

Working with GUI for .Net  
**Roundtable TSMS**

Roundtable Development Team



# Table of Contents

---

<b>1</b>	<b>Overview .....</b>	<b>1</b>
<b>2</b>	<b>Analysis .....</b>	<b>1</b>
<b>3</b>	<b>Managing .resx Files.....</b>	<b>1</b>
<b>4</b>	<b>Managing the assemblies.xml File .....</b>	<b>1</b>
<b>4.1</b>	<b>Using the Roundtable TSMS Plug-in .....</b>	<b>1</b>
4.1.1	Creating assemblies.xml Using OpenEdge Architect 10.2B and higher .....	2
4.1.2	Creating assemblies.xml Using OpenEdge Architect 10.2A .....	2
<b>4.2</b>	<b>Using the Roundtable TSMS Windows GUI client .....</b>	<b>3</b>
<b>5</b>	<b>Conclusion .....</b>	<b>4</b>

---

## 1 Overview

The addition of .NET UI and the OpenEdge Visual Designer in OpenEdge Architect 10.2A introduces two new file types that should be managed as part of application lifecycle management activities. This white paper provides recommendations for managing these files using Roundtable TSMS and the Roundtable TSMS plug-in for OpenEdge Architect/Eclipse.

## 2 Analysis

When using .NET controls in your application, two new files come into play: .resx files and the assemblies.xml file. A .resx file is an XML file that contains instance-specific settings for controls used by an ABL Form. The assemblies.xml file is an assembly references file that is required to compile and run an ABL class or procedure that instantiates a .NET class or references any .NET object type that is not loaded by default.

For each ABL Form that contains or one or more .NET controls, OpenEdge will automatically generate a .resx file containing the instance-specific properties for those controls. OpenEdge Architect also automatically updates the .resx file when any of the properties are changed.

The assemblies.xml file pertains to an OpenEdge project. During development or at runtime, there will be a single assemblies.xml file for your OpenEdge project.

## 3 Managing .resx Files

For each ABL class that contains a .NET control, a .resx file is generated with the same base name and in the same location as the ABL class. For example, if you add a .NET control to MyForm.cls, OpenEdge Architect will automatically create the file MyForm.resx in the same directory.

A multi-part PCODE Object Subtype should be created to manage these file pairs as a single object. The first part of the Subtype will have a "cls" extension, and the second part will have a "resx" extension. Using this Subtype for any .NET forms will ensure that both the manual changes to the ABL class and the corresponding automatic changes to the .resx file will get stored together when checking in modifications to the object.

## 4 Managing the assemblies.xml File

### 4.1 Using the Roundtable TSMS Plug-in

Since one assemblies.xml file is used by the OpenEdge project, it is best if there is one shared instance of the file at each stage of development. This will prevent the challenges of having to merge differences and manage conflicts from privately-maintained copies. Using Roundtable TSMS, the assemblies.xml should be an object managed in the Roundtable Workspace.

Although the assemblies.xml file is automatically updated as new assemblies are required, it is used by the AVM, and not related to a specific form class. The assemblies.xml file should also be checked out when performing modifications to UI that will change the contents of the assemblies.xml file.

---

Before creating the first .NET form, be sure to create the aforementioned PCODE Object Subtype described in the previous section, and a PCODE Object Subtype for XML files (if you do not already have one). Also, you should create an "empty" assemblies.xml file in the development Workspace.

Follow the steps below applicable to your OpenEdge Architect version to create an "empty" assemblies.xml file in the development Workspace using the Roundtable plug-in.

#### 4.1.1 Creating assemblies.xml Using OpenEdge Architect 10.2B and higher

1. Create a project from the appropriate repository Workspace.
2. Create a new Central Task, and set it as the active Task.
3. Right-click the project in the Resources view and select Properties from the context menu.
4. Select the OpenEdge/Assemblies property page.
5. Uncheck the Use Default Location option, and choose the Workspace button to locate the appropriate module (as determined by your organization) subfolder of the active Task's folder in the project.
6. Click OK to save the location.
7. Using the RTB Imports view, import the new assemblies.xml file as a Roundtable-managed object of the aforementioned XML Code Subtype.
8. Once imported, return to the project's OpenEdge/Assemblies properties page, and change the location of the assemblies.xml file to the absolute path – including the repository Workspace root path – of the folder where the new assemblies.xml object file resides.

**NOTE:** For any new project created from a Roundtable Workspace, you will need to change the new project's assemblies file location to reference the physical assemblies.xml file that is associated with the Roundtable Workspace from which the new project was created.

#### 4.1.2 Creating assemblies.xml Using OpenEdge Architect 10.2A

1. Create a project from the appropriate repository Workspace.
2. Create a new Central Task, and set it as the active Task.
3. Create a new Object named "assemblies.xml" (sans quotes) of the aforementioned XML Code Subtype in the appropriate Module (As determined by your organization).
4. Edit the new object , inserting the following text into it:

```
<?xml version="1.0" encoding="UTF-8"?>
<references/>
```

5. Save and close the object.
6. Add the following parameter to the project AVM startup parameters:

```
-assemblies path_to_assemblies.xml
```

where *path\_to\_assemblies.xml* is the absolute path – including the repository Workspace root path – of the folder where the new assemblies.xml file resides. For example:

```
-assemblies //devserver/devshare/rtb/workspaces/devel
```

7. Restart the project AVM.

---

**NOTE:** You will need to modify the project AVM startup parameters to include the `-assemblies` parameter for any new project created from the Roundtable Workspace. The `-assemblies` parameter must reference the physical `assemblies.xml` file that is associated with the Roundtable Workspace from which the new project was created.

## 4.2 Using the Roundtable TSMS Windows GUI client

By default, the .NET classes recorded in an `assemblies.xml` file are loaded at session startup – either from the `assemblies.xml` file in the “Start in” directory, or a path specified by the `-assemblies` session startup parameter. Some Roundtable TSMS users have asked about managing Workspace-specific `assemblies.xml` files using the Roundtable TSMS Windows GUI client. As Releases are promoted through the workflow via Imports, it is customary to perform a Selective Compile in the receiving Workspace after the Import. If the selected Workspace contains objects that use non-default .NET classes that are not in the startup `assemblies.xml` file, those Workspace objects will not compile, since the referenced classes will not have been loaded into the OpenEdge session.

The challenge, then, is how to load additional .NET classes from an `assemblies.xml` file in the selected Workspace during the same OpenEdge session. There is a solution available using one of Roundtable TSMS’s event hooks to dynamically load the .NET classes in an `assemblies.xml` residing in root directory of the selected Workspace.

By intercepting the `changeWorkspace` event hook (see `rtb_events.p`), you can pass the `Pother` parameter value (the ID of the selected Workspace) to the code listed in Figure 1 below. The code will cause the OpenEdge session to load the .NET classes listed in an `assemblies.xml` file located in the selected Workspace's root folder.

**IMPORTANT:** There are two significant caveats with this code.

1. Although it will load .NET classes (if not already loaded) listed in an `assemblies.xml` file, it will not unload classes already loaded. Consequently, if an object in the selected Workspace references a .NET class that is not listed in the `assemblies.xml` file in that Workspace, but the referenced .NET class is in the `assemblies.xml` file in a previously selected Workspace (during the same session), the object will compile without error, since the referenced .NET class has already been loaded into memory. You will not know from the compile results that the `assemblies.xml` in that Workspace is "out-of-synch" with the .NET classes referenced by code in the selected Workspace.

For example, suppose that the `assemblies.xml` file in Workspace “Dev” includes .NET class “A”. When Workspace “Dev” is selected. Class “A” will be loaded into memory. Subsequently, Workspace “Test” is selected, but the `assemblies.xml` file in Workspace “Test” does not include class “A”. If an object in Workspace “Test” references class “A”, it will compile without error, since class “A” has already been loaded.

2. The `Progress.ClrBridge.AssemblyStore` class used here is undocumented and unsupported, and is therefore subject to change in future OpenEdge releases, without notification.

Given these uncertainties, you may decide that it is safer to maintain a single, `assemblies.xml` file – apart from Roundtable control – that is shared by all Workspaces, located either in the startup directory, or a location specified by the `-assemblies` session startup parameter.

**Figure 1. Code to dynamically load assemblies.xml**

```
/*
Dynamically load .NET classes from assemblies.xml file in the root folder
of the specified Workspace.

The drawback to this method is that any assemblies that are currently
loaded will NOT be unloaded.
*/
USING Progress.ClrBridge.AssemblyStore FROM ASSEMBLY.

DEFINE INPUT PARAMETER pcWspace AS CHARACTER NO-UNDO. /*Workspace ID */

DEFINE VARIABLE cWorkspacePath AS CHARACTER NO-UNDO.
DEFINE VARIABLE hWorkspaceSDO AS HANDLE NO-UNDO.
DEFINE VARIABLE assemblyStore AS Progress.ClrBridge.AssemblyStore NO-UNDO.

IF pcWspace <> "" THEN DO:

    assemblyStore = Progress.ClrBridge.AssemblyStore:Instance.

    /* Get handle to Workspace SDO */
    PUBLISH "evRtbGetWsSDOHandle" (OUTPUT hWorkspaceSDO).

    /* Get the path for the selected Workspace */
    cWorkspacePath = DYNAMIC-FUNCTION('fnRtbGetWorkspacePath':U
                                     IN hWorkspaceSDO, pcWspace).

    /* Set assemblies.xml path to Workspace root folder */
    assemblyStore:AssembliesPath = ENTRY(1, cWorkspacePath).

    /* Load the .NET classes listed */
    assemblyStore:Load() NO-ERROR.

    DELETE OBJECT assemblyStore.

END.
```

## 5 Conclusion

Although the addition of .NET UI to ABL introduces new files to application development and deployment, Roundtable TSMS and the Roundtable TSMS Plug-in can be used to effectively manage these files and ensure their promotion through the development lifecycle.