

CHAPTER 6

Resource File Tools

THIS CHAPTER IS DEVOTED to those tools that .NET provides to help edit and manipulate resource files without using code. Many of these tools are external to the IDE. They either require you to dig up old “DOS days” memories or to acquire new ones. It all depends on how old you are!

I also go into quite a bit of depth concerning XML resources. While this is not a book on XML, it is necessary that you understand the XML files that are generated by .NET and why they are necessary.

Resource Tools

There are a few tools that .NET provides to aid in handling resource files.

XML Designer in Visual Studio

Here is where I think most people will externalize their resource files. It's easy and the casual programmer never need know the files he or she creates are XML files.

To invoke this tool you need to be editing a resource file from within the VS IDE. Bring up a new project; either form-based or console-based. Go to the project window and add a new item of type “Assembly Resource File.”

Alternately, you can do this from the File/Add new menu selection. You are immediately taken into the XML designer in data view. Anytime you want to edit this file just double-click it and the XML designer will appear.

There are two views for this designer. One is the XML view and the other is the data view. The XML view includes color-coding, IntelliSense, word completion, and member lists. The data view is simple. Just type in the key, comment (if any), and value. There you go. Easy as pie.

OK, so there you are, happy as a clam inputting hundreds of strings into this resource file. It is time to translate the strings. What now? Try sending the .resX file to a translation service. It is not the best format for translating strings. Not only that, but what about pictures and sounds, and so on? The XML designer does not even allow you to input objects.

Chapter 6

The XML designer is okay for simple programs but becomes woefully inadequate for large ones. You are better off making your own resource file editor. You do just that in the next chapter.

The XML designer is not just there for resource files. It is there as an aid in developing XSD schemas and XML documents. It has a great design and layout facility for creating schemas and editing data sets. This is where it really shines. I could go much further into this aspect of XML in .NET but I promised I would focus on the resource file aspects.

ResGen.exe

ResGen is a great utility. It allows you to take a resource file in any of its three forms and translate it to any of the other three forms. You can do the following:

- Convert a .txt resource file into a .resources file
- Convert a .txt file into a .resx file
- Convert a .resx file into a .resources file
- Convert a .resx file into a .txt file
- Convert a .resources file into a .resx file
- Convert a .resources file into a .txt file

ResGen is a must-use utility. To embed a resource file into an assembly or into a satellite resource file you must first convert it from one of the text forms to the binary .resources form.

ResGen also takes as a command line argument, the /Compile switch. This allows ResGen to do batch processing. You can put in any number of *.txt or *.resx files and it processes each in turn. Table 6-1 shows a matrix of ResGen uses.

Table 6-1. ResGen Uses

COMMAND LINE	DESCRIPTION
Resgen myres.txt	Compiles myres.txt into myres.resources
Resgen myres.txt mybinaryres.resources	Compiles myres.txt into mybinaryres.resources
Resgen myres.txt myresx.resx	Converts myres.txt into myresx.resx XML file
Resgen myxres.resx	Compiles myxres.resx into myxres.resources
Resgen myxres.resx mybinaryres.resources	Compiles myxres.txt into mybinaryres.resources
Resgen myxres.resx myres.txt	Converts myxres.resx into myres.txt text file
Resgen mybinaryres.resources myres.txt	Converts mybinaryres.resources into myres.txt text file
Resgen mybinaryres.resources myxres.resx	Converts mybinaryres.resources into myxres.resx
Resgen /compile res1.txt res2.txt res3.txt	Compiles all res files into separate .resources files

ResGen Caveats

- Compiling from a text file to .resources file loses all your comments. This means that converting from a .resources file to a text file leaves you with your data only.
- Converting a text resource file to an XML resource file also loses the comments.
- Place a comment on its own line. Comments tacked on the end of a resource line make the ResGen compiler think it is part of the string text.
- Converting a .resources file to a text file gives a text representation of any objects such as pictures, and so on. This means that you cannot reverse this process and reconvert to a .resources file. The text file output will have an entry stating what the picture and its type was. The data will not be converted.

The following entries illustrate how to write a text-based resource file.

```
STR_ONE = one
STR_two = two
Str_three = three
;This is a legal comment line
str_4 = four;This comment is taken to be part of the string value
str5 = "String five"
```

Chapter 6

The ResGen compiler happily compiles this file to a .resources file. The only problem is that it believes everything after the = sign on the str_4 line is the actual string.

Al.exe

This is the assembly generation tool. It has only one use as far as resource files go.

- Generate a satellite assembly out of a resource file.

Remember the satellite assembly? This is the binary .resources file compiled into a separate DLL. The binary resource file should have the following name convention: <basename>.<culture>.resources. These satellite DLLs should be put in their proper directories for the resource fallback mechanism to work.

There are quite a few options for the Al tool. I'll go over how to use this tool to embed resource files into satellite resource files.

The following is a way to make a satellite resource file that can be used in a resource fallback scheme.

```
Al /out:Myprog.Resources.DLL /v:1.2.3.4 /c:es-ES /embed:MyStrings.es-  
ES.resources,MyStrings.es-ES.resources,private
```

- /out: Specifies the output satellite file
- /v: Specifies the version of this assembly
- /c: This is the internal culture identifier
- /embed: <file>,<name><file> is the name of the file that is getting embedded. <name> is the internal identifier for the resource
- private This specifies that the resource is not visible to other assemblies.

The following use is also legal. The missing options are all defaulted.

```
Al /out:MyprogRes.dll /embed:MyprogRes.resources
```

Note that the Al tool only takes a binary resource file as an argument. It cannot make a DLL out of a text file or an XML file.

Embedding

What about embedding a resource file directly into the executable? Why should you use embedding again? It is best practice to embed a culture-invariant resource file in the program executable. This allows for resource fallback. If you have a small program that you are not translating, the embedded resource file could also serve as the only resource file. This makes for a simpler installation. You can embed a resource file into an executable using the command line compiler.

Here is a sample used to generate a C# program with an embedded resource file:

```
csc /target:winexe /out:Myprog.exe /res:MyStrings.resources Myprog.cs
```

The csc compiler can also be used to generate a VB program with an embedded resource file.

```
csc /target:winexe /out:Myprog.exe /res:MyStrings.resources Myprog.vb
```

Here is a scenario. You have a large program written in VB that also has a large string resource file. Perhaps it has a few hundred strings. Your boss has just told you to convert the program to VB.NET. What do you do about the resource file? Do you chuck it? Do it over? Well it just so happens that the Al.exe tool will also build a satellite assembly out of a Win32 resource file. This is the file you create using Visual Studio V 5 or V 6. The Win32 resource text file has the extension of .rc and the compiled resource file has the extension of .res. Here is the command to compile a .res file into a .NET satellite DLL.

```
Al /embed:MyStrings.res /win32res:MyStrings.res
```

Keep in mind what I went through in Chapter 5. To reference these resources in this file you need to use numbers.

The intricacies of the Al.exe tool and the csc.exe tool are beyond the scope of this book. What I have explained here is the most basic use. But I want to remind you of something about compiling .NET code. Perhaps you did not know this.

The Visual Studio .NET IDE compiles only single-file assemblies. In order to create multifile assemblies you need to resort to DOS and use the command line Al.exe or csc.exe compilers. The command line csc.exe compiler can be used to create the individual assemblies and the Al.exe tool can be used to create a manifest that contains all references to external modules.

I strongly encourage you to read and understand the documentation concerning the command line compilers. If you want to do any kind of sophisticated compiling and linking you will need to use them.

Chapter 6



NOTE *It seems to me that we have come full circle here. I have been programming long enough to remember using command line compilers and linkers with long lists of .obj files to be linked in. The past several years have been spent inside the IDE. It is now necessary to go outside the IDE to the command line to do any sophisticated compiling and linking. This after Microsoft has been trying to hide the DOS box from the casual user.*

IDE Forms Designer

Here is an interesting way to design a localized program. It is possible using the IDE in either C# or in VB to create a form for each language you want your program to run in. The IDE does all the work for you insofar as making resource files and satellite assemblies and putting them in all in their correct directories.

This process involves changing some of the forms' properties at design time and typing in your translations. In a complex form with many controls, this can be quite the time-consuming and error-prone task.

I will say this, however, for the IDE designer approach. It may be the best way to achieve a form that looks like it was written in the target language. This is because for each language you are able to move and resize the controls on the form. No longer do your controls have to be much larger than necessary to accommodate possibly longer translated strings. They can all be just the right size for the particular languages needs. This is pretty neat if you ask me.

Start up a Windows Forms project in either C# or VB, and click the form itself. Look at the properties box. You should see something like what you see in Figure 6-1.

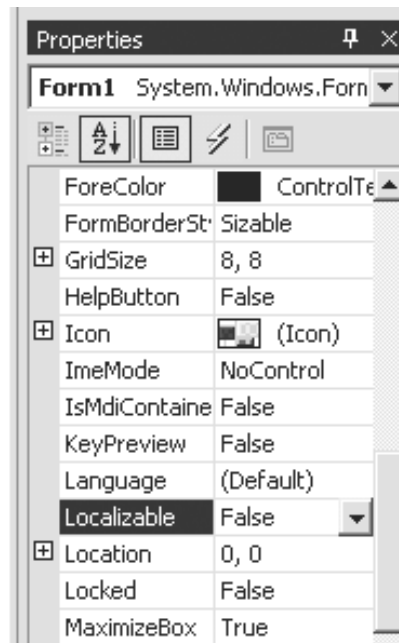


Figure 6-1. Form properties showing the localizable property

The two properties we are interested in are the localizable and the language properties.

If we want to start this process we need to change the localizable property from false to true. The language property is by default; default. What this means is that the resource file that gets generated is the culture-invariant resource file for this project. This form resource file is created if you use localization or not.

You may wonder if you can see this resource file. Yes, you can. Go into the Solution Explorer pane and turn on "Show all files." You should now see what is shown in Figure 6-2.

Chapter 6

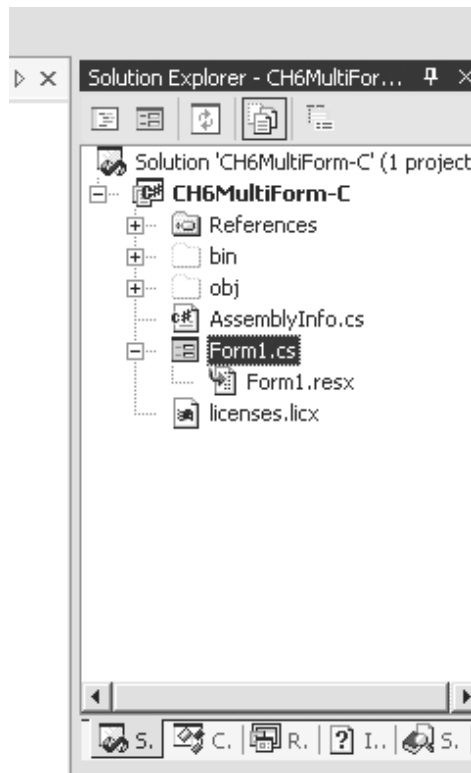


Figure 6-2. Click “Show All Files” to see the localized form resource

As you can see, the file generated is `Form1.resx`. This is an XML-based resource file. You see what is inside in a little while. First, expand the references node in this pane. See anything interesting? I am talking about the reference to the `System.Windows.Forms.dll`. Of course you cannot get a form on the screen without this dll but remember also that it contains the `ResXResourceWriter`, `ResXResourceReader`, and other `ResXnnn` classes. These are the classes in the `System.Resources` namespace that are necessary to write XML based resource files.

Are you starting to understand a little of how .NET was put together?

Back at the ranch . . . Drag a control; let's say a label onto the form. Drag a few more controls on the form. Enter text values for all these controls. Here is what I placed on my form shown in Figure 6-3.

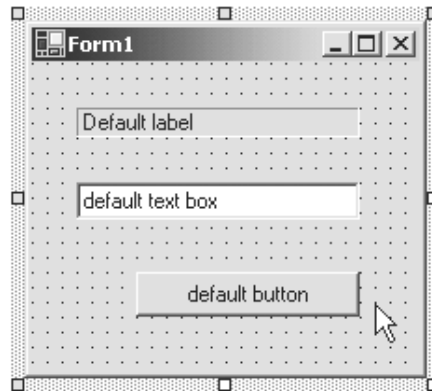


Figure 6-3. Basic form with some controls to show localization

Not much here but there is enough to continue. Click the form again and change the language property to Azeri (Cyrillic)(Azerbaijan). Look in the Solution Explorer and you should see two more resource files. They are shown in Figure 6-4.

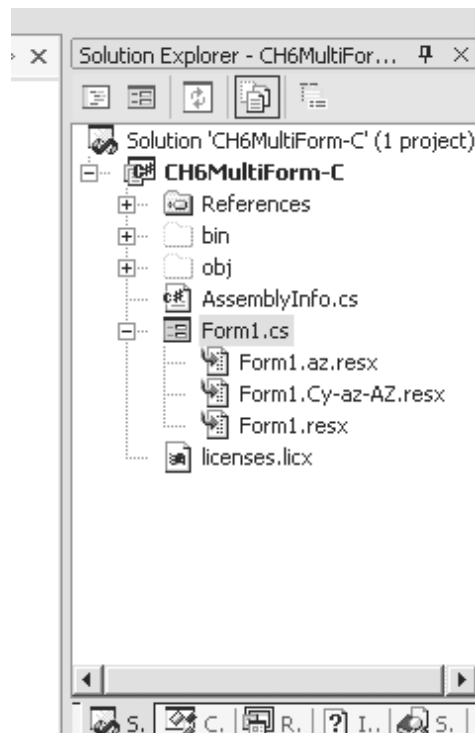


Figure 6-4. All localized forms shown for this example

Chapter 6

See how they are named? They are named according to the culture and region. Now go ahead and change the text in the controls to some thing else that identifies the form. Here is what I have in Figure 6-5.

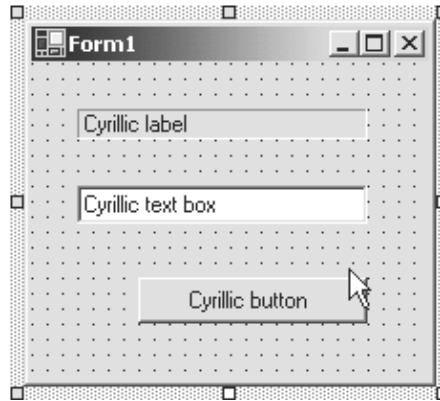


Figure 6-5. Cyrillic form with text to identify it as such

Now is a good time to save and compile your program. This is so cookie cutter that you should have no errors. You have yet to type any code! Okay so now is the time to go answer that question that has been burning in your mind. Why were two resources files generated when I changed languages in the form, and what is in them?

A Look at IDE-Generated Resource Files

Go into the Windows Explorer and find the resource file named Form1.az-AZ-Cyrl.resx. Open this file with notepad or a similar text editor. Listing 6-1 is what you should see.

Listing 6-1. Incremental form resource file

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" targetNamespace="" xmlns=""
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata=
      "urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
```

```

<xsd:element name="data">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0"
        msdata:Ordinal="1" />
      <xsd:element name="comment" type="xsd:string" minOccurs="0"
        msdata:Ordinal="2" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="mimetype" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
<xsd:element name="resheader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0"
        msdata:Ordinal="1" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<data name="label1.ImeMode" type="System.Windows.Forms.ImeMode,
  System.Windows.Forms">
  <value>NoControl</value>
</data>
<data name="label1.Text">
  <value>Cyrillic label</value>
</data>
<data name="textBox1.Text">
  <value>Cyrillic text box</value>
</data>
<data name="button1.ImeMode" type="System.Windows.Forms.ImeMode,
  System.Windows.Forms">
  <value>NoControl</value>
</data>
<data name="button1.Text">
  <value>Cyrillic button</value>
</data>

```

Chapter 6

```

<resheader name="ResMimeType">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="Version">
  <value>1.0.0.0</value>
</resheader>
<resheader name="Reader">
  <value>System.Resources.ResXResourceReader</value>
</resheader>
<resheader name="Writer">
  <value>System.Resources.ResXResourceWriter</value>
</resheader>
</root>

```

First, notice that it starts out with XSD schema commands that define the data that follows. Next notice that some of the properties of all the forms' constituent controls are in here. These properties relate to what you have done to localize this form. So far you see just the text changes you made. Later you will make some other positional changes and revisit this file.

One important thing to note here is that this resource file is an incremental change file from the base form. Open up the form1.resx file in another instance of notepad. See that it is much larger. It contains everything having to do with the form including a binary representation of the form's icon. Including it here would add too much weight to the book. And this is a small resource file! The size of this file and what is—or is not—in it will be important to remember when you get to the WinRes.exe tool. Stay tuned!

Go back to the project and make the button the same size as the other controls on the form. You should still be in the Cyrillic form. Here is what you should see in Figure 6-6.

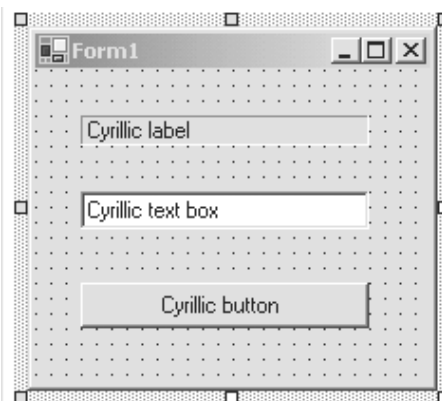


Figure 6-6. Cyrillic form showing different button size

Save and recompile your code. Here is the cool part. Open up the Form1.az-AZ-Cyrl.resx file again. Notice any differences? Listing 6-2 is mine.

Listing 6-2. Incremental form resource file with changes

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" targetNamespace="" xmlns=""
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata=
      "urn:schemas-microsoft.com:xml-msdata">
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                  msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
                  msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                  msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
    <xsd:schema>
      <data name="label1.ImeMode" type="System.Windows.Forms.ImeMode,
        System.Windows.Forms">
        <value>NoControl</value>
      </data>
```

Chapter 6

```

<data name="label1.Text">
  <value>Cyrillic label</value>
</data>
<data name="textBox1.Text">
  <value>Cyrillic text box</value>
</data>
<data name="button1.ImeMode" type="System.Windows.Forms.ImeMode,
                                System.Windows.Forms">
  <value>NoControl</value>
</data>
<data name="button1.Location" type="System.Drawing.Point, System.Drawing">
  <value>24, 112</value>
</data>
<data name="button1.Size" type="System.Drawing.Size, System.Drawing">
  <value>152, 24</value>
</data>
<data name="button1.Text">
  <value>Cyrillic button</value>
</data>
<resheader name="ResMimeType">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="Version">
  <value>1.0.0.0</value>
</resheader>
<resheader name="Reader">
  <value>System.Resources.ResXResourceReader</value>
</resheader>
<resheader name="Writer">
  <value>System.Resources.ResXResourceWriter</value>
</resheader>
</root>

```

You see a few more lines this time. Now that you changed the size of the button from that of the original form, you get size and positional information in the resource file. As you can see, these files are only as big as they need to be.

Click the form and change the language back and forth from the Cyrillic to the default. See the form change? You now effectively have two forms.

Second Resource File

I told you I would explain both resource files. The second one was made when you originally changed the form to Cyrillic and is called Form1.az.resx. Open this form with Notepad. What do you see? It is empty. Why? It has the name of a language-only resource file. No region is included in the name of the file. This leads me to believe that it was created as a string resource file for your program. Being empty however is puzzling. It should at least have the necessary XSD information so it can be edited with the XML editor. Well, I am using the Beta2 CD for this book and I can only hope that this file will be properly formed in the release version. By the way, if you try another nonregionalized language such as German you will not get the second resx file.

It is time to see your program at work. Press F5 and start the program. You will see the original default form show up. Close the program and open your code pane.

If you have made this program in C# put the following lines at the top of your code.

```
using System.Threading;  
using System.Globalization;
```

Now type the following line of code in the form1 section just before the “InitializeComponent();” line.

```
Thread.CurrentThread.CurrentUICulture=new CultureInfo("az-AZ-Cyrl");
```

If you have made this program in VB put the following lines at the top of your code.

```
Imports System.Threading  
Imports System.Globalization
```

Now type the following line of code in the New function just before the “InitializeComponent” line.

```
Thread.CurrentThread.CurrentUICulture=new CultureInfo("az-AZ-Cyrl")
```

Now run your program again. You should see the Cyrillic version of your form appear. Comment out this “Thread” line, run the program again and you should be back to the default form.

Well, this was easy enough. You made a small program that had two different versions of the same form. You saw that the controls on each form could be resized and even moved around. If you took a peek in the forms resource file to

Chapter 6

see what was going on in there and you saw that .NET makes a base resource file for the form and subsequent incremental form resource files for each language you choose.

Let's take a last peek at the program you just made. Go into the forms code and expand the Windows Form Designer generated code section. Take a look at the code in the InitializeComponent method. This method makes a ResourceManager and goes out to the resource file and gets all the information pertaining to the form. Notice the one thing that does not get saved to the resource file. The name of each control is hard coded. Go back to the form and change the forecolor of one of the controls. Rebuild the project and look at this section of code again. You will see that the color of the control is now also hard coded. There are obviously some things that you are not allowed to externalize in a resource file. Listing 6-3 shows some of this code.

Listing 6-3. Windows generated code to retrieve resources

C#

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    System.Resources.ResourceManager resources = new
    System.Resources.ResourceManager(typeof(Form1));
    this.label1 = new System.Windows.Forms.Label();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AccessibleDescription =
        ((string)(resources.GetObject("label1.AccessibleDescription")));
    this.label1.AccessibleName =
        ((string)(resources.GetObject("label1.AccessibleName")));
    this.label1.Anchor =
        (System.Windows.Forms.AnchorStyles)(resources.GetObject("label1.Anchor"));
        this.label1.AutoSize = ((bool)(resources.GetObject("label1.AutoSize")));
        this.label1.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
```

```

        this.label1.Cursor =
((System.Windows.Forms.Cursor)(resources.GetObject("label1.Cursor")));
        this.label1.Dock =
((System.Windows.Forms.DockStyle)(resources.GetObject("label1.Dock")));
        this.label1.Enabled = ((bool)(resources.GetObject("label1.Enabled")));
        this.label1.Font =
((System.Drawing.Font)(resources.GetObject("label1.Font")));
        this.label1.ForeColor = System.Drawing.Color.Red;
        this.label1.Image =
((System.Drawing.Image)(resources.GetObject("label1.Image")));
        this.label1.ImageAlign =
((System.Drawing.ContentAlignment)(resources.GetObject("label1.ImageAlign")));
        this.label1.ImageIndex =
((int)(resources.GetObject("label1.ImageIndex")));
        this.label1.ImeMode =
((System.Windows.Forms.ImeMode)(resources.GetObject("label1.ImeMode")));
        this.label1.Location =
((System.Drawing.Point)(resources.GetObject("label1.Location")));
        this.label1.Name = "label1";
        this.label1.RightToLeft =
((System.Windows.Forms.RightToLeft)(resources.GetObject("label1.RightToLeft")));
        this.label1.Size =
((System.Drawing.Size)(resources.GetObject("label1.Size")));
        this.label1.TabIndex = ((int)(resources.GetObject("label1.TabIndex")));
        this.label1.Text = resources.GetString("label1.Text");
        this.label1.TextAlign =
((System.Drawing.ContentAlignment)(resources.GetObject("label1.TextAlign")));
        this.label1.Visible = ((bool)(resources.GetObject("label1.Visible")));
        this.label1.Click += new System.EventHandler(this.label1_Click);
        //
    ...

```

This is a very powerful way to localize your program at the design stage. However, it soon gets unwieldy when you have many forms with many controls running under many languages. It is virtually impossible to manage and localize any kind of large program in this manner. .NET does provide a tool to help localize the forms themselves. This tool is not much better than using the IDE. The tool is called WinRes.exe.

A list of some IDE designer caveats follows:

- The same controls must be on all versions of the form.
- Several properties of the constituent controls are global to all versions of the form.

Chapter 6

- All text must be typed by hand into each control on the form
- Incremental form resource files based on language cannot be edited by WinRes.exe.

WinRes.exe

This program is an adjunct to the IDE forms designer. I stated that using the forms designer would be tedious and error prone for the developer to use. This program allows that burden to be put upon the transition service. Is this better? I revisit this question at the end of the section.

WinRes.exe takes as an argument any .NET resource file except for a .txt file. The reason it does not take a text file is that this is a form resource editor. As you recall text-based resource files contain only strings. No objects are allowed. You can either supply a command argument to WinRes or you can open a resource file from within WinRes. Your choice.



NOTE *WinRes is a program that requires certain environment variables to be set if used from the command line. When in DOS you must run the Corvars.bat program located in <drive>:\ProgramFiles\Microsoft.NET\FrameworkSDK\Bin. You can also use Windows Explorer to go to this directory and double-click directly in WinRes.exe to start it.*

Let's start with the IDE forms example you just finished. I'll do this the easy way.

1. Find WinRes.exe (in <drive>:\Program Files\Microsoft.NET\FrameworkSDK\Bin).
2. Drag a shortcut to this program on your desktop.
3. Open the folder that has your last program—the one for the IDE forms designer.
4. You should now see the forms resource files that pertain to the program. My folder looks like what you see in Figure 6-7.

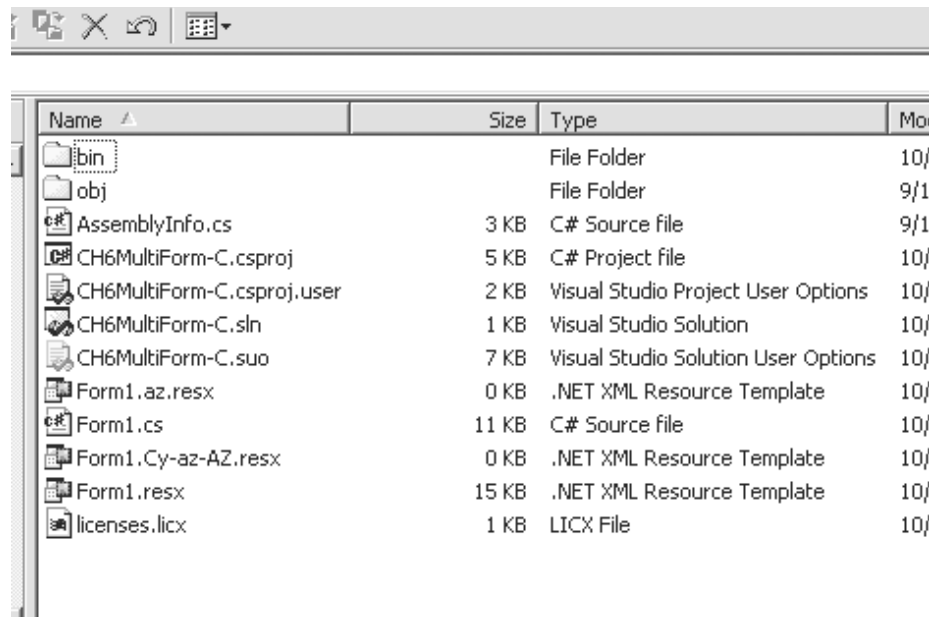


Figure 6-7. Explorer listing showing all resource file forms

Open each of the resx files in turn. First, open the Form1.resx file by dragging it on top of the shortcut to Winres program. The Winres program contains a visual designer and a properties window. Figure 6-8 shows what you should see.

Chapter 6

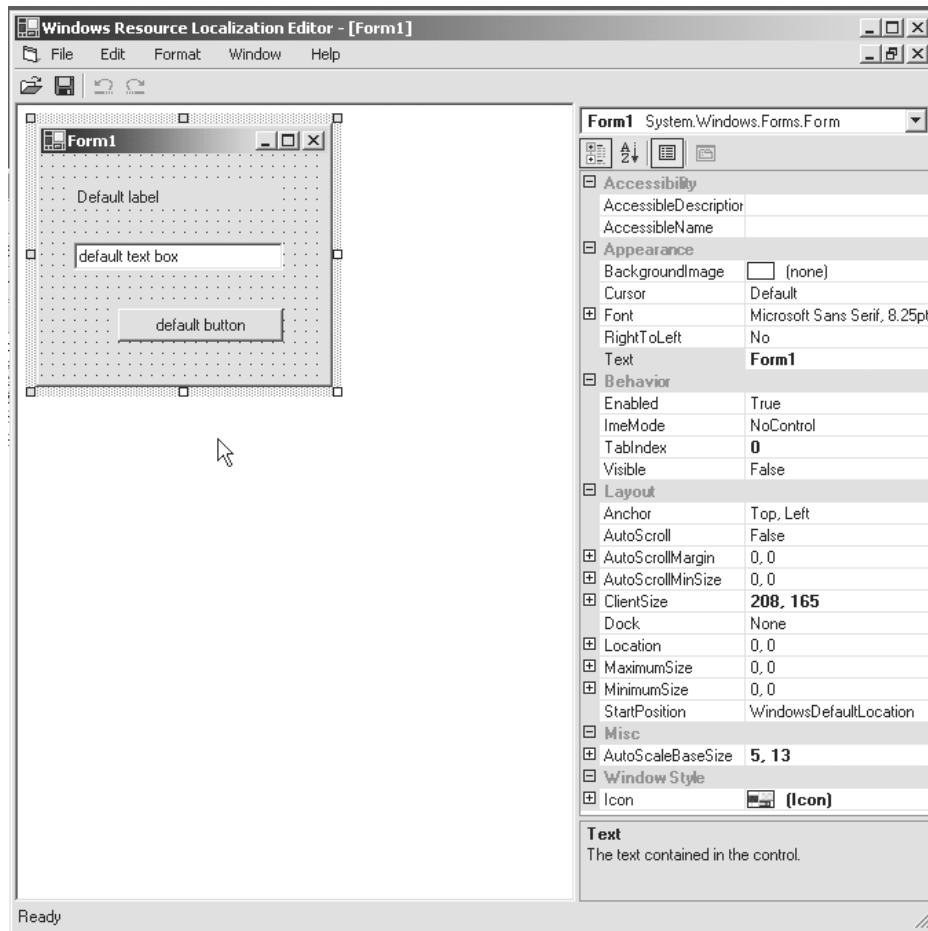


Figure 6-8. WinRes editor screen

What is most noticeable is not what is here but what is missing. Try deleting any of the components on the form. Try right-clicking any of the components. Nothing happening? That is the whole point. The WinRes editor is designed as a tool for the localizer. It is meant as an editing tool only.

The Visual Studio IDE allows you to create any number of forms with any number of controls and abstract them from your code by externalizing the parameters in an XML resource file. The WinRes.exe program allows you to edit these forms by changing certain properties without ever needing access to the original source code. It does not allow the editor to remove or add any controls.

You can think of this tool as a way to externalize what you did in the IDE forms designer example. Inside the IDE you were able to move and resize some of the controls. You were also able to change the displayed text. You are able to do the same thing with this WinRes tool.

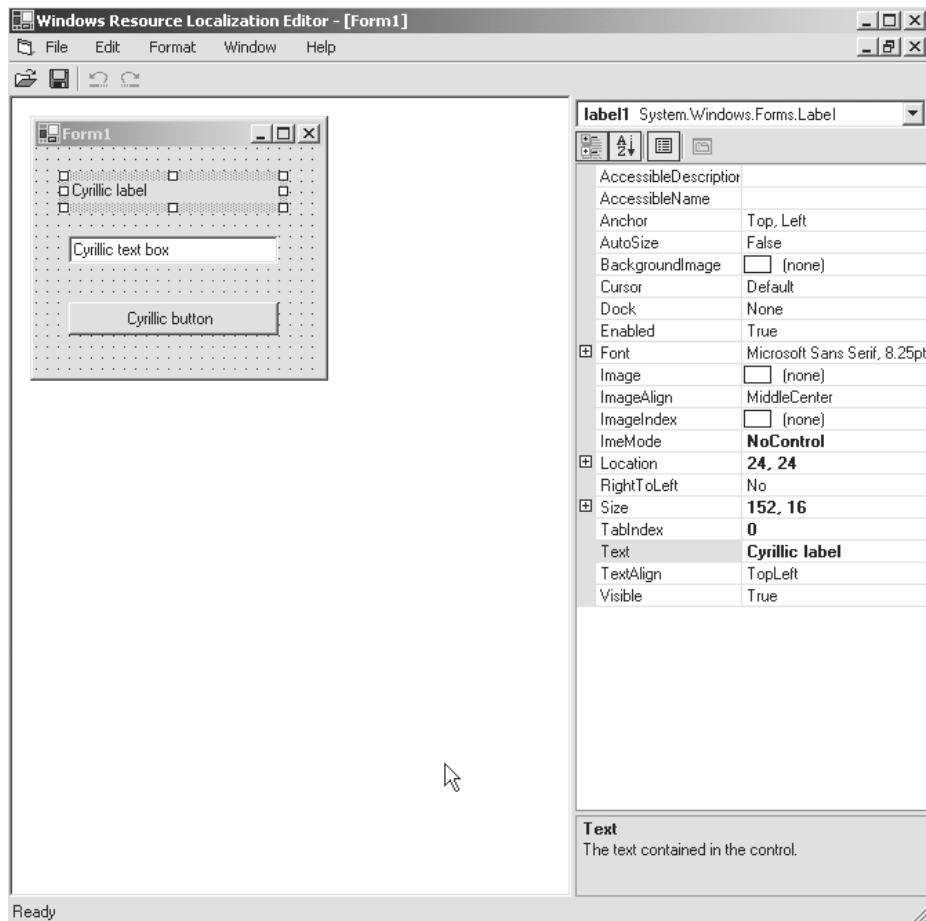


Figure 6-9. Using the WinResEditor to edit the Cyrillic form

Change the size of the button to match that of the text box and label. Now replace the text in these controls with the text we had in the Cyrillic controls from the last example. You should end up with a screen that looks like Figure 6-9.

Now save the file to a different name but in the same directory as Form1.resx. Save it as Form1.az.resx. Open this file with notepad and compare it with the Form1.resx file. You will see that they are identical in content but with some of the properties changed.

Notice that the new resource file is not an incrementally changed resource file like the Cyrillic one that was created by the IDE forms designer. You now have a twin of the original file.

OK, I know you have been dying to do this. Open the incremental form resource file that you created with the IDE forms designer. It is called Form1.az-AZ-Cyrl.resx. How does it look to you? That's right you get an error.

Chapter 6

There is not enough information in this file to create the whole form. Remember that this resource file only captures the differences between forms.

Now that we have gone through this WinRes tool you must be wondering what you can really do with it. It is not really a tool for the developer so much as it is a tool for the localization service. It allows you to make a form and have the form itself localized visually without sending out any code. While this is really neat, and it works well, it still suffers from the same problems I mentioned while working in the IDE forms designer. The problem is the tediousness of translation and inputting all the correct strings directly into the form. All you have done is transfer this problem to the translator. It does have the advantage, however, of being able to get a form back from the translation service whose controls are all sized and placed according to the language. You need to weigh the pros and cons of using this tool as opposed to using a straight string file and a resource manager.

Design Issue

Before we begin a working program I need to tell you a pet peeve of mine—a lack of good design. I began my software career writing C code for embedded systems, some that I designed even. (I started out life not as a programmer but as an electrical engineer). Anyway, I gravitated to the DOS world and have been programming in Windows since the start. I have done quite a few projects in C++ and have written several controls in ATL.

When I discovered VB at Version 5 I was amazed at how easy it was to write complex Windows code in such a short time. No longer did I need to write lots of arcane code just to have a window show up. I could concentrate on the code that actually did the work. Visual Basic was a great time saver for me and other programmers. However, like they say, what is good for the goose is good for the gander.

Visual Basic became the great enabler. All of a sudden managers found out that they need not wait for IT to assign a programmer to a task. Visual Basic was so easy to use they could get a nonprogrammer to write something up “real quick.” “Honest it’s just a demo!” “It will never make it to the public!” How often have we heard that one?

After a while, people who were not trained as programmers found they could do an OK job at it and they became VB programmers. What was missing was the thought process behind many of these programs. Design took a back seat to expediency and therefore maintenance and extendibility suffered.

I have been involved in many language retrofit projects. Some have been easy and some have taken months. It is much harder to go back and internationalize a program that was never designed for that option. Often whole screens have to be changed around. Quite a bit of code must be touched. No matter how careful you are you introduce new bugs. A program that was debugged and working perfectly could go through the localization process and wind up not working as it did before.

Upfront design is the key. As far as multilanguage programs go, do your homework first and design your program with this in mind. These days I never write a production program without designing in multilanguage capability. If Marketing says it will only be sold here it will be back in a year asking how long it will take to make it so it works in Brazil. If you have designed it correctly you can whine and say six weeks and then be the hero and do it in two days. If you have done it right you may not even have to recompile the code!

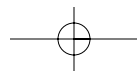
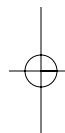
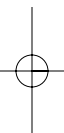
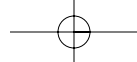
.NET was designed up front to make localization easy and complete. Make sure your design takes advantage of this work.

Summary

Here you went over the tools provided by .NET to manage resource files internally and externally to the IDE. These tools do not require writing any code. These tools are:

- XML Designer in Visual Studio. This tool allows you to generate and edit a string resource file for your program. The output of this tool is a binary .resources file. Objects such as pictures are not allowed in this tool.
- ResGen.exe. This tool allows you to convert a resource file between any of its three forms. The forms are .txt, .resx, and .resources.
- Al.exe. This tool compiles binary resource files into satellite assemblies.
- CSC.exe. This tool compiles .NET code into executables and also embeds a resource file into that executable.
- IDE forms designer. This tool allows the programmer to edit a program's form of visual localization and generate new forms for new cultures. This is a form of visual localization. He or she can then type in new translated text in the different forms and they will be displayed according to the current culture.
- WinRes.exe. This is a tool for the localizer. A programmer can create a Windows Form and send the resulting XML form resource file to a localizer. The localizer can translate text and rearrange controls on the forms to conform to the localized text.

Next I take you through an example that uses all the knowledge gained so far to make a localized resource editor. Chapter 7 is dedicated to making the editor and Chapter 8 is dedicated to localizing it.





<http://www.springer.com/978-1-59059-002-7>

Internationalization and Localization Using Microsoft
.NET

Symmonds, N.

2002, XIX, 352 p. 52 illus., Softcover

ISBN: 978-1-59059-002-7

A product of Apress