Using Resources in Visual C#.NET Applications

Conf. Marian DÂRDALĂ PhD, lect. Adriana REVEIU
Academy of Economic Studies, Bucharest

This paper presents many aspects regarding embedding resources in applications. There are analyzed *the main advantages of using resources in software applications. We introduce several ways to build and explore the application's resources. Embedding software resources in application is helpful for distributing purposes. This technique allows protecting software resources for unauthorized access.*
Keywords: *resources, assembly, image, embedding resources.*

1 Introduction

Windows applications developed using Visual C++ 6.0 software environment use a lot of software resources. They help us to describe dialog boxes, to store bitmaps like user images, icons and cursors, strings and so on. Resources are separately compiled in a project against the C++ source code and then all of them are linkedited to obtain the application. The .NET Framework also provides support for resources creation and localization.

A resource can be defined like a nonexecutable data that is logically deployed with an application. A resource might be displayed in an application like messages or like part of the user interface. Resources can contain data in a various forms: strings, images, and another kind of objects. If we store data in a resource file we can change them without recompiling the entire application. Using an embedded resource of image type we can protect it thus it can't be changed or copied by unauthorized users.

2. Using resources directly

Working with resources in this way requires including the resource in the project directly without using a specialized file for storing resources. We have to execute the following steps to include a resource directly in the project:

- In the *Solution Explorer* window right click on the project name and *Add Existing Item*;
- Choose the type of the resource (e.g. *Image Files*) and the file that contain it; after that in the *Solution Explorer* window, the name of the file will appear like a new item;
- To embed this resource in the application, right click on this new item and select for the *Build Action* property *Embedded Resource*.

For instance we embedded at the *res* project the *pz.jpg* file following the previous steps. The goal of this application is to display in the application form the bitmap that exist like an embedded resource. To do that is necessary to follow the next steps:

- To define a *Bitmap* object reference: `Bitmap poza;`
- To obtain the names of all resources that were added to the project in a string vector (at the default assemblies):

```
string[] rs = GetType().Assembly.GetManifestResourceNames();
```

GetType() is a method that returns the type of the object (*this*) and the property *Assembly* gets the assembly that the type is declared in. For an Assembly the *GetManifestResourceNames*() method obtains the name of the all embedded resources.

- Build the *Bitmap* object (*poza*) from the resource identified by its index in a *rs* vector:

```
poza=new Bitmap(GetType().Assembly.GetManifestResourceStream(rs[0]));
```

The *GetManifestResourceStream*() method returns effectively the resource like a data stream. It's possible to load the resource using its name. The name has the structure: *the project name.the file name.the file extension*:

```
poza=new Bitmap(
        GetType().Assembly.GetManifestResourceStream("res.pz.jpg"));
```

- Display the resource, I mean the image at the original size:

```
this.CreateGraphics().DrawImage(poza,0,0,poza.Width,poza.Height);
```

3. Using resources stored in resources files

The files that contain resources are the *resx* or *resources* type. To create a resource file of the *resx* type that contain a string type resource we have to follow the next steps:

- In the *Solution Explorer* window right click on the project name and *Add New Item*;
- In the dialog box at *Categories* choose *Resource File* and at *Templates* select *Assembly Resource File* then we can update the name of the *resx* file (e.g. *rs*); after we perform this operation in the *Solution Explorer* window will appear a new item (*rs.resx*);
- To add a new resource in the *rs.resx* file right click on *rs.resx* item and choose *Open*.

After this operation on the screen will appear a form like in figure 1. A resource can be identified by a name and the programmer can associate it a value according with this name. The value represents the resource. For instance, as we can see in the figure 1 the name of the entered resource is *s1* and the value is *resursa de tip sir*.

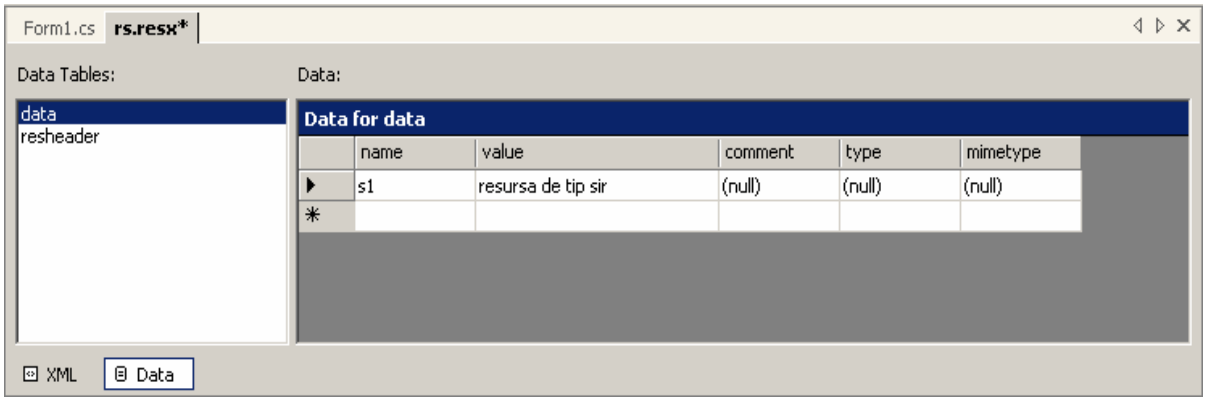

Figure 1. Form to work with resources in a resx file

In a *resx* file the data are stored in the *XML* format. To see the *resx* file data in the *XML* format click on the *XML* button.

To access the resources from a *resx* file we use the *ResXResourceReader* class. To use classes for accessing resources we have to include the next sentence: `using System.Resources;`. The next code sequence opens the *rs.resx* file, accesses each resource within it and finally the file is closed.

```
ResXResourceReader rsxr = new ResXResourceReader("rs.resx");
foreach(DictionaryEntry d in rsxr)
{
        MessageBox.Show(d.Key.ToString() + "  >>\t" +
d.Value.ToString());
}
rsxr.Close();
```

Resources are organized by the *ResXResourceReader* class like a collection of the *DictionaryEntry* elements. A *DictionaryEntry* object has two main properties: *Key* to identify the name of the resource; and *Value* to access the resource.

We can create resources files in a programmatically way. If we want to create a resource file we'll use the *ResXResourceWriter* class. In the next program sequence we'll create a new resource file named *resp.resx* then add an image resource from the *pz.jpg* file and a string resource.

```
Bitmap poza=new Bitmap("pz.jpg");
string s = "resursa sir";
ResXResourceWriter res=new ResXResourceWriter("resp.resx");
res.AddResource("p1",poza);
res.AddResource("s1",s);
res.Close();
```

The *AddResource* method adds a resource given its name and its value.
Having the resources in a *resx* file we can convert it in a *resources* file using the *ResGen* application. For instance, to convert the *resp.resx* file in a *resp.resources* we use the following command: *>ResGen resp.resx resp.resources*.

Of course we'll find in the *resp.resources* file the same resources like in the *resp.resx* file but to retrieve them we have to use the *ResourceReader* class. The following code sequence is used to retrieve a string and image resources from the *resp.resources* file:

```
ResourceReader rsxr = new ResourceReader("resp.resources");
foreach(DictionaryEntry d in rsxr)
        {
                if (d.Key.ToString() == "s1") eb.Text=d.Value.ToString();
                if (d.Key.ToString() == "p1") poza = new Bitmap((Image)d.Value);
        }
rsxr.Close();
```

The variables *poza* and *eb* are defined in this way:

```
Bitmap poza;
private System.Windows.Forms.TextBox eb;
```

4. Using resources in assemblies

Assemblies are the building blocks of the .NET Framework; they form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions.
An assembly can contain a collection of resources of various types and that are built to work together and form a logical unit of functionality. Resource assemblies are useful when we need to update resources frequently without to recompile the entire solution.
To create an assembly only with resources we have to follow the next steps:
- Create a new application with the *Empty Project* template named for example *resursa*;
- In the *Solution Explorer* window right click on the project name and choose *Add Existing Item* to add the *Resource1.resx* resource files to the project;
- Add in the *Resource1.resx* file the string type resource having the name *s1* and the value *sir resursa*;
- In the *Solution Explorer* window, right click on the project name and choose *Properties*; then for *Output Type* property select *Class Library*;
- Build the project.

Thus, the resources are compiled into the assembly and they will be packed in the *resursa.dll* file.
Resources compiled into resource assemblies can be retrieved using the *ResourceManager* class. We've just built the *resursa.dll* file that contains the resources and we'll develop a new application to retrieve them.
In first step we have to add the assembly to the new project. To do that, in the *Solution Explorer* window right click on *References*

and select *Add Reference*. Search for *resursa.dll* and add it to the project references.

To retrieve the resource that was packed in a *resursa.dll* we used the next sequence:

```
System.Reflection.Assembly ass;
ass = System.Reflection.Assembly.Load("resursa");
ResourceManager rm = new ResourceManager("resursa.Resource1", ass);
string sir;
sir = rm.GetString("s1");
MessageBox.Show(sir);
```

We create an *Assembly* object loading the *resursa* assembly. From this assembly we access the resources from *Resource1* (the name of the *resx* file from the *resursa* project) and then create a *ResourceManager* object (*rm*). With this object, using the specialized methods according to the resource type, we can retrieve the resources identified by its name (in our example *s1*).

Conclusions

Software resources are often used in professional applications. They allow programmers to build multilingual interfaces. Texts that occur in the interface are displayed in a language selected by the user when he/she installs the software package. More than this, using resources, we can manage errors more efficient; we can associate the error messages with the error codes.

Bibliography

- * * * *MSDN Library*, Microsoft Coporation, 2005;
- Richter J. *Applied Microsoft .NET Framework Programming*, Microsoft Press, 2002;
- Smeureanu I., Dârdală M., Reveiu A., *Visual C# .NET*, CISON, Bucureşti, 2004;
- Wigley A., Wheelwright S., *Microsoft .NET Compact Framework*, Microsoft Press, 2003;