A Couple of Gadgets for Windows® SideShow™, Using VB.NET

By Sean D. Liming and John R. Malin
SJJ Embedded Micro Solutions / www.sjjmicro.com

January 2008

There have been a few articles, different tools discussed, and many blog posts about SideShow gadgets. Much of this information is scattered across the internet. Since we have been putting together a few articles that bring all this information together, we decided to put our spin on Gadgets with a VB.NET solution. It is always good to see a different solution on a subject. Let's first start off by summarizing Windows® SideShow and Gadgets.

A Little Background

The initial idea behind SideShow is to have small, low-powered, second display that provides content when a PC is turned off. The display will be able to show the latest e-mails and calendar activity, and it can have its own local applications. The original name was the Auxiliary Display, but the name was changed to SideShow. You may see references in various APIs that call out the old name. You may also see references to Windows Sidebar that is part of the Vista Operating Systems. Gadgets are also created for Sidebar as well, but we are going to focus on SideShow in this article.

SideShow itself is simply a shell application that runs on the Microsoft® .NET Micro Framework. Using the *Windows SideShow Device SDK*. OEMs can modify the SideShow shell application to create a different look and feel, as well as, create custom applications that run in SideShow.

To download data from a PC to the device, there needs to be an application that sends the data, a driver to support data transport , and a client that receives and processes the data. The application that sends the data is known as a Gadget. The client that receives and processes the data is known as an Endpoint. The Endpoint is simply a .NET Micro Framework application that is running in SideShow. A Windows driver acts as the transport, when the device is connected to the PC. Typically, this is USB but it could be any wired or wireless connection means, like Bluetooth for example. You can think of Gadgets as nothing more than a web server providing content to a connected device.

Note: SideShow is only supported in Vista Ultimate and Vista Business Editions. To develop Gadgets you will have to be using one of these platforms.

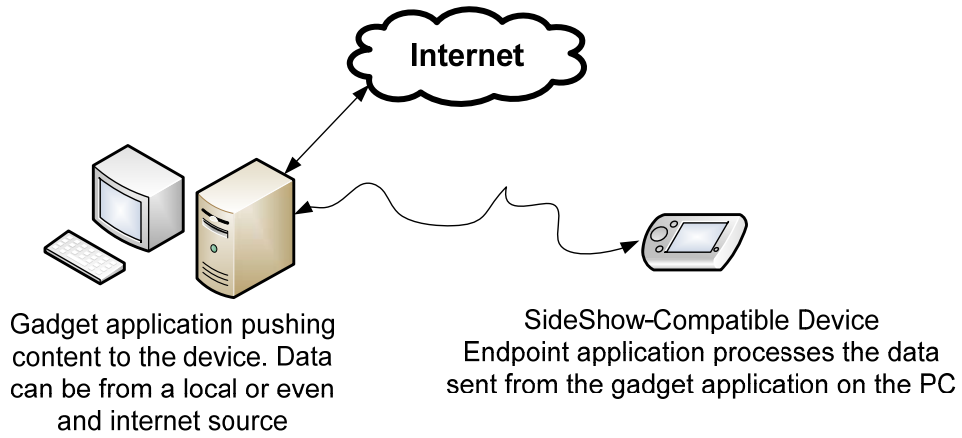


Figure 1 - Basic Concept of a Gadget

There are two built in Endpoints in SideShow. One is for mail, contacts, and calendar information from Microsoft Outlook or Windows Mail, and this is called iCal. Gadgets are already available for this application. iCal is sparsely documented at the time of this article. The other Endpoint is called Simple Content Format (SCF). SCF was specially designed by the SideShow team to provide a basic content renderer that anyone can use to develop Gadgets. SCF allows you to develop Gadgets that pass down text and image information to be displayed on a SideShow compatible device.

You can create your own Endpoint using the Windows SideShow Device SDK, but the discussion on how to create an Endpoint is for a topic for another article. In this article we will stay focused on creating a Gadget using the SCF Endpoint.

A Gadget itself is nothing more than a Windows application, but the application itself has to be registered as a Gadget in the registry. If you have already installed a few Gadgets, you can see a couple of places in the registry where the information is stored. Figure 2 shows the HKEY_Local_Machine (HKLM) location and Figure 3 shows the HKEY_Current_User (HKCU) location. Our custom gadget will be placed in the HKCU.

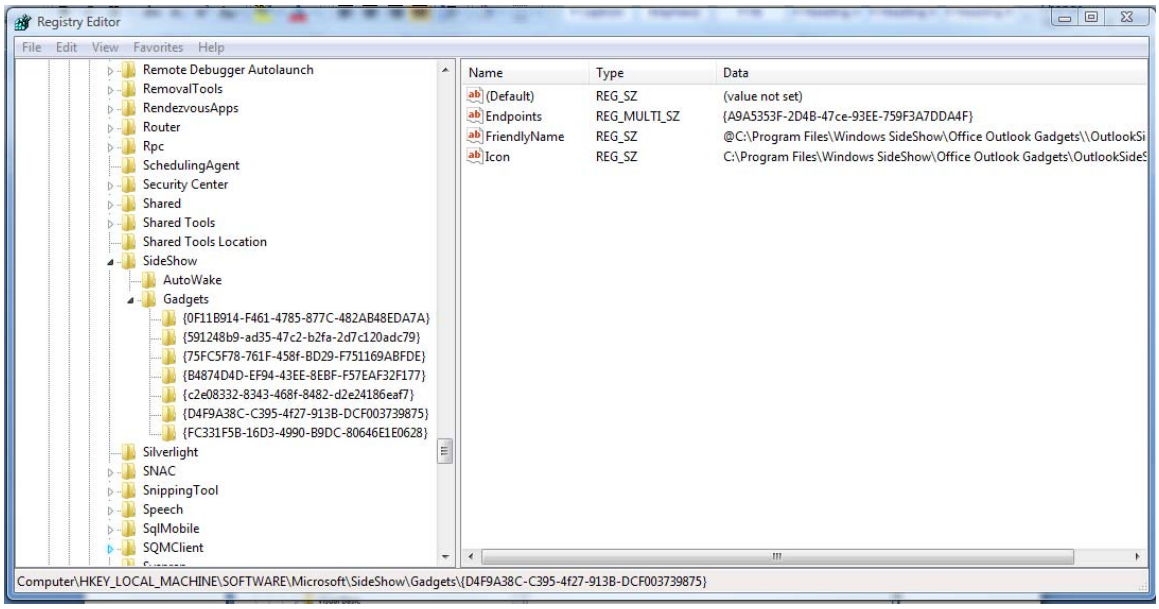


Figure 2 - HKLM Location for Gadgets

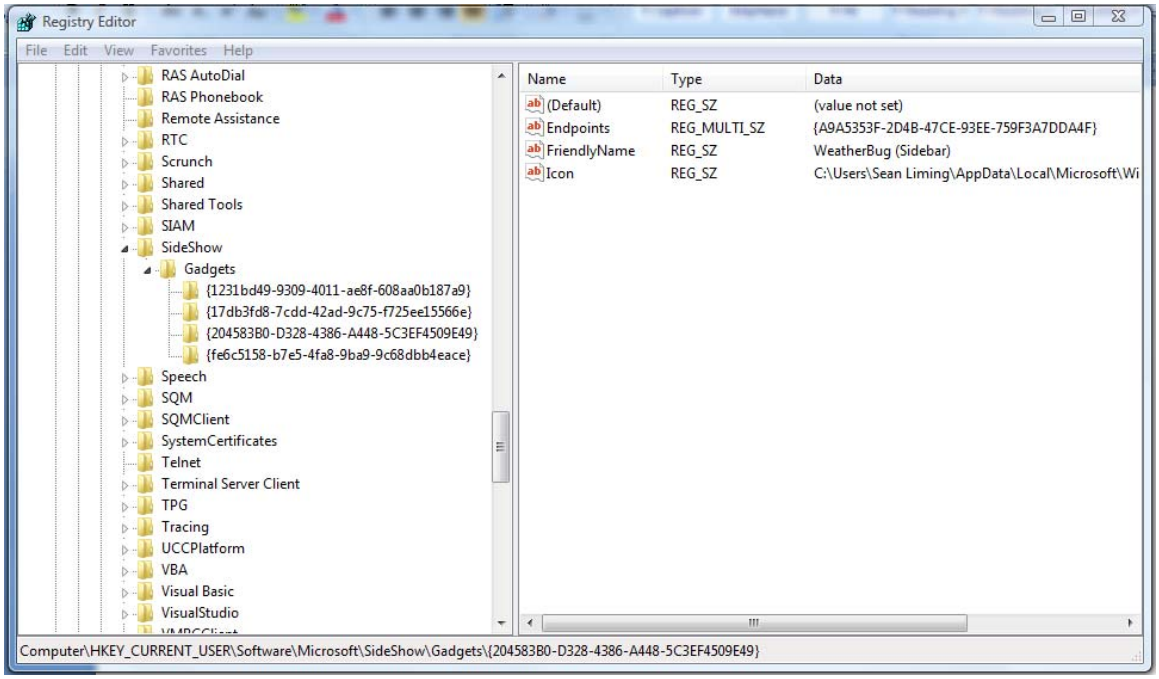


Figure 3 - HKCU Location for Gadgets

The easiest way to see the installed gadgets is in the Windows SideShow Control panel. Figure 4 shows the Windows SideShow Control panel and a few installed Gadgets. From the control panel, you can enabled and disable the gadgets for a specific device. If you are looking for sample gadgets to test, you can click on the link to “Get More Gadgets on Line”, which will take you to a web page with various SideShow and Sidebar gadgets.

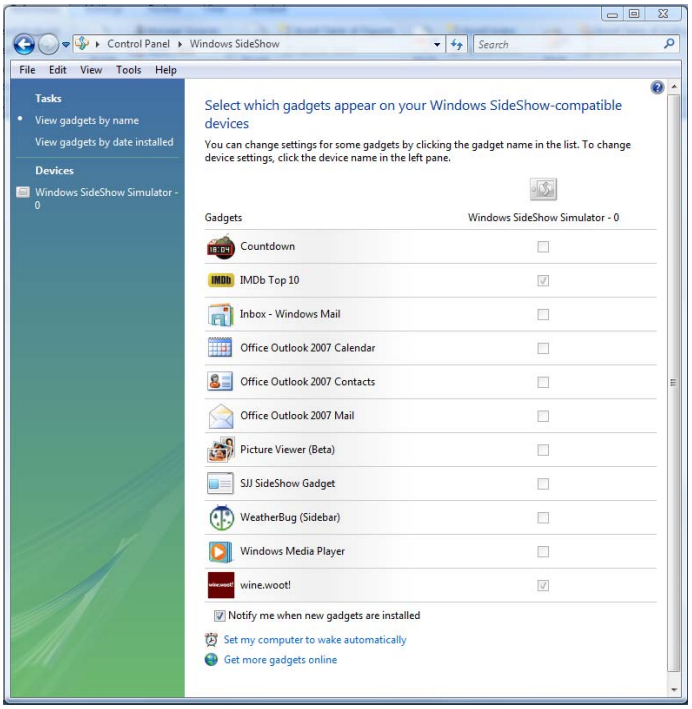


Figure 4 - Windows SideShow Control Panel Interfaces to the Registry to display the available SideShow gadgets.

A gadget can be created using the native programming language like C/C++ or using managed code with C# or VB.NET. To create gadgets in C/C++, you will need the Windows SideShow API header files and library files, which are found in the *Windows SDK Update for Windows Vista*. This SDK includes sample gadget applications. Details on how to use the API and COM programming can be found on MSDN.

Now for the rest of us, there is a managed code wrapper for the Windows SideShow API known as the *Windows SideShow .NET Framework Components 1.0*. The components provide a name space Microsoft.SideShow. The components are downloadable from MSDN. You can find information about the Managed API by doing a search in MSDN for Microsoft.Sideshow: http://msdn2.microsoft.com/en-us/library/aa970188(VS.85).aspx.

If you do a web search, you will find information discussing the managed code API and a few sample gadget solutions:

- Jeffrey Richter wrote an excellent article on SideShow Gadgets in the January 2007 edition of MSDN magazine titled "*Get Started Writing Gadgets For Windows SideShow Devices".* The article is posted here: http://msdn.microsoft.com/msdnmag/issues/07/01/SideShow/default.aspx. There is also a C# Gadget application associated with the article.

- Daniel Moth, a Microsoft Developer in the UK, also has some SideShow Gadget information and video posted on his blog - http://www.danielmoth.com/Blog/.

Most of the examples have been written in C#, but you can write a gadget in VB.NET as well.

With a Gadget, you can download glance content or endpoint content. Glance content is data that appears next to each Gadget icon in the SideShow-compatible device. You can pass specific information to the Gadget such as weather, stock information, tracks playing in media player, etc.

Figure 5 – A Weather Gadget with Glance Data

When the user selects the Gadget, the Gadget opens and based on the endpoint, attempts to process any content that has been downloaded to the device. In our case we will be using the

SCF endpoint to render SCF pages. Gadgets using the SCF endpoint are simply downloading data. Creating mini-web pages is another way of thinking about SCF. When you are developing a gadget, you are creating pages of content that will display information.

A Basic Demo Gadget Application

Figure 6 sketches 3 pages for this basic demo gadget application. The first page will be a menu, the second will have some text, and third page will show a picture. Navigation between the pages will be using the menu or the left and right navigation buttons on the SideShow-compatible device.

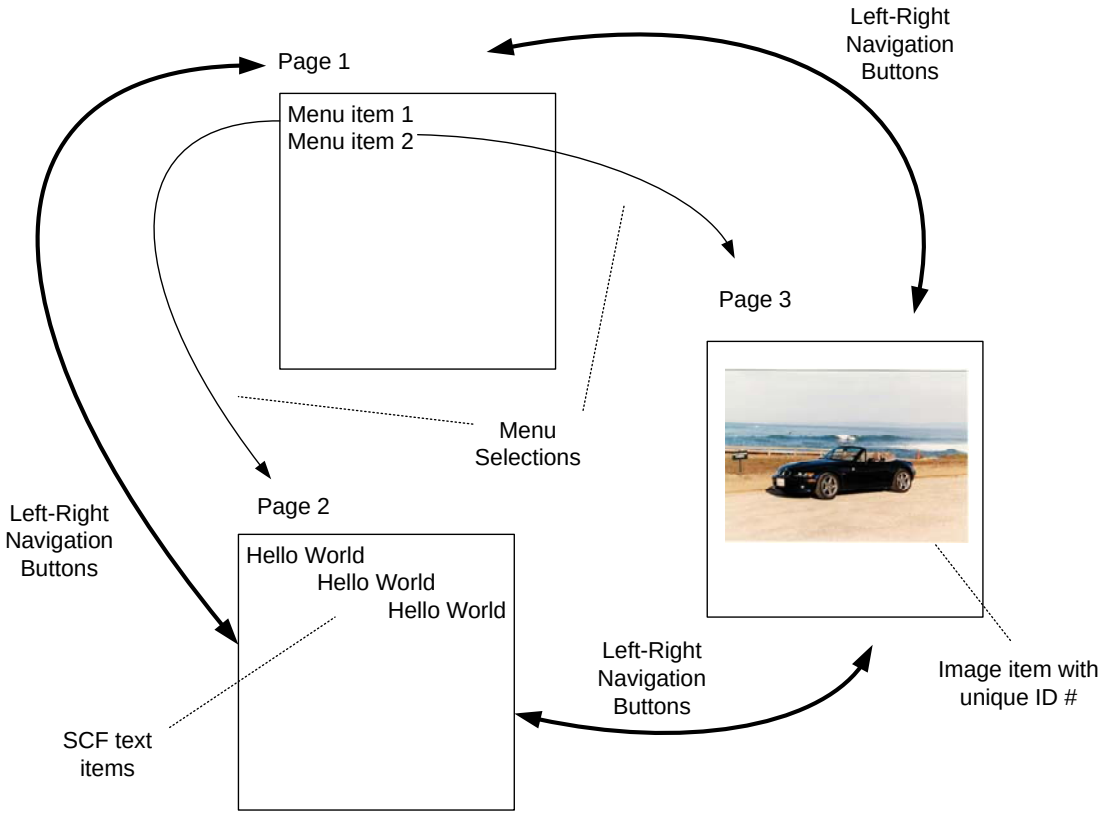


Figure 6 - Content Setup Diagram for Example

Each page has a unique ID number. Pages can have text, images, images and text, a dialog box, menus, and defined actions for what the external buttons on the device will do. SCF itself is really defining XML elements. In the end, there are two ways to send data to a device: via XML or using the wrapper APIs. We will save the XML discussion for another application note.

We want our first Gadget to just download data when the device is connected. The example application in the January 2007 issue of MSDN has a good setup and structure for what we want to do. We will create a variant of this in VB.NET.

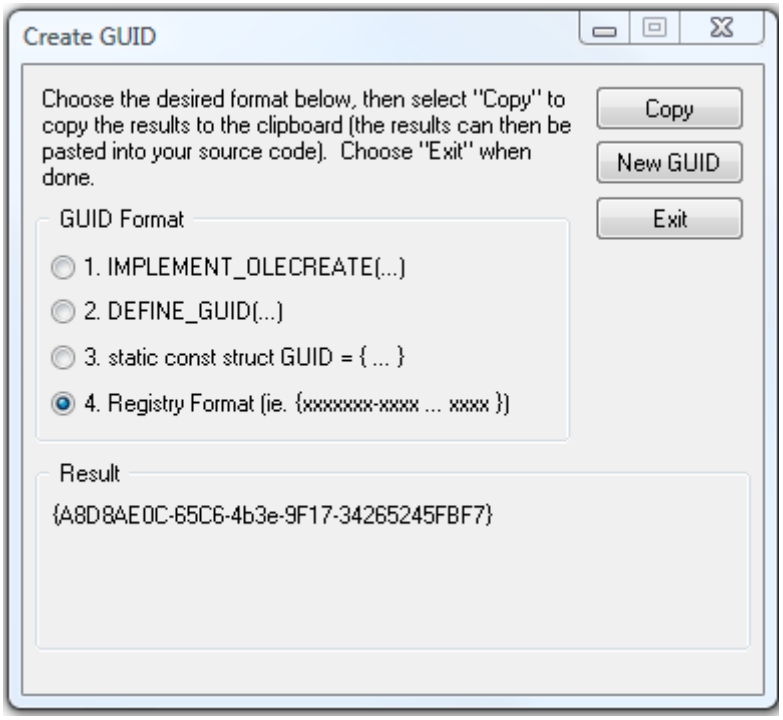
1. Make sure you have downloaded and installed the Windows SideShow .NET Framework Components 1.0 from MSDN.
2. Open Visual Studio 2005, and create a Visual Basic Console Application. You could create the application with a form, and every time the device is attached the form will

appear, All we want for this example is the content to be downloaded to the device, so a console application will do nicely.

3. Add References to the project for Microsoft.SideShow and System.Drawing. We need the later for working with images.
4. Add the Imports so we can take advantage of the namespaces:

```
Imports System
Imports System.Drawing
Imports System.IO
Imports System.Reflection
Imports Microsoft.SideShow
Imports Microsoft.SideShow.SimpleContentFormat
```

5. We are going to build in the ability to Register and Unregister the Gadget, which was a nice feature from the MSDN example. Use the application to register and Unregister itself makes it easier than having to maintain separate .REG files or create a sophisticated installer. Use GUIDGen.exe that comes with Visual Studio to create a GUID, and in the code, create a new GUID instance just before the Main subroutine.




```
Dim Gadget_Demo_GUID As New Guid("{A8D8AE0C-65C6-4b3e-9F17-34265245FBF7}")
```

6. After the Main subroutine, add a DownloadData subroutine. We will come back to fill this in a little later.
7. In the Main subroutine add the following:

```
Sub Main(ByVal args() As String)

        If (args.Length = 0) Then
                Console.WriteLine("SJJ DEMO Gadget [/R] [/U] [/D]")
                Console.WriteLine("   /R   Register the Gadget")
                Console.WriteLine("   /U   Unregister the Gadget")
                Console.WriteLine("   /D   Download Data")
        Return

        End If
```

```
        For Each arg As String In args

                Select Case arg.ToUpper

                        Case "/R" 'Registers the Gadget and sets the location for execution
                and an icon. You can change these to fit your system
                                GadgetRegistration.Register(False, Gadget_Demo_GUID,
                        ScfSideShowGadget.ScfEndpointId, "SJJ Gadget Demo",
                        "C:\dotNETMF\SideShow\Samples\SJJ_Gadget_Demo\SJJ_Gadget_Demo\bin\D
                        ebug\SJJ_Gadget_Demo.exe /D",
                        "C:\dotNETMF\SideShow\Samples\SJJ_Gadget_Demo\SJJ_Gadget_Demo\bin\D
                        ebug\cab.ico", False, GadgetCachePolicies.KeepNewest, Nothing)

                        Case "/U"
                                GadgetRegistration.Unregister(False, Gadget_Demo_GUID)

                        Case "/D"
                                DownloadData()

                        Case "?"
                                Console.WriteLine("SJJ DEMO Gadget [/R] [/U] [/D]")
                                Console.WriteLine("   /R   Register the Gadget")
                                Console.WriteLine("   /U   Unregister the Gadget")
                                Console.WriteLine("   /D   Download Data")
                                Return

                End Select

        Next
End Sub
```

We include the basic command line help information for the end user. When the user registers the Gadget, the GadgetRegistration.Register call creates a new registry entry for our application. First we allow everyone access to the gadget, the GUID is stored in the registry and uniquely defines our gadget, and the endpoint is set to SCF. The next entry is used to launch the gadget and download the data once the device is attached. We set the file path with the /D option. An icon for our gadget will also be downloaded. In these last two entries, we placed the path to the development path of the project, which is why it is so long. Normally, once we have complete the project, we would place the gadget in the \Program Files\Windows SideShow\<Gadget> directory, thus we would have to change the registration paths to the binary. The last three entries deal with actions when entering the gadget, caching, and a GUID link to a parameter table.

The /U option simply unregisters the gadget with a call to GadgetRegistration.Unregister. The Boolean is for the user access to the gadget and in this case it matches what was in the register operation: false.

Having the /D option allows us to manually download data if we want too.

8. In the DownloadData subroutine add the following:

```
Dim Gadget_Demo As New ScfSideShowGadget(Gadget_Demo_GUID)

'Download Glance Data
Gadget_Demo.AddGlanceContent("Data downloaded on " + Date.Now)

Page 1 - Menu
Gadget_Demo.AddContent(Scf.Menu(ScfSideShowGadget.HomePageContentId, "Main Menu",
ScfSelectAction.Target, Scf.Item(2, "Show me some text"), Scf.Item(3, "Take me to the
picture page"), Scf.Btn(DeviceButton.Left, "", 3), Scf.Btn(DeviceButton.Right, "", 2)))

‘Page 2 - The Text Page
Gadget_Demo.AddContent(Scf.Content(2, "Welcome to the Text page", Scf.Txt(ScfAlign.Left,
False, Color.Red, "Welcome to SideShow"), Scf.Txt(ScfAlign.Center, False, Color.White,
"from"), Scf.Txt(ScfAlign.Right, True, Color.DarkBlue, "SJJ Embedded Micro Solutions"),
Scf.Btn(DeviceButton.Left, "", 1), Scf.Btn(DeviceButton.Right, "", 3)))
```

```
'Page 3 - Picture page with picture content
Gadget_Demo.AddContent(Scf.Content(3, "Picture page", Scf.Img(100, ScfAlign.Center,
ScfImageFit.Auto, "No picture found"), Scf.Btn(DeviceButton.Left, "", 2),
Scf.Btn(DeviceButton.Right, "", 1)))
Gadget_Demo.AddContent(100, ImageContentTransforms.ReduceColorDepth Or
ImageContentTransforms.StretchToFit, Image.FromFile("c:\pictures\Z3.jpg")) 'You can
change the to the picture of your choice
```

The first line simple gets an instance of the Gadget. The AddGlanced data just sends a string to the device to be displayed next to the gadgets icon. The glance data here simply sends a string with the data and time of the last data download. You could put a timer in the application so it periodically downloads fresh data.

The next line is the first page of our SCF content. The first page is a menu page that has two menu items that will take you to page 2 or page 3. In addition, the left and right navigation buttons are used to jump to different pages. Figure 7 explains the different entries.

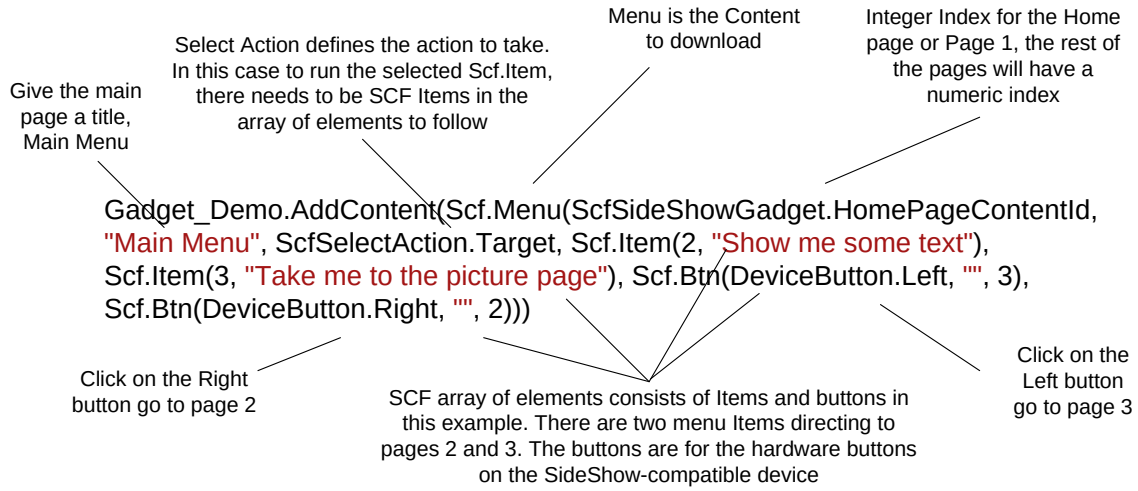


Figure 7 - Break down of an SCF page

The next line adds page 2 – content ID 2, which is nothing more than 3 lines of text with different colors and justification placement. The left and right navigation buttons are also setup to navigate to the other 2 pages.

The final 2 lines add the 3rd page (content ID 3) and the graphic to be displayed. The graphic in this case is from the local file system so you can change the path to any picture in your system. The link between the page and the graphic is the Integer value 100. The value 100 is a content ID like the content IDs for the different pages, but the page 3 line is what displays the data since the endpoint is SCF. It is important to note that all content download to the device must have a unique content ID. The value 100 was used, but we could have easily chosen 4 or 50. Images themselves can be from an external source or built into the application.

The next step is to test the Gadget. Build the gadget application, and then register it from the command line:

```
C:\>SJJ_Gadget_Demo /R
```

To test the gadget, you can use a Windows SideShow-Compatible Device or use the SideShow Device Simulator. There are two versions of the simulator available. One is a standalone version that can be downloaded from MSDN. The other comes with the *Windows SideShow Device SDK*.

The important thing to know is that only one of these simulators can be installed/register in the system at a time. If you attempt to install both simulators, they will conflict with one another.

For this example, we used a simulator for testing. You have to launch the simulator and enable the newly registered gadget in the SideShow control panel for the device. Once enabled, you can access the gadget. Figure 8 shows the glance data and the 3 SCF pages.

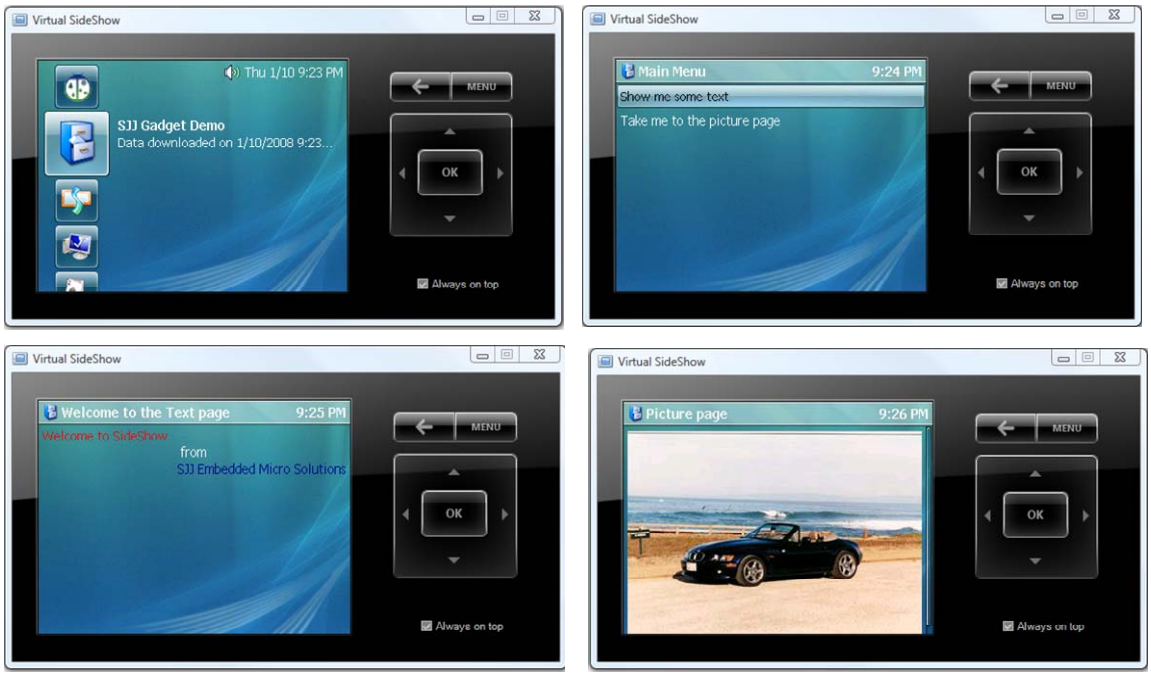


Figure 8 - Using the SideShow-Device Simulator to test the Gadget, we see Glance Content and the 3 pages

A Little Debug Knowledge

A good development method is to have the project still open in Visual Studio. If the application crashes you can quickly debug it. Since this is a Windows Application, you can debug it directly. We added a call to DownloadData() at the end of the Main subroutine, and commented out the Return statement in the zero arguments test. With the Simulator running, you can then start the debugger and step through the application.

A MyPics Gadget

There is a C/C++ pictures gadget in the *Windows Software Development Kit Update for Windows Vista*. The gadget downloads pictures from the MyPictures folder. Each picture is its own page where navigation is with the left and right buttons. Figure 9 show a drawing of how this would work. In this example, we will demonstrate something similar. The twist is that JPG files will be downloaded from a directory called c:\Pictures.

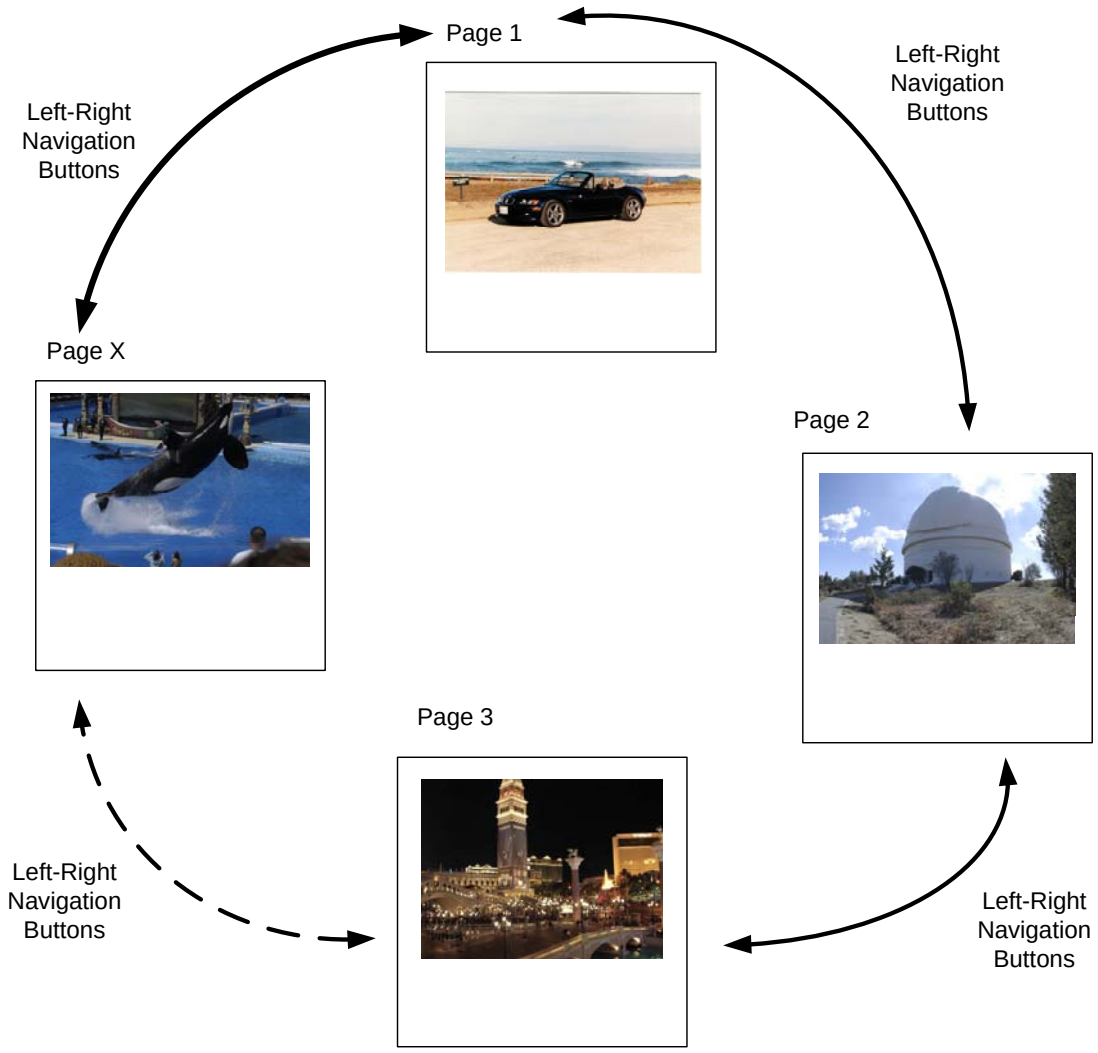


Figure 9 - MyPic Gadget Drawing

There are two areas that have to be addressed. First is the transition between the last page and the first page. We will not know the exact number of pictures that will be found in the directory. Of course, there will be memory size limitations for specific hardware which will force a specific number to be downloaded, and this could arise to a Form application where one picks and chooses the pictures to be downloaded to the device. In this example, we will limit the number of pictures in that we put in the c:\pictures directory.

Second is the number of picture and pages. Images and pages each get a unique ID. Pages are going to start with 1 and count up. Since we will be loading these pictures in a series, we will need to start numbering them such that the number of pictures doesn't overrun the number of pages. The application could start the pictures content ID at 10 and download 15 pictures. As per the first issue, we will limit the pictures that are placed in the subdirectory and start the image content IDs at a 100.

Per the previous example, create a Visual Basic console application, add the references, include the same Imports, generate a GUID, and create the select-case to handle the input parameters. Set the correct path for the application and icon registration. Instead of DownloadData, create a DownloadPics subroutine. The DownloadPcis subroutine would look like the following:

```
Dim MyPics_Gadget As New ScfSideShowGadget(MyPics_GUID)

'Get a list of the JPG files in the c:\pictures directory
Dim PicFiles As String() = Directory.GetFiles("c:\pictures", "*.jpg")

‘Send down the glance data
MyPics_Gadget.AddGlanceContent(String.Format("There are " + PicFiles.Length.ToString + "
pictures downloaded on " + Environment.NewLine + Date.Now.ToString))

'Setup the First Page
MyPics_Gadget.AddContent(Scf.Content(1, PicFiles(0), Scf.Img(100, ScfAlign.Center,
ScfImageFit.Screen, "picture not there"), Scf.Btn(DeviceButton.Right, "", 2),
Scf.Btn(DeviceButton.Left, "", PicFiles.Length)))
        MyPics_Gadget.AddContent(100, ImageContentTransforms.ReduceColorDepth Or
ImageContentTransforms.StretchToFit, Image.FromFile(PicFiles(0)))

'Add the pages between the first and the last
For x As Integer = 1 To PicFiles.Length - 2

        MyPics_Gadget.AddContent(Scf.Content((x + 1), PicFiles(x), Scf.Img((100 + x),
        ScfAlign.Center, ScfImageFit.Screen, "picture not there"),
        Scf.Btn(DeviceButton.Right, "", (x + 2)), Scf.Btn(DeviceButton.Left, "", x)))
        MyPics_Gadget.AddContent((100 + x), ImageContentTransforms.ReduceColorDepth Or
        ImageContentTransforms.StretchToFit, Image.FromFile(PicFiles(x)))

Next

'Setup last page
MyPics_Gadget.AddContent(Scf.Content(PicFiles.Length, PicFiles(PicFiles.Length - 1),
Scf.Img((100 + (PicFiles.Length - 1)), ScfAlign.Center, ScfImageFit.Screen, "picture not
there"), Scf.Btn(DeviceButton.Right, "", 1), Scf.Btn(DeviceButton.Left, "",
(PicFiles.Length - 1))))
MyPics_Gadget.AddContent((100 + (PicFiles.Length - 1)),
ImageContentTransforms.ReduceColorDepth Or ImageContentTransforms.StretchToFit,
Image.FromFile(PicFiles(PicFiles.Length - 1)))
```

As with the first example, we create an instance o the Gadget. The next line gets the file path for the JPG files found in the c:\pictures directory. The PicFiles array information is then used to send out the glanced data and the number of pictures found.

To resolve the first issue, we setup the first page, and set the left right buttons to navigate to the second page and the last page. Next, we use the For-Next loop to setup the pages between the first and the last, and we maintain the navigation buttons. Finally, we create the last page, where the navigation buttons must go to the first page and the previous. The tricky part was correlating the indexing of the picture file array with the page numbering. Since our PicFiles array starts with 0 and pages start with 1, the indexing had to be adjusted by 1. If this was not taken into account some pictures might be missing when scrolling through them or some pages might not be setup properly.

You can add other features to this application like a timer to periodically download pictures or check of the number of pictures found and the starting count ID of the picture images.

You will have to create a directory called c:\pictures and copy some JPG files to it. Running the gadget, you should see the glance data with the number of JPG files found. Selecting the Gadget you should be able to use the left and right navigation buttons to flip between the picture pages.

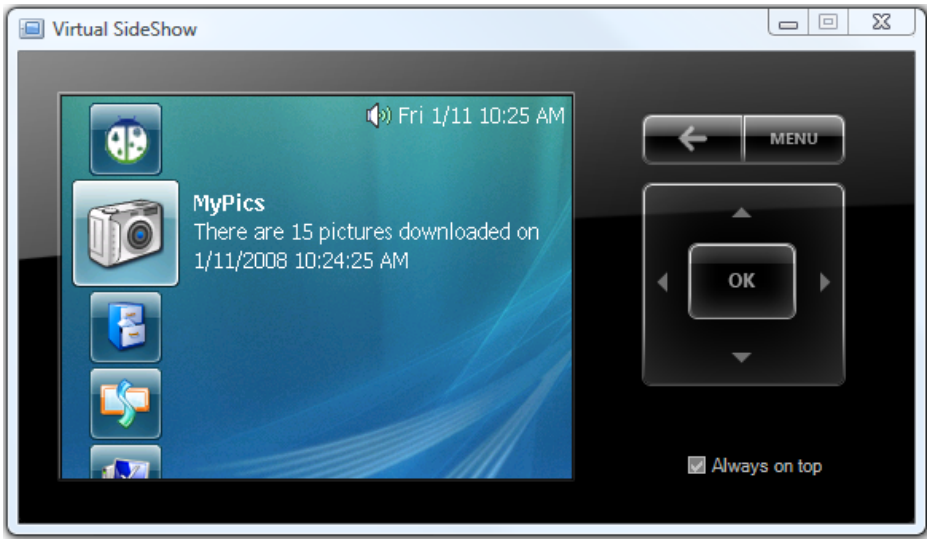


Figure 10 - MyPics Gadget with 15 Pictures Found

Summary on Gadgets

The endpoint for these gadgets was SCF. If a device manufacturer creates a custom endpoint, you would have to write the application for that specific endpoint. With SCF, you should diagram the pages before writing the code. This will help create a good picture layout of how the gadget will perform on the device.

We used a console application project for these examples. If you ran the gadgets, you will notice that nothing appeared on the PC when the SideShow-compatible device ran the gadget. Gadgets working behind the scenes, but you could have easily created a Form application and have some GUI interface to interact with the device.

Finally, gadgets are nothing more than PC applications that are registered to work with SideShow. For example, we could have a SideShow compatible device that a maintenance person plugs into an embedded PC platform. The connectivity action launches an application on the embedded PC system. The application can either interact with the SideShow-compatible device or it simply allows back end access for the maintenance personal.

Windows is a registered trademark of Microsoft Corporation.