
Q-flow 3.0: Designing custom forms

Manual code: Qf304013ENG
Version: 1.0
Applies to: Q-flow 3.04
Latest revision: 29/5/2009

Q-flow 3.0

Designing custom forms

© Urudata Software
Canelones 1370 • Piso 2 CP11200
Montevideo, Uruguay
Teléfono: (598 2) 900 76 68 • Fax: 900 78 56

Table of Contents

Introduction	4
Designing custom forms	4
Configuring Visual Studio 2005 for using the control library	4
Q-flow forms controls.....	7
Validation resources	8
Compatibility with forms of previous versions	10

Introduction

The purpose of this manual is to explain how to design custom forms. Custom forms are ASP .Net pages that can be used to replace standard Q-flow forms. The concept of custom form is explained in the business process modeler manual, which also explains how you can associate a form with a template or a step of a template. This manual is about building custom forms only.

Readers of this manual should be familiar with the most important aspects of Q-flow and with process design with Q-flow. In addition to that, they should possess knowledge of ASP .Net 2.0 programming.

Designing custom forms

Custom forms are aspx pages. They include HTML code and are linked to a file containing C# code. The development of a custom form requires programming skills, knowledge of HTML and of the chosen design tool (usually, Visual Studio 2005 or Visual Web Developer 2005 Express Edition, which available for free).

The class corresponding to a start form should be derived from the `Qflow.Web.CustomForms.StartFlowBase` class. The class corresponding to a response form should be derived from the `Qflow.Web.CustomForms.TaskResponseBase` class. The class corresponding to a workflow form should be derived from the `Qflow.Web.CustomForms.FlowFormBase` class.

Q-flow provides a start form sample (`StartFlowSample.aspx`), located on the `CustomForm` subfolder of the Q-flow site folder. Other sample forms may be found in that folder as well.

To create a custom form out of an already existing one, do the following:

1. Copy and rename both files (the one with extension `aspx` and the one with extension `cs`) of the original form.
2. Open both files in Visual Studio 2005, or in any other design application.
3. Modify the "CodeFile" attribute for the `aspx` page with the new file name followed by the `cs` extension.
4. Perform the needed changes to both files.
5. Copy them to the `CustomForm` subfolder, located in the Q-flow site folder. Remember to create, in the template, a custom form with a URL that points to the new form. You also must assign the custom form to the step you designed it for.

Q-flow provides a control library for designing custom forms. These controls allow you to include in your custom forms all elements Q-flow uses in its standard forms, and to have access to all functionalities of Q-flow's standard forms. The control library has been implemented in the `Qflow.Web.CustomForm.dll` file.

Configuring Visual Studio 2005 for using the control library

Prior to the actual design work, it is convenient to add the Q-flow controls to the Visual Studio 2005 toolbox. In order to do that, do the following:

1. If the Q-flow web site is not installed in the computer where will work, copy the file `Qflow.Web.CustomForm.dll` from the web site server. This file is located in the subfolder "bin" of

- the folder where the Q-flow web site is installed. By default, the folder where the file is located on the website server is: C:\inetpub\wwwroot\QflowWebSite\bin.
2. Open Visual Studio 2005.
 3. Select the “Toolbox” option from the “View” menu. This will cause Visual Studio to open the toolbox.
 4. Right click on the toolbox. Visual Studio will open the context menu. Select the “Select items ...” option. Visual Studio will open a window as shown in Figure 1.

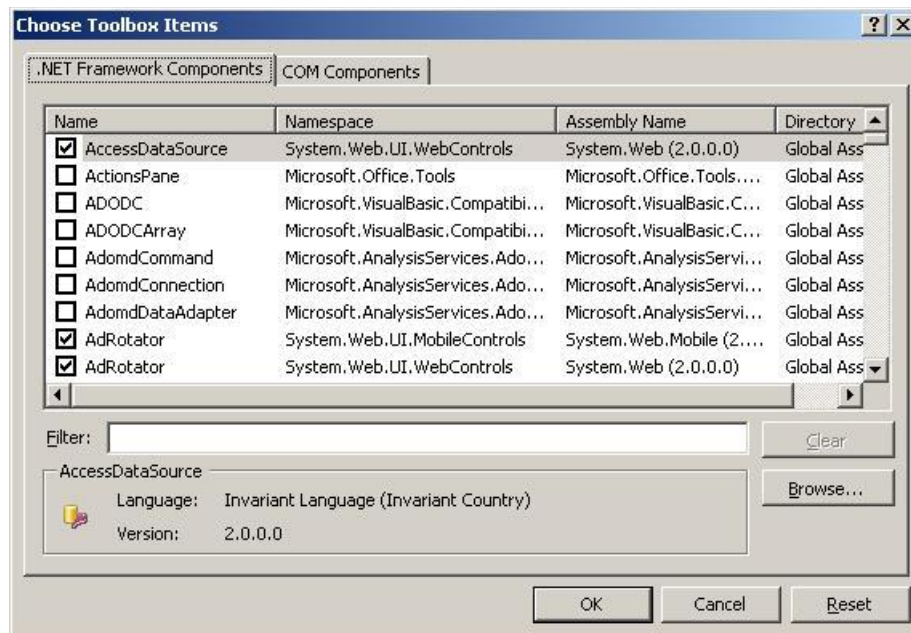


Figure 1 The window which allows you to add controls to Visual Studio's toolbox

5. Click on “Browse ...” to search for the Qflow.Web.CustomForms.dll file. Browse until you find the folder containing the file, and double click on the file. This will add the Q-flow controls to the list (Figure 2).
6. Click on “OK”. This will add the Q-flow controls. Next time a web project is opened on Visual Studio, the Q-flow controls will be loaded in the toolbox (Figure 3).

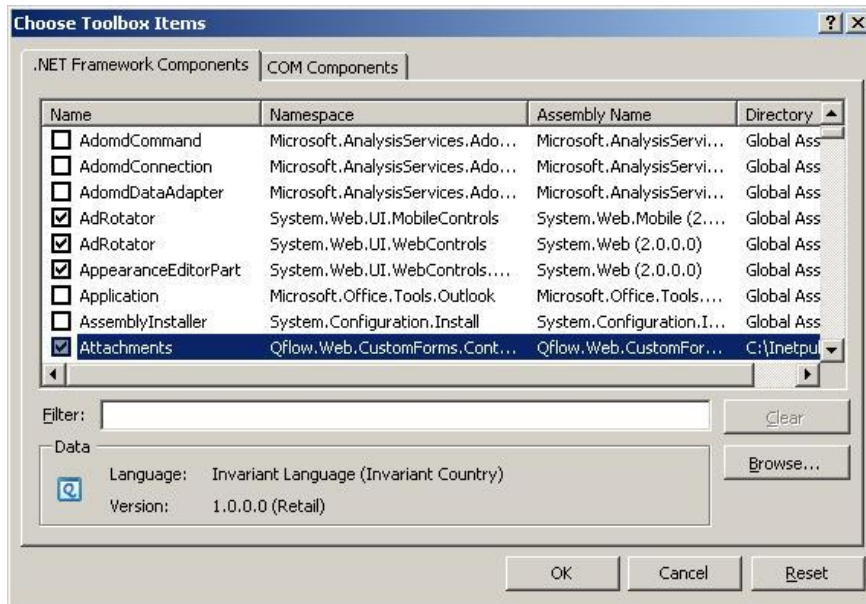


Figure 2 The Q-flow form controls have just been added

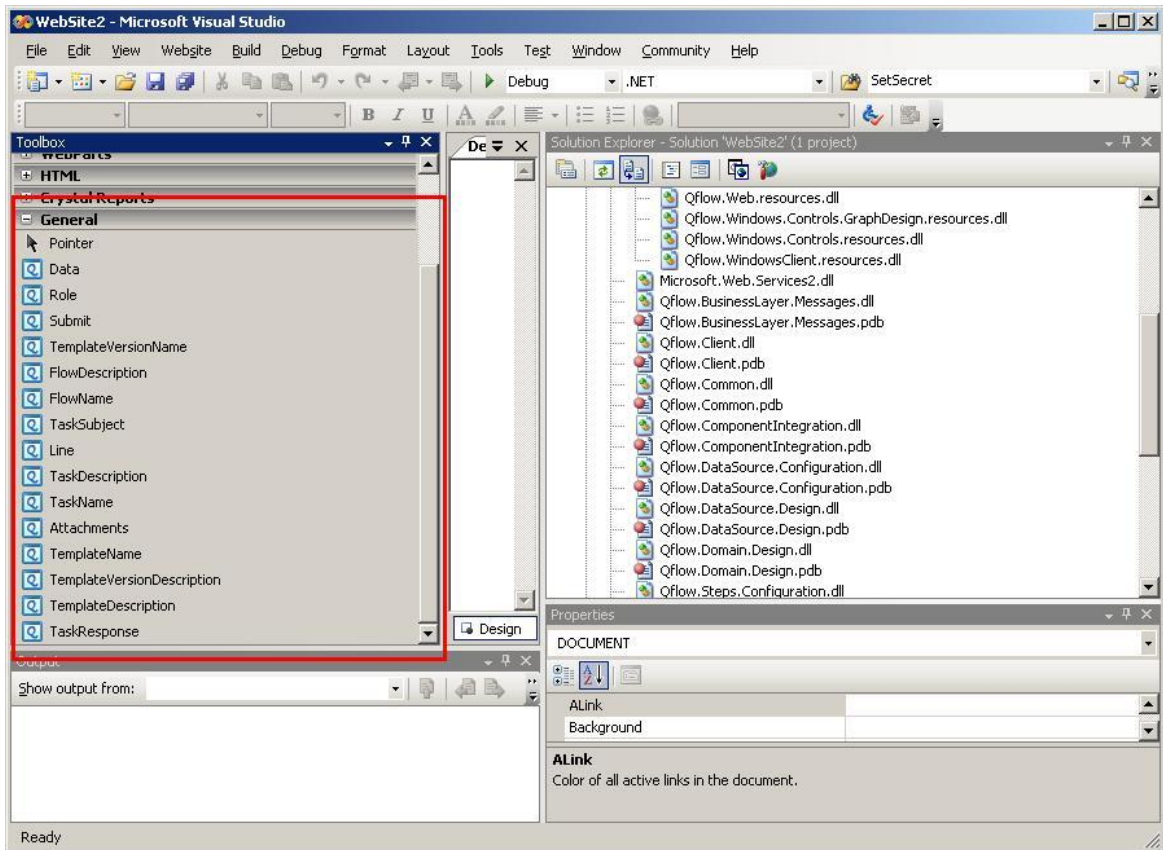


Figure 3 Visual Studio's toolbox with Q-flow controls

Q-flow forms controls

All controls have the properties of the WebControl class, as well as some additional properties. Besides, all controls have the **DisplayMode** property. The value of this property specifies how Q-flow should show the control in the web site. There are three options:

- **ControlWithLabel:** Q-flow will show the control next to a label that will show its name.
- **Control:** Q-flow will show the control only, with no label.
- **Label:** Q-flow will show the label only.

The controls for Q-flow forms are the following:

- **Data:** represents an application datum. When Q-flow shows the form in the web site, it will replace this control with the control associated with the domain to which the datum belongs. The value of the **DataName** property is the name of the datum represented by the control. When Q-flow shows the form, it will use this name in order to know what datum it must show in this control's position.
- **Role:** represents a template role. When Q-flow shows the form in the web site, it will replace this control with a control that allows one to choose the user that will perform this role, or with a control that shows what user is performing that role, depending on whether the role is editable in the step associated with the step. The value of the **RoleName** property is the name of the role. When Q-flow shows the form in the web site, it will use this name in order to know what role it must show in this control's position.
- **Submit:** it is a button. It acquires a different meaning according to the type of form in which it is used:
 - In a start form, it is a button that allows one to start a workflow.
 - In a response form, it is a button that allows one to respond a task.
 - In a workflow form, it is meaningless and it does not make sense to use it.
- **TemplateVersionName:** this control shows the name of the version of the template which is being used.
- **FlowDescription:** This control shows a description of the current workflow, or allows one enter it, in the case of a start form.
- **FlowName:** this control shows the current workflow's name, or allows one to enter it, in the case of a start form.
- **TaskSubject:** This control shows the subject of a task. It is useful in response forms.
- **Line:** This control defines a line block. It is useful to group Data controls that reference data belonging to line blocks.
- **TaskDescription:** This control shows the description of a task. It is useful in response forms.
- **TaskName:** This control shows the name of a task. It is useful in the response forms.
- **Attachments:** Shows the files attached to a workflow. If the attached files are modifiable in the step associated to the form, it allows one to add and remove attached files.
- **TemplateName:** Shows the name of the template to which the workflow associated with the form belongs.
- **TemplateVersionDescription:** Shows the description of the template's version being used.
- **TemplateDescription:** Show the description of the template to which the workflow associated with the form belongs.
- **TaskResponse:** Shows a list with the possible options of response, and allows one to choose one of the responses.

Validation resources

Q-flow provides mechanisms to interact with its controls so that client side validations can be made. Q-flow's controls internally use simpler controls (such as a text box), and allow access to those internal controls so that a script may interact with them and make validations to their content. A developer may include these mechanisms in a custom form. To do so, he must use the scripts provided in the ElementValidationHandlerScript.js file, which is in the Q-flow website folder.

This file provides the following functions:

- **GetFlowNameElement():** gets the HTML element that contains the workflow's name.
- **GetFlowDescriptionElement():** gets the HTML element that contains the workflow's description.
- **GetTaskNameElement():** gets the HTML element that contains the task's name.
- **GetTaskDescriptionElement():** gets the HTML element that contains the task's description.
- **GetTaskSubjectElement():** gets the HTML element that contains the task's subject.
- **GetTaskResponseElement():** gets the HTML element that contains the task's response.
- **GetTaskProgressElement():** get the HTML element that contains the task's progress percentage.
- **GetSubmitElement():** gets the HTML element corresponding to the button that allows one to answer the task.
- **GetDataElement(dataName, instance):** gets the HTML element that represents the application datum whose name equals the value of the dataName parameter. If the datum allows multiple values, the instance parameter specifies what line should be obtained (0 corresponds to the first line). Otherwise, it's 0.
- **GetDataCount(dataName):** gets the number of lines of the datum whose name is specified by the dataName parameter.
- **GetRoleElement(roleName, instance):** obtains the HTML element that represents the role whose name is specified in the roleName parameter. If the role allows multiple values, the instance parameter specifies which value should be obtained (0 corresponds to the first value). Otherwise, it is 0.
- **GetRoleCount(roleName):** gets the number of lines of a role whose name is specified in the roleName parameter.
- **GetLineDataElement(lineBlock, dataName, instance):** gets the HTML element that represents the datum whose name is specified by dataName and that belongs to the line block specified by lineBlock. The instance parameter specifies which value is to be obtained (0, for the first value, or for the only value, if the datum does not allow multiple values).
- **GetLineCount(lineBlock):** gets the number of lines of the line block specified by the lineBlock parameter.

In some cases, it is possible to modify the values of these elements. For instance, if the template is configured so that a specific datum may be modified in the step associated with the form, then it is possible to modify this datum's value. In other cases, it is not possible to change the datum's value. For example, it is not possible to change a workflow's name once the workflow is running.

Example: assign the current date to a datum of the "Date" type

Suppose that a process template has an application datum of the "Date" type. You wish the form to automatically load the current date into this data when a user is about to respond to the step. In order to achieve that, simply paste the following script to that step's custom form.

```
<script for=window event=onLoad language=vbscript>
  dim dateString
  dim element
  dim dateTypeDatumName
  dateTypeDatumName = "DATE"
  dateString = FormatDateTime(Date(),2)
  set element = GetDataElement(dateTypeDatumName)
  element.value = dateString
  element.nextSibling.childNodes(0).value = dateString
</script>
```

The first line of the declarations assigns the “dateTypeDatumName” variable the name of the datum, i.e. “DATE”. Data names are case sensitive. In the following line the current date is obtained, and it is stored in the dateString variable. Then, the GetDataElement function is used in order to assign to a variable the element which represents the “DATE” datum in the screen. Finally, the value of the dateString variable is assigned to the value of the element that represents the datum. The last line modifies the value that is shown to the user: the value of the datum is the one kept in element.value, but the value shown to the user is the one that is kept in element.nextSibling.childNodes(0). The last line is not necessary for data of the text type. With this script, when the user opens the form, the text box that shows the value of the “DATE” value will show the current date.

Example: Load a text into a workflow’s name

The following script may be used inside a start form in order to assign a text (for example, “My flow:” followed by the current date) to a workflow’s name.
El siguiente script puede ser utilizado en un formulario personalizado de inicio para asignarle un texto (en el ejemplo “Mi flow:” seguido de la fecha actual) al nombre del workflow.

```
<script for=window event=onLoad language=vbscript>
  dim automaticFlowName
  dim flowName
  automaticFlowName = "My flow: " & Date
  set flowNameElement = GetFlowNameElement()
  flowNameElement.value = automaticFlowName
</script>
```

Example: Roles

A report elaboration process has two roles: Writers and Reviewers. Both allow multiple values. No writer may be part of the reviewers’ team. The start form for workflows based on this process must prevent any workflow from starting if a writer is found to be one of the reviewers. The following script solves the problem:

```
<script for=aspnetform event=onsubmit language=vbscript>
  dim numberOfWriters
  dim roleElement
  'Gets the number of members of a role
  numberOfWriters = GetRoleCount("Writers")
  Dim i
  'The first element corresponds to i=0:
  For i=0 To numberOfWriters - 1
```

```
'Gets the element in the i position:
set roleElement = GetRoleElement("Writers",i)
If IsReviewer(roleElement) Then
    'Shows en error message,
    'roleElement.nextSibling.value contains the role member's
name
    MsgBox("ERROR: The writer " & roleElement.nextSibling.value
& " is also a reviewer.")
    'Cancels the event, preventing the form from being sent
    window.event.returnValue = false
    Exit For
End If
Next
</script>
<script language=VBScript>
function IsReviewer(roleElement)
    dim numberOfReviewers
    numberOfReviewers = GetRoleCount("Reviewers")
    Dim i
    For i=0 To numberOfReviewers - 1
        If roleElement.value = GetRoleElement("Reviewers",i).value Then
            IsReviewer = true
            exit function
        End If
    Next
    IsReviewer = false
end function
</script>
```

The script must be executed when the user clicks on the “Start” button, which causes the form to be sent to the server. The “for=aspnetform” attribute specifies that the script will be applied to the form, and the “event=onsubmit” attribute specifies that it will be executed when the user orders that the form be sent, that is, when the user click on the “Start” button.

The script iterates through all role members of the “Writers” role and checks, for each of them, whether it is also a member of the “Reviewers” role. If a role that satisfies that condition is found, the script shows an error message and cancels the event, thus preventing the form from being sent. The error message will show the name of the user that belongs to both roles.

Notice that in order to obtain the name of a role member, the “roleElement.nextSibling.value” expression is used. However, the “IsReviewer” function uses the “roleElement.value” expression to compare two users. This is possible only because roleElement.value contains the user identifier, while roleElement.nextSibling.value contains the user name, which is what the form displays.

Compatibility with forms of previous versions

Custom forms made to work with previous versions of Q-flow 3 (Q-flow 3.0 and Q-flow 3.01) are not compatible with Q-flow 3.02. However, they can be made to work if a few changes are made.

- If a form includes the Q-flow.Common.Exceptions namespace in the code behind, replace the importation sentence with one that refers to Qframework.Common.Exceptions.
- Standard Q-flow controls, which were in the Qflow.Web.dll component, in the Qflow.Web.Controls namespace, are now in the Qframework.Web.dll component, in the

Qframework.Web.Controls namespace. If some form uses them in its markup, modify the references.

- In the code behind, all references to the Interaction object must be replaced by references to FlowInteraction. In places where this new property is used, the Qframework.Web.Interaction namespace must be imported.
- Most enumerations (enums) were moved from the Qflow.Common namespace to the Qframework.Common namespace. Therefore, if you use an enum, you must change the reference.
- In general, in those exceptional cases in which forms use some other functionality of Q-flow, if the form does not find one of the classes of Q-flow, seek a class with the same name in a similar namespace whose name begins with "Qframework".

Additionally, because of an improvement on the behavior of the "Submit" control of Q-flow, some changes regarding client-side validations in custom forms have appeared.

In the previous version, it was possible to add javascript validation routines to events such as the form's "onsubmit" and the "Submit" button's "onclick", by means of snippets like the following:

- `document.forms[0].attachEvent("onsubmit",validarForm)`
- `GetSubmitElement().attachEvent("onclick",validarForm)`

From this version onwards, these validations, although they will be executed, will not stop the page's postback operation from occurring. However, it is possible to cause these validations written in javascript to be executed normally inside the page's cycle. The key is to use the ASP .Net validation controls, specifying that a client-side validation routine will be used. To do that:

1. In the page's markup, add the ASP .Net validation control, with a definition like the following:

```
<asp:CustomValidator ID="CustomValidator1" runat="server" ClientValidationFunction="validateForm"
EnableClientScript="true"
ValidationGroup="QCommandButtonValidationGroup"></asp:CustomValidator>
```

2. Modify the javascript validation routine so that it can receive the arguments required by the ASP .Net validation framework. In this example, the signature would be the following:

```
function validateForm(source, clientside_arguments)
```

3. Inside the javascript validation routine, the boolean property `clientside_arguments.IsValid` is used to specify whether the validation has been successful or not. If the validation has not been successful, the postback is not performed.