

How to write plugins for BP2X?

This how to tutorial will explain what you need to do to write plugins for Broadcast Power.

Prerequisites:

1. Knowledge of C# or .NET language
2. Understand .NET remoting
3. .NET compiler
4. Version 2.5.4.1 of Broadcast Power

In its simplest form, a plugin is an extension to Broadcast Power that enhances it with features.

A plugin is a class in .NET that inherits from an interface. In BP2X the interface is called **IServerPlugin** and can be found in the **PluginInterfaces.dll**.

Below is the source code for the server plugin interface.

```
public interface IServerPlugin
{
    /// <summary>
    /// place initialization code in this function
    /// this function will be called after the assembly is loaded
    /// </summary>
    void Start();
    /// <summary>
    /// Cleanup code for your plugin goes here
    /// </summary>
    void Dispose();
    /// <summary>
    /// sets the plugin directory so plugins can load their config files when start is called
    /// this function will be called by the plugin host before Start() is called
    /// </summary>
    /// <param name="pluginDir"></param>
    void SetPluginDirectory(string pluginDir);
    /// <summary>
    /// Displays a dialog box to gather config data specific to plugin.
    /// </summary>
    void ConfigurePlugin();
    /// <summary>
    /// Function to allow host to decide whether a particular function is implemented.
    /// returns true if plugin has configurable aspects and implements ConfigurePlugin.
    /// </summary>
    /// <returns>bool</returns>
    bool IsConfigurable();
    /// <summary>
    /// Returns description of the plugin.
    /// </summary>
    /// <returns>string</returns>
    string GetDescription();
    /// <summary>
    /// The name of this plugin.
    /// </summary>
    /// <returns>name</returns>
    string GetName();
}
```

You have to implement 6 or 7 function interfaces and you should have a functional plugin.

To help you start coding, I am also attaching below the source code for the plugin that ships with version 2.5.4.1 of Broadcast Power.

To compile this plugin you need to have the necessary dll's.

Make sure you reference:

1. BusinessObjects.dll (comes with Broadcast Power)
2. EnterpriseObjects.dll (comes with Broadcast Power)
3. Log4net.dll (can be downloaded from the Internet)
4. Melih.FTPClient.dll (can be downloaded from the Internet)
5. PluginInterfaces.dll (comes with Broadcast Power)
6. RemotePlayer.dll (comes with Broadcast Power)
7. RemotingLibrary.dll (comes with Broadcast Power)
8. System.Data (comes with .NET)
9. System.Xml (comes with .NET)
10. System.Windows.Forms (comes with .NET)
11. System.configuration (comes with .NET)

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Configuration;
using System.Threading;

using RemotePlayer;
using RemotingLibrary;
using BroadcastPower.Enterprise;

using Melih.FTPClient;
using log4net;

namespace BP2X.ServerPlugins
{
    public class NowPlayingWebExportPlugin : MarshalByRefObject,
    BP2X.PluginInterfaces.IServerPlugin, IPlayerEventClient
    {
        // Remote Player variables
        FirstPlayer _FirstPlayer;
        SecondPlayer _SecondPlayer;

        private string _output;
        private string _input;
        private string _outputfilename;
        private string _templatefilename;
        private string _ftpserver;
        private string _ftpusername;
        private string _ftppassword;
        private string _ftpdirectory;

        private string _pluginsDirectory;
        private ILog _logger = null;
    }
}
```

```

private NowPlayingWebExportPluginSettings _settings;

private System.Threading.Timer _timer;

private RemotePlayer.Player.PlayerEventType _EventType;
private RemotePlayer.Player.PlayerData _Data;

public NowPlayingWebExportPlugin()
{
}

public string GetName()
{
    return "Now Playing Web Export Plugin";
}

public string GetDescription()
{
    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Now Playing Web Export Plugin.");
    sb.AppendLine("-----");
    sb.AppendLine("This plugin is used to publish the currently playing media file's
details to a website. ");
    sb.AppendLine("It works by parsing a template file and replacing predefined tags with
the values of the currently playing media file from the database.");
    sb.AppendLine("Then, it takes the resulting file and FTP's it to a predefined FTP
server.");
    sb.AppendLine("You will have to define certain parameters for the plugin to function.
This plugin will log any errors in the standard plugins log file of BP2X.");
    return sb.ToString();
}

public void SetPluginDirectory(string dir)
{
    _pluginsDirectory = dir;
    // setup log4net
    log4net.Config.DOMConfigurator.Configure(new FileInfo(dir +
Path.DirectorySeparatorChar + "NowPlayingWebExportPlugin.log4net"));
    _logger = log4net.LogManager.GetLogger("Plugins.NowPlayingWebExportPlugin");
}

public void Start()
{
    _logger.Debug("Start called for NowPlayingWebExportPlugin");
    // let's read the configuration file
    ReadConfigurationFile();

    // maybe this is the first time the user runs the plugin
    // let's ask them to configure it
    if (_input == null || _input.Length < 1)
    {
        ConfigurePlugin();
    }

    try
    {
        // initialize the remote players
        RemotingLibrary.ClientTools client = new RemotingLibrary.ClientTools();
        client.RegisterRemotePlayers(); // to be able to get events from the players
        client.RegisterBusinessObjects(); // to be able to call the Database

        // create remote players
        _FirstPlayer = FirstPlayer.Player;
        _SecondPlayer = SecondPlayer.Player;

        // setup the resource manager instance
        //_resourceManager = new ResourceManager("AudioPlayers.AudioPlayers",
typeof(RPlayer).Assembly);

```

```

        // _resourceManager.IgnoreCase = true;

        // initialize the callbacks / events from the server
        AttachEvents();
    }
    catch (Exception ex)
    {
        System.Collections.Specialized.NameValueCollection info = new
System.Collections.Specialized.NameValueCollection();
        info.Add("Working Set", Environment.WorkingSet.ToString());
        _logger.Error(ex.Message, ex);
        MessageBox.Show(null, ex.Message, "Exception", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }

    // Create the delegate that invokes methods for the timer.
    TimerCallback timerDelegate =
        new TimerCallback(_timer_ontick);

    _timer = new System.Threading.Timer(timerDelegate, null, 500, 30000);
}

public void Dispose()
{
    _timer.Dispose();
    DetachEvents();
}

private void _timer_ontick(object state)
{
    // check remoteplayers and register event if disposed
    if (!_FirstPlayer.IsClientAttached((IPlayerEventClient)this))
        _FirstPlayer.AttachClient((IPlayerEventClient)this);
    if (!_SecondPlayer.IsClientAttached((IPlayerEventClient)this))
        _SecondPlayer.AttachClient((IPlayerEventClient)this);
}

private void ReadConfigurationFile()
{
    _logger.Debug("Reading Configuration File for NowPlayingWebExportPlugin");
    ExeConfigurationFileMap fileMap = new ExeConfigurationFileMap();
    fileMap.ExeConfigFilename = _pluginsDirectory + Path.DirectorySeparatorChar +
@"NowPlayingWebExportPlugin.config"; // relative path names possible

    // Open config file
    Configuration config =
        ConfigurationManager.OpenMappedExeConfiguration(fileMap,
ConfigurationUserLevel.None);

    //read/write from it as usual
    //ConfigurationSection mySection =
config.GetSection("NowPlayingWebExportPluginSettings");
    _settings = config.GetSection("NowPlayingWebExportPluginSettings")
        as NowPlayingWebExportPluginSettings;

    if (_settings != null)
    {
        // now let's try to open the template file
        if (File.Exists(_settings.TemplateFilename))
        {
            StreamReader sr = new StreamReader(_settings.TemplateFilename);
            _input = sr.ReadToEnd();
            sr.Close();

            _outputfilename = _settings.OutputFilename;
            _templatefilename = _settings.TemplateFilename;
            _ftpdirectory = _settings.FTPDirectory;
            _ftppassword = _settings.FTPPassword;
            _ftpserver = _settings.FTPServer;
            _ftpusername = _settings.FTPUsername;
        }
    }
}

```

```

    }
    else
    {
        MessageBox.Show(null, "The template file '" + _settings.TemplateFilename + "'
does not exist. This plugin will not function correctly or will fail to run.", "Now Playing Web
Export Plugin - File not found!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
else
{
    MessageBox.Show(null, "The configuration file '" + fileMap.ExeConfigFilename + "'
does not exist. This plugin will not function without it. Check the plugin documentation and re-
create this configuration file.", "Now Playing Web Export Plugin - Config file not found!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

public void ConfigurePlugin()
{
    _logger.Debug("Configuring plugin for NowPlayingWebExportPlugin");
    // open the configuration window
    NowPlayingWebExportPluginConfigDialog dlg = new
NowPlayingWebExportPluginConfigDialog();

    // populate existing values
    ReadConfigurationFile();
    if (_settings != null)
    {
        dlg.FTPDirectory = _settings.FTPDirectory;
        dlg.FTPPassword = _settings.FTPPassword;
        dlg.FTPServer = _settings.FTPServer;
        dlg.FTPUsername = _settings.FTPUsername;
        dlg.TemplateFilename = _settings.TemplateFilename;
        dlg.OutputFilename = _settings.OutputFilename;
    }

    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // get data elements from dialog box and save to config file
        _settings = new NowPlayingWebExportPluginSettings();
        _settings.FTPDirectory = dlg.FTPDirectory;
        _settings.FTPPassword = dlg.FTPPassword;
        _settings.FTPServer = dlg.FTPServer;
        _settings.FTPUsername = dlg.FTPUsername;
        _settings.TemplateFilename = dlg.TemplateFilename;
        _settings.OutputFilename = dlg.OutputFilename;

        // save the settings to the config file
        ExeConfigurationFileMap fileMap = new ExeConfigurationFileMap();
        fileMap.ExeConfigFilename = _pluginsDirectory + Path.DirectorySeparatorChar +
@"NowPlayingWebExportPlugin.config"; // relative path names possible

        // Open config file
        Configuration config =
            ConfigurationManager.OpenMappedExeConfiguration(fileMap,
                ConfigurationUserLevel.None);

        //write from it as usual
        // You need to remove the old settings object before you can replace it
        config.Sections.Remove("NowPlayingWebExportPluginSettings");
        // with an updated one
        config.Sections.Add("NowPlayingWebExportPluginSettings", _settings);
        // Write the new configuration data to the XML file
        config.Save();

        // let's try to re-load
        ReadConfigurationFile();
    }
    else
    {
        ReadConfigurationFile();
    }
}

```

```

        if (_settings == null)
            MessageBox.Show(null, "You did not configure the plugin. This plugin might
fail to run until properly configured.", "Now Playing Web Export Plugin - Configuration Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

public bool IsConfigurable()
{
    return true;
}

public void AttachEvents()
{
    _timer_ontick(null);
}

public void DetachEvents()
{
    _FirstPlayer.DetachClient((IPlayerEventClient)this);
    _SecondPlayer.DetachClient((IPlayerEventClient)this);
}

#region IPlayerEventClient Members

// TODO: Add code to reconnect to remote players if connection fails and remoteplayers
drop the link. This can be done by polling every minute, two or so.

public void ReceivePlayerMessage()
{
    _logger.Debug("Processing ReceivePlayerMessage for NowPlayingWebExportPlugin");
    try
    {
        // take template and replace all tags with valid values from player _Data
        if (_EventType == Player.PlayerEventType.Play)
        {
            System.Diagnostics.Debug.WriteLine("Received message for file: " +
_Data.MediaId);

            Media media = null;

            if (_Data.MediaId < 1)
            {
                // let's try to retrieve media id from filename
                if (_Data.Filename != null && _Data.Filename.Length > 0)
                    media = Media.GetByFilename(_Data.Filename);
            }
            else
            {
                media = Media.GetById(_Data.MediaId);
            }

            if (media != null)
            {
                // save result file to directory where required, if web location, then
ftp
                _output = _input.Replace("{%SongName%}", media.Name);

                switch (media.MediaTypeId)
                {
                    case Media.MEDIA_TYPE_SONG:
                        Song song = Song.GetByMedia(media.Id);
                        if (song != null)
                        {
                            _output = _output.Replace("{%Album%}", song.AlbumName);

                            Performer performer = Performer.GetById(song.PerformerId);
                            if (performer != null)

```

```

        _output = _output.Replace("{%Performer%}",
performer.Name);
    }
    break;
case Media.MEDIA_TYPE_COMMERCIAL:
    break;
case Media.MEDIA_TYPE_NEWSCLIP:
    break;
} // switch

// save output file in a temporary location
string localfile = _templatefilename.Substring(0,
_templatefilename.LastIndexOf("\\") + 1) + _outputfilename;
StreamWriter writer = new StreamWriter(localfile, false, Encoding.UTF32);
writer.Write(_output);
writer.Close();

try
{
    // start ftp client and send output file
    FTPClient ftp = new FTPClient();
    ftp.FTPOpen(_ftpserver, 21);
    ftp.FTPConnect(_ftpusername, _ftppassword);
    ftp.FTPSetCurrentDirectory(_ftpdirectory);
    ftp.FTPPutFile(localfile, _outputfilename, FileTransferType.Binary);
    ftp.FTPClose();
}
catch (Exception ex)
{
    _logger.Error(ex.Message, ex);
}

} // media not null

} // event is play
}
catch (Exception ex)
{
    _logger.Error(ex.Message, ex);
}
}

public object ReceivePlayerMessage(RemotePlayer.Player.PlayerEventType EventType,
RemotePlayer.Player.PlayerData Data)
{
    _logger.Debug("About to create a new thread in ReceivePlayerMessage for
NowPlayingWebExportPlugin");

    _EventType = EventType;
    _Data = Data;

    // we will start this process in a separate thread
    // so that the server doesn't time out and remove the client from its
    // list of event receivers
    System.Threading.Thread t = new System.Threading.Thread(new
ThreadStart(ReceivePlayerMessage));
    t.Start();

    return null;
}

#endregion

}

public class NowPlayingWebExportPluginSettings : ConfigurationSection
{
    [ConfigurationProperty("TemplateFilename")]
    public string TemplateFilename
    {

```

```

        get { return (string)this["TemplateFilename"]; }
        set { this["TemplateFilename"] = value; }
    }

    [ConfigurationProperty("OutputFilename")]
    public string OutputFilename
    {
        get { return (string)this["OutputFilename"]; }
        set { this["OutputFilename"] = value; }
    }

    [ConfigurationProperty("SaveAs")]
    public string SaveAs
    {
        get { return (string)this["SaveAs"]; }
        set { this["SaveAs"] = value; }
    }

    [ConfigurationProperty("FTPServer")]
    public string FTPServer
    {
        get { return (string)this["FTPServer"]; }
        set { this["FTPServer"] = value; }
    }

    [ConfigurationProperty("FTPUsername")]
    public string FTPUsername
    {
        get { return (string)this["FTPUsername"]; }
        set { this["FTPUsername"] = value; }
    }

    [ConfigurationProperty("FTPPassword")]
    public string FTPPassword
    {
        get { return (string)this["FTPPassword"]; }
        set { this["FTPPassword"] = value; }
    }

    [ConfigurationProperty("FTPDirectory")]
    public string FTPDirectory
    {
        get { return (string)this["FTPDirectory"]; }
        set { this["FTPDirectory"] = value; }
    }
}
}

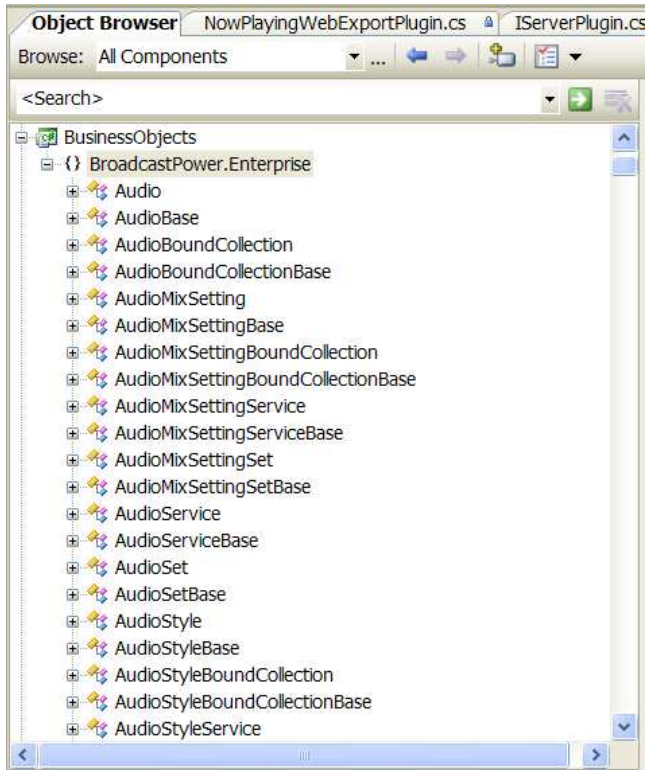
```

Depending on the complexity of the task that the plugin has to achieve, it can be small or large.

The plugin that ships with BP2X version 2.5.4.1 is moderate in complexity.

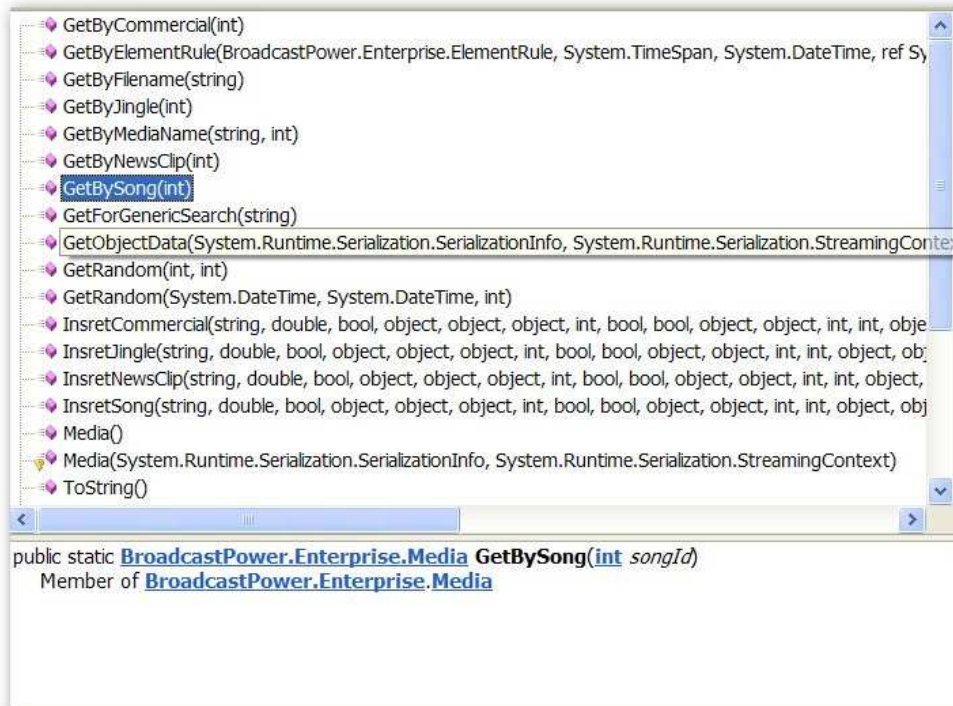
BP2X Libraries

You can browse the objects in each of the referenced libraries to learn more about what they offer. Simply double click on the library under References in your project.



Highlighting any specific function in the library will expose the additional properties of this function so you know how to call it.

Here's an example of the Media object under BroadcastPower.Enterprise:



I do not currently provide documentation for the libraries that ship with Broadcast Power. If you are interested in documenting these libraries, please let me know and I will try to assist you understand the functions that you are not able to decipher.

The libraries can be used to give you access to the database tables or records stored by BP2X, in addition to many other functions. In the sample code above, the BP2X libraries are used to retrieve details of the audio media being played by the players to publish it on a web page.

How to use the plugin when it is written?

Once a plugin is written and compiled into a dll, all you need to do to run it is place it under the plugins directory inside the directory for Broadcast Power.

The default location of the plugins directory is: **C:\Program Files\Media Bonza\Broadcast Power\2.5\plugins**

Restart **RemotePlayerHost.exe** when you're done and you should be able to see the outcome of your plugin.

Please note that there is provision in **RemotePlayerHost.exe** to pick up any dll that has been modified or added to the plugins directory in realtime and reloads it, but I have not done much testing on this logic. Feel free to try it and report back any issues.

The players

In Broadcast Power, we expose several audio / media players to external applications:

1. FirstPlayer
2. SecondPlayer
3. CommercialPlayer
4. JinglePlayer
5. NewsPlayer

You can control these players or receive events from them by referencing them in your plugin, similar to the sample code above.

Once you have a reference to these players, you can do almost anything that is possible in the user interface of Broadcast Power.

Well, that's it for now. I hope you will have fun writing plugins for BP2X. Drop me a line if you have any question.