



---

**Q-Pulse Web Service APIs**  
**Developing a SharePoint Web Part**

# Contents

---

Introduction.....	3
Assumptions .....	3
Overview of the Example .....	3
Strategy .....	3
Initial Solution Setup for the Server Control.....	3
Initial Solution Setup for the User Control.....	4
Implementing the Server Control.....	4
Implementing the User Control.....	5
Implementing the Login panel .....	5
Implementing the Search panel .....	6
Implementing the Results panel .....	7
Implementing the Logic .....	7
Button Click logic.....	12
Deployment to SharePoint .....	14
Adding the Web Part to a Web page .....	15
Limitations/Improvements .....	15

## Introduction

The purpose of this tutorial is to demonstrate developing a SharePoint Web Part application which makes use of QPulse Web Services to query Incidents in QPulse.

## Assumptions

The user is assumed to be familiar with C# development in Visual Studio 2008. We also assume that the reader already has IIS 6.0 and SharePoint 2007 installed and configured.

## Overview of the Example

The example application should initially allow the user to authenticate with QPulse. After successful authentication, a search dialog should appear, allowing the user to search Incidents using a keyword. After clicking on a button, the user should then be presented with the results obtained from the Web Service query.

## Strategy

Sharepoint Web Parts are Server controls. In order to make development and customisation as easy as possible, we will develop a Server control which loads in a User control. The majority of the functionality will be contained in the User control.

## Initial Solution Setup for the Server Control

1. Create a new Server Control project by going to File -> New Project -> Visual C# -> Web -> ASP.NET Server Control and name it QPulseSharePointServerControl.
2. In Solution Explorer, expand the Properties folder and open AssemblyInfo.cs. Add the following namespace directive to the top of the file:

```
using System.Security;
```

3. Add the following to the bottom of the file:  
[assembly: AllowPartiallyTrustedCallers]

This allows the Web Part to run under the WSS\_Medium trust level (otherwise trust level 'Full' would be required).

4. Rename ServerControl1.cs to QPulseWebPart.cs and similarly rename the ServerControl1 class to QPulseWebPart.
5. Right-click on the project in Solution Explorer and click on the Signing tab. Tick the 'Sign the assembly' tick box and choose a name key file (or create one).

## Initial Solution Setup for the User Control

1. Create a new User Control by going to File -> New Project -> Visual C# -> Web -> ASP.NET Web Application.
2. In Solution Explorer, expand the Properties folder and open AssemblyInfo.cs. Add the following namespace directive to the top of the file:  

```
using System.Security;
```
3. Add the following to the bottom of the file:  

```
[assembly: AllowPartiallyTrustedCallers]
```
4. Right-click on the project in Solution Explorer and click on the Signing tab. Tick the 'Sign the assembly' tick box and choose a name key file (or create one).

## Implementing the Server Control

1. In QPulseWebPart.cs, change the ToolboxData class attribute to read :  

```
[ToolboxData("<{0}:QPulseWebPart  
runat=server></{0}:QPulseWebPart>")]
```
2. Change the class that is inherited from to  
System.Web.UI.WebControls.WebParts.WebPart
3. In the RenderContents method, replace `output.Write(Text);` with  
`RenderChildren(output);`
4. Override the CreateChildControls method to load in the User control:

```
protected override void CreateChildControls()
{
    base.CreateChildControls();
    try
    {
        Controls.Add(Page.LoadControl("~/controls/QPulseShareP
ointUserControl.ascx"));
    }
    catch (Exception e)
    {
        Debug.WriteLine(e.ToString());
    }
}
```

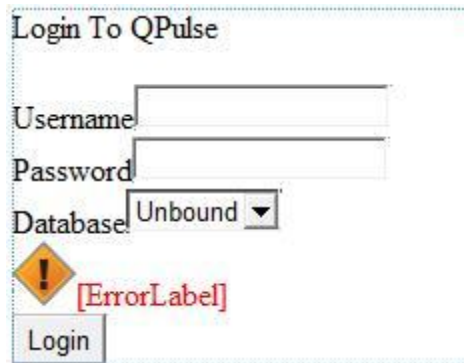
## Implementing the User Control

1. In Solution Explorer, right-click on the project, go to Add -> New Item -> Web User Control and name it QPulseSharePointUserControl.ascx.
2. In Solution Explorer, right-click on the QPulseSharePointUserControl project and click on Add Web Reference. Enter the URL to the service (of the form `http://machineName/QPulseWebServices/services/core.svc`) in the text box and click Go. If successful, this should display the Web Service's web page. In the 'Web reference name' text box, enter 'CoreService' and click the Add Reference button.
3. Similarly, add a Web Reference to `http://machineName/QPulseWebServices/services/incident.svc` and name it 'IncidentService'.
4. Add a new item of type C# -> Web -> Web User Control and name it QPulseSharePointUserControl.ascx.
5. In the design view drag Panels onto the control from the toolbox. Name them LoginPanel, SearchPanel and ResultsPanel and set their visibility property to false.

## Implementing the Login panel

1. Click in the Login panel and type 'Login To QPulse'
2. Add three labels, one beneath the other:
3. UsernameLabel with text 'Username'
4. PasswordLabel with text 'Password'
5. DatabaseLabel with text 'Database'
6. Adjacent to the UsernameLabel and Password label, add a Textbox and name them UsernameTextBox and PasswordTextBox.
7. For the PasswordTextBox, change the TextMode property to Password.
8. Adjacent to the DatabaseLabel, add a DropDownList component and name it DatabaseDropDownList.
9. Beneath the DatabaseLabel, add an Image component and another label. Name them ErrorImage and ErrorLabel respectively and set the Visible property on both to false. Set the ForeColor property of the label to Red and remove the text from the Text property.
10. In the Solution Explorer, create a new folder in the project called Images and add an image to the folder to indicate an error.
11. Set the ErrorImage's ImageUrl property to the error image added to the images folder.
12. Beneath the ErrorImage, add a Button called LoginButton.

The end result should look like:



A screenshot of a login form titled "Login To QPulse". The form contains three input fields: "Username", "Password", and "Database". The "Database" field is a dropdown menu with "Unbound" selected. Below the "Database" field is a red error icon (a diamond with an exclamation mark) and the text "[ErrorLabel]". At the bottom of the form is a "Login" button.

## Implementing the Search panel

1. Click in the Search panel and type 'Search for Incidents'.
2. Add a label to the panel, call it KeywordLabel and set the Text to 'Keyword'.
3. Beneath the label, add a TextBox and call it KeywordTextBox.
4. Beneath the TextBox, add a button called SearchButton and set the Text property to 'Search'.

The end result should look like:



A screenshot of a search panel titled "Search for Incidents". The panel contains a label "Keyword" above a text input field. Below the input field is a "Search" button.

## Implementing the Results panel

1. Click in the Results panel and type 'Results'.
2. Beneath this, add a GridView component and name it IncidentGridView.
3. Beneath the GridView, add two buttons called BackButton and LogoutButton and set their Text properties to 'Back' and 'Logout' respectively.

The end result should look like:

Results		
Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
Back Logout		

## Implementing the Logic

The main logic should decide which panel to display to the user. In order to make the decision, we need to make use of session data. There are three pieces of data we would like to store as session data: the authentication token obtained from successful authentication, the search criteria used to query the Incidents Web Service and any error message obtained while trying to authenticate. The first two pieces of data will be used to determine which panel to display.

On loading of the page, the values of the authentication token and search criteria should be checked. If the authentication token is not set, the Login panel should be displayed. If the authentication token has been set and the search criteria data is empty, the Search panel should be displayed. If both the authentication token and search criteria have been set, the query should be performed against the QPulse Incidents Web service and the results should be displayed in the Results panel.

The method to override in `QPulseSharePointUserControl.ascx.cs` is called `Page_Load` and should look like:

```
private void Page_Load(object sender, EventArgs e)
{
    // reset the visibility of the panels
    LoginPanel.Visible = false;
    SearchPanel.Visible = false;
    ResultsPanel.Visible = false;

    // check to see if the user is authenticated
    if
(string.IsNullOrEmpty((String)GetSessionData(authenticationToken))
)
    {
        DisplayLogin();
    }
    else{
        // check to see if search criteria has been set
        if (GetSessionData(searchCriteria) == null)
        {
            // display search panel
            DisplaySearch();
        }
        else
        {
            // query Web Service and display results
            DisplayResults();
        }
    }
}
```

This method relies on four other methods `GetSessionData`, `DisplayLogin`, `DisplaySearch` and `DisplayResults` which we will now implement.

**GetSessionData** is a utility method designed to retrieve information from the Session and deal with any `NullReferenceExceptions` by catching them and returning null to the method caller:

```
private object GetSessionData(string name)
{
    object data = null;
    try { data = Session[name]; }
    catch (NullReferenceException) {}
    return data;
}
```

**DisplayLogin** displays the Login panel. It has to display any error associated with a previous login attempt. It must also query the `QPulseWebService` to obtain the list of databases available and populate the `DatabaseDropDownList` (we offload this functionality to a utility method called `PopulateDatabaseDropDownList`).

```
private void DisplayLogin()
{
    // reset error image and label visibility
    ErrorLabel.Visible = false;
    ErrorImage.Visible = false;

    // fetch list of databases
    PopulateDatabaseDropDownList();

    // attempt to get any error message from a previous
login attempt
    var errorMessage = (String)GetSessionData(loginError);

    // if there as an error message, display the error
image and message
    if (!string.IsNullOrEmpty(errorMessage))
    {
        ErrorLabel.Text = errorMessage;
        ErrorLabel.Visible = true;
        ErrorImage.Visible = true;
    }

    // reset the login error session data
    Session[loginError] = null;

    // display the panel
    LoginPanel.Visible = true;
}
```

```

private void PopulateDatabaseDropDownList()
{
    using (var client = new CoreService.Core())
    {
        // get the available databases from the Core
Service
        string[] databases =
client.AvailableConnections();

        // clear the list of any previous items
DatabaseDropDownList.Items.Clear();

        // populate the drop down list
foreach (string s in databases)
        {
            DatabaseDropDownList.Items.Add(s);
        }
    }
}

```

**DisplaySearch** displays the search panel and has minimal functionality:

```

private void DisplaySearch()
{
    // display the panel
    SearchPanel.Visible = true;
}

```

**DisplayResults** makes a request of the Incident Service to return results based on the search criteria. The results are then displayed to the user. We present lightweight objects to the GridView instead of the Incident objects received from the Web Service, in order to display only the data we are interested in. This approach is documented here: <http://support.microsoft.com/kb/316303>.

```

private void DisplayResults()
{
    Incident1[] incidents;

    // query the Web Service for Incidents using the
authentication token and query object
    using (var client = new IncidentService.Incident())
    {
        incidents =
client.GetIncidents((String)GetSessionData(authenticationToken),
(IncidentQuery)GetSessionData(searchCriteria), true, true);
    }

    // list of lightweight objects to bind GridView to
var gridList = new ArrayList();
    // dictionary of tooltips for titles that are too long
var toolTipDict = new Dictionary<int, string>();

    for (int i = 0; i < incidents.Length; i++ )

```

```

        {
            var incident = incidents[i];
            var title = incident.Title;

            // shorten title if required and add to dictionary
of tooltips
            if (title.Length > 30)
            {
                // add title to toolTipDict using row number
and full title
                toolTipDict.Add(i, title);
                // shorten the title to be displayed in the
GridView
                title = title.Substring(0, 27) + "...";
            }

            // add an anonymous lightweight object to the list
to be bound to the GridView
            gridList.Add(new { Number = incident.Number, Title
= title, ReportType = incident.ReportTypeName, Status =
incident.Status });
        }

        // bind the list to the GridView
        IncidentGridView.DataSource = gridList;
        IncidentGridView.DataBind();

        // set the tooltip on the row of each element with a
title too long
        foreach(var element in toolTipDict)
        {
            IncidentGridView.Rows[element.Key].ToolTip =
element.Value;
        }

        // display the panel
        ResultsPanel.Visible = true;
    }

```

## Button Click logic

There are four buttons: LoginButton, SearchButton, BackButton and LogoutButton. To implement the logic associated with a button click event, double-click the button in the design view and paste in the code listed below.

**LoginButton\_Click** should take the name, password and database from the ASP.net controls defined in the Login panel and send the data to the Core QPulseService in the hope of obtaining an authentication token. If authentication succeeds and a token is received back, it should be stored in the Session data for future use:

```
protected void LoginButton_Click(object sender, EventArgs e)
{
    using(var client = new CoreService.Core())
    {
        string token = null;

        try
        {
            // attempt authentication with QPulse via the
Web Service
            token =
client.Authenticate(UsernameTextBox.Text, PasswordTextBox.Text,
DatabaseDropDownList.SelectedValue);
        }
        catch(Exception ex)
        {
            // store the exception message for displaying
in the Login panel
            Session[loginError] = ex.Message;
        }
        // store the authentication token in the session
        Session[authenticationToken] = token;

        // reload the page
        Page_Load(sender, e);
    }
}
```

**SearchButton\_Click** should take the data entered into the SearchTextBox and create a new IncidentQuery object in the Session data. The query object will be sent to the Incident Web Service when the Results panel is to be displayed.

```
protected void SearchButton_Click(object sender, EventArgs
e)
{
    // create a new IncidentQuery object based on the
    TextBox data and store it for later use
    Session[searchCriteria] = new IncidentQuery { Keywords
= KeywordTextBox.Text };

    //reload the page
    Page_Load(sender, e);
}
```

**BackButton\_Click** simply sets the searchCriteria Session data to null, causing the Search panel to be displayed when the page is loaded:

```
protected void BackButton_Click(object sender, EventArgs e)
{
    // nullify searchCriteria to display search dialog
    Session[searchCriteria] = null;

    // reload the page
    Page_Load(sender, e);
}
```

**LogoutButton\_Click** sets the authenticationToken and searchCriteria Session data to null, causing the Login dialog to be displayed when the page is loaded. We also need to remove the text from the textboxes that was entered at the previous login.

```
protected void LogoutButton_Click(object sender, EventArgs
e)
{
    // nullify authenticationToken and searchCriteria to
display login dialog
    Session[authenticationToken] = null;
    Session[searchCriteria] = null;

    // clear the username and password textboxes
    UsernameTextBox.Text = String.Empty;
    PasswordTextBox.Text = String.Empty;

    // reload the page
    Page_Load(sender, e);
}
```

## Deployment to SharePoint

Firstly, you must be aware of the application root folder you would like to install and run the Web Part under (by default it will probably be C:\inetpub\wwwroot\wss\VirtualDirectories\80)

1. In Windows Explorer, navigate to the application root folder. In the Web.config find the trust section and change it to read: `<trust level="WSS_Medium" originUrl="" />`
2. In the Web.config, under the SafeControl section add the following entry:  

```
<SafeControl Assembly="QPulseSharePointServerControl,
Version=1.0.0.0, Culture=neutral,
PublicKeyToken=xxxxxxxxxxxxxxxxxxxx"
Namespace="QPulseSharePointServerControl"
TypeName="QPulseWebPart" Safe="True"/>
```
3. Remember to put in the value of the Public Key token (this can be found by running 'sn.exe -T locationToSolution\bin\QPulseSharePointUserControl.dll' which is available from the Visual Studio 2008 Command prompt.
4. Similarly, add the following entry as well `<SafeControl Assembly="QPulseSharePointUserControl, Version=1.0.0.0, Culture=neutral, PublicKeyToken= xxxxxxxxxxxxxxxxxxxx " Namespace="QPulseSharePointUserControl" TypeName="*" Safe="True"/>`
5. Also in Web.config, locate the 'pages' xml element and change the enableSessionState from false to true: `enableSessionState="true"`
6. Build both solutions and copy the assemblies (QPulseSharePointServerControl.dll and QPulseSharePointUserControl.dll) from the project output folders to the bin folder in your application root folder.
7. In the application root folder, create a new folder called 'controls' and copy QPulseSharePointUserControl.ascx from the User control project into the new folder.
8. Create a folder in the application root folder called 'images' and copy the error icon image from the User control project to there.
9. In the application root folder, create a new folder called 'images' and copy the error image you added previously, into the folder.
10. In Internet Explorer, navigate to the SharePoint home page (e.g. <http://localhost/Pages/Default.aspx>). Click on the Site Actions button and go to Site Settings -> Modify All Site Settings. Under the Galleries section, click on Web Parts. In the Web Part Gallery page click on the New button, near the top. In the New Web Parts page, scroll down to locate QPulseSharePointServerControl.QPulseWebPart, tick the associated box and click the Populate Gallery at the top of the page.

## Adding the Web Part to a Web page

1. Navigate to a page in SharePoint (e.g. the home page) and click the Edit Page button. In the zone you would like to display the Web Part, click on the 'Add a Web Part' button. On the page that pops up, expand 'All Web Parts'. Under the Miscellaneous section tick the box next to QPulseWebPart and click the Add button

## Limitations/Improvements

As this is an example, in a real world deployment scenario there would be other considerations to address. Here are a few suggestions:

- The Web Service calls would be made across an encrypted connection (SSL).
- The user session might time out after a period of inactivity.
- Any exception or error occurring while communicating with the Web Service would be handled (e.g. fetching the database list, authenticating and performing the query) and the user notified accordingly.
- The user input for the text boxes could be validated.



**Gael Ltd.**

Orion House,  
S. E. Technology Park,  
East Kilbride,  
Scotland. G75 0RD

**t:** +44 1355 593400

**f:** +44 1355 579191

**e:** [info@gaelquality.com](mailto:info@gaelquality.com)

**w:** [www.gaelquality.com](http://www.gaelquality.com)

Q-Pulse is a registered trademark of Gael Products Ltd. All rights reserved worldwide.  
Copyright © 2011 Gael Products Ltd. Gael Quality is a trading division of Gael Ltd.