekla Open API.

1 Self learning material for Tekla Open API of Tekla Structures

Purpose and structure of exercises

This self learning material contains exercises to help you learn and practice basic concepts of Tekla Open API (the API). Since it is not possible to cover every conceivable issue developer might face, the exercises work through some typical cases faced by many. Once you become familiar with the API you may develop your own application for other cases.

This exercise is divided into several smaller exercises, which build up into an application that creates a small building and later produces drawings from that. The exercises contain some exercises for modeling and for drawings. Also usage of dialogs is included.

In each exercise, you will see an image of the end result in Tekla Structures and you can get the example code too. The same example code is available in the beginning of the next exercise. That way you can use the sample code base, if your own code is too different to continue easily the exercises

Reference Manual

Before starting the exercises, it is strongly suggested you read the reference manual of the Open API. The reference manual **TeklaOpenAPI_Reference.chm** is included in Open API Start-up package, and the manual is available also in folder ..\Tekla Structures\[version]\nt\help\enu\.

The reference manual is very valuable resources when building new software using the API. There are a lot of good examples on how to use different objects and methods. The reference manual should be used throughout these exercises.

The following chapters include exercises for self learning the API. In Chapter 1.1, there is an exercise on how to set up a new project in Microsoft Visual Studio and how to take the API into use. Modeling and dialog exercises are in Chapter 2 and drawing exercises in Chapter 3.

1.1 Create a new project in Visual Studio

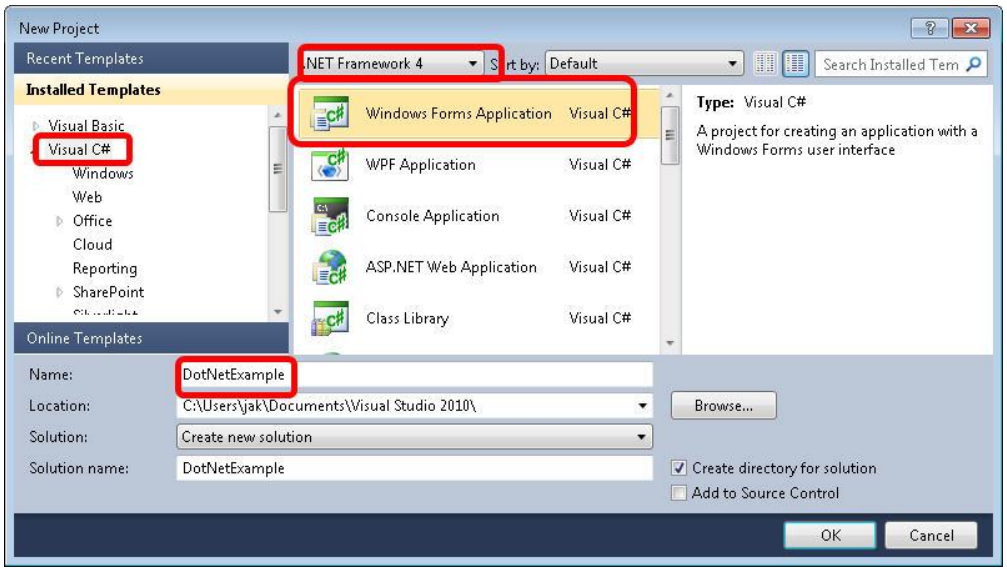
You will learn what you need to do to use the API in your project.

Precondition

You need to have Tekla Structures running and a model opened, preferably an empty model, when you want to test your application.

1. Create a new .NET application project.

The first step to take is to start Microsoft Visual Studio and create a new project (**New -> Project** from the **File** menu).

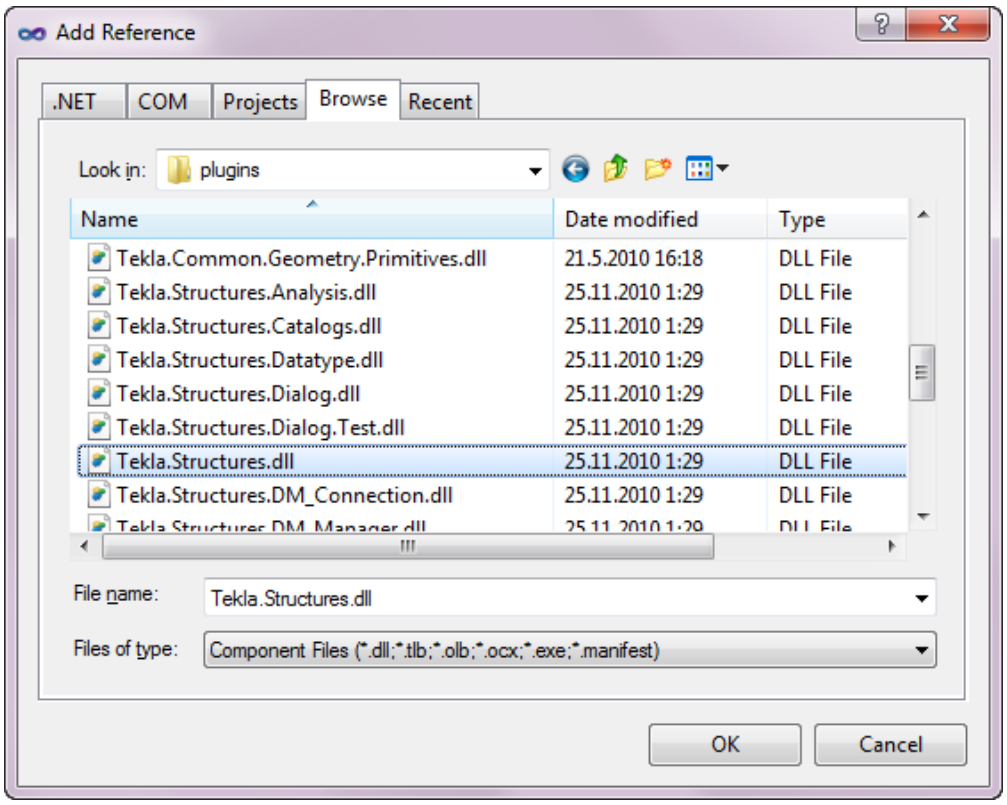


Fill out the details as shown in the screenshot and press **OK** to create an empty windows application.

2. Add references to the API assemblies.

Once you have created the project, you need to add a reference to the Tekla Open API assemblies you will use, like **Tekla.Structures.Model.dll** and **Tekla.Structures.dll**. Adding these references will enable you to use objects included in those assemblies.

To add the references you can either right-click on **References** in the Solution Explorer and select **Add Reference** from the pop-up menu, or you can select **Add Reference…** from the **Project** menu.



Click **Browse…** in the **Add Reference** menu to locate the Tekla.Structures.Model.dll and Tekla.Structures.dll files. You will find these files in the **\nt\bin\plugins** folder of your Tekla Structures version. Once you have located these select them both and select **Open** and press **OK** in the **Add Reference** dialog.

In the exercises you will need these references:

- Tekla.Structures.dll
- Tekla.Structures.Model.dll
- Tekla.Structures.Drawing.dll
- Tekla.Structures.Dialog.dll (found from \nt\bin\dialogs\)
- Tekla.Structures.Catalogs.dll

If you want to create a plug-in for Tekla Structures, then you need to add Tekla.Structures.Plugin.dll.

From now on you are able to use the classes and methods of the API in your project.

3. Add directives to the namespaces of the API assemblies.

View the code of Form1.cs (right-click on the form and select **View code**) and add the *directive* lines to the beginning of the code:

```
using Tekla.Structures.Model;
using TSG3D = Tekla.Structures.Geometry3d;
```

Now you are ready to start the development of your application using the API.

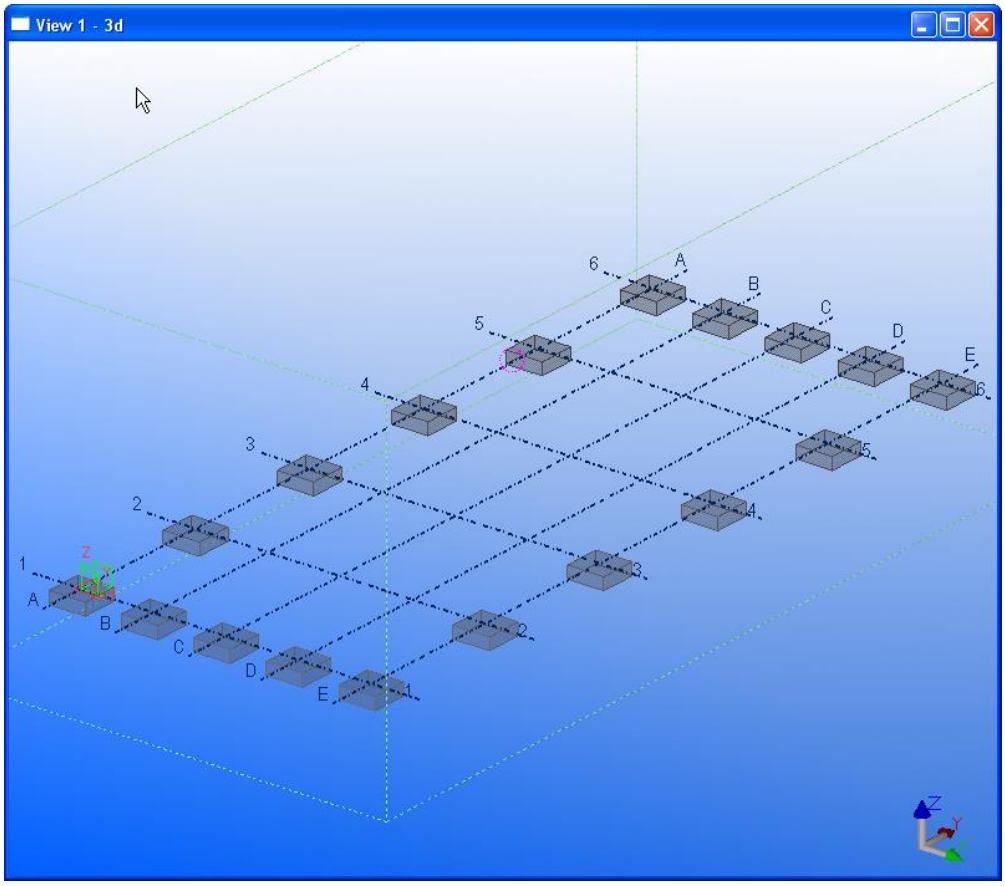
2 Exercises for the Modeling API

Here are the topics of the modeling exercises:

2.1 Create pad footings

You will learn how to create basic objects in Tekla Structures through the API.

The result after this exercise should look like this



Precondition

You need to have Tekla Structures running and an empty model opened. Use this model, so that the grid spacing is the same as used in the example:

Exercise-model.zip

Please remember to read the hints on this page.

Read and learn from Chapter 1.1 about creating new a project in Microsoft Visual Studio.

Create application that creates pad footings

Add a button "Create Pad Footings" to the application form and pad footing creation to Click-event.

Create pad footings on each grid intersections on grid lines A, E, 1 and 6. So create 18 pad footings in total.

Hints:

- Pad footing can be created using the Beam class.
- You should make a method that creates one pad footing to a coordinate that is given to it.
- Then you can use a *for* loop to create the positions for the pad footings.

After the creation a Model instance, you should check that the connection is valid by calling Model.GetConnectionStatus(). If it returns true, then you can use the API.

Help:

See at least following topics from the Reference Manual:

- Beam class

Explanation:

A brief explanation about what code should do:

1. Create a new Model object that represents the Tekla Structures model you have opened in Tekla Structures.
2. Check if you have a Tekla Structures model open that you can connect to.
3. Create a method that creates and inserts one pad footing based on the two input positions.

   Pad footing should have the following properties set up:

Name = "FOOTING"
Profile.ProfileString = "1500*1500"
Material.MaterialString = "K30-2"
Class = "8"
StartPoint
EndPoint (change Z coordinate 500.0 mm lower than in startpoint)
Position.Rotation = Position.RotationEnum.FRONT
Position.Plane = Position.PlaneEnum.MIDDLE
Position.Depth = Position.DepthEnum.MIDDLE

   Please note, that these values might need to be localized and profile and material needs to be found in environment.
4. Create two *for* loops to cycle through the X and Y directions of the positions and create pad footings with the method mentioned in step 3.
5. *CommitChanges()* makes sure all changes that have been done are updated in Tekla Structures and the model view is updated accordingly. If this command isn't called, then you will have to refresh the view manually in Tekla Structures to see the changes.

After this has been done, all you need to do is make sure you have a Tekla Structures model opened and choose **Start** from the **Debug** menu in Visual Studio to run your project. Your form with the **Create Pad Footings** button should now appear and pressing it should place pad footings at the grid intersections of the model.

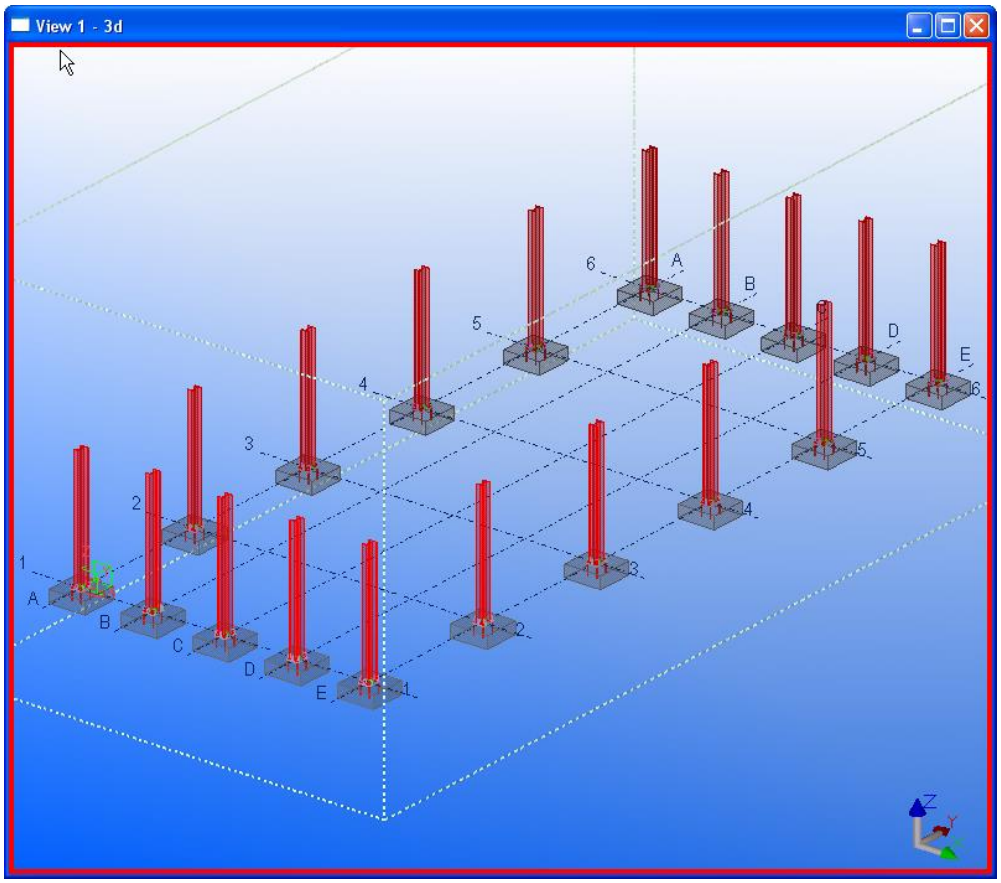
Code

The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise2.1-PadFootings**.

2.2 Create columns on top of the pad footings and connect columns to the pad footings

You will learn how to create connections to the model.

The result should look like this



Precondition

You need to have Tekla Structures running and an empty model opened.
You will also need code from the first exercise; you can get it from the link below or use your own that you have after exercise 2.1.

Example solution for last exercise

Help:

See at least following topics from the Reference Manual:

- Beam class
- Connection class

Create column on top of each pad footing and connect them with stiffened base plate (1014)

Modify the last exercise so that you add two more methods, one for adding columns on top of the pad footings and the second for adding stiffened base plate (1014) between the pad footings and columns. You need to return the inserted pad footings and columns and then feed them to this method that creates connection between those (you can also use some other architecture for your code).

Explanation:

1. Create a method that creates columns (Beam class) on the pad footings.
2. Connect the columns to the pad footings using the Connection class, stiffened base plate (1014); use the columns as the primary part.

   Use these values for the columns (remember to localize needed values):

```
Name = "Column";
Profile.ProfileString = "HEA400";
Material.MaterialString = "S235JR";
Class = "2";
EndPoint.Z = 5000.0;
Position.Rotation = Position.RotationEnum.FRONT;
Position.Plane = Position.PlaneEnum.MIDDLE;
Position.Depth = Position.DepthEnum.MIDDLE;
```

   Use this attribute for stiffened base plate to enable anchor rods:

```
SetAttribute("cut", 1)
LoadAttributesFromFile("standard")
UpVector = new Vector(0, 0, 1000)
```

A note on the "cut" attribute. It is a option menu in Tekla Structures and it's defined in an .inp file for that particular connection.

Also note that for historical reasons, the value returned by option menus are such that:

- First value (usually 0) means empty value internally
- Second value (usually 1) means value 0
- Third value (usually 2) means value 1.

That's why you need to set that attribute as 1, while in .inp it says that anchor rods have the value 2.

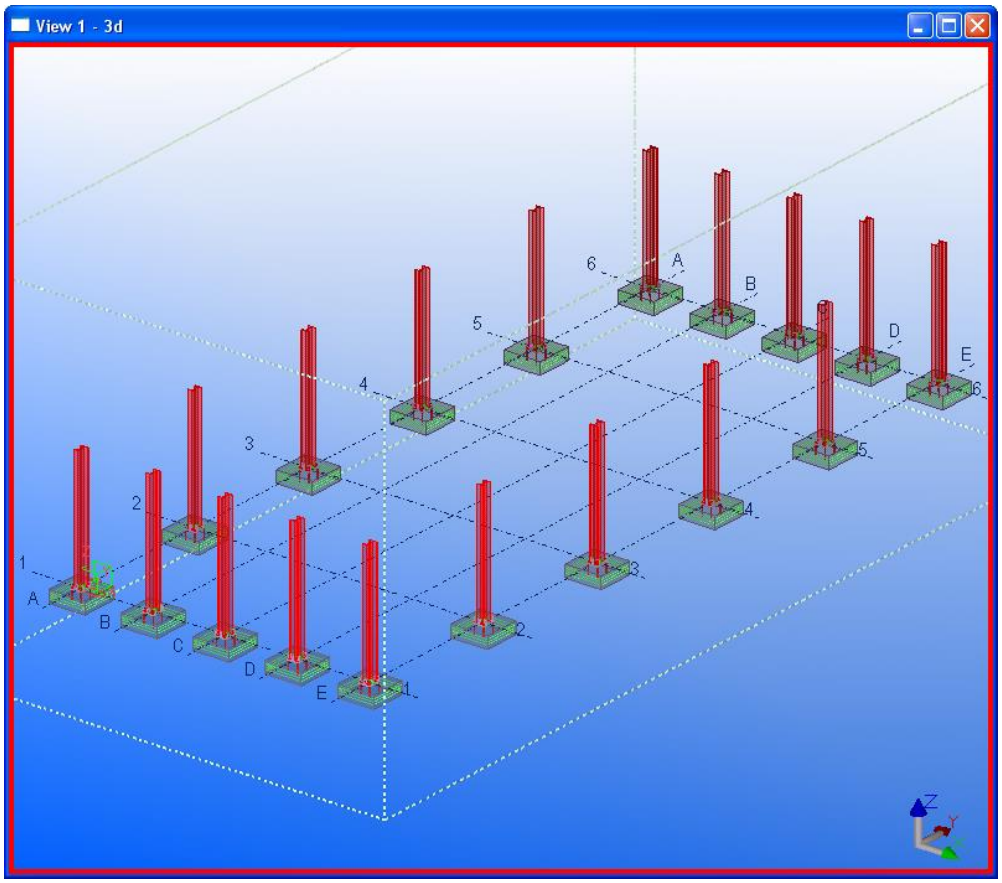
Code

The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise2.2-Columns**.

2.3 Create rebars to the pad footings

You will learn how to select objects from the model and then use an enumerator to loop through those objects.

The result should look like this



Precondition

You need to have Tekla Structures running and a model opened, preferably an empty model.

You also need the source code from exercise 2.2.

Example solution for last exercise

Help:

See at least following topics from the Reference Manual:

- Reinforcement and RebarGroup classes
- Polygon and Point classes

Explanation:

1. Create rebars for each pad footing.
   Create a field in the dialog for user to give the size of the footings.
2. Add a new button to the dialog, that creates rebars using the following method.
3. Select all objects from model to enumerator (ModelObjectEnumerator).
4. Loop through enumerator.

5. If an object is a pad footing, then create rebar group to that using the pad footing's StartPoint as position.
6. Get the size of the footing from the dialog.
7. Create circular rebars around the footing.

   Use these values (remember to localize needed values):

Polygon, use pad footing's corner points as polygon points, remember to give starting point also as last point *(see point 6.)*
StartPoint and EndPoint, use pad footings position and height.
Class = 3
Name = "FootingRebar"
Father, use pad footing.
Grade = "A500HW"
Size = "12"
Radiusvalues.Add(40.0)
SpacingType = SPACING_TYPE_TARGET_SPACE
Spacings.Add(100.0)
ExcludeType = EXCLUDE_TYPE_BOTH
StartNumber = 0
Prefix = "Group"
OnPlaneOffsets.Add(25.0)
FromPlaneOffset = 40

Code

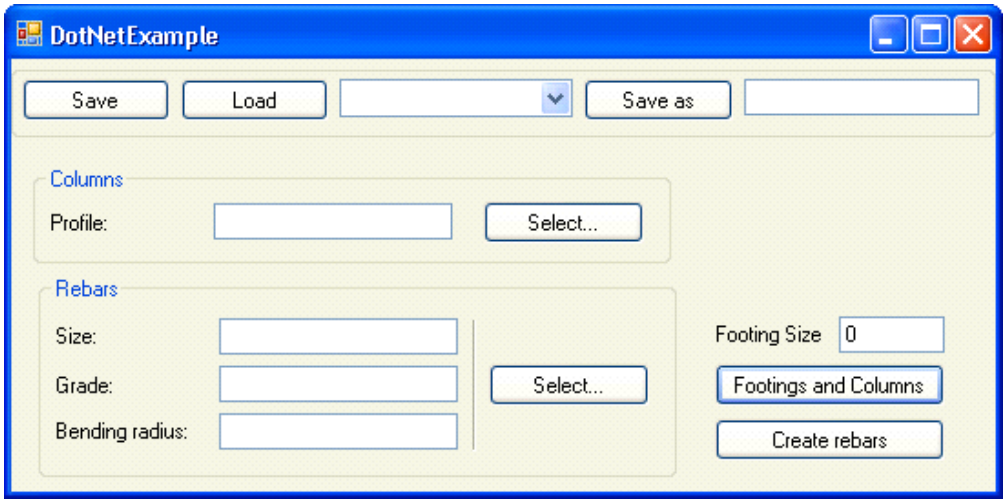
The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise2.3-Rebars**

2.4 Use Catalog UI controls

API contains Forms dialogs for profile catalog, rebar catalog, and mesh catalog to help developers of applications (and plug-ins) that have Forms and need to provide for example profile selection in UI. You will now learn how to use readymade customized user interface controls to select profiles and meshes from the catalogs in Tekla Structures.

You will also learn how to enumerate through the catalogs and customize the list of items shown in the dialog.

After this exercise your application may look like this



Precondition

You need to have Tekla Structures running and a model opened, preferably an empty model.

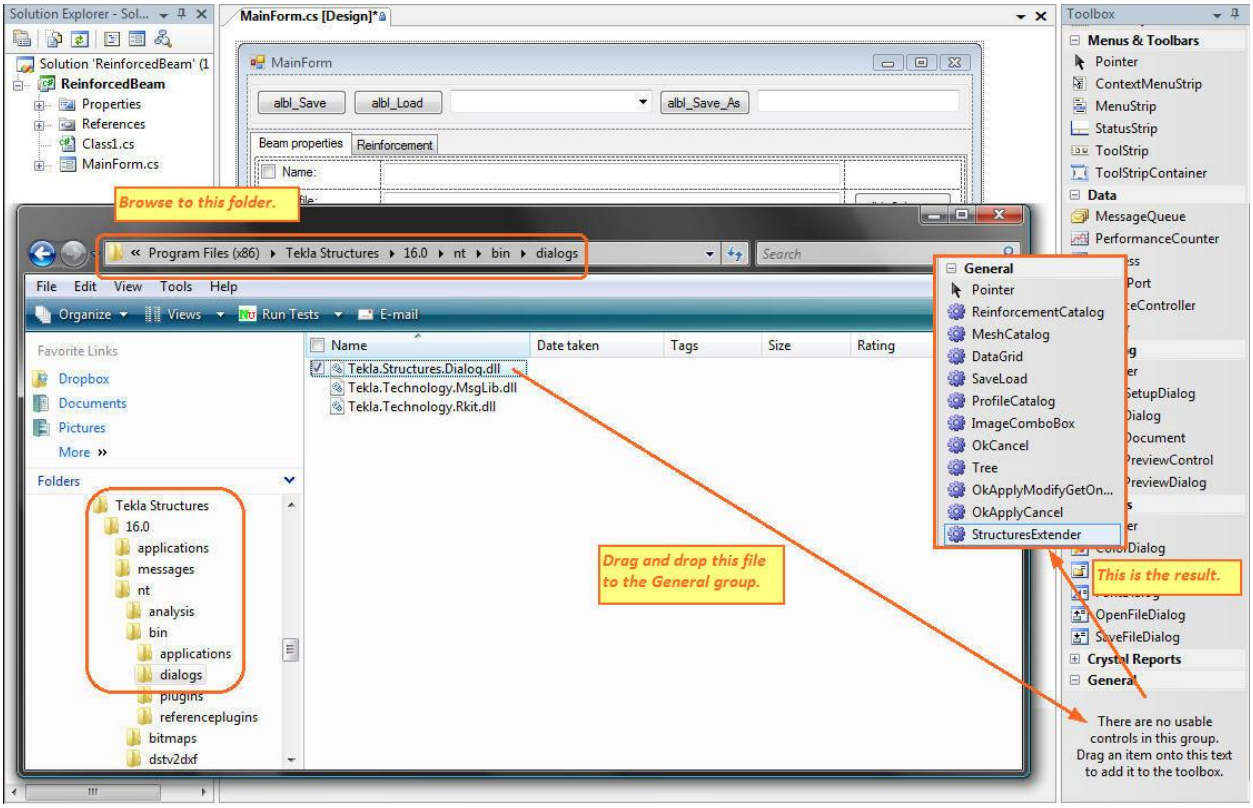
You also need the source code from exercise 2.3.

Example solution for last exercise

Add user interface controls to Visual Studio's Toolbox:

There are two ways to add controls to Visual Studio's Toolbox:

- Right click on the Toolbox and select "Choose items". Then click the browse button to open Tekla.Structures.Dialog.dll. The controls will be added to the list. You can select one control or all the controls to be added to the Toolbox.
- Use Windows Explorer to navigate to the \Program Files\Tekla Structures\[version]\nt\bin\dialogs\Tekla.Structures.Dialog.dll, open the Designer and drag and drop the dll into the General tab of the Toolbox. All the controls will be added to the Toolbox. See the snapshot below.



Help:

See at least following topics from the Reference Manual:

- FormBase class
- ProfileCatalog class
- ReinforcementCatalog Class
- SaveLoad Class

Explanation:

1. In order to get user interface controls to work, your Form must inherit from FormBase.

   Also in the Form's constructor there has to be a call to

   ```
   base.InitializeForm();
   ```

2. Let the user select the profile for the columns. For that you can use the user interface control called "ProfileCatalog". Add this control to your Form. You can drag and drop a control to your Forms dialog.

   The item, that user selects from the dialog, will be stored to a SelectedProfile property.

   Add a text box to show the selected profile.

   When user types a profile to the text box and clicks Select button, the typed profile should be selected automatically from the profile catalog dialog. Setting the property SelectedProfile to a certain value will select the value from the dialog when the dialog opens (if the value is included in the catalog).

3. Add the "ReinforcementCatalog" user interface control to select the rebars for pad footings.

   Add three text boxes to show the size, grade and bending radius of the selected rebar. Use the values of these text boxes also to select values in the reinforcement catalog dialog when the dialog opens.

4. Substitute the hardcoded values for the profile and rebars in the code.

   Rebar bending radius can have several values in the catalog e.g. “20 40” so those values need to be handled by the application. So instead of defining the radius values in code as we did earlier

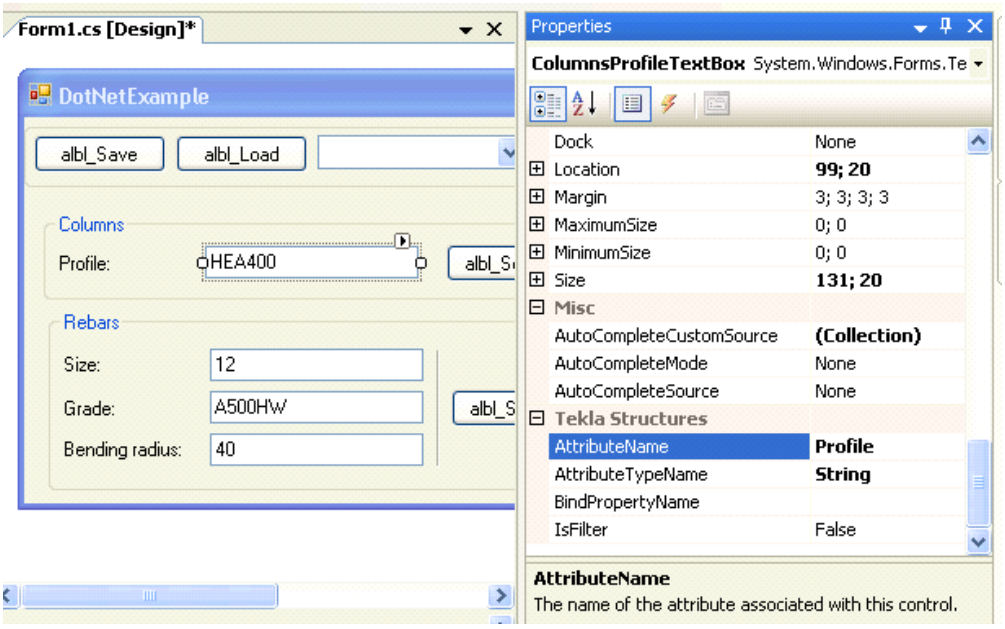
```
Radiusvalues.Add(40.0)
```

   we can ask the values from the user:

```
char[] Separator = {' '};
string[] Radiuses =
                    BendingRadiusTextBox.Text.Split(Separator,
                    StringSplitOptions.RemoveEmptyEntries);
foreach(string Item in Radiuses)
          Rebar.RadiusValues.Add(Convert.ToDouble(Item));
```

5. Add the “SaveLoad” control to your Form.

   Only the values of controls which have are bound to some attribute will be saved, meaning they have “AttributeName” and “AttributeTypeName”. To set those, open the Visual Studio Design view and go to the Properties tab. AttributeName and AttributeTypeName are found under title Tekla Structures:



   AttributeName is the name by which you identify the control.

   AttributeTypeName is the name of Tekla.Structures.Datatype for the control's contents. Possible AttributeTypeNames are Boolean, Distance, DistanceList String, Integer, and Double.

Hints:

When using the user interface controls “ProfileCatalog” and “ReinforcementCatalog” remember to create callbacks for the events “SelectClicked” and “SelectionDone”.

Modify the SelectClicked callbacks of both dialogs. SelectClicked event is raised when user clicks the Select button in your form. Modify the callback so that the values of SelectedProfile, SelectedRebarGrade, SelectedRebarSize, and SelectedBendingRadius are set according the values in the Form, for example:

```
profileCatalog1.SelectedProfile = ColumnsProfileTextBox.Text;
```

The SelectionDone event is raised when the profile or rebar selection dialog is closed. To update the text shown in your Form, use the method SetAttributeValue() and the values stored in SelectedProfile, SelectedRebarGrade, SelectedRebarSize, and SelectedBendingRadius properties. For example:

```
SetAttributeValue(ColumnsProfileTextBox, profileCatalog1.SelectedProfile);
```

To substitute the hardcoded values you need to use values defined in the text boxes. For example:

```
Column.Profile.ProfileString = ColumnsProfileTextBox.Text;
```

Code

The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise2.4-UIControls**.

2.5 Add template form

You will learn how to use some inherited forms available in the Tekla.Structures.Dialog.UIControls namespace. Those inherited forms can be used as templates for some most common Tekla Structures dialogs.

The available Forms are:

- CommitAction
- CreateDialog
- OrganizerDialog
- PropertiesDialog
- TreeViewDialog

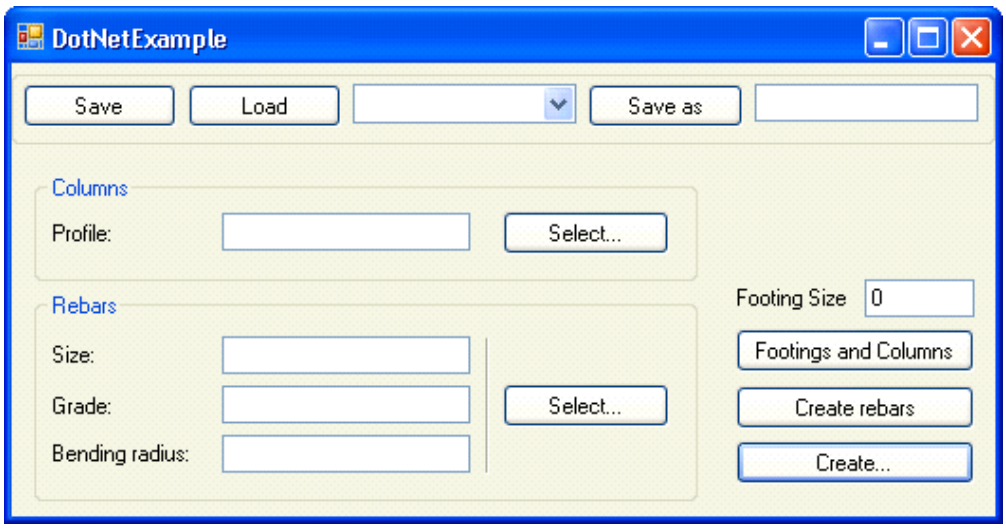
Precondition

You need to have Tekla Structures running and a model opened, preferably an empty model.

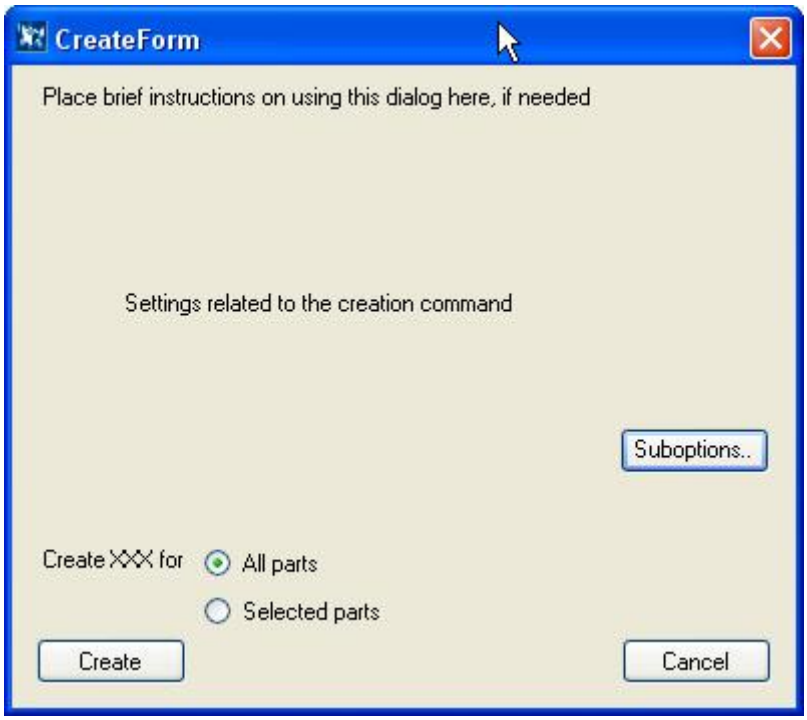
You also need the source code from exercise 2.4.

Example solution for last exercise

The main dialog of the application may look like this



The creation dialog looks like this



Precondition

You need to have Tekla Structures running and model opened, preferably empty model.
You also need the source code from exercise 2.4.

[Example solution for last exercise]

Explanation:

In this exercise we will not add any new functionality but we will just show how to add a template dialog to your project.

1. In order to use one of the template dialogs you need to insert a new Form to your project. This new Form must inherit from the dialog you want to use, for example:

```
public partial class CreateForm : Tekla.Structures.Dialog.UIControls.CreateDialog
```

2. Add a new button to the main Form of your application that creates a new instance of the inherited dialog and shows it.
3. Try different templates and modify the controls included in those.

   The dialogs don't contain any logic, so everything will need to be added. Templete dialogs' purpose is just to have a "Tekla look".

You could for example use “CreateDialog” to implement a dialog similar to “Create NC Files” dialog.

4. If you like you can continue the modeling exercises by creating a document that includes information about the selected parts (or all parts).

Hints:

The dialog can be modified since all the controls included in it are protected, so texts can be modified, new controls added, events can be added to the existing ones.

Controls in the dialog cannot be removed, though they can be hidden.

Code

The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise2.5-TemplateForms**.

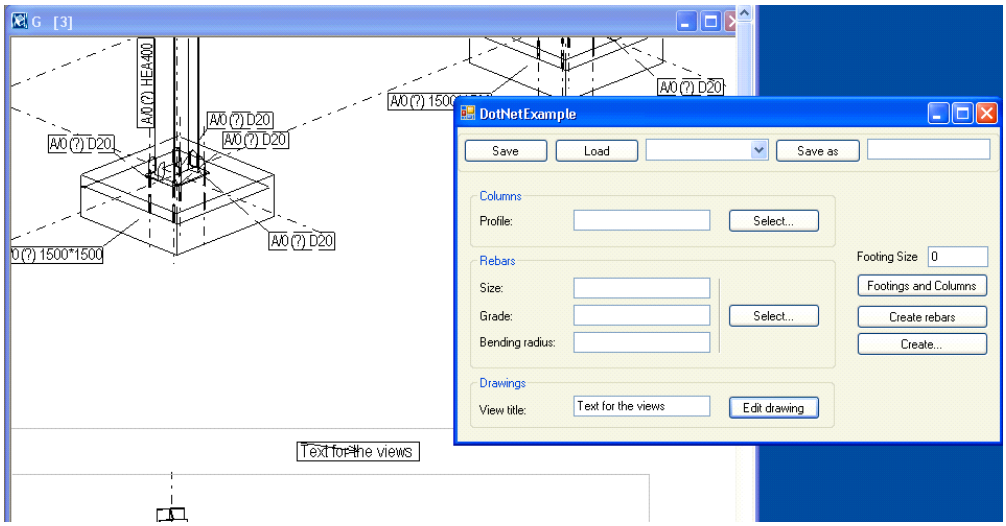
3 Exercises for the Drawing API

Here are the topics of the drawing exercises:

3.1 Easy editing of an opened drawing

You will learn how to do simple editing of a drawing that is opened in Tekla Structures. In this exercise you will add lines and texts to a drawing.

The result of this exercise should look like this



Precondition

You need to have Tekla Structures running, a model opened and a GA drawing opened.

You can use source code from the last exercise, or use the one below.

Below is a link to a zipped model, after exercise 3 and with drawings created. Open the drawing with the name "GA-drawing before editing".

Example solution for last exercise
Model after exercise 3 - added with some drawings

In this exercise you will add a text field and a button to the your main Form.The idea is that the user can enter text to the field and by pressing the button, that text is inserted to the drawing.

The text will be created centered under every view in the drawing and there should be a rectangular box drawn around it.

Get drawing from DrawingHandler and then loop through views in it. Calculate location for the text and then insert it. Get the bounding box for the text and draw a rectangle based on it.

Help:

See at least following topics from the Reference Manual:

- DrawingHandler class
- Drawing and DrawingObjectEnumerator classes
- Text class
- RectangleBoundingBox and Rectangle classes
- View and ContainerView classes

Explanation:

1. Add reference to Tekla.Structures.Drawing.dll.
2. Create an instance of DrawingHandler.
3. Add a text field to the dialog, for the text to be inserted.
4. Add a button to the dialog that inserts the text to the drawing and also draws a box around the text.
5. The text should be inserted below every view in the drawing, in a centered position.
6. Get the bounding box of the text after it has been inserted.
7. Draw a box around the text according to the bounding box.

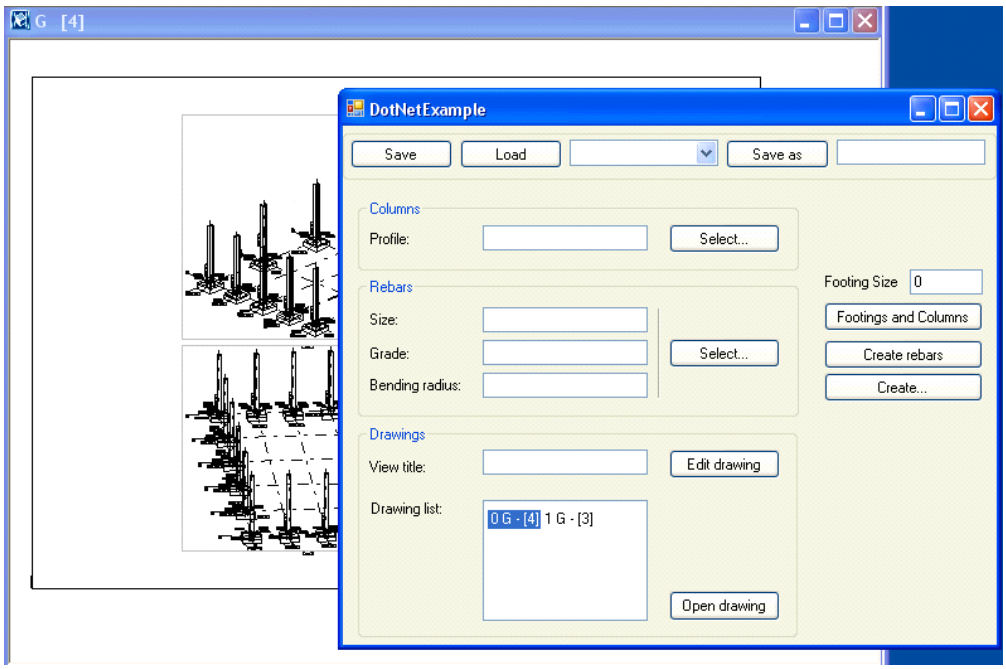
Code

The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise3.1-DrawingEditing**.

3.2 Browse through drawings list and open a drawing

You will learn how to open the drawings list and loop through the drawings in it. Also, you will learn how to open a drawing.

The result should look like this



Precondition

You need to have Tekla Structures running and a model opened; the model needs to have some drawings created already. Here is the model from exercise 3.1:

Model after exercise 3 - added with some drawings

You can use the source code after the last exercise, which is included below; or, you could create this exercise as a new project.

Example solution for last exercise

In this exercise you should add a ListView control to the main Form for listing the available drawings. Add also a button to open the selected drawing.

In the ListView you should have some attributes of the drawing visible. The user should be able to select only one drawing from the list.

With the Open button, that drawing is opened in Tekla Structures; and after that, you can check that previous exercise 3.1 works also with drawings that are opened using the API.

Help:

See Microsoft's reference for how to use System.Windows.Forms.ListView class.

Explanation:

1. Add reference to Tekla.Structures.Drawing.dll.
2. Add a ListView control to the dialog for listing the available drawings.
3. Create a button to the dialog, for opening the selected drawing.
4. Use DrawingEnumerator from DrawingHandler to loop through the available drawings.
5. Use Tag in the ListViewItem to store the drawing. ListViewItem represents an item in the ListView.
6. When opening the selected drawing (ie. pushing the button), set that Tag as active drawing; that opens the drawing.

Code

The example solution for this exercise can be found from the Open API Start-up package under the folder **Exercises/Exercise3.2-DrawingList**.