



(22) **Date de dépôt/Filing Date:** 2003/09/11

(41) **Mise à la disp. pub./Open to Public Insp.:** 2005/03/11

(45) **Date de délivrance/Issue Date:** 2016/08/30

(51) **Cl.Int./Int.Cl. G06F 17/00** (2006.01),  
**G06F 9/44** (2006.01)

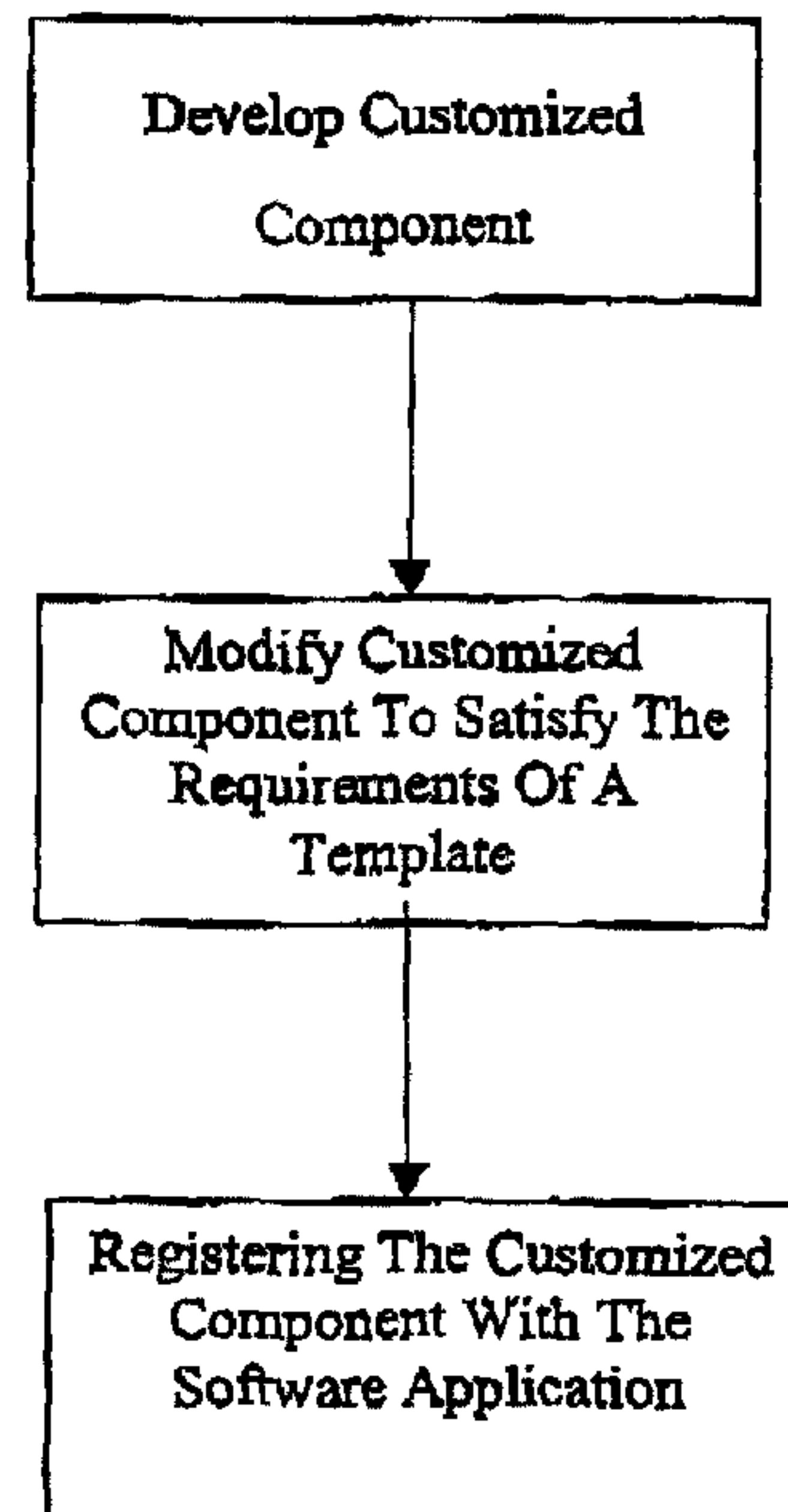
(72) **Inventeurs/Inventors:**  
TATTRIE, SCOTT I., CA;  
DINN, TRENT SHELDON, CA

(73) **Propriétaire/Owner:**  
OPEN TEXT ULC, CA

(74) **Agent:** BORDEN LADNER GERVAIS LLP

(54) **Titre : ELEMENTS PERSONNALISABLES**

(54) **Title: CUSTOMIZABLE COMPONENTS**



(57) **Abrégé/Abstract:**

A method for expanding the functionality of a software application. More specifically, the present invention relates to a method for incorporating customized components into a software application, such as a workflow application, without reliance upon programming scripts to call the customized component from outside of the application.

**ABSTRACT**

A method for expanding the functionality of a software application. More specifically, the present invention relates to a method for incorporating customized components into a software application, such as a workflow application, without reliance upon programming scripts to call the customized component from outside of the application.

## **CUSTOMIZABLE COMPONENTS**

### **FIELD OF THE INVENTION**

The present invention relates generally to a method for expanding the functionality of a software application. More particularly, the present invention relates to a method for incorporating customized components into a software application, such as a workflow application, without reliance upon programming scripts to call the customized component from outside of the application.

### **BACKGROUND OF THE INVENTION**

Traditionally, computer programmers had limited access to any components outside of a compiled software application from within the compiled software application. In order to add to components and functionality to software, it was necessary to modify the software's source code. In many cases functionality could not be added through components as the software was distributed in object code and, accordingly, the source code was not available for modification.

As a result of traditional programming, it was necessary to create individual programs for each desired function or to attempt to create software that contemplated all possible functions an end user may desire.

With the advent of more contemporary programming languages, it became possible to "call" external programs and provide information to external programs to enhance or add to the overall productivity of software. However, this approach suffers the disadvantage of having to have sufficient programming knowledge to enable the two programs to communicate effectively. While calling programs has been made easier through the use of scripting programs and languages, programming knowledge is still required to effectively use such scripting programs and languages to call external programs to add functionality to software. Accordingly, it is difficult for an average end user to add functionality of software by calling external components, even with scripting language available.

Another method of enhancing the functionality of a software application is by the addition of "plug-ins" to the software application. A plug-in is typically a separate file which

meets certain criteria and can thus be called by the software application without the use of scripting language. However, plug-ins are limited to relying on the existing functionality of the software application and therefore merely enhance the software application rather than adding a core component to the software.

It is, therefore, desirable to provide a method for adding to the functionality of software through the incorporation of customized components (rather than calling the customized component from the software) and thereby alleviate the need for programming knowledge.

## **SUMMARY OF THE INVENTION**

It is an object of the present invention to provide a method for adding to the functionality of software for end users without requiring programming knowledge through the use of customized components and without accessing the software's source code.

In one embodiment of the present invention, a customized component is incorporated directly into the software using a template which alleviates the need to write programming code in a scripting language in order for an end user to communicate with the customized component.

Other aspects and features of the present invention will become apparent to those ordinarily skilled in the art upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

Embodiments of the present invention will now be described, by way of example only, with reference to the attached Figures, wherein:

FIG. 1 is a flow diagram demonstrating the method of adding functionality to a software application in accordance with the invention.

FIGS. 2A-2T are exemplary user interface diagrams illustrating use of embodiments of the invention.



## **DETAILED DESCRIPTION**

### **Overview**

It is not unusual that, when using a software application, an end user will need some additional functionality added to suit the end user's particular needs. While this functionality may require a separate software application, often the required functionality could be better provided in a customized component created by a skilled programmer.

Generally, the present invention provides a method and system for expanding the functionality of a software application for an end user through the use of customized components. More particularly, the present invention relates to a method for incorporating customized components into a software application, such as a workflow application, without reliance upon programming scripts to call the customized component from outside of the application.

Once the customized component is created by a skilled programmer using any suitable programming language, the customized component then needs to be incorporated into the software to be useful to the end-user. The present invention provides a template which defines functions and criteria which must be satisfied by the customized component in order to be recognized and plugged into the software. That is, the template provides readily understandable rules and definitions in a form useful to the skilled programmer to cause the customized component to enable the software to use the customized component. Using this method, the present invention alleviates the need for the software to call the customized component which tends to complicate the overall process of adding to the software's functionality and thereby avoids introducing increased probability of programming errors.

The template is tailored specifically for the software as the software's need to understand the customized component will vary depending on the complexity of the software and the proposed complexity of the customized component. However, some information such as the name (or some other representation) of the customized component is minimally required by the template to enable the ordinarily skilled end user to easily find and use the customized component once it is integrated with the software.

One embodiment of the present invention is a workflow application. The workflow application may define a number of actions which can be incorporated into a workflow model

by a relatively unsophisticated user. Such actions may include, for instance, the automatic generation of an e-mail to a specified recipient, the automatic completion of a standard form word processing document or a reminder added to a calendaring system.

Despite the best efforts of the workflow application programmers, almost inevitably the end user will desire to use an action which is not programmed into the workflow application. The present invention allows a customized component (or in this example a custom action) to be added to the software without accessing the software's source code and without calling the custom action from the software or involving external scripting languages or calling any external application. That is, while it is still necessary for a skilled programmer to initially create the custom action and integrate the custom action with the software by following the template, the present invention allows the end user to have a custom action added to the software without the end user having any knowledge of how the custom action was created or integrated with the software. Once the custom action is successfully integrated with the software, the custom action can be used seamlessly and repeatedly by the end user without any requirement or need for programming language on the part of the end user, by using relatively conventional "drag and drop" and iconic user interface elements and techniques.

An example of a custom action in the above example of a workflow application may be an action which integrates with third-party products or involves a business practice that has been specifically created or customized for the end user. A more specific example may be that the custom action allows the end user to build in automatic communications with a supplier in the workflow model, despite the fact that the workflow application may not have included any telephony functionality.

As exemplified in the example above, the present invention also alleviates the need for scripting language to call the custom action and therefore reduces the potential number of idiopathic errors or, alternatively, reduces the troubleshooting required to find any such errors which are generated.

Another embodiment of the present invention is a graphics application or drawing program. While it is possible to add a "plug-in" to some graphics applications, the present invention further allows an end user to benefit from an addition to the graphics application's



core functionality through the incorporation of a customized component. That is, while a plug-in may allow modified use or a specified use of an existing tool in the graphics application, the present invention may provide a completely new tool thereby adding core functionality rather than enhancing existing functionality. By way of example only, the present invention may add a tool which, when invoked by the end user, distributes a thumbnail of the graphic created by the end user to a predefined group of companies interested in purchasing electronic art and automatically complete a licensing transaction with an interested buyer based on a price and terms predefined by the end user. This type of unexpected functionality (for a drawing program) and very end user-specific functionality may be added to the graphics application without modifying the graphics application and without calling external programs through programming a script. Further, this functionality increases the usefulness of the graphics application to the user, notwithstanding that such core functionality (such as accessing the Internet) may not have been conceived by the original programmers of the graphics application. It may also be done without access to the source code or API structures of the graphics program through the use of provided templates.

### **Custom Actions**

Customs Actions are a new and powerful feature that has been added to the Teamplate Integrated Development Environment (IDE).

While Teamplate has supplied you with a wide variety of Actions to use in the development of a Model, every development scenario is unique and may require a level of customization or integration not readily available. To provide for organizations in this type of situation, the IDE has been extended to use Custom Actions.

This feature has to two important aspects to it. The new and improved IDE interface is in part to reduce the "learning" curve involved with the Teamplate IDE by providing any already familiar environment. As part of the new interface, the comfort zone is "taken up a notch" by being able to create a development environment that is unique and comfortable to you, the Developer. As with Visual Studio for .NET, the Teamplate IDE "remembers" your setup and is ready "the way you want it" and available to you whenever you open the IDE.

The second aspect is the ability for the Developer to re-use previously created coding. This is powerful as this code may be for the integration of third party products or involve a business practice that has specifically created for and/or customized for use by your organization. The crux of the .NET philosophy is the ability to create and re-use components (code) and therefore is one of the main purposes of Custom Actions in Teamplate.

By providing this functionality, Teamplate no longer requires you to adapt your Script into an acceptable Teamplate-like format within its IDE. The Custom Action in Teamplate allows you to add additional functionality to your Model or Workflow Application. You simply add Custom Actions that you have created in Visual Studio for .NET to Toolbox View.

It should be noted that integration with Teamplate or any product for that matter has two basic methodologies. The first is to use Teamplate called by another application. This is done by using the Workflow API to manipulate a Teamplate Workflow Application. For more information of the Teamplate Workflow API, refer to the Workflow API Guide.

Teamplate also can integrate with a third-party product by using or calling the application's exposed API within ScriptVlew for an Object, thereby invoking the third-party product as part of a Workflow Application.

The purpose of any Action, custom or otherwise, within the IDE is for the “non-programmer” Teamplate Developer to create automated workflow without having programming knowledge. In other words, drag and drop an Action onto a Workflow Object, fill in the blanks and click the Finish button. The Script is created for you without having to manually code or write the text required to create the automation,

This is also helpful for the programming-savvy Developer who can create code and then customize it quickly and easily, thereby saving an organization valuable development time and resources in the deployment of an automated workflow solution. The drag and drop Graphical User Interface (GUI) provides a mechanism for this type of Developer to create functionality In Visual Studio for the “non-programmer” Developer.

The Action provides an easy-to-use Graphical User Interface (GUI). The interface is created to ensure all aspects of the scripting are covered to generate script that is both seamless and useful to the end User.



The Action also provides the Developer with the ability to create and customize repetitive code while reducing the risk of "human error" when script is copied and pasted or entered manually with errors.

As Actions are viewed by displaying ActionView popup for the Object, you can edit any Action by double clicking on it. The Action is displayed and when you click the Finish button, the Script is updated. This means you do not have to manually modify Script but can do it in a graphical environment. You can also delete an Action from an Object by using Action View.

Custom Actions are powerful and beneficial to you as they provide one of the easiest ways possible to re-use customized script and integrations within Teamplate. The true power of these Actions is realized when they can be invoked for several Objects within a Workflow Application or even more powerful, used for other Models you are developing or will develop.

There are several important steps in creating a Custom Action for use in Teamplate. Each of these steps is explained in the following sections. It is highly recommended that you read and follow these steps to ensure your Custom Action(s) will work as anticipated.

The first and most obvious step is to create the Custom Action. Although the creation of the code is important, there are also some customizations that must occur within Visual Studio so that the Custom Action will function as anticipated.

Once you have created a Custom Action, you will need to register it in the Global Assembly Cache (GAC) using any one of the available methods. After which, you can then add it to Teamplate by right clicking on Toolbox View and selecting Customize Toolbox from the shortcut menu.

### **Creating a Visual Studio Project**

This is the first step involved in creating a Custom Action for use in the Teamplate IDE. Although the depicted examples in the following explanations are in C# (C Sharp) and Visual Basic (VB) you can use any of the project types available in Visual Studio for .NET.

This means that you can create or re-use code in Visual Basic or C++ project and invoke the code even though ScriptView in the Teamplate IDE uses scripting language based

on Microsoft's Visual Studio for .NET. This includes all Controls, which have all the functionality, Properties and Events including syntax found in that development environment.

Note: An example Custom Action can be found in the

C:\ProgramFiles\Teamplate\Teamplate.Net\Samples\SampleAction folder located on a Server-side installation of the Teamplate for .NET product.

It is also assumed that as a Developer, you have had previous exposure or development experience with Microsoft's Visual Studio for .NET.

#### Creating a Visual Studio Project (C#)

1. Open the VS .NET Development Environment

(Start - Programs - Microsoft Visual Studio .NET - Microsoft Visual Studio .NET)

2. Create a New Project. From the File menu (FIG. 2A), select New and then select Project from the Submenu. (File → New → Project)

OR

By pressing the CTRL+Shift+N keys for the keyboard shortcut.

3. Select Windows Control Library Project from the Templates list and select Visual C# Projects from the Project Types list (FIG. 2B).

4. Enter a name for the Project in the Name field. E.g. AccountsPayableAction

5. Enter the path or location for the Project in the Location field.

OR

Click the Browser button and select a location for the Project.

6. Click the OK button.

#### Creating a Visual Studio Project (VB)

1. Open the VS .NET Development Environment.

(Start - Programs - Microsoft Visual Studio .NET - Microsoft Visual Studio .NET)

2. Create a New Project. From the File menu (FIG. 2C), select New and then select Project from the submenu. (File - New - Project)

OR

By pressing the CTRL+Shift+N keys for the keyboard shortcut



3. Select Windows Control Library Project from the Templates list and select Visual Basic Projects from the Project Types list (FIG. 2D).
4. Enter a name for the Project in the Name field. E.g. AccountsPayableAction
5. Enter the path or location for the Project in the Location field.

OR

Click the BROWSER button and select a location for the Project.

6. Click the OK button.

### **Adding the Teamplate Reference**

Once you have created the initial Windows Control Library Project, the next step is to add the Teamplate Reference to your project. Without this reference, your Custom Action will not work as anticipated in the Teamplate IDE.

Adding the Teamplate Reference (C#) (These steps assume that you have created a Windows Control Library Project.)

1. Ensure your Windows Control Library Project for your Custom Action is open.
2. Display or create a Blank Control.
3. Display the Solutions Explorer by clicking on the Solutions Explorer tab.

OR

From the View menu, select Solutions Explorer.

OR

By pressing the CTRL+ALT+L keys for the keyboard shortcut.

4. If necessary, click on the +ProjectName expand node.
5. Right click on the References node and select Add Reference from the shortcut menu. (FIG. 2E)

6. In the Add Reference dialog box, on the .NET tab select the TeamplateWin/IDE.dll from the Components: list (FIG. 2F).

7. Click the SELECT button.

OR

If the TeamplateWinIDE.dll is not available in the Components list, click the Browse button and browse to the C:\Program Files\Teamplate\Teamplate.NET folder

In the Select Component dialog box (FIG. 2G), select the TeamplateWinIDE.dll and click the OPEN button (You can also double click on the file.)

8. Also select the System.Drawing.dll from the Components list and click the Select button.

9. Click the OK button.

You will see the Teamplate Reference has been added under the References node of the Solution Explorer as seen in FIG. 2H.

Adding the Teamplate Reference (VB)(These steps assume that you have created a Windows Central Library Project).

1. Ensure your Windows Control Library Project for your Custom Action is open.
2. Display or create a Blank Control.
3. Display the Solutions Explorer by clicking on the Solutions Explorer tab.

OR

From the View menu, select Solutions Explorer.

OR

By pressing the CTRL+ALT keys for the keyboard shortcut.

4. If necessary, click on the +ProjectName expand node.
5. Right click on the References node and select Add Reference from the shortcut menu. (FIG. 2I)

6. In the Add Reference dialog box (FIG. 2J), on the .NET tab select the TeamplateWinIDE.dll from the Components: list.

7. Click the SELECT button.

OR

If the TeamplateWinIDE.dii is not available in the Components list, dick the Browse button and browse to the C:\Program Files\TeamplateTeamplate.NET folder

In the Select Component dialog box (FIG. 2K). select the TeamplateWinIDE.dll and click the Open button (You can also double click on the file.)



8. Also select the System.Drawing.dll from the Components list and click the SELECT button.
9. Click the OK button.

You will see the Teamplate Reference has been added under the References node of the Solution Explorer as seen in FIG. 2L (You can also double click on the file.)

### **Developing the Custom Action**

The development of a Custom Action and its use in Teamplate is solely at your discretion. You must decide how this Action or wizard will look and interact with the Teamplate Developer as well as its function. Remember that a Developer may or may not have previous programming experience.

Prior to the development of the Custom Action you need to ensure that you perform the following steps so that your Custom Action will work as anticipated in the Teamplate IDE. When the Teamplate Reference is added to your Custom Action Project, Stub Functions also get created and are automatically inserted into the code. You will need to write code for each Stub Function.

#### Developing the Custom Action (C#)

1. From the View menu (FIG. 2M), select Code.

OR

Press the F7 key for the keyboard shortcut

2. Rename the Class. It is recommended that the name you use has "Action" appended to it.

These steps assume that you have created a Windows Control library project and the Template Reference has been added.

E.g. AccountsPayableAction

```
public class AccountsPayableAction
```

3. Update the System.Windows.Form.UserControl to:  
System.Windows.Form:UserControl; Teamplate:Windows.IDE.Design.

4. From Intellisense, select IActionWizardDesign
5. Change UserControl to the name of your Custom Action.

E.g. AccountsPayableAction

6. Display Class View by clicking on the Class View tab.

OR

From the View menu, select Class View.

OR

By pressing the CTRL+Shift+C keys for the keyboard shortcut.

7. Click the +ProjectName expand node.
8. Click the +ProjectName NameSpace node.
9. Click the +Bases and Interfaces node.
10. Right click on the iActionWizardDesign node and select Add from the shortcut

menu and then select Implement this Interface from the submenu. (FIG.2N)

At this point the Stub Functions will be generated and you must write code for each. However, when using Visual Basic to create your project, the Stub Functions must be added manually.

Developing the Custom Action (VB) (These steps assume that you have created a Windows Control Library Project and the Teamplate Reference has been added.)

1. From the View menu (FIG. 2O), select Code.

OR

Press the F7 key for the keyboard shortcut.

2. Rename the Class. It is recommended that the name you use have "Action" appended to it. E.g. AccountsPayableAction

Public Class AccountsPayableAction

3. After the Inherits System.Windows.Forms.UserControl, press the ENTER key and add the following code (You can use Intellisense.):

Implements Teamplate.Windows.IDE.Design.IActionWizardDesign

4. Change UserControl1 to the name of your Custom Action.

E.g. AccountsPayableAction



5. From the Class Name drop down list (FIG. 2P), select IActionWizardDesign (Design).
6. From the Methods drop down list, select ActionImage.
7. From the Class Name drop down list, select IActionWizardDesign (Design).
8. From the Methods drop down list, select DisplayWizard.
9. From the Class Name drop down list, select IActionWizardDesign (Design).
10. From the Methods drop down list, select ExecuteParameters.
11. From the Class Name drop down list, select IActionWizardDesign (Design).
12. From the Methods drop down list, select ReturnParameter.
13. From the Class Name drop down list, select IActionWizardDesign (Design).
14. From the Methods drop down list, select SaveXMLAction.
15. From the Class Name drop down list, select IActionWizardDesign (Design).
16. From the Methods drop down list, select Execute.
17. Write the code for the Stub Functions.

For more information, refer to the Stub Functions section.

### **Stub Functions**

The Stub Functions are important as they define how the Custom Action will interact with the Teamplate Developer within the Teamplate IDE as well as control the display of the Custom Action in Toolbox-View. Each Stub Function is detailed in the following-sections.

Once these Stub Functions have been defined, you can create the code for your Custom Action.

```
#region Implementation of IActionWizardDesign
public System.Windows.Forms.DialogResult DisplayWizard (string xml,
Teamplate.Windows.IDE.Design
{
    return new System.Windows.Forms.DialogResult ();
}
```

### **ActionImage**

This Stub Function controls or determines the image that will be displayed for the Custom Action in Toolbox View as well as the icon that is displayed in the ActionView popup.

The image you use must be 16 x16 pixels in size.

ActionImage Stub Function (C#):

```
public System.Drawing.Image ActionImage
{
    get
    {
        return null;
    }
}
```

Note: You must implement an image. Do not leave the return as "null".

ActionImage Stub Function (VB):

```
Public ReadOnly Property ActionImage() As System.Drawing.Image
Implements Teemplate.Windows.IDE..Designs.IActions:
    End Get
End Property
```

DisplayWizard

This Stub Function will pass the first XML parameter into the Custom Action into the RDE. This XML parameter is a description of the Custom Action that is being built.

This is also defined in the SaveXMLAction Stub Function.

There are three options that are displayed in Intellisense.

The GetDataBindingInformation requests a list all the XML DataBinding Objects in the Custom Action. These are the objects that can be bound to an XML or Database Data Object that is being used by the Model. There are three options for Data Binding:



- All Objects which will bind all available Objects in the Custom Action
- Just Database which will bind only Database Fields in the Custom Action
- Just XML which will bind only XML Nodes In the Custom Actions

The XMLCreateObject is used by the Custom Action and creates an XML Document for the Object. There are two parameters that must be configured:

- Name of the XML Object which determines which XML Object will be used by the Custom Action

- XML Definition which determines what the XML Object should look like

The XMLAppendNode is used to extend the XML Object that is currently used by the Model. There are two parameters that must be configured:

- Reference path so that the Model "knows" where to access the XML Document

- Node to Append which is the name of the Node that is to be appended to the XML Data Object for use by the Custom Action.

DisplayWizard Stub Function (C#):

```
public System.Windows.Focus.DialogResult DisplayWizard(string xml,
Teamplate.Windows.IDE.Design.IModelDES:
{
    return new System.Windows.Forms.DialogResult()
```

DisplayWizard Stub Function (VB):

```
Public Function DisplayWizard(ByVal xml As String, ByVal modelDesign As
Teamplate.Windows.IDT.Design.IModelDesign
End Function
```

### ExecuteParameters

This Stub Function returns a string of parameter definitions for the Execute Stub Function.

ExecuteParameters Stub Function (C#):

```
public string Execute Parameters
```

16

```

    {
        get
        {
            return null;
        }
    }

```

Note: The return can be left as "null".

ExecuteParameters Stub Function (VB):

```

    Public Read Only Property Execute Parameters [] As String Implements
    Teamplate.Windows.IDE.Design.
    Get
    End Get
    End Property

```

### ReturnParameter

This Stub Function will return a value and determines where the values are to be stored.

Return Parameter Stub Function (C#):

```

    public string ReturnParameter
    {
        get
        {
            return null;
        }
    }

```

Note: The return can be left as "null".

Return Parameter Stub Function (VB):

Public ReadOnly Property Return Parameter() As String Implements  
Teamplate.Windows.IDE.Designs.

Get

End Get

End Property

### SaveXMLAction

This Stub Function is to save an XML description of your Custom Action. This XML String Is then passed into the DisplayWizard Stub Function.

SaveXMLAction Stub Function (C#):

```
public string SaveActionXml()
{
    return null;
}
```

SaveXMLAction Stub Function (VB):

```
Public Function SaveActionXML() As String Implements
Teamplate.Windows.IDE.Design.IActionWizardDesign.SaveActions
End Function
```

### Execute

This Stub Function executes your Custom Action in the Teamplate IDE.

You must add "Static" at the beginning of this Stub Function so that your Action will function as anticipated in the Teamplate IDE.

Execute Stub Function (C#):

```
public object Execute(params object[] objs)
{
    return null;
}
#endregion
```



In C#, you must copy this Stub Function and paste it below the original. Then add “Static” at the beginning of second instance of the Stub Function. Remove params object [] objs and replace it with the specific action parameters required by your Custom Action.

Execute Stub Function (VB):

Public Function Execute. As object implements Teamplate.Windows.IDE.Design End Function

In VB, you must copy this Stub Function and paste it below the original. Then add "Shared" at the beginning of second instance of the Stub Function. Remove params object [] objs and replace it with the specific action parameters required by your Custom Action. Remove the Implements Teamplate.Windows.IDE.Design.IActionWizardDesign.Execute from the second instance of the Stub Function.

### **Building the Project**

After creating the code for your Custom Action as well as the Stub Functions, you will need to build your project.

#### Building the Project

1. Display Class View by clicking on the Class View tab.

OR

From the View menu, select Class View.

OR

By pressing the CTRL+Shift+C keys for the keyboard shortcut.

2. Right click on the Project Name node and select Build from the shortcut menu.

OR

From the Build menu, select Build Solution.

OR

Press CTRL+Shift+B keys for the keyboard shortcut.

### **Assigning a Strong Name**

After you have created your Custom Action in Visual Studio for .NET and built the Project. The next step Is to assign a Strong Name to the file and then enter the resulting filename (Strong Name Key) into the Assembly Key File parameter jn your project.

The Project Is then rebuilt and you will then be ready to register your Custom Action in the Global Assembly Cache (GAC).

#### Assigning a Strong Name

1. Open the Visual Studio Command Prompt window.

(Start - Programs -- Microsoft Visual Studio .NET - Visual Studio .NET  
Tools - Visual Studio .NET Command Prompt)

(Where projectname is the Name of your Project for your Custom Action)

2. Navigate to the folder where the Project is saved.

3. Enter:

sn -k projectname.snk

4. In your Project, open the Assembly Information. From the File menu (FIG. 2Q), select Open and from the submenu select File.

OR

Press the CTRL+O keys for the keyboard shortcut.

5. In the Open File dialog box (FIG. 2R), select the Assemblyinfo.cs file and then click the OPEN button.

6. Locate the AssemblyKeyFile parameter.

[assembly: AssemblyDelaySign(false)]

[assembly: AssemblyKeyFilei ("")]

[assembly: AssemblyKeyName ("")]

7. Enter the path and Strong Name Key filename in between the quotation marks of this parameter.

8. Rebuild your Project.

#### **Registering in the Global Assembly Cache**

After creating and building the Project which Includes adding the Teamplate Reference, writing code for the Stub Functions, assigning a Strong Name and configuring the AssemblyInfo.cs file you are then ready to register the Custom Action in the Global Assembly Cache.

There are several methods you can use to register a Custom Action in the GAG. It does not matter which method you prefer or use.

If the Custom Action Is not registered in the GAC, it will not appear in the Select Actions dialog box while customizing Toolbox View in the IDE.

#### Registering a Custom Action in the GAC (Drag & Drop Method)

1. In Windows Explorer or tiled My Computer windows, drag and drop the Custom Action DLL file into the C:\Windows\assembly folder.

#### **Adding Custom Actions to Toolbox View**

The final step for creating a Custom Action for use in Teamplate is to add the Action to Toolbox View. This step is the same as adding any of the other Actions that are available for use in Toolbox View.

#### Adding Custom Actions to Toolbox View

1. Ensure that Toolbox View is display In the IDE.
2. Right click on Toolbox View (FIG. 2S) and select Customize Toolbox from the shortcut menu.
3. In the Select Actions dialog box (FIG. 2T), click on the checkboxes of the Custom Actions you would like to include In Toolbox View.
4. Click the OK button.

Note: Custom Actions are removed by de-selecting the Actions in the Select Actions dialog box.

#### **Notes**

While the steps for creating and configuring a Custom Action have a level of complexity all their own, there are a few important notes that you should keep in mind:



### **Updating Custom Actions**

Whenever a Custom Action is updated or changed then Teamplate must be closed. The Custom Action is updated or re-registered in the GAC and then Teamplate can be re-opened. Toolbox View will automatically reference the updated Custom Acton.

Keep in mind, that If you leave the asterisk (“\*\*”) in the AssemblyVersion parameter (see Figure 15) In the AssemblyInfo.cs file, It will build another separate rile (.dll) each time the Project for your Custom Action is updated and then re-built.

```
[assembly: AssemblyVersion("1.0.**")
```

AssemblyVersion parameter

It is highly recommended that you manually change the AssemblyVersion parameter. Each time the Custom Action is re-registered into the GAO, and the version number is incremented or different, the GAC registers multiple versions of the Custom Action.

Teamplate however, will only recognize the Custom Action that was first added to Toolbox View by the AssemblyVersion parameter.

You must remember to delete the old Custom Action and use the updated version of the Custom Action if the AssemblyVersion parameter Is different. If you leave the Assembly Version number the same, regardless of the changes and rebinding the Project for the Custom Action, Teamplate will reference and use the updated Custom Action automatically based on the original Custom Action's AssemblyVersion parameter.

### **Deleting Custom Actions**

If the Custom Action is deleted from the GAC then Teamplate will throw an exception when the Action is used from Toolbox View. You must remember to remove the Custom Action from Toolbox View after you delete the Custom Action from the GAC.

**APPENDIX**

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Xml;
using Teamplate.Windows.IDE.Design;
using System.Web;

namespace Teamplate.Actions.Samples
{
    /// <summary>
    /// Summary description for SendEmail.
    /// </summary>
    public class SendEmail :
        System.
        Windows.Forms.UserControl, Teamplate.Windows.IDE.Design.IActionWizardDesign
    {
        private System.ComponentModel.IContainer components;

        private System.Windows.Forms.ImageList imageList1;
        System.Web.Mail.MailMessage m_webmail= new System. Web.Mail.MailMessageO;
        string m_smtp;

        public SendEmailO
        {
            // This call is required by the Windows.Forms Form Designer. InitializeComponent;

            //TODO: Add any initialization after the InitForm call }

            ///<summary>
            /// Clean up any resources being used.
            /// </summary>
            protected override void Dispose( bool disposing )
            {
                if( disposing )
                {
                    if(components !=null)
                    {
                        components.Dispose();
                    }
                }
            }
        }
    }

```

```

base.Dispose( disposing );
}

#region Component Designer generated code
///<summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.Resources.ResourceManager resources = new
    System.Resources.ResourceManager(typeof(SendEmail));
    this.imageList1 = new
    System.Windows.Forms.ImageList(this.components);
    //
    // imageList1
    //
    this.imageList1.ColorDepth =
    System.Windows.Forms.ColorDepth.Depth16Bit;
    this.imageList1.ImageSize = new System.Drawing.Size(16, 16);
    this.imageList1.ImageStream =
    ((System.
    Windows.Forms.ImageListStreamer)(resources.GetObject("imageList1.ImageStream")));

    this.imageList1.TransparentColor =
    System.Drawing.Color.Transparent;
    //
    // SendEmail
    //
    this.Name = "SendEmail";

} #endregion
#region Implementation of IActionWizard

object IActionWizardExecute.Execute(params object[] objs)
{
    return null;
}

static public object Execute(string to,string cc,string bcc,string from,string subject,string
body,string smtp)
{
    try
    {

```



```

System.Web.Mail.MailMessage webmail = new
System.Web.Mail.MailMessage();
webmail.Bcc = bcc;
webmail.Body = body;
webmail.Cc = cc;
webmail.From = from;
webmail.Subject = subject;
webmail.To = to;

```

```

System.Web.MaiLSmtpMail.SmtpServer=smtp;
System.Web.Mail.SmtpMail.Send(webmail);

```

```

return true;
}
catch(System.Exception e)
{
throw new System.Exception(e.ToSiring());
}
return null;
}

```

```

public System. Windows.Forms.DialogResult DisplayWizard(string xml,
Teamplate.Windows.IDB.Design.IModelDesign modelDesign)
{
System.Windows.Forms.DialogResult result;
try
{
EmailForm wiz= new EmailForm(modelDesign);

```

```

System.Windows.Forms.TreeView treeView =
modelIDesign.GetDataBindingInfo(Teamplate.Windows.IDE.Design.DataBinding.Xml);
// load binding data

```

```

this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBoxBcc);
this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBoxCc);
this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBoxFrom);
this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBoxBody);
this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBaxSMTP);
this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBoxSubject);
this.PopulateDataBinding(treeView.Nodes[0],wiz.comboBoxTo);

```

```

if (xml != null)
{
XmlTextReader xmlReader = new

```

```
XmlTextReader(xml,XmlNodeType.Element,null);

xmlReader.Read();
while(!xmlReader.EOF)
{
    if(xmlReader.NodeType ==
    XmlNodeType.Element)

    {
        switch(xmlReader.LocalName)
        {
            case "TO" :

                wiz.comboBoxTo.Text=xmlReader.ReadInnerXml();
                break;
            case "FROM" :

                wiz.comboBoxFrom.Text=xmlReader.ReadInnerXml();
                break;
            case "CC":

                wiz.comboBoxCc.Text=xmlReader.ReadInnerXml();
                break;
            case "BCC" :

                wiz.comboBoxBcc.Text=xmiReader.ReadInner.Xml();
                break;
            case "SUBJECT" :

                wiz.comboBoxSubject.Text=xmlReader.ReadInnerXml();
                break;
            case "BODY" :

                wiz.comboBoxBody.Text=xmlReader.ReadInnerXml();
                break;
            case "SMTP" :

                wiz.comboBoxSMTP.Text=xm1Reader.ReadInnerXml();
                break;
            default:
                xmlReader.Read();
                break;

        }
    }
}
```

```
else
xmlreader.Read();

}
}

result=wiz.ShowDialog();

if (result = System.Windows.Forms.DialogResult.OK)

if (wiz.comboBoxTo.Text = "")
m_webmail.To="\"";
else
m_webmail.To=wiz.comboBoxTo.Text;

if (wiz.comboBoxFrom.Text = "")
m_webmail.From="\"";
else

m_webmail.From=wiz.comboBoxFrom.Text;

if (wiz.comboBoxCc.Text = "")
m_webmail.Cc="\"";
else
m_webmail.Cc=wiz.comboBoxCc.Text;

if (wiz.comboBoxBcc.Text == "")
m_webmail.Bcc="\"";
else
m_webmail.Bcc=wiz.comboBoxBcc.Text;

if (wiz.comboBoxSubject.Text == "")
m_webmail.Subject="\"";
else

m_webmail.Subject=wiz.comboBoxSubject.Text;

if (wiz.comboBoxBody.Text== "")
m_webmail.Body="\"";
else

m_webmailBody=wiz.comboBoxBody.Text;

if (wiz.comboBoxSMTP.Text== "")
m_smpt="\"";
```



```

else
m_smtp=wiz.comboBoxSMTP.Text;

}

}
catch(System.Exception e)
{
System.Windows.Forms.MessageBox.Show(e.Message);
return System.Windows.Forms.DialogResult.Cancel;
return
}

return result;
}
public string SaveActionXml()

{
String xml="<SampleSendEmailAction";
xml+=" "<TO>" +this.m_webmail.To + "</TO>";
xml+=" "<CC>" +this.m_webmail.Cc + "</CC>";
xml+=" "<BCC>" +this.m_webmail.Bcc + "</BCC>";
xml+=" "<FROM>" +this.m_webmail.From + "</FROM>";
xml+=" "<BODY>" +this.m_webmail.Body + "</BODY>";
xml+=" "<SUBJECT>" +this.m_webmail.Subject + "</SUBJECT>" ;
xml+="- "<SMTP>" +this.m_smtp + "</SMTP>";
xml+=" "<SampleSendEmailAction>";

return xml;
}

public System.Drawing ActionImage
{
get
{
return this.imageList1.Images[0];
}
}
public string ExecuteParameters
{
get (return    m_webmail.To+","    +    m_webmail.Cc+"," +
m_webmail.Bcc+"," + m_webmail.From +  ","    + m_webmail.Subject +    "," +

```

```

m_webrmail.Body + "," + m_smtp;)
}
public string ReturnParameter
{
get {return "";}
}

internal void PopulateDataBinding(TreeNode node,
System.Windows.Forms.ComboBox cb)
{
if (node == null)
return;

for (int i=0; i < node.Nodes.Count;i++)
{
string item=node.Nodes[i].Text+"(''"
buildpath(node.Nodes[i],cb,item );
}
}
internal void buildpath(TreeNode node, System.Windows.Forms.ComboBox
cb,string path)
{
if (node == null)
return;

for (int i=0; i < node.Nodes.Count;i++)
{

cb.Items.Add(path + node.Nodes[i].Text+"\"");
buildpath(node.Nodes[i],cb,path+ node.Nodes[i].Text+"/");

}

}
#endregion

}

}

```

The above-described embodiments of the present invention are intended to be examples only. Alterations, modifications and variations may be effected to the particular embodiments by

those of skill in the art without departing from the scope of the invention, which is defined solely by the claims appended hereto.



**CLAIMS:**

1. A method of adding new core components into existing host applications, comprising:  
creating, by a first computer and in a development environment, a custom component at least partially based on a custom component definition, the custom component definition providing criteria for all custom components for use in a host application, the host application having an integrated development environment and associated with a shared cache;  
registering the custom component with the shared cache associated with the host application, the host application running on a second computer;  
adding the custom component registered with the shared cache to a view of the host application that is part of the integrated development environment of the host application and that is accessible by an end user of the host application; and  
responsive to an activity of the end user of the host application, automatically displaying the view of the host application containing the custom component such that the custom component is accessible by the end user and adds a custom functionality to the host application already running on the second computer.
2. The method according to claim 1, wherein the creating comprises:  
creating code and a custom action interface for the custom component.
3. The method according to claim 1, wherein the creating comprises:  
developing the custom component using, at least partially, features provided by the host application.
4. The method according to claim 1, wherein the creating further comprises modifying the custom component to satisfy template parameters described in the criteria provided by the host application.

5. The method according to claim 1, wherein the creating further comprises:  
creating a project for the custom component in the development environment.
6. The method according to claim 5, wherein the project comprises a control library project.
7. The method according to claim 5, wherein the creating further comprises:  
adding a host application reference to the project for the custom component.
8. The method according to claim 7, wherein the adding comprises associating a file from a components list within the project in the development environment with the integrated development environment of the host application.
9. The method according to claim 1, wherein the creating further comprises:  
adding functionality to the custom component using at least one third party application.
10. The method according to claim 1, wherein the custom component is configured for use by the end user to develop workflow models.
11. The method according to claim 1, wherein the shared cache includes custom components shared by a plurality of host applications.
12. The method according to claim 1, wherein the host application comprises a workflow application and wherein the custom component is available for use by a plurality of workflow processes or workflow models in the workflow application.

13. The method according to claim 1, wherein the host application comprises a workflow application and wherein the custom component is useable with a plurality of workflow objects within a single workflow model in the workflow application.

14. A computer program product comprising at least one non-transitory computer readable medium storing instructions executable by at least one processor to perform:

creating, in a development environment running on a first computer, a custom component at least partially based on a custom component definition, the custom component definition providing criteria for all custom components for use in a host application, the host application having an integrated development environment and associated with a shared cache;

registering the custom component with the shared cache associated with the host application, the host application running on a second computer;

adding the custom component registered with the shared cache to a view of the host application that is part of the integrated development environment of the host application and that is accessible by an end user of the host application; and

responsive to an activity of the end user of the host application, automatically displaying the view of the host application containing the custom component such that the custom component is accessible by the end user and adds a custom functionality to the host application already running on the second computer.

15. The computer program product of claim 14, wherein the creating comprises:  
creating code and a custom action interface for the custom component.

16. The computer program product of claim 14, wherein the creating comprises:  
developing the custom component using, at least partially, features provided by the host application.



17. The computer program product of claim 14, wherein the creating further comprises modifying the custom component to satisfy template parameters described in the criteria provided by the host application.
18. The computer program product of claim 14, wherein the creating further comprises: creating a project for the custom component in the development environment.
19. The computer program product of claim 14, wherein the project comprises a control library project.
20. The computer program product of claim 18, wherein the creating further comprises: adding a host application reference to the project for the custom component.
21. The computer program product of claim 20, wherein the adding comprises associating a file from a components list within the project in the development environment with the integrated development environment of the host application.
22. The computer program product of claim 14, wherein the creating further comprises: adding functionality to the custom component using at least one third party application.
23. The computer program product of claim 14, wherein the custom component is configured for use by the end user to develop workflow models.
24. The computer program product of claim 14, wherein the host application comprises a workflow application and wherein the custom component is available for use by a plurality of workflow processes or workflow models in the workflow application.

25. The computer program product of claim 14, wherein the host application comprises a workflow application and wherein the custom component is useable with a plurality of workflow objects within a single workflow model in the workflow application.

26. A system, comprising:

at least one processor; and

at least one non-transitory computer readable medium storing instructions executable by the at least one processor to perform:

creating, in a development environment running on a first computer, a custom component at least partially based on a custom component definition, the custom component definition providing criteria for all custom components for use in a host application, the host application having an integrated development environment and associated with a shared cache;

registering the custom component with the shared cache associated with the host application, the host application running on a second computer;

adding the custom component registered with the shared cache to a view of the host application that is part of the integrated development environment of the host application and that is accessible by an end user of the host application; and

responsive to an activity of the end user of the host application, automatically displaying the view of the host application containing the custom component such that the custom component is accessible by the end user and adds a custom functionality to the host application already running on the second computer.

27. The system of claim 26, wherein the creating comprises:

creating code and a custom action interface for the custom component.

28. The system of claim 26, wherein the creating comprises:

developing the custom component using, at least partially, features provided by the host application.

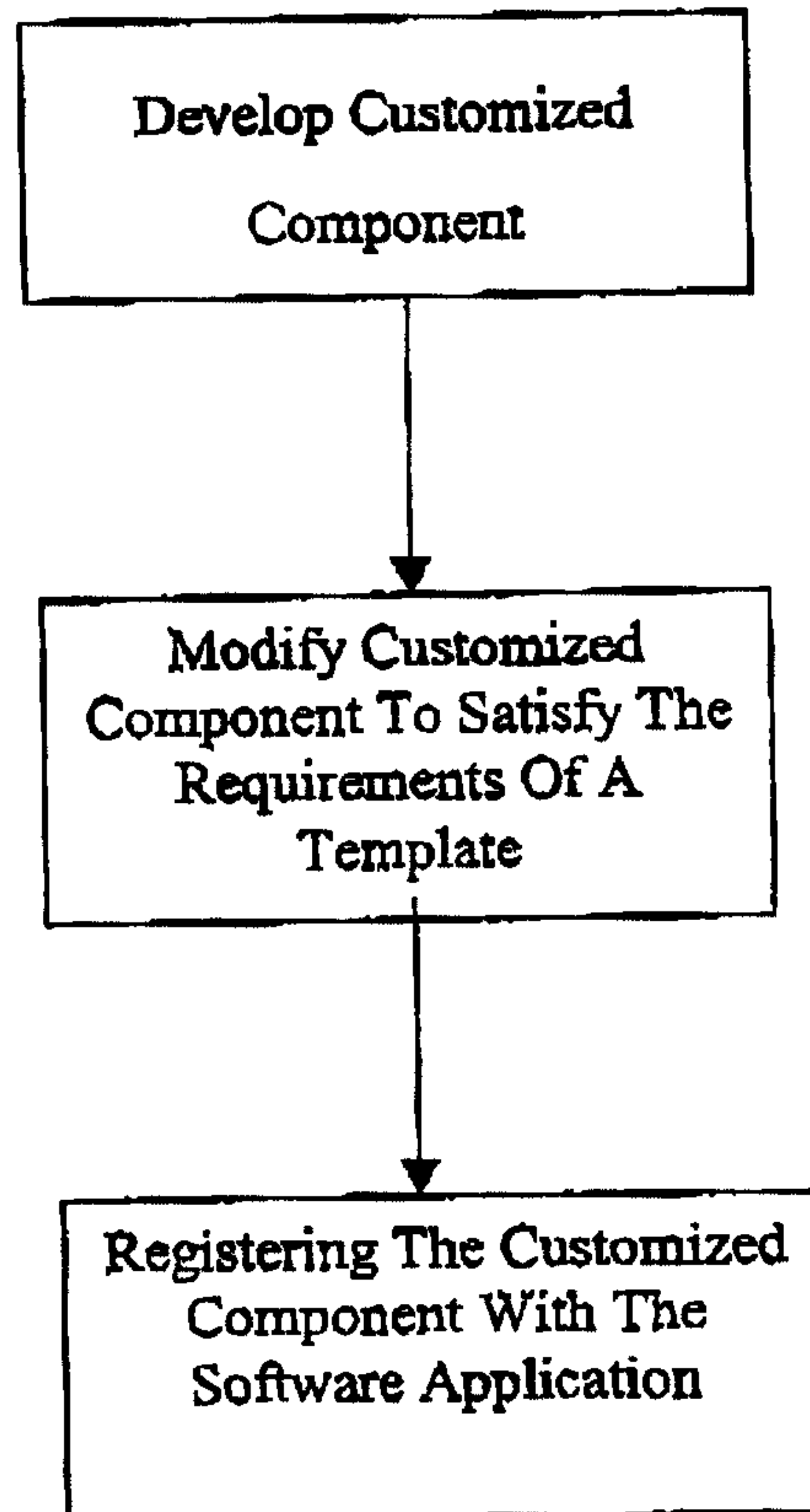
29. The system of claim 26, wherein the creating further comprises modifying the custom component to satisfy template parameters described in the criteria provided by the host application.
30. The system of claim 26, wherein the creating further comprises:  
creating a project for the custom component in the development environment.
31. The system of claim 30, wherein the project comprises a control library project.
32. The system of claim 30, wherein the creating further comprises:  
adding a host application reference to the project for the custom component.
33. The system of claim 32, wherein the adding comprises associating a file from a components list within the project in the development environment with the integrated development environment of the host application.
34. The system of claim 26, wherein the creating further comprises:  
adding functionality to the custom component using at least one third party application.
35. The system of claim 26, wherein the custom component is configured for use by the end user to develop workflow models.
36. The system of claim 26, wherein the shared cache includes custom components shared by a plurality of host applications.



37. The system of claim 26, wherein the host application comprises a workflow application and wherein the custom component is available for use by a plurality of workflow processes or workflow models in the workflow application.

38. The system of claim 26, wherein the host application comprises a workflow application and wherein the custom component is useable with a plurality of workflow objects within a single workflow model in the workflow application.

**FIGURE 1**














File	
New ▶	 Project... Ctrl + Shift + N
Open ▶	 File... Ctrl + N
Close	 Blank Solution...
 Add New Item... Ctrl + Shift + A	
 Add Existing Item... Shift + Alt + A	
Add Project ▶	
 Open Solution...	
 Close Solution	
 Save Form1.cs Ctrl + S	
Save Form1.cs As...	
 Save All Ctrl + Shift + S	
Source Control ▶	
 Page Setup...	
 Print... Ctrl + P	
Recent Files ▶	
Recent Projects ▶	
Exit	

FIG. 2A



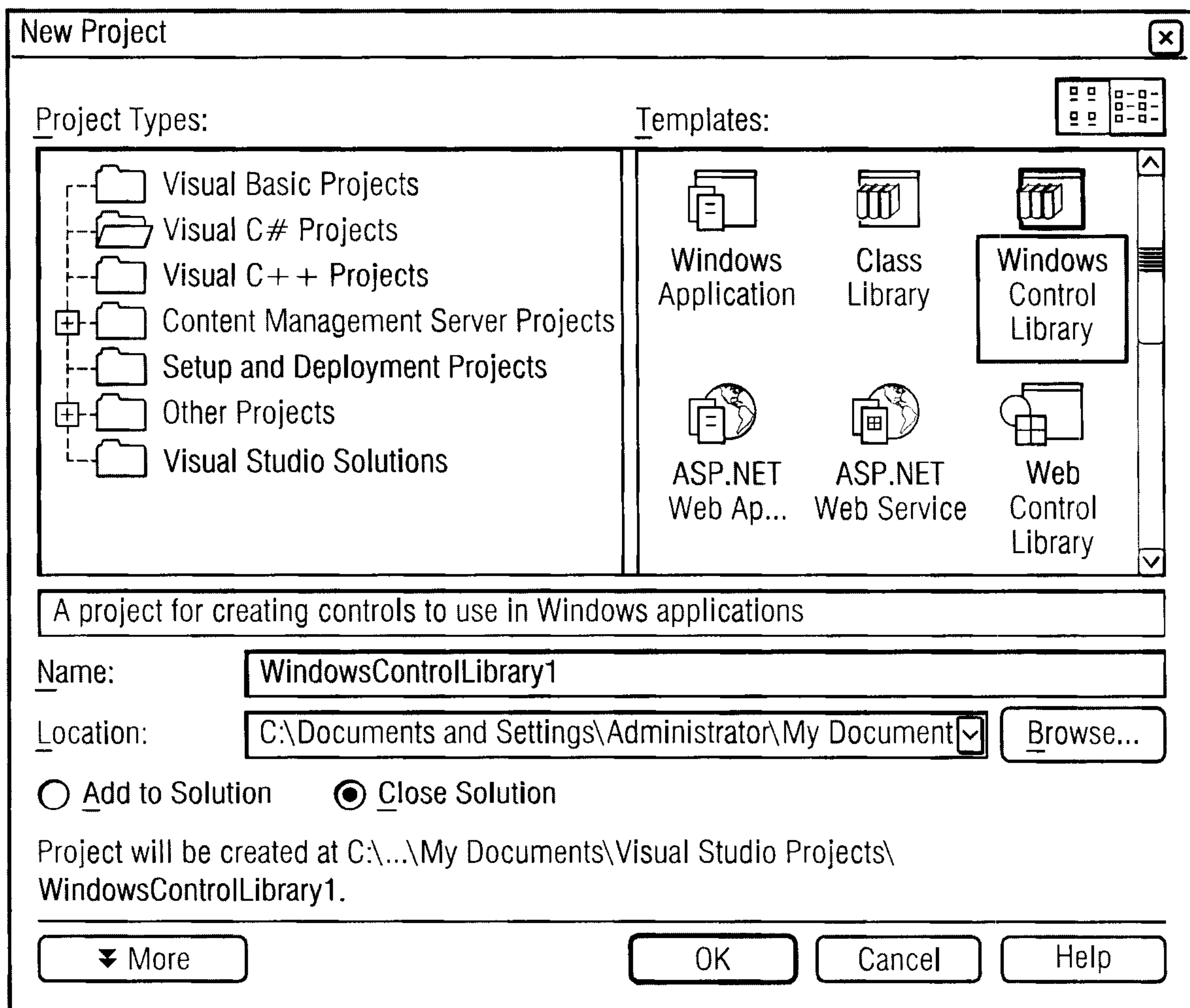


FIG. 2B

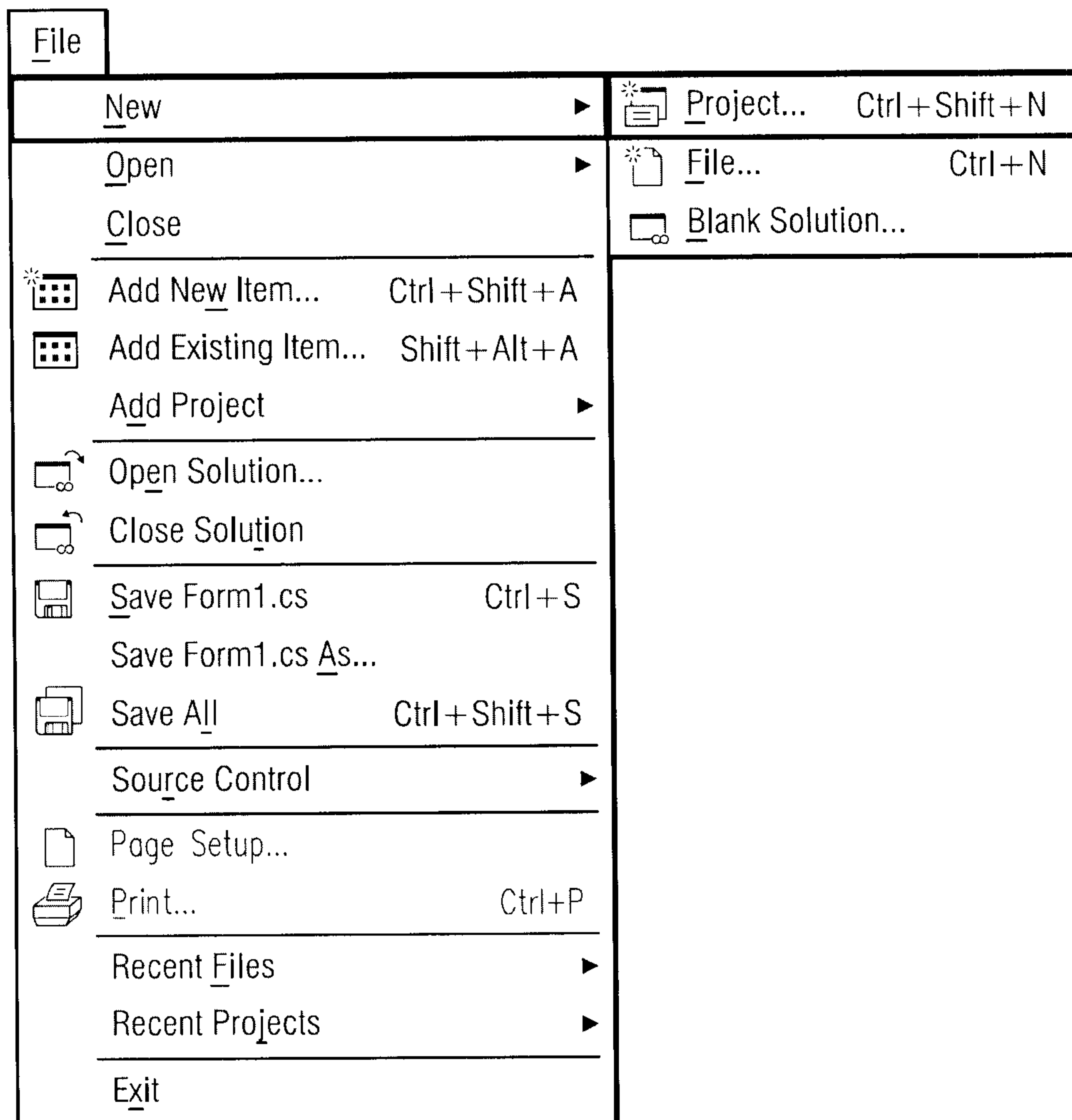


FIG. 2C

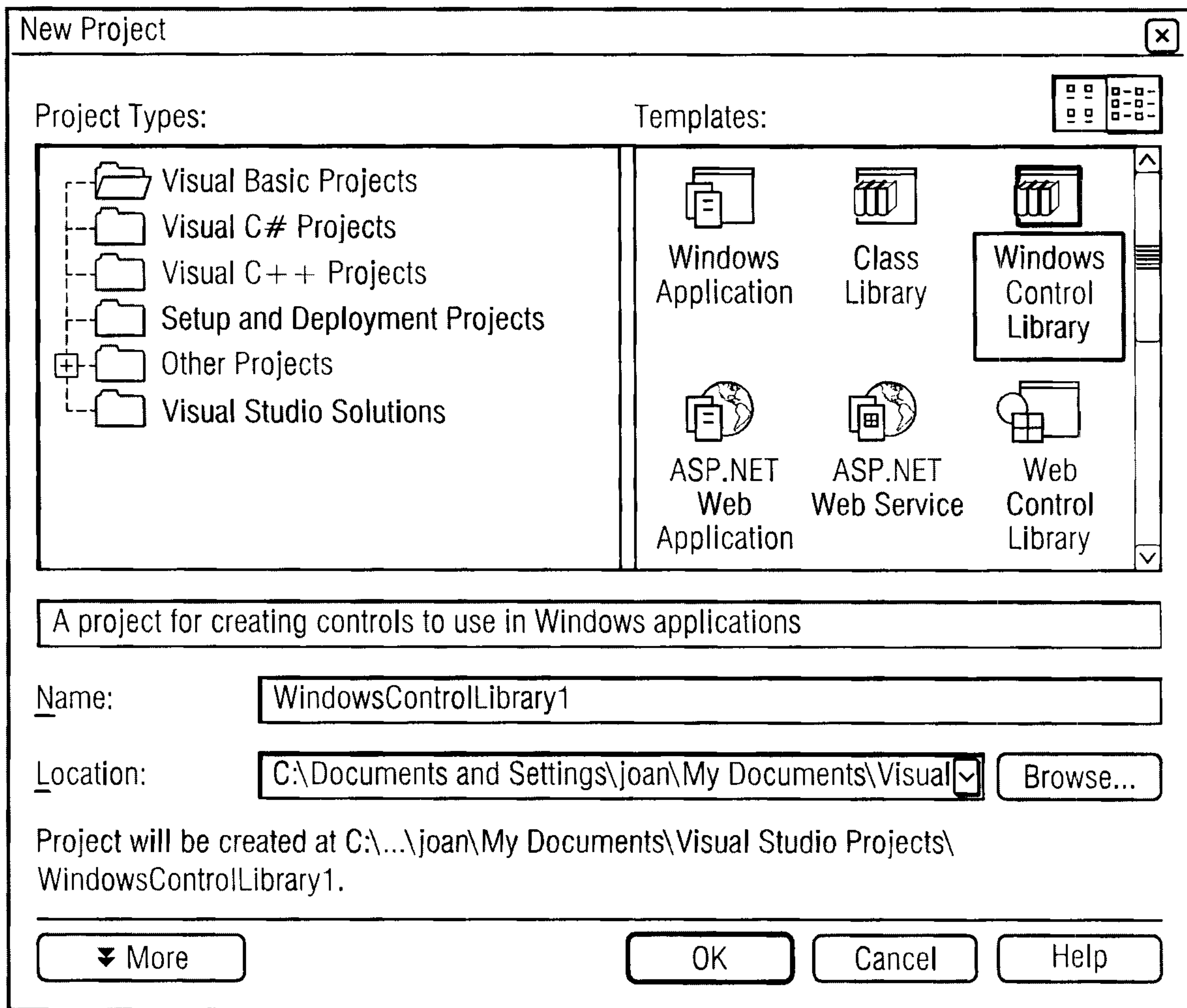


FIG. 2D

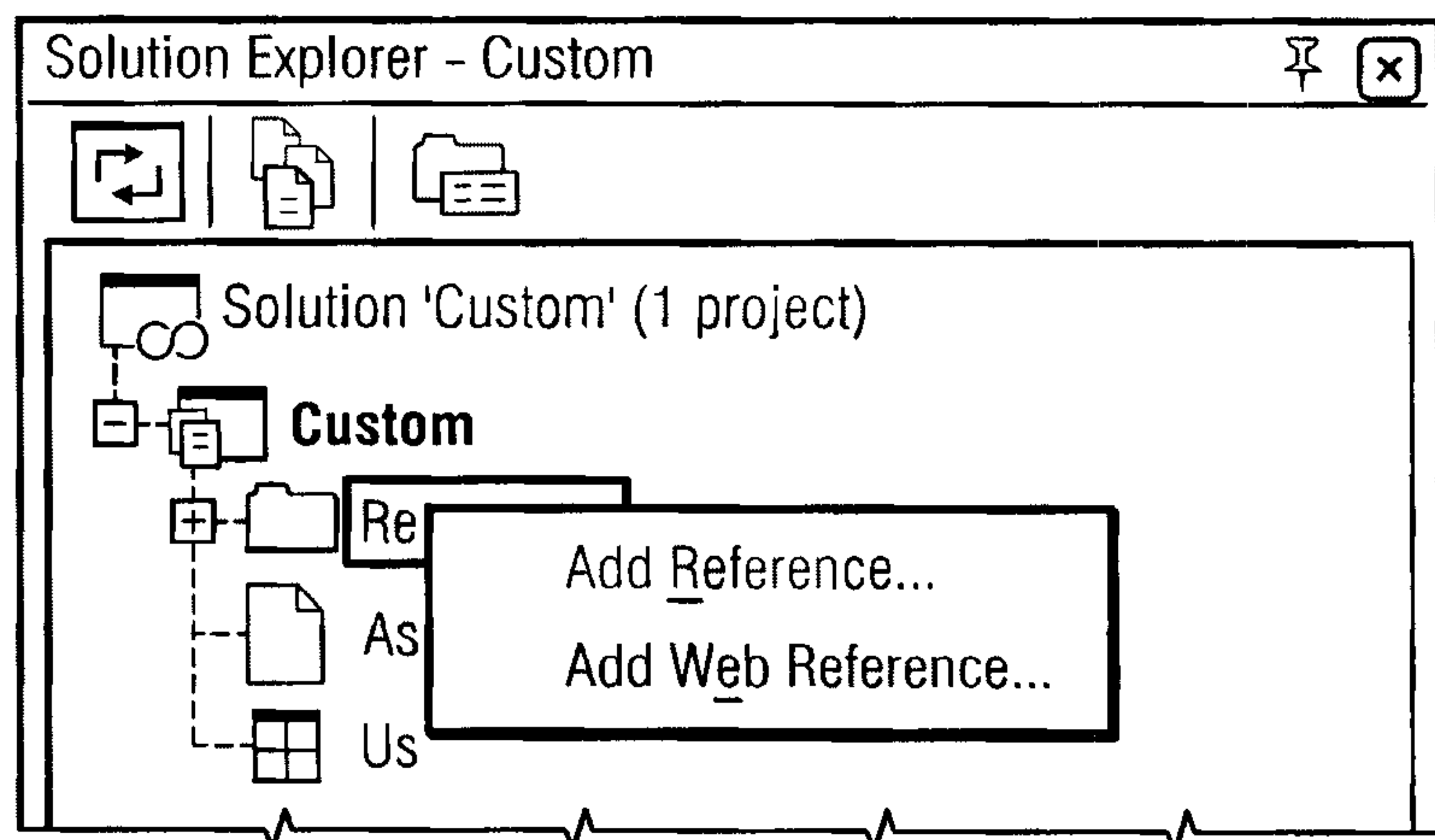


FIG. 2E



Add Reference

.NETCOMProjects

Component Name

Version

Path

Accessibility.dll

1.0.3300.0

C:\WINDOWS\Microsoft.NET\F...

adodb

7.0.3300.0

C:\Program Files\Microsoft.NET...

CRVsPackageLib

1.0.0.0

C:\Program Files\Common Files...

CrystalDecisions.CrystalReports...

9.1.3300.0

C:\Program Files\Common Files...

CrystalDecisions.ReportSource

9.1.3300.0

C:\Program Files\Common Files...

CrystalDecisions.Shared

9.1.3300.0

C:\Program Files\Common Files...

CrystalDecisions.Web

9.1.3300.0

C:\Program Files\Common Files...

CrystalDecisions.Windows.Forms

9.1.3300.0

C:\Program Files\Common Files...

CrystalEnterpriseLib

1.0.0.0

C:\Program Files\Common Files...

CrystalInfoStoreLib

1.0.0.0

C:\Program Files\Common Files...

CrystalKeyCodeLib

1.0.0.0

C:\Program Files\Common Files...

CrystalPluginMorLib

1.0.0.0

C:\Program Files\Common Files...

Browse...

Select

Selected Components:

Component Name

Type

Source

TemplateWinIDE.dll

File

C:\Program Files\Teamplate\Tea...

Remove

OK

Cancel

Help

FIG. 2F

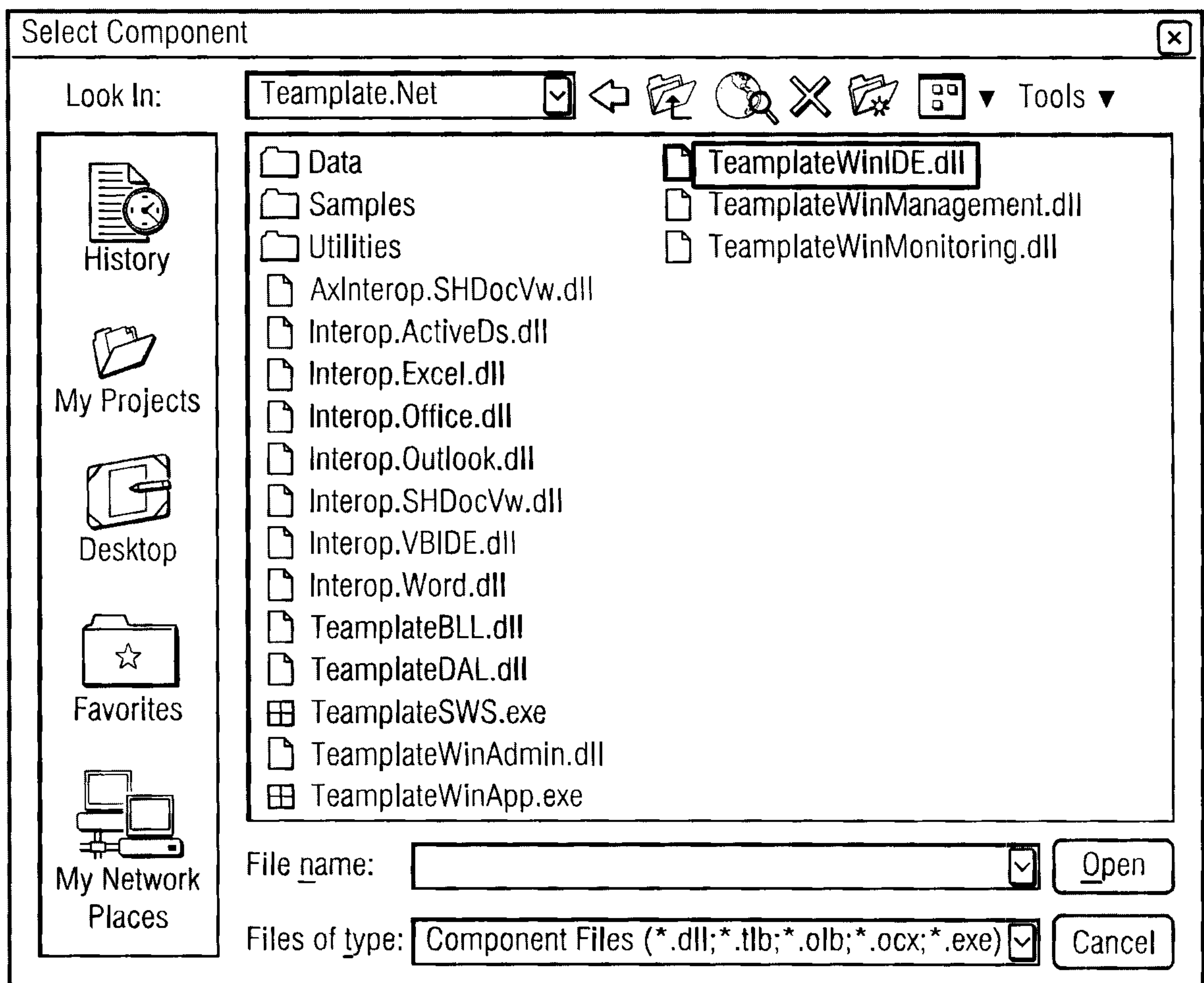


FIG. 2G

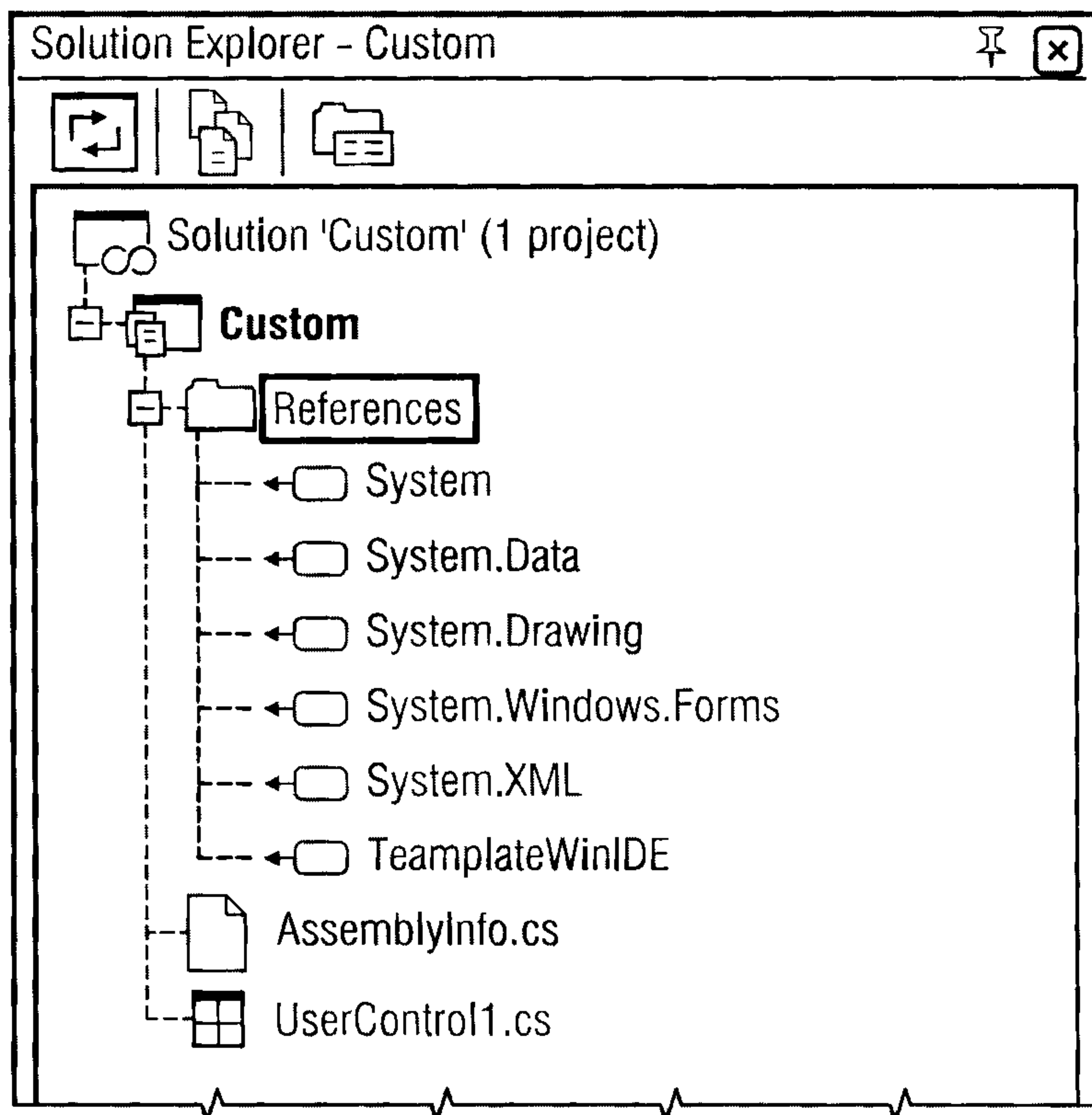


FIG. 2H

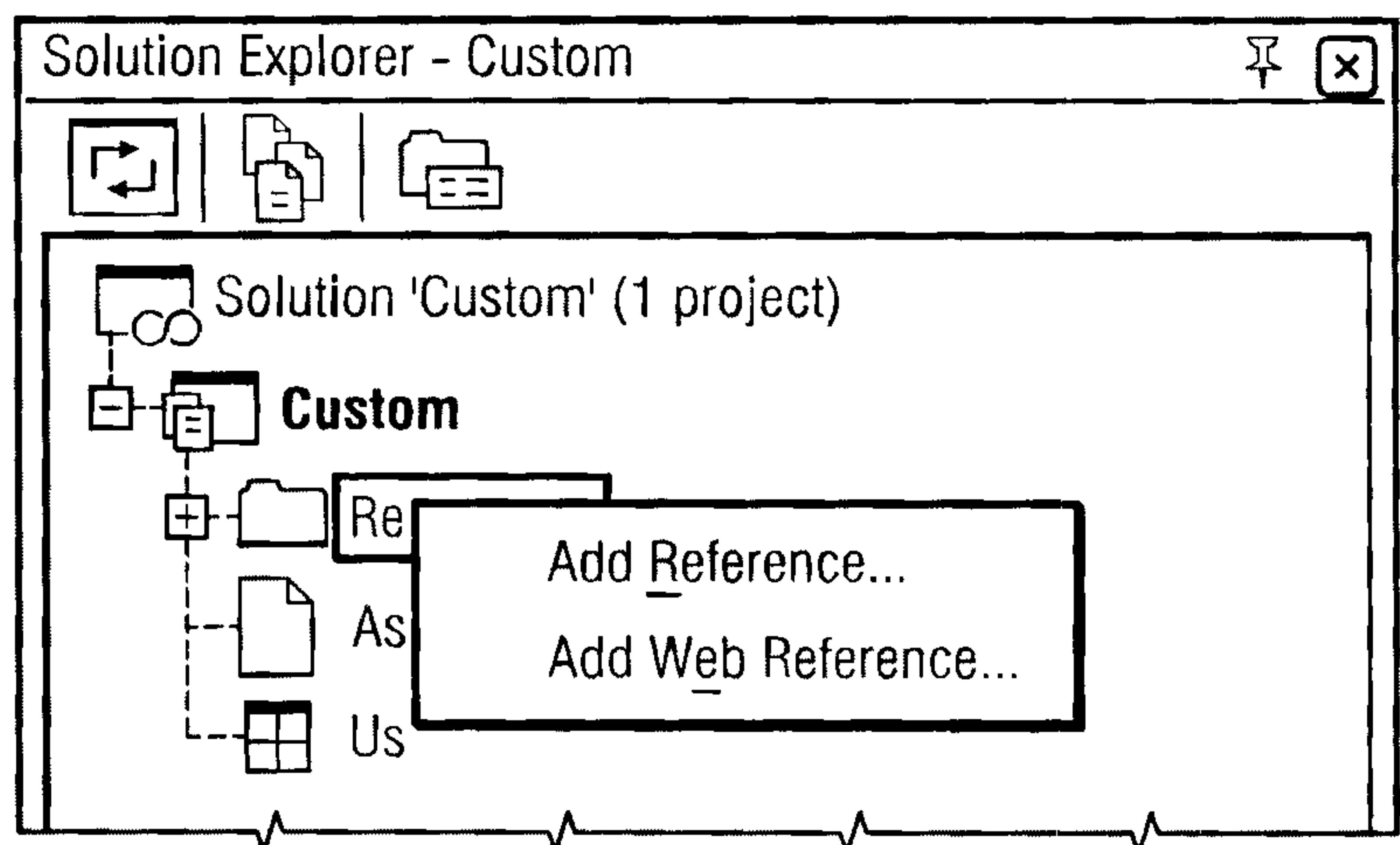


FIG. 2I



Add Reference

.NETCOMProjects

Component Name	Version	Path
Accessibility.dll	1.0.3300.0	C:\WINDOWS\Microsoft.NET\F...
adodb	7.0.3300.0	C:\Program Files\Microsoft.NET...
CRVsPackageLib	1.0.0.0	C:\Program Files\Common Files...
CrystalDecisions.CrystalReports...	9.1.3300.0	C:\Program Files\Common Files...
CrystalDecisions.ReportSource	9.1.3300.0	C:\Program Files\Common Files...
CrystalDecisions.Shared	9.1.3300.0	C:\Program Files\Common Files...
CrystalDecisions.Web	9.1.3300.0	C:\Program Files\Common Files...
CrystalDecisions.Windows.Forms	9.1.3300.0	C:\Program Files\Common Files...
CrystalEnterpriseLib	1.0.0.0	C:\Program Files\Common Files...
CrystalInfoStoreLib	1.0.0.0	C:\Program Files\Common Files...
CrystalKeyCodeLib	1.0.0.0	C:\Program Files\Common Files...
CrystalPluginMorLib	1.0.0.0	C:\Program Files\Common Files...

Browse...

Select

Selected components:

Component Name	Type	Source
TemplateWinIDE.dll	File	C:\Program Files\Teamplate\Tea...

Remove

OK

Cancel

Help

FIG. 2J

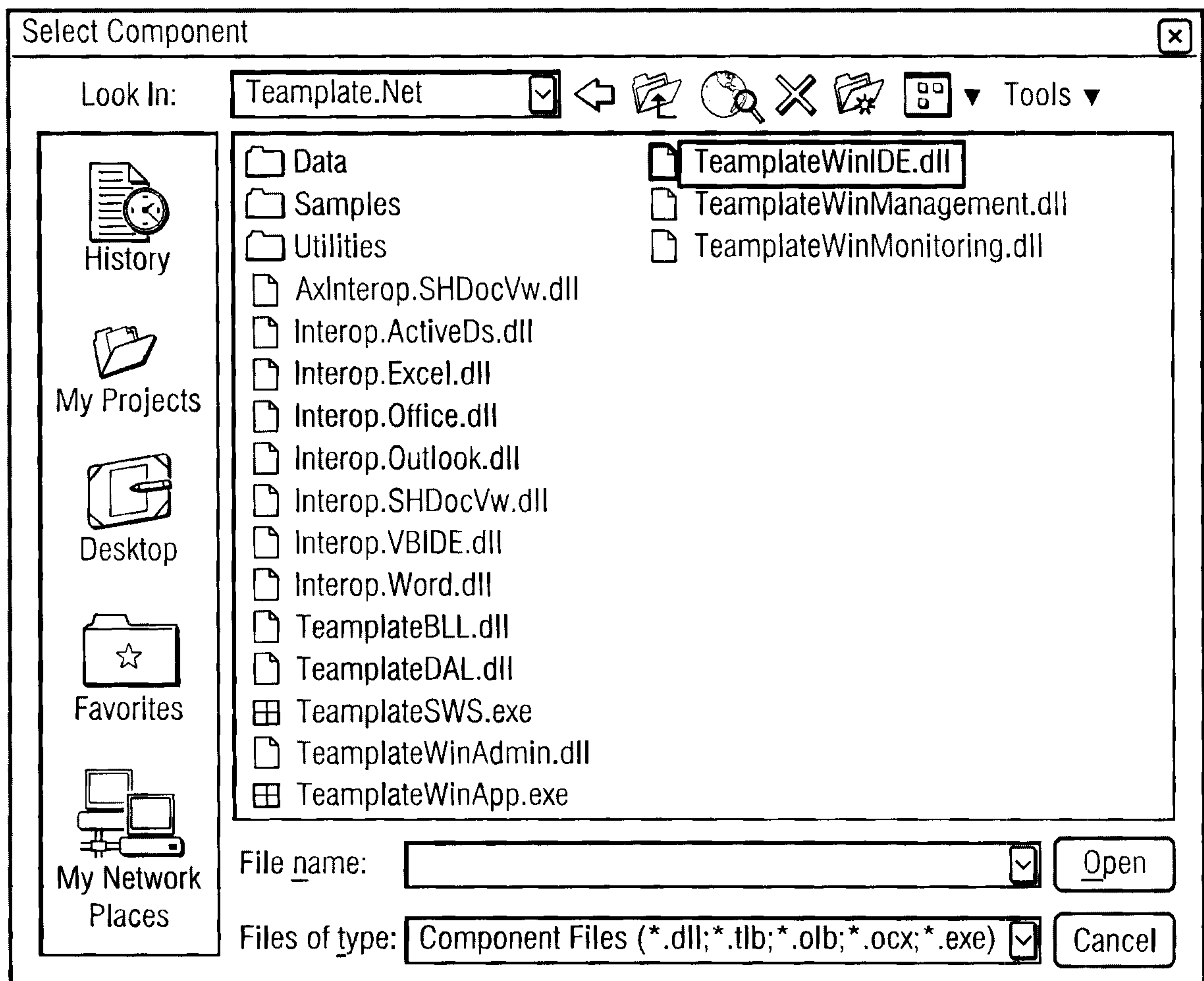


FIG. 2K

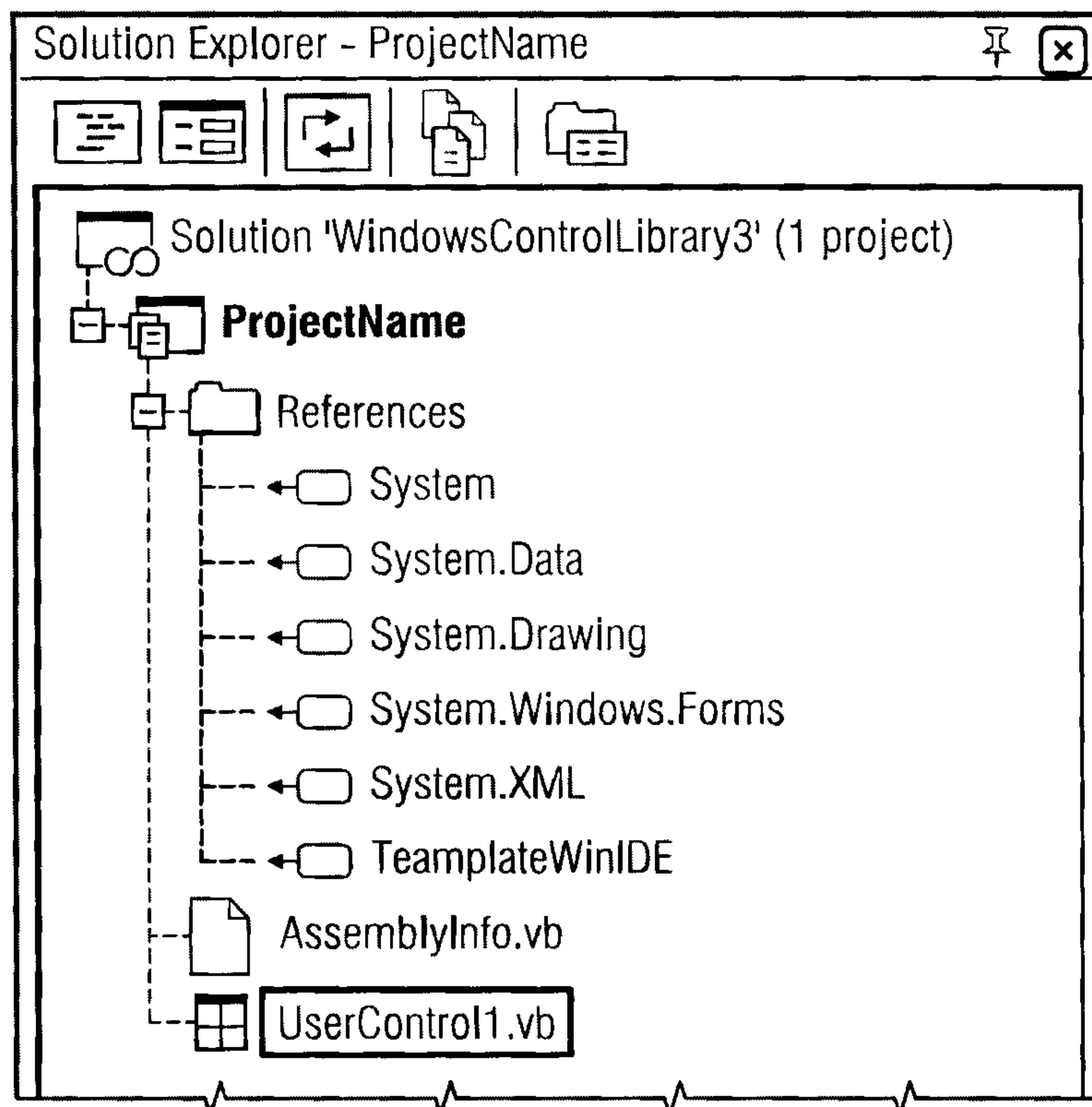


FIG. 2L

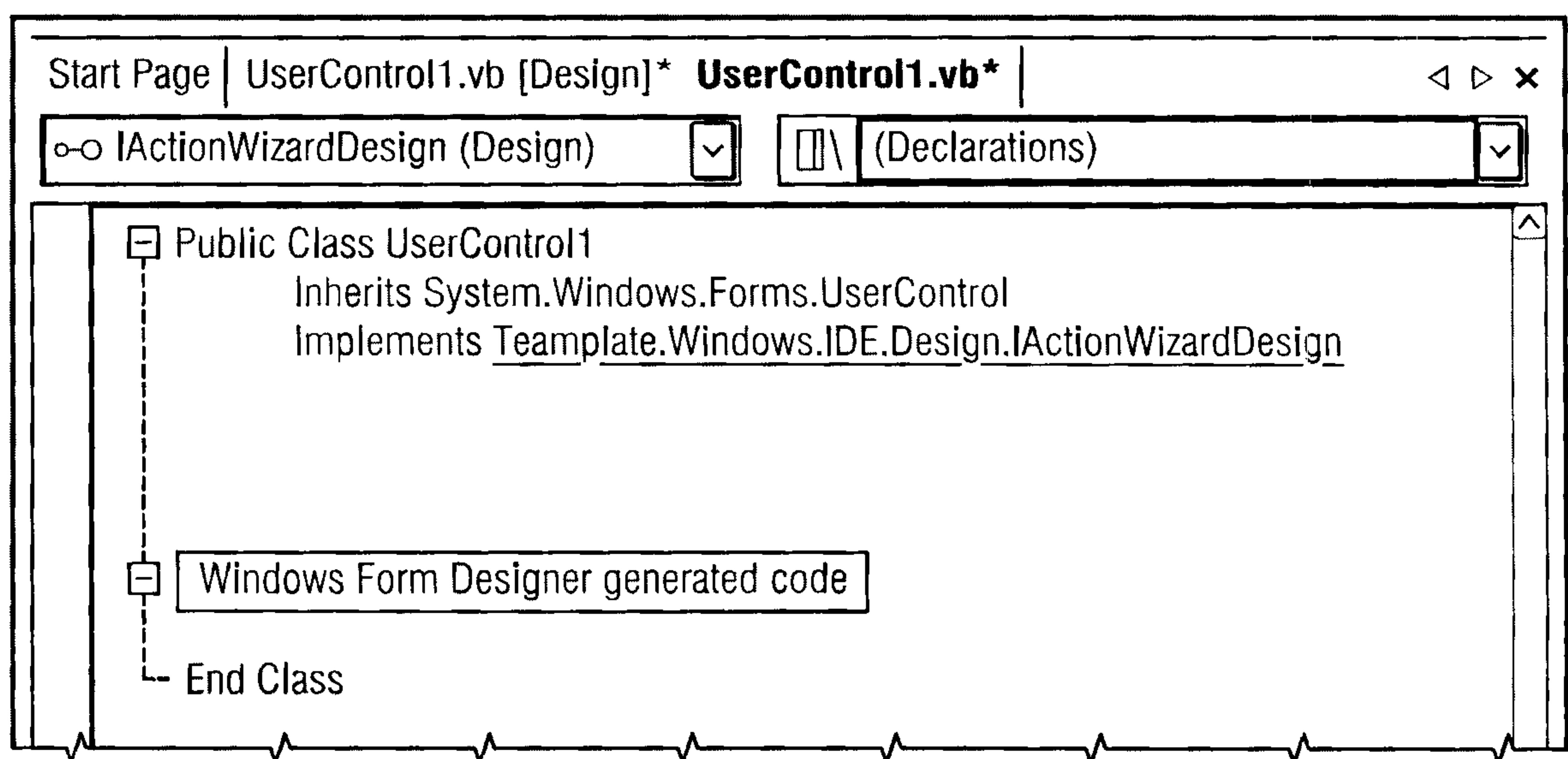


FIG. 2P



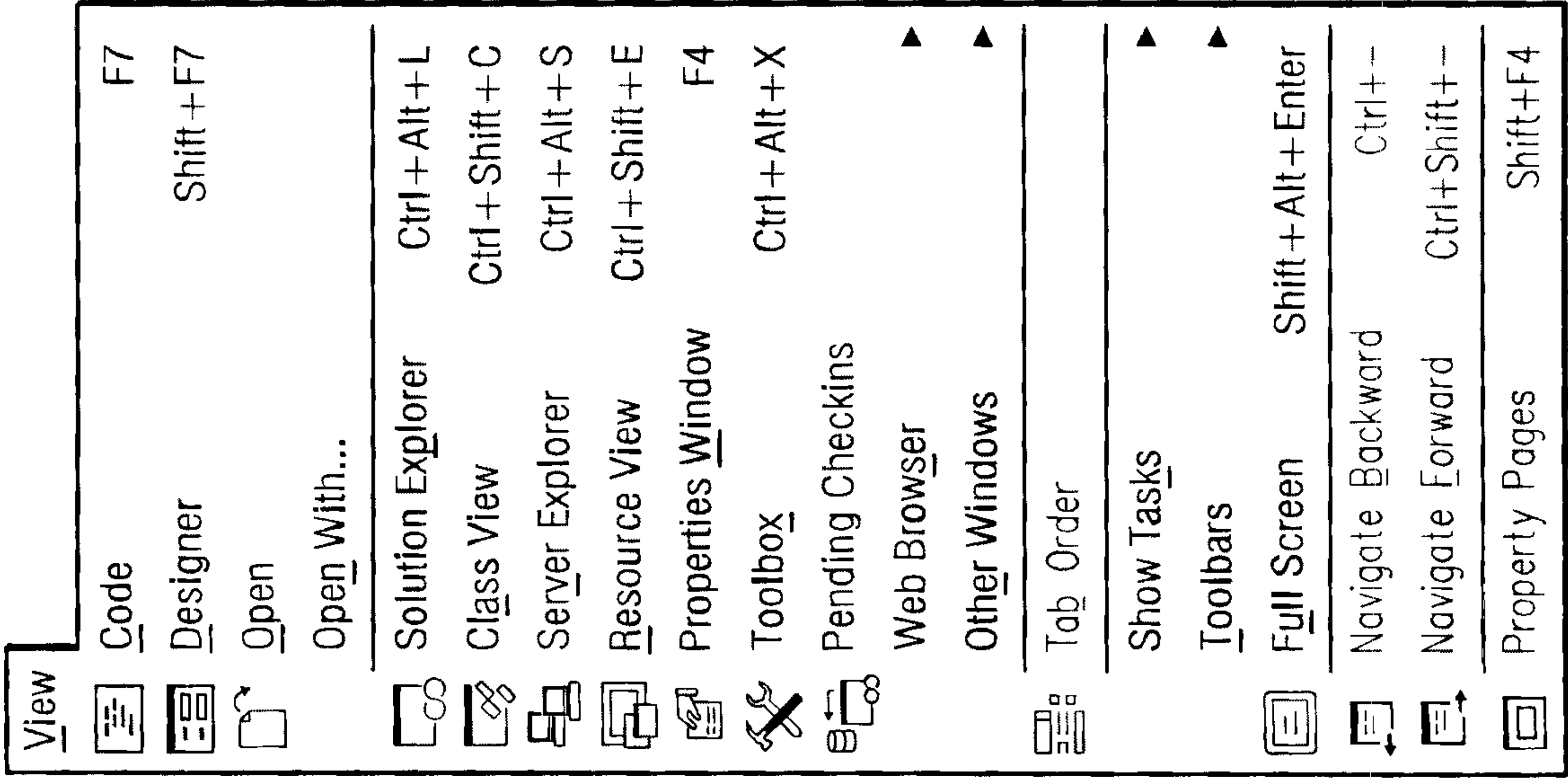


FIG. 2M

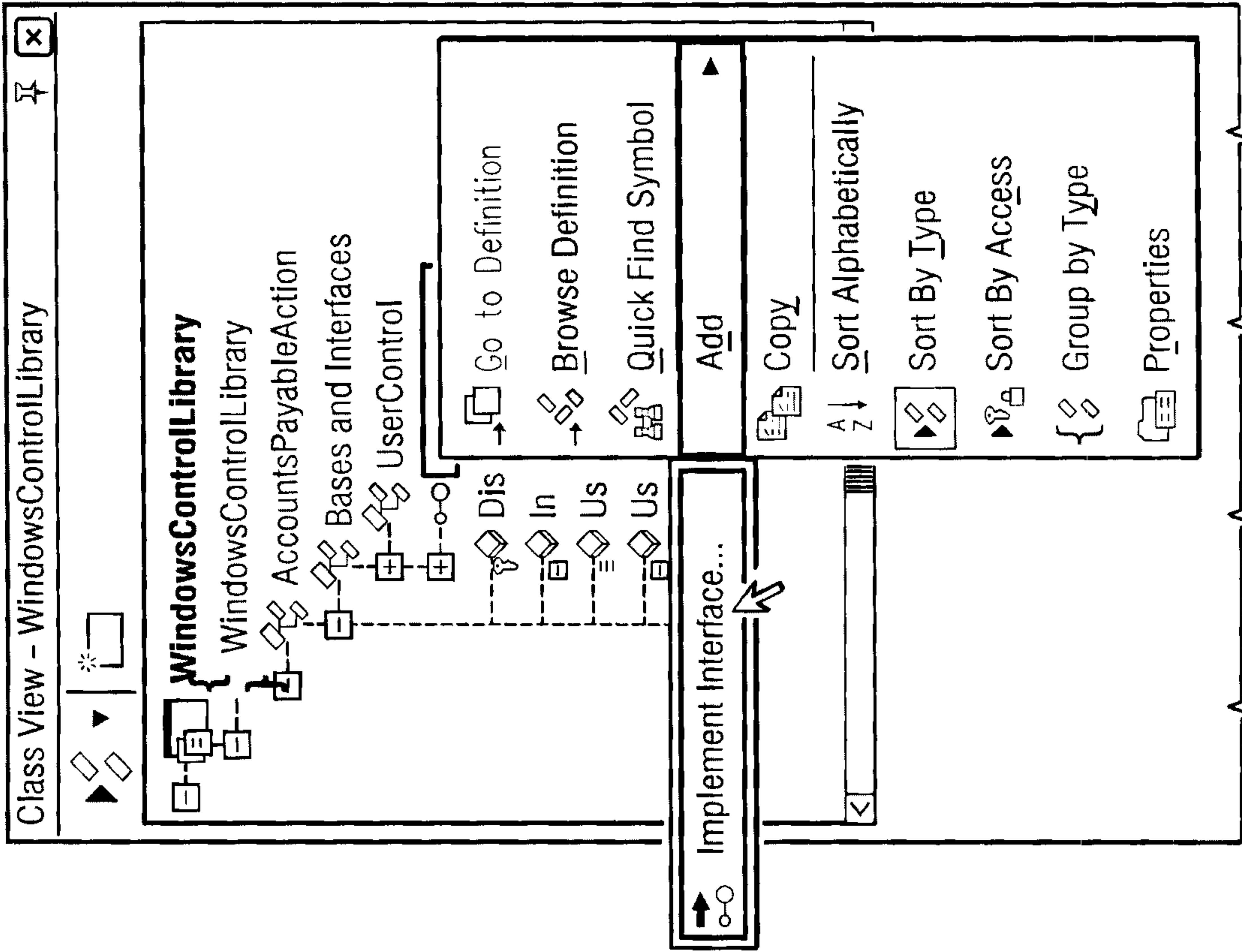


FIG. 2N

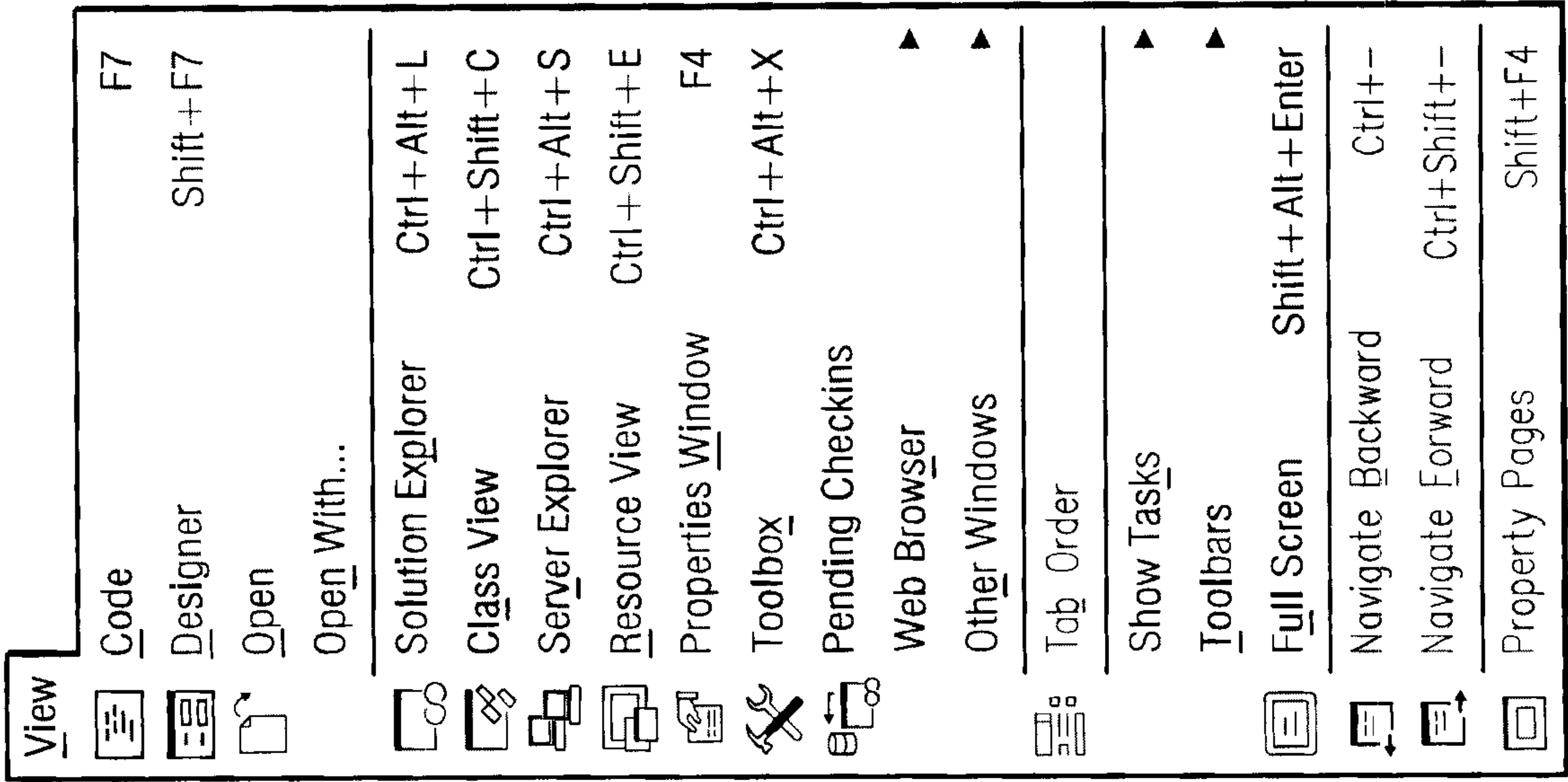


FIG. 20

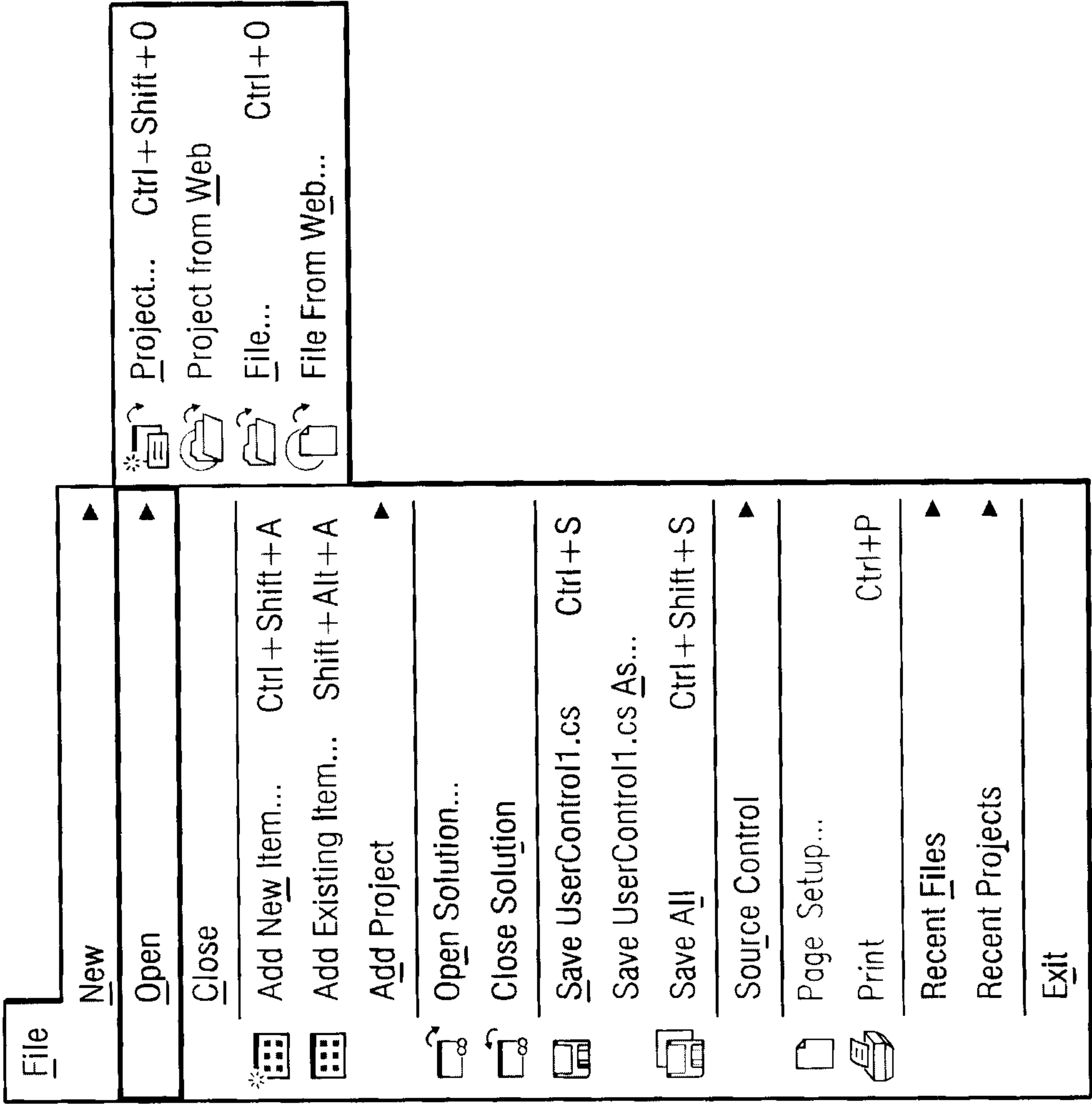


FIG. 2Q

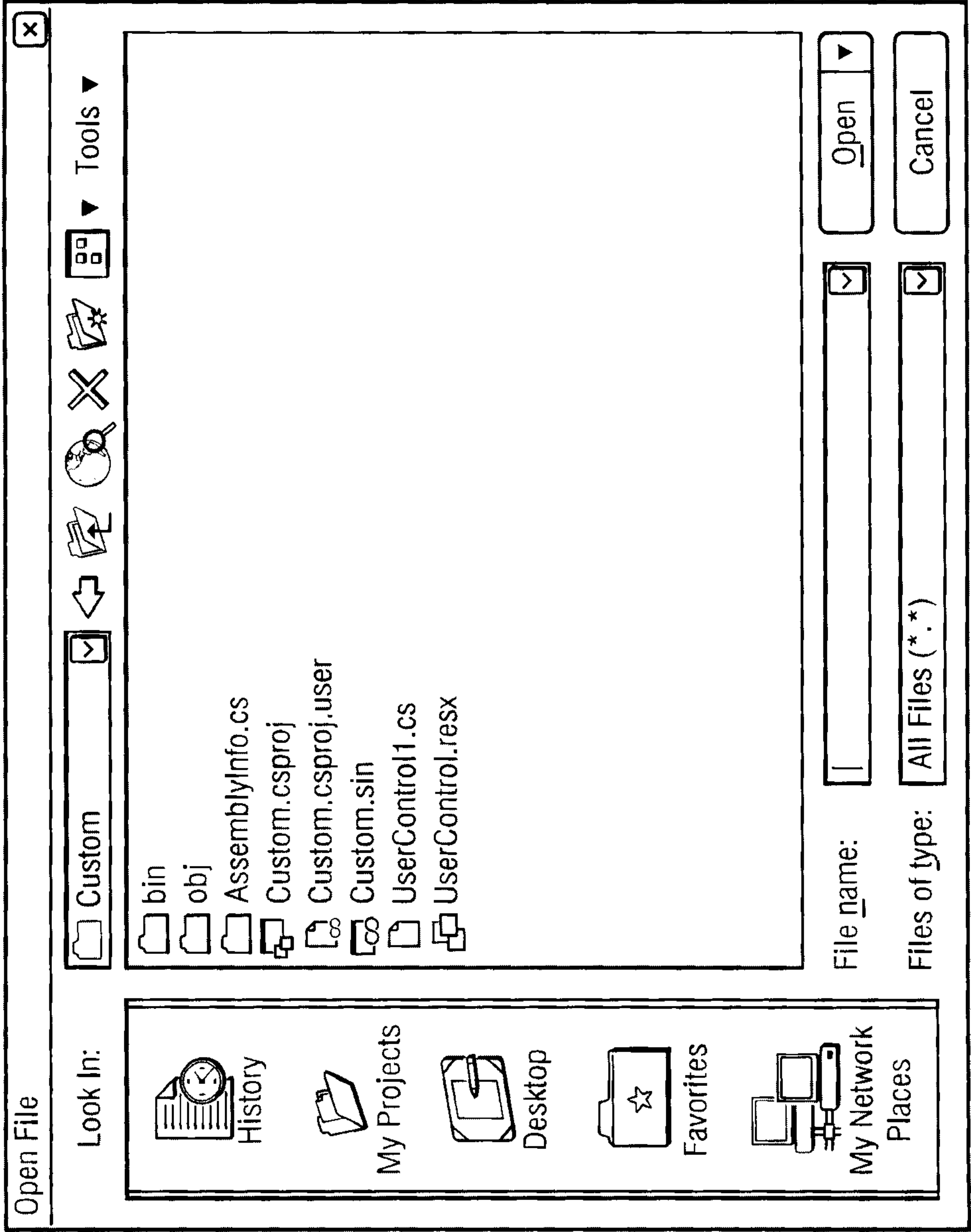


FIG. 2R

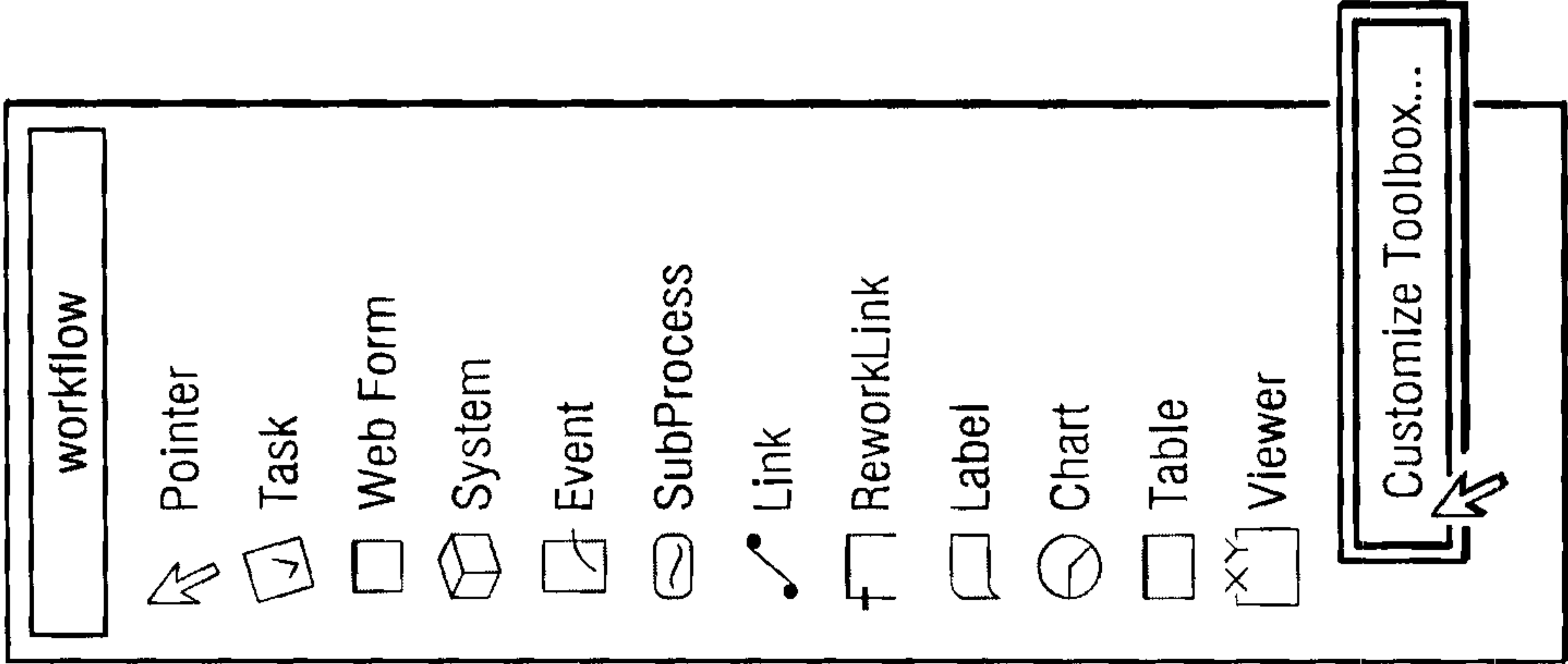


FIG. 2S



Select Actions

Assemble Actions

Action	Assembly
<input type="checkbox"/> CheckBizTalkQueue	Template.Actions.Biztalk
<input type="checkbox"/> ReceiveFromBizTalk	Template.Actions.Biztalk
<input type="checkbox"/> SendToBizTalk	Template.Actions.Biztalk
<input type="checkbox"/> Computation	Template.Actions.Core
<input type="checkbox"/> ConfigureTask	Template.Actions.Core
<input type="checkbox"/> ConsumeWebService	Template.Actions.Core
<input type="checkbox"/> DefaultAction	Template.Actions.Core
<input type="checkbox"/> TaskDueDate	Template.Actions.Core
<input type="checkbox"/> TaskResponsibility	Template.Actions.Core
<input type="checkbox"/> GenDocument	Template.Actions.Office
<input type="checkbox"/> GenExcel	Template.Actions.Office

OK

Cancel

Reset

FIG. 2T

**Develop Customized  
Component**

```
graph TD; A[Develop Customized Component] --> B[Modify Customized Component To Satisfy The Requirements Of A Template]; B --> C[Registering The Customized Component With The Software Application];
```

**Modify Customized  
Component To Satisfy The  
Requirements Of A  
Template**

**Registering The Customized  
Component With The  
Software Application**