Macromedia Flash Article

Embedding and Communicating with the Macromedia Flash Player in C# Windows Applications

Stock History is written in C# and it embeds the Macromedia Flash Player ActiveX control, which loads a Macromedia Flash movie (Flash\chart.swf) that handles all of the charting requirements.

Embedding the Macromedia Flash Player ActiveX control is a relatively simple way to do the following:

- Add functionality, graphics, or animation to your Windows application.
- Extend the capabilities of your stand-alone Macromedia Flash application (such as adding file I/O capabilities to Macromedia Flash).

Before you decide to embed the Macromedia Flash Player ActiveX control within your Windows application, however, there are a number of things that you should consider:

- Macromedia does not officially support embedding the Macromedia Flash Player ActiveX control within a Windows application. This means that future versions of the player may not work with your application.
- It requires users to install the Macromedia Flash Player ActiveX control on their computers. This can be done by installing the player within Microsoft Internet Explorer. Although it is technically possible to distribute the player, that is restricted by the player's end-user license agreement. If you need to distribute the Macromedia Flash Player ActiveX control, contact Macromedia for permission.
- Because it uses ActiveX technology, it works only on Windows OS.
- Versions of the Macromedia Flash Player prior to version 6 did not support embedding within Windows applications. Users need version 6.0r79 or higher.

Making the Macromedia Flash Player ActiveX Control Available Within Visual Studio .NET

The easiest way to use the Macromedia Flash Player ActiveX control within your C# Windows applications is to add it to the Toolbox in Visual Studio. This allows you to use the player in the same manner as any other Windows Forms components. It also makes it easy to insert the player into your application.

Here's how you add the Macromedia Flash Player ActiveX control to Visual Studio .NET:

1. Open Visual Studio .NET.
2. Make sure the Toolbox is open (select View > Toolbox).
3. Open the Windows Forms section of the Toolbox.
4. Right-click the Windows Form section and select Add/Remove Items from the pop-up menu. This opens the Customize Toolbox window.
5. Select the COM Components tab, which lists all of the available COM components on your computer, including the Macromedia Flash Player ActiveX control.
6. Scroll down to Shockwave Flash Object and select it (see Figure 1). If this item is not listed, make sure that the Macromedia Flash Player ActiveX control is installed on your system.

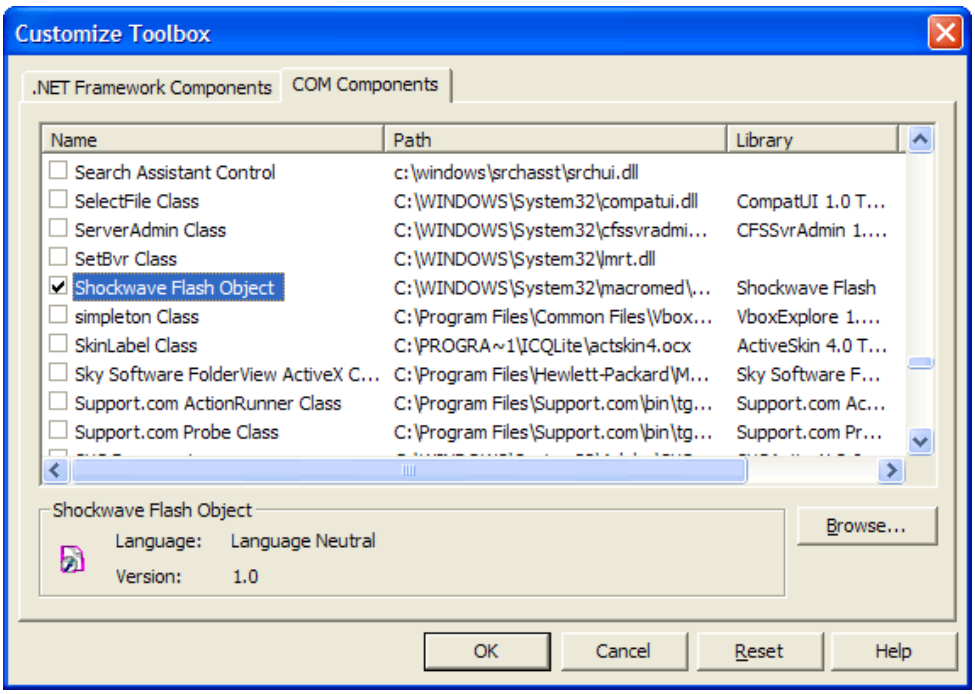

Figure 1. Customize Toolbox window with Shockwave Flash Object selected

7. After you select the item, click OK. This will add "Shockwave Flash Object" to the Windows Forms section of the Toolbar (see Figure 2).

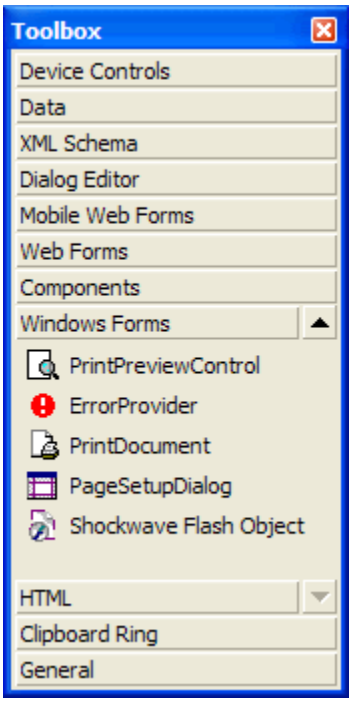

Figure 2. Macromedia Flash Player ActiveX control added to the Toolbox

You can now embed the Macromedia Flash Player ActiveX control into your Windows application by dragging the control from the Toolbox to your Windows Form in Design mode.

Creating the StockHistory Windows Application

Using the Macromedia Flash Player ActiveX control within the Stock History application requires a couple of steps:

1. Drag the Macromedia Flash Player ActiveX control into the Chart section of the tab control in your application.
2. Set some default properties for the Macromedia Flash Player ActiveX control in the Properties window.
3. Write some C# code to control the player at runtime.

This article assumes that you have the StockHistory solution open within Visual Studio .NET. Open StockHistory.cs in Design and Code mode.

Adding the Macromedia Flash Player ActiveX Control to the Application in Design Mode

Embedding the Macromedia Flash Player ActiveX control into the application is simple. It involves dragging the Shockwave Flash Object control from the Toolbox into the application. For this application, we place the control on the Chart tab of the Tab control.

It may be difficult to view the actual control because its background color is set to match that of the application. However, if you select the stage of the Chart section of the Tab control, you can view the outline of the Macromedia Flash Player ActiveX control (see Figure 3). Use the outlines and anchors of the control to resize it in Design mode.

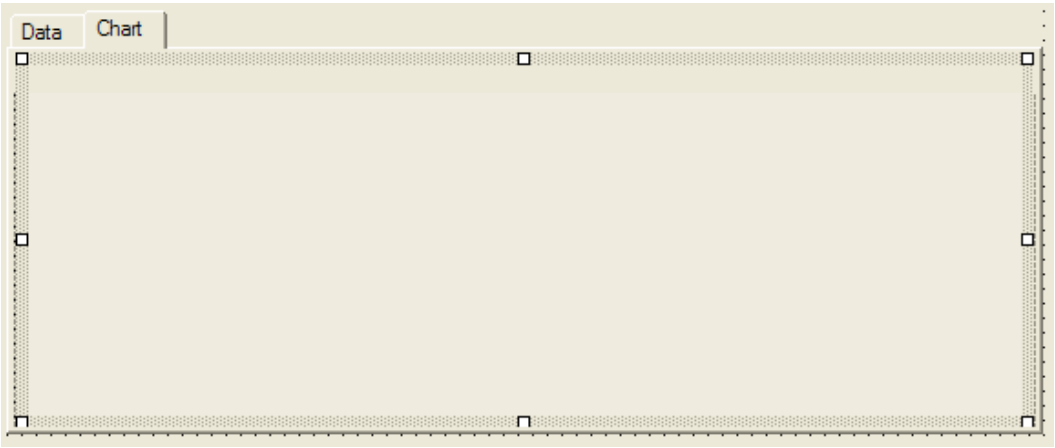

Figure 3. Macromedia Flash Player ActiveX control within the application in Design mode

Setting the Default Properties of the Flash Player Control

Once you've added the Macromedia Flash Player ActiveX control to your application, you can set a number of its properties within the Properties window (View > Properties; see Figure 4).

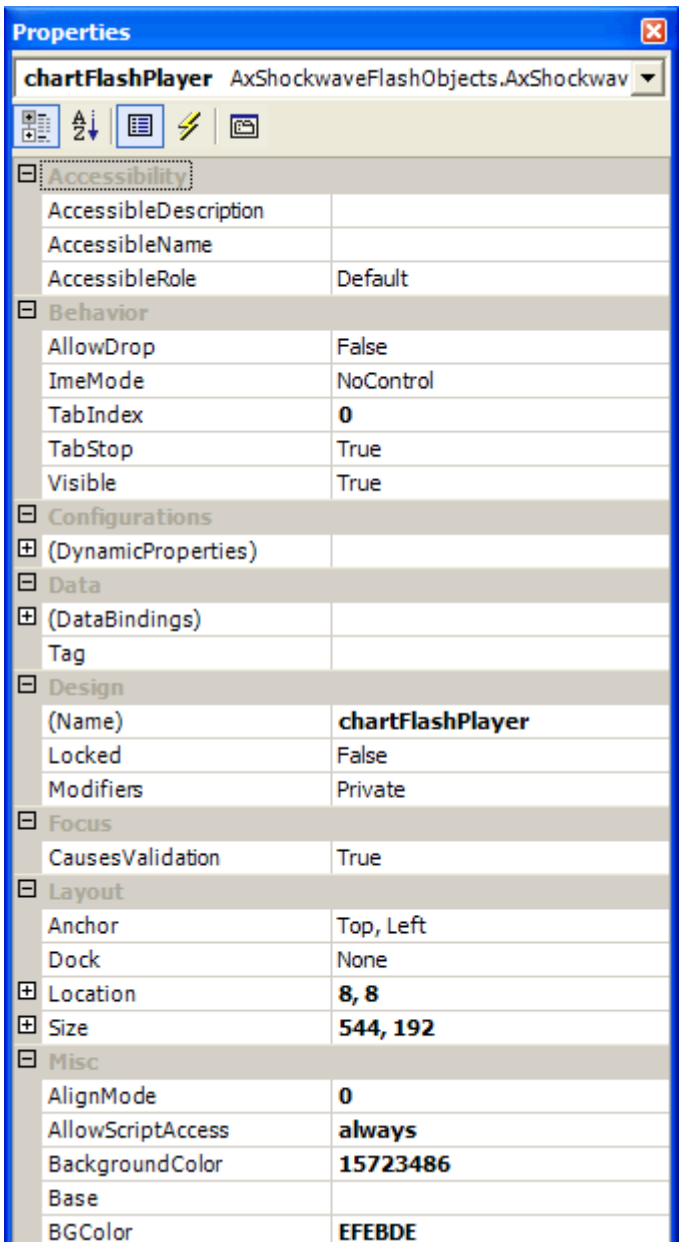

Figure 4. chartFlashPlayer properties in the Properties window

As you can see from Figure 4, quite a number of properties are available. For the Stock History application, you only need to set a couple of properties:

- **Name:** This is the instance name of the control within the C# code. We changed this to **chartFlashPlayer**.
- **Background Color:** We changed this to match the background of the Windows application. If you specify the color as a hexadecimal value, Visual Studio converts it to the necessary format.

You can also set these properties from C#. In fact, if you look at the code generated by Visual Studio, you will see how it sets some of these properties:

```
//
// chartFlashPlayer
//
```

```
this.chartFlashPlayer.ContainingControl = this;
this.chartFlashPlayer.Enabled = true;
this.chartFlashPlayer.Location = new System.Drawing.Point(8, 8);
this.chartFlashPlayer.Name = "chartFlashPlayer";
this.chartFlashPlayer.OcxState =
   ((System.Windows.Forms.AxHost.State)
   (resources.GetObject("chartFlashPlayer.OcxState")));
this.chartFlashPlayer.Size = new System.Drawing.Size(544, 192);
this.chartFlashPlayer.TabIndex = 0;
```

Depending on your project, you may need to adjust additional properties.

Controlling the Macromedia Flash Player ActiveX Control with C#

Now that you have inserted and configured the Macromedia Flash Player ActiveX control within your Windows application, all that remains is to write the C# code needed to manipulate and control the player at runtime.

For the Stock History sample application, there are a number of things you need to do:

1. Load chart.swf into the player. This SWF contains the Macromedia Flash code and components necessary to create charts.
2. Send the stock data to the Macromedia Flash movie when the data has loaded.
3. Reset the movie to clear its charts when the Reset button in the application is clicked.

Loading Chart.swf into the Macromedia Flash Player ActiveX Control

The first thing you need to do when the application runs is load chart.swf. Place the following two lines of code in the constructor of the StockHistory class (which is called when the application is loaded):

```
String swfPath = Directory.GetCurrentDirectory() +
   Path.DirectorySeparatorChar + "chart.swf";

chartFlashPlayer.LoadMovie(0, swfPath);
```

The first line creates the file path to chart.swf. It finds the path from where the application is being run and concatenates the name of the Macromedia Flash movie to it. Of course, this assumes that chart.swf is in the same directory as the application's executable file.

The second line tells the Macromedia Flash Player ActiveX control to load the SWF. It calls the LoadMovie function on the Macromedia Flash Player ActiveX control. This function takes two parameters. The first specifies the level that the movie should be loaded into the player. Since you are loading only one movie, just use level 0. The second parameter specifies the path to the Macromedia Flash movie that should be loaded. In this case, pass the absolute path to the movie that you created earlier.

Once this is run, the movie is loaded into the application and begins to play. In the Stock History application, the default state of the application does not do much; it only displays the outlines of the chart (see Figure 5).

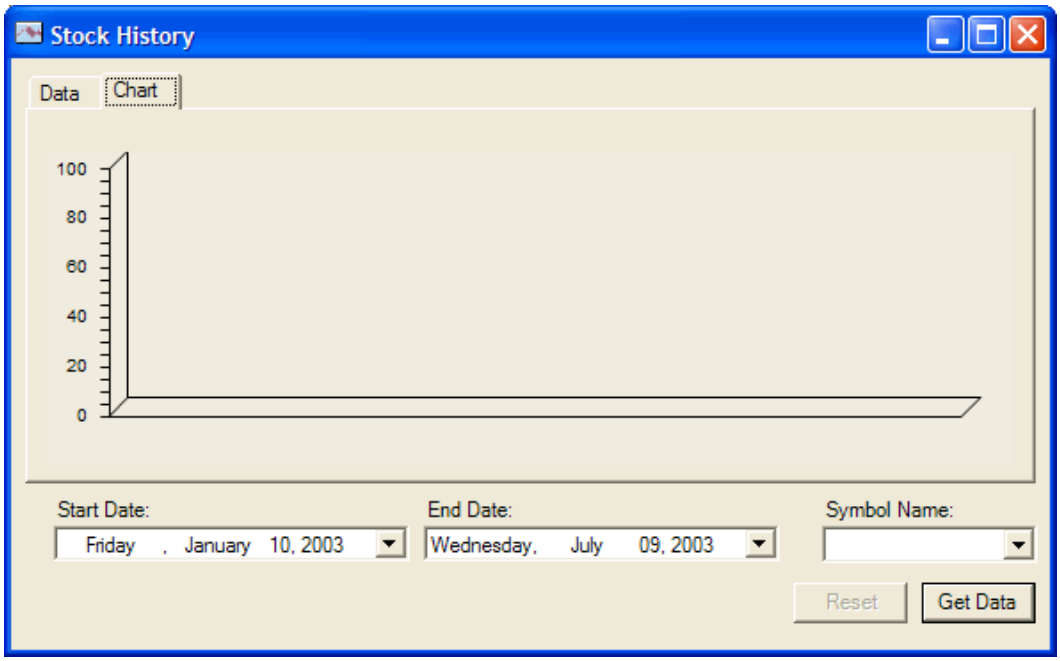

Figure 5. The default state of the Macromedia Flash movie when it initially loads into the application

Sending Stock Data to the Application

Once the Macromedia Flash movie is loaded and initialized, it does not do anything again until the stock data loads. Once that happens, it's serialized into a string and then passed to the movie. The movie deserializes the string and uses the data to create a chart.

Before examining why the data has to be serialized and deserialized, let's look at the code necessary to pass the data to Macromedia Flash:

```
string flashData = "stockData=" +
   URLEncode(stock.SerializeToString()) + "&stockSymbol=" +
   URLEncode(stockSymbol);

chartFlashPlayer.SetVariable("chartData",flashData);
```

The first line creates the serialized string of data that will be passed to Macromedia Flash. In this case, we are passing two name/value pairs of data as a URL-encoded query string. Here is the data we are passing:

- **stockData:** This contains a serialized DataSet/DataProvider containing the stock information.
- **stockSymbol:** This is the stock symbol with which the data is associated.

You will see how data is deserialized later in more detail (see "Deserializing the Data").

The next line actually passes the data to the movie running within the Macromedia Flash Player ActiveX control. This is done with the SetVariable method, which takes two parameters: the name of the ActionScript variable that is being set/modified within the movie, and the value/data to which the ActionScript variable is being set.

Let's take a step back for a second. There is no way to call a function within a Macromedia Flash movie from C#. However, you can set and modify ActionScript variables within Macromedia Flash. Because of this, you have to call methods in Macromedia Flash indirectly. This is done by placing a watch on a variable within ActionScript. When that variable is modified, a function that you specify will be called and passed the new data. Thus from C#, you pass the data to the variable in ActionScript which is being watched, and then ActionScript passes that to your function.

Let's look at the Macromedia Flash code that's necessary to set up the watch variable and the function that handles the data from C#:

```
_root.chartData = "";

function onChartData(prop, oldVal, newVal, userData)
{
   var lv = new LoadVars();
   lv.decode(newVal);

   var dataArray = decodeData(lv.stockData);

   renderChart(lv.stockSymbol, dataArray);
}

_root.watch("chartData", onChartData);
```

First, the code initializes the chartData variable in ActionScript. This allows you to place a watch on the variable to watch for changes to its value.

The code then defines the onChartData function, which gets called when the value of the chartData variable is modified. This variable is modified from C#; it's how the Windows application passes the data into the Macromedia Flash movie.

Finally, the code adds the watch for the chartData variable. It basically specifies that when the chartData variable changes, the onChartData method should be called and passed information about the change.

Now it should make a little more sense why you send a URL-encoded query string into Macromedia Flash from C#. Remember that you can only pass a single string into Macromedia Flash at a time. However, you need to pass in two parameters, which you do by passing in a URL-encoded query string.

In Macromedia Flash, you now have a URL-encoded query string that you have to deserialize into the individual name/value pairs. Luckily, the LoadVars object contains a method that deserializes query strings and places the name/value pairs within itself.

This happens within the onChartData method (which is called when the data is passed from C#):

```
var lv = new LoadVars();
lv.decode(newVal);
```

The newVal variable contains the URL-encoded string sent from C#.

Once you call lv.decode, you can then access the name/value pairs as properties of the LoadVars instance, like so:

```
trace(lv.stockData);
trace(lv.stockSymbol);
```

You may be thinking that if you only need to pass in one name/value pair, you can avoid these steps and pass the data in directly. In general, however, it is a good idea to always pass data in like this because it gives you the flexibility of adding more name/value pairs without making major modifications to your code.

The lv.stockData property contains a serialized set of data (similar to a DataProviderClass) that has been sent from C#. Before you can use it, you must deserialize it into a format that the charting components can use (in this case an array of data values). However, before you do that, let's look at how the code is serialized within C#.

Serializing the Data in C#

In C#, the data associated with the stock symbol is contained within a DataTable object. This is convenient because it can be passed directly to the DataGrid component within your Windows application. In order to pass it to Macromedia Flash, however, you must serialize it into a string. Do this by delineating columns with commas and delineating rows with line returns. If your data contains these characters, you need to select different characters as delineators.

The serialization occurs within the SerializeToString method of the C# Stock class (Stock.cs):

```
public string SerializeToString()
{
   DataColumnCollection columns = dt.Columns;
   int colLength = columns.Count;

   StringBuilder sb = new StringBuilder();

   for(int i=0; i < colLength; i++)
   {
      sb.Append(columns[i].ColumnName);
      sb.Append(",");
   }

   sb.Remove(sb.Length - 1, 1);
   sb.Append("\n");

   DataRowCollection rows = dt.Rows;
   int rowLen = rows.Count;

   for(int i=0; i < rowLen; i++)
   {
      for(int k = 0; k < colLength; k++)
      {
         sb.Append(rows[i][k]);
         sb.Append(",");
      }
      sb.Remove(sb.Length - 1, 1);
```

```
        sb.Append("\n");
    }

    sb.Remove(sb.Length - 1, 1);


    return sb.ToString();
}
```

SerializeToString loops through the DataTable object and serializes it into a string. Below is an example of what the serialized string will look like:

```
Date,Open,High,Low,Close,Volume
10-Jan-03,12.85,13.24,12.65,13.15,839500
13-Jan-03,13.35,13.55,13.11,13.50,818300
14-Jan-03,13.40,13.55,12.87,13.34,801700
```

This string is then URL-encoded and passed to Macromedia Flash (as discussed earlier in "Sending Stock Data to the Application").

Deserializing the Data

Once the data passes to Macromedia Flash, it is passed to the decodeData ActionScript method (shown below), which deserializes it into an array of numbers which will then be passed to the charting components. Notice that decodeData starts looping through the data on the second row (index 1), not the first (index 0). This is because the first row contains the column names which are not needed by the Line Chart component. Also notice in SerializeToString that we only need one column of the data (columns[4]). We send in the entire data set, though, just in case any of the other data is needed in future revisions.

Here is the decodeData ActionScript method:

```
function decodeData(data)
{
    var dataArray = new Array();
    var rows = data.split("\n");
    var rowLen = rows.Length;
    var columns = new Array();

    for(var i = 1; i < rowLen; i++)
    {
        columns = rows[i].split(",");
        dataArray.push(parseInt(columns[4]));
    }

    return dataArray
}
```

This data, along with the stock symbol, is passed to the renderChart method, which adds the data to the Line Chart component:

```
function renderChart(stockSymbol, dataArray)
{
   myLineChart.addItem(stockSymbol, dataArray);
}
```

At this point, the component renders the line chart and displays it within the Windows application.

The Reset button in the application works in a similar way. However, instead of sending data in Macromedia Flash, it just sends an empty string, which prompts the onReset method to be called within Macromedia Flash.

‹ Back | Contents