

**DevSource**<sup>TM</sup>  
Sponsored by **Microsoft**

Microsoft  
**Visual Studio.net**

Microsoft

Introducing Visual Studio 2005 Team System.  
Communicate. Collaborate. Create.

Microsoft  
**Visual Studio**

Get the Visual Studio Beta 2 Now 

## Streaming with Windows Media Services and ASP.NET

March 2, 2005

By Shawn Wildermuth

I love my music. Like many people these days, I have ripped my entire CD collection to MP3 and I buy all my music online. I am also pretty impatient. I never jumped on the 256MB MP3 player bandwagon because I couldn't figure out what songs I couldn't live without. Now things have changed. I use a 30GB MP3 player but I am running out of room. What's a music addict to do? I realized that since I am pretty connected to the Internet these days, all I should need to do is expose my music collection on an external server and stream them to wherever I am. No need to pick and choose — every album would be at my finger tips.



Luckily, Windows Server 2003 has come to the rescue in the form of Windows Media Services. Included in Windows Server 2003 licensing (Standard and above) is the use of Media Streaming. In this article, I show you how to get started streaming your music or video over the Internet with Windows Streaming Services and ASP.NET.

### Preparing Your Computer

Using this technology takes a little set-up. Bear with me for a little bit, before we get to code examples.

Before we can get started, we need a Windows Server 2003 computer with IIS and ASP.NET set up. Once the computer's set up with the operating system, we can add Windows Media Services (WMS). To do this, go to Add/Remove Programs in the Control Panel, where you will see the dialog in Figure 1:

Figure 1: Add/Remove Programs

Click on "Add/Remove Windows Components" to show the Windows Components dialog, as seen in

Figure 2:

*Figure 2: Windows Components*

If you scroll to the bottom of the list, you will see Windows Media Services. Click on the check box, and press Next to install Windows Media Services. Now we're ready to add our content.

## Exposing Content

Once you have a computer ready for Windows Media Services, you want to expose some content. To do this, launch the Windows Media Services Management Console, as seen in Figure 3:

*Figure 3: Windows Media Service Console*

Once in the console, we can expose our content. This is done through what Windows Media Services calls a *publishing point*. Right-click the publishing-points node, and select "Add Publishing Point (Advanced)" to see the dialog shown in Figure 4:

*Figure 4: New Publishing Point*

In this dialog, you specify the type of stream, the location, and the name. In our case, we want to expose an on-demand publishing point: a directory where our MP3s are stored. Do the following:

- Click the "On-demand" radio button.
- Enter a name for the publishing point. This name becomes part of the URL to your content, so short and memorable is important.
- In the "Location of content" text box, type the directory where the media is stored.
- Last, pick Directory as the Content Type. (There are a myriad of publishing points, but for the purposes of this article we only discuss the Directory type.)

After doing this, your dialog will look something like Figure 5:

*Figure 5: Finished Dialog*

Now you have a publishing point. Before you can use it, you have to figure out how to handle authentication. By default, new publishing points have no authentication enabled, which means your content isn't available outside your own computer. Click on the Properties tab of your publishing point, and select "Authentication" from the property list to reveal a screen like the one shown in Figure 6:

*Figure 6: Authentication*

In my case, I chose just to use Negotiate Authentication (which in turn uses NT authentication), as I am the only one to whom I want to stream my content. Logistically, more than one stream would eat up all my upstream bandwidth, so I decided to rely on NT authentication more for practical reasons than legal ones. Now we are ready to try out our stream.

## Testing Our Stream

Now that we have our server set up, we want to test out to see if it can stream content. Windows Media Services supports three out of the box streaming types:

- RTSP with TCP-based transport (RTSPT)
- RTSP with UDP-based transport (RTSPU)
- HTTP transport

By default, RTSP uses port 1755 and the HTTP transport uses port 80. Depending on your network configuration, you either need to open a port in your firewall or change these default ports. In general, I use the HTTP transport since it works at most of my clients with the fewest headaches.

There are two ways to construct an URL to a specific song on your server, using http and using mms.

The mms protocol is a Windows Media Player specific protocol that supports fail-over to all three transport types; RTSPT, RTSPU then HTTP. Assuming you can use the standard port (80) for http, using mms URLs is most effective.

Crafting an URL is quite simple: the protocol, the machine name, the publishing point name, and the path to the media. For example, using mms, to a machine name called `music.mydomain.com`, on the publishing pointed called `mymusic` for the song in `d:\ourmusic\10,000 Maniacs\In My Tribe\What's the Matter Here.mp3` would look like so:

```
mms://music.mydomain.com/mymusic/10,000 maniacs/in my tribe/what's the matter here.mp3
```

We don't include the `d:\ourmusic` as part of the path, since that was what we used for the root to our publishing point. If you put your new URL in Windows Media Player, you should hear the sweet sound of Natalie Merchant (or your favorite artist).

One caveat is that the mms protocol expects port 80 for the failover to the http protocol. To expose your music over another http protocol, you will be forced to use the http protocol instead. The form is identical except for the replacing the mms with http and postpending the port (e.g. 8080):

```
http://music.mydomain.com:8080/mymusic/10,000 maniacs/in my tribe/what's the matter here.mp3
```

Pointing to specific files is interesting, but I want to be able to browse my music and put together ad-hoc playlists. Unfortunately, there is nothing built-in with Windows Media Services to allow this. To create our user interface, we can go to ASP.NET.

## Creating the Site

Now we can create our site. My goal was to produce a single page that could give me a tree list of all of my music by album; there is so much that I am not interested in picking individual songs. The user interface will look like figure 7 when we're done.

I am not going to go step-by-step of how I created this site, but focus on the code that is specific to Windows Media Services and playlists. The idea is to pick artists or albums to include to a playlist and click "Add to Playlist" to create a playlist of songs on the right side. Once the playlist is created, the user can click either "Playlist (mms)" or "Playlist (http)" to get a playlist of either mms or http (with my 8080 port) saved to his browser, so he can open it in Windows Media Player.

*Figure 7: The Site*

In my example, I use Infragistics' UltraWebTree control for the tree control; you can replace it with any Web tree control.

The core of the code is still simple enough. I go through the directory with all my MP3s and re-create the directory structure in the tree without the individual files. Once the user has checked all the nodes he wants, I add the songs to a simple array and bind it to a list of songs on the right side of the page, as seen in figure 8:

*Figure 8: Songs Selected*

To send each of the songs individually (so the player knows what songs it is playing), we have to put together a playlist. The playlist format here is the ASX file.

The ASX file looks like so:

```
<ASX VERSION="3.0" PREVIEWMODE="NO" BANNERBAR="auto" >
  <ENTRY >
    <TITLE>Sample Song</TITLE>
    <REF HREF="http://sample.microsoft.com/sample1.mp3" />
```

```
</ENTRY>
</ASX>
```

The format of the file is simple. For every song in the playlist, you need an ENTRY section. The entry has a title and ref section to name and point to the media.

While this looks innocently like XML, it is not. The elements are not case sensitive, and the parser in Windows Media Player choke if you include a processing header (e.g. `<? Xml version="1.0">`). But I hate munging text, so I go ahead and use an `XmlDocument` to create the playlist:

```
MemoryStream CreateASX(SortedList songs, string server)
{
    const string PUBPOINT = "OurMusic";

    XmlDocument asx = new XmlDocument();
    XmlElement element = asx.CreateElement("ASX");
    element.Attributes.Append(asx.CreateAttribute("VERSION"));
    element.Attributes["VERSION"].Value = "3.0";
    element.Attributes.Append(asx.CreateAttribute("PREVIEWMODE"));
    element.Attributes["PREVIEWMODE"].Value = "NO";
    element.Attributes.Append(asx.CreateAttribute("BANNERBAR"));
    element.Attributes["BANNERBAR"].Value = "auto";
    XmlElement title = asx.CreateElement("TITLE");
    title.InnerText = "Shawn Wildermuth's Remote Media Player";
    element.AppendChild(title);
    foreach (DictionaryEntry entry in songs)
    {
        FileInfo file = entry.Value as FileInfo;
        if (file != null)
        {
            XmlElement media = asx.CreateElement("ENTRY");
            XmlElement href = asx.CreateElement("REF");
            href.Attributes.Append(asx.CreateAttribute("HREF"));
            href.Attributes["HREF"].Value =
                string.Format("{0}{1}{2}",
                    server,
                    PUBPOINT,
                    file.FullName.Substring(ROOTDIR.Length).Replace("\\", "/"));
            XmlElement mtitle = asx.CreateElement("TITLE");
            mtitle.InnerText = file.Name;
            media.AppendChild(mtitle);
            media.AppendChild(href);
            element.AppendChild(media);
        }
    }
    asx.AppendChild(element);
    MemoryStream stream = new MemoryStream();
    StreamWriter writer = new StreamWriter(stream);
    writer.Write(asx.OuterXml);
    writer.Flush();
    return stream;
}
```

This code is pretty straightforward. If the user has songs in the song list (the files object), we clear the response and send down a playlist file. The `CreateASX` method puts together the playlist file, so we can send it down to the client.

I had some trouble getting it to return the file as anything other than `default.aspx` as a file name. I had to play around with this code for quite a while to allow me to open it as an `.asx` file (the playlist format we're using). The key seems to be to add the "Content-Disposition" header to allow you to specify a filename.

Now that this all works, I can click on the Playlist button and open the `.asx` file (which launches

Windows Media Player), as seen in figure 9:

*Figure 9: Playing the Playlist*

I have only scratched the surface of using Windows Media Services. For larger media-based business, Windows Media Services supports subscriptions, banner ads, and more. It is a mature platform for delivering content. Since it is an included part of Windows Server 2003, don't forget to include it in your next project!

*Download the source code for this article here.*

Learn more programming techniques in DevSource's [Techniques Section](#).

Copyright (c) 2005 Ziff Davis Media Inc. All Rights Reserved.