How to create and use HTML resource files

Contents

Introduction

Article #2 noted the benefits of embedding data within your program's executable file or within a DLL. Often this is preferable to supplying a lot of separate support files that the user can accidentally alter or delete. The same principle applies to HTML and related files displayed by your application in a *TWebBrowser* control.

In fact Microsoft® have encouraged this by creating the `res://` protocol. This is used by *TWebBrowser* and Internet Explorer to access HTML etc. files that are stored in a module's resources. This protocol makes it very easy to navigate to HTML resources using the *TWebBrowser* control.

No query strings

The `res://` protocol doesn't interpret query strings that are appended to the URL. Pity!

Accessing HTML resources from the browser

When displaying data from embedded resources we usually have to do some work – we have to read the data out of resources and display it in a suitable control (see *article #3*). However, with *TWebBrowser*, Microsoft® has made things easier for us. All we have to do is specify where to find the resource by using a special `res://` protocol URL. The browser will extract the data from the resource and display it as it would a conventional file.

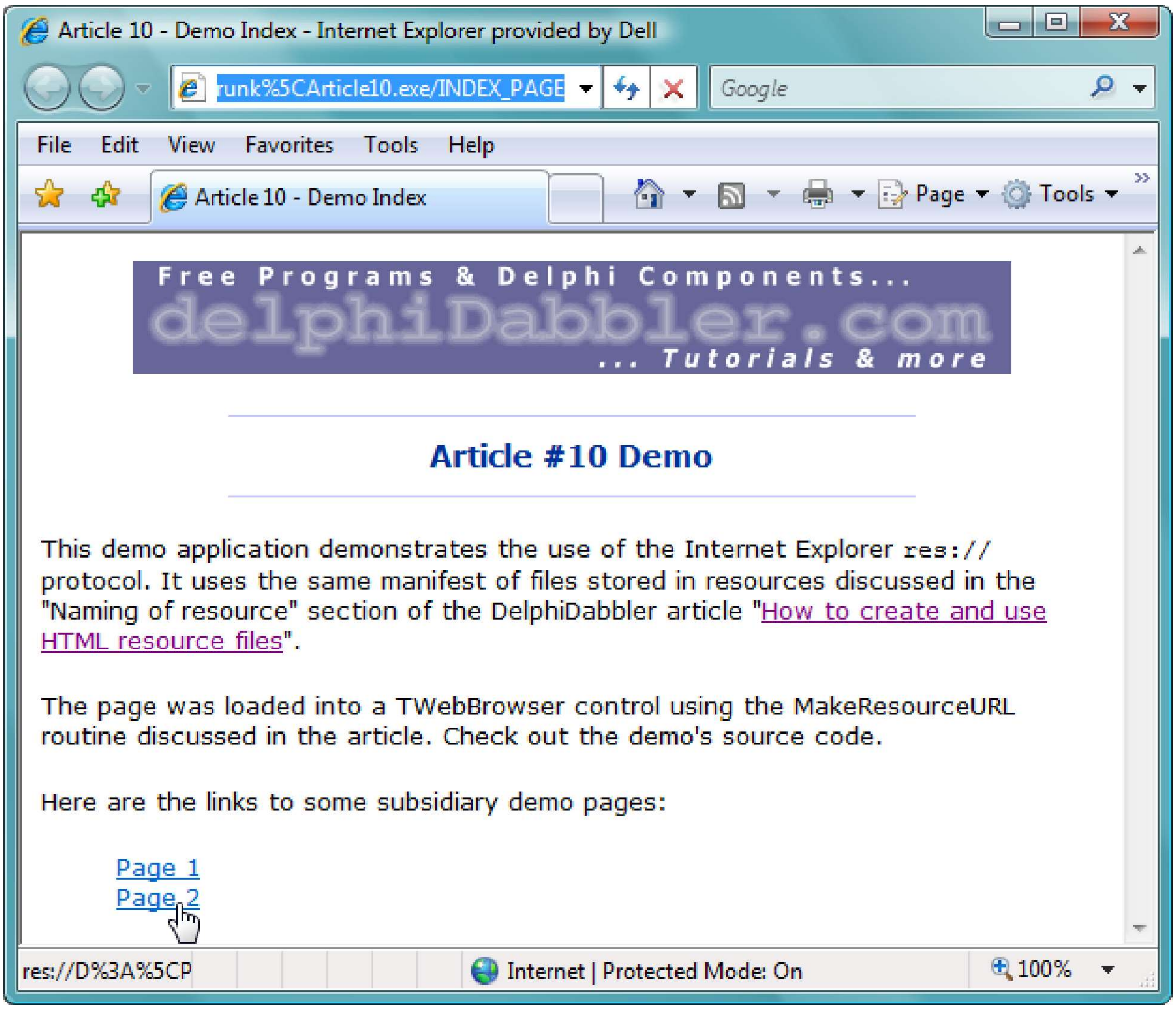
This approach works in both the *TWebBrowser* control and by typing the `res://` protocol URLs directly into the *Internet Explorer* location bar.

IE based browsers

Browsers built around the IE web browser control may also support the `res://`

protocol.

By way of an example, the following screenshot shows Internet Explorer displaying one of the web pages included in the resources of this article's demo program.


Article 10 - Demo Index - Internet Explorer provided by Dell
runk%5CArticle10.exe/INDEX_PAGE
Google
File Edit View Favorites Tools Help
Article 10 - Demo Index
Page Tools
Free Programs & Delphi Components...
delphiDabbler.com
... Tutorials & more
res://D%3A%5CP
Internet | Protected Mode: On
100%

Article #10 Demo

This demo application demonstrates the use of the Internet Explorer res:// protocol. It uses the same manifest of files stored in resources discussed in the "Naming of resource" section of the DelphiDabbler article "How to create and use HTML resource files".

The page was loaded into a TWebBrowser control using the MakeResourceURL routine discussed in the article. Check out the demo's source code.

Here are the links to some subsidiary demo pages:

Page 1
Page 2

To replicate this you must first compile the demo program then enter the following URL in *Internet Explorer*'s location bar:

```
res://[Path to Demo]\Article10.exe/INDEX_PAGE
```

Where `[Path to Demo]` is the full path to the demo program. Strictly speaking you should "escape" any backslash, space, hash ("#") and colon characters in the path – see *Making res:// protocol URLs safe* below for details. If you copy the URL from the demo program it will already be escaped.

What is more, if a document is loaded from a `res://` URL, that URL becomes the browser's base URL. This means that any relative links in the document refer back into the resources. We can get the same effect by setting a document's `<base>` address to a `res://` URL.

Understanding the res:// protocol

Overview

Valid `res://` protocol URLs have the form:

*res://**module**[/**restype**]/**resid***

where the segments of the URL have the following meanings:

module

The name of the module (DLL or executable program) containing the resource. The full path to the module is required, unless it is on the search path.

restype

An optional segment that specifies the resource type identifier. If the segment is omitted the resource type is assumed to be *RT_HTML*.

resid

The resource name identifier.

About resource identifiers

An attribute of Windows® resource names and resource types is that they can either be identified by a character string or by a numeric value.

When specifying resource identifiers in URLs we must obviously represent both kinds as strings, but we must also be able to distinguish between the two different kinds. Numeric identifiers are flagged by prepending a "#" character to the decimal value of the identifier. For example the *RT_HTML* resource type would be represented as "#23". String identifiers are represented literally.

Making res:// protocol URLs safe

We have to be careful when entering a `res://` protocol URL into the browser bar, or when embedding the URL as an attribute of a HTML tag such as the `<a>` tag. The URL must not include "special characters". Any such characters need to be replaced by URL escape characters.

URL escape characters

These characters have the form %HH where HH is the character code in hexadecimal.

In particular, since our `res://` protocol URL main contain a path to the module that holds the required resource, we need to escape any back-slashes and spaces in the path. Additionally, since numeric resource identifiers begin with a "#" character, and these have special meaning in a URL, it may be wise to escape these too.

For example this URL:

```
res://C:\Program Files\MyModule.dll/#42
```

would be escaped as:

```
res://C%3A%5CProgram%20Files%5CMyModule.dll/%2342
```

The escape codes used are: %3A (":"), %5C ("\"), %20 (space) and %23 ("#").

Generating res:// protocol URLs from Delphi

Overview

When developing an application that uses the web browser control, we may need to build `res://` protocol URLs programatically. In this section we present the code to do this.

Resource identifiers

Recall that resource names and resource types can be either strings or have a numeric value. In code both kinds of resource identifier are represented by a *PChar* value. When the identifier is a string the *PChar* value points to the required string of characters. For numeric identifiers the high order word of the pointer is zero and the low order word contains the value. In a URL we know that the numeric indentifier is a series of digits preceded by a "#" symbol.

Our job is to develop a Delphi function that transforms a resource identifier referenced by a *PChar* into an appropriately formatted string for use in a URL. *Listing 1* shows a solution:

```
function FormatResNameOrType(ResID: PChar): string;
begin
  if HiWord(LongWord(ResID)) = 0 then
    // high word = 0 => numeric resource id
    // numeric value is stored in low word
    Result := Format('#%d', [LoWord(LongWord(ResID))])
  else
    // high word <> 0 => string value
    // PChar is implicitly converted to string
    Result := ResID;
end;
```

Listing 1

We simply test the resource identifier pointer's high word to see if it is zero. If so we return the value of the low word preceeded by "#". Otherwise we return the resource identifier unchanged. Delphi silently converts the result to the compiler's default string type.

Using *PChar*

PChar works here on any version of Delphi, regardless of whether Delphi defines it as *PAnsiChar* or *PWideChar*. This is because *FormatResNameOrType* checks the pointer itself, not the character it points to.

Don't try to use the wide char version in non Unicode Delphis and vice versa – always use the native *PChar*.

URL encoding

Earlier we discussed the need to escape special characters in a `res://` protocol URL. The simple function presented in *Listing 2* takes the safest possible approach and escapes all characters that are neither alphanumeric nor one of "-", "_" or ".". Conditional compilation is used to test set membership correctly depending on if an ANSI or Unicode version of Delphi is being used.

Library routine

A more flexible version of this routine see *URLEncode* in the *Code Snippets Database*.

```
function URLEncode(const S: string): string;
var
  Idx: Integer; // loops thru characters in string
begin
  Result := '';
  for Idx := 1 to Length(S) do
  begin
    {$IFDEF UNICODE}
    if CharInSet(S[Idx], ['A'..'Z', 'a'..'z', '0'..'9', '-', '_', '.']) then
    {$ELSE}
    if S[Idx] in ['A'..'Z', 'a'..'z', '0'..'9', '-', '_', '.'] then
    {$ENDIF}
      Result := Result + S[Idx]
    else
      Result := Result + '%' + IntToHex(Ord(S[Idx]), 2);
  end;
end;
```

Listing 2

Building the URL

We now move on to take a look at two overloaded functions that can create a `res://` protocol URL.

Both functions take a reference to a module, a resource name and an optional resource type as parameters. If the resource type parameter is omitted, or is nil, it is not included in the URL and a resource type of *RT_HTML* will be assumed.

The routines differ in how they handle the module component of the URL. The first function (*Listing 3*) is passed the module file name as a string while the second routine (*Listing 4*) takes a handle to the required module. First we will consider the function that accepts the module name as a string:

```
function MakeResourceURL(const ModuleName: string; const ResName: PChar;
  const ResType: PChar = nil): string; overload;
```

```
begin
  Assert(ModuleName <> '');
  Assert(Assigned(ResName));
  Result := 'res://' + URLEncode(ModuleName);
  if Assigned(ResType) then
    Result := Result + '/' + URLEncode(FormatResNameOrType(ResType));
  Result := Result + '/' + URLEncode(FormatResNameOrType(ResName));
end;
```

Listing 3

This function records the protocol name and appends the name of the module to it. It then appends the resource type if specified and finally adds the resource name. URL segments are separated by slash characters. Notice how we call *FormatResNameOrType* to get the resource type and resource name identifiers and we use *URLEncode* to ensure each segment of the URL is URL safe.

The second form of the function is for use with loaded modules. Such modules are indentified by means of their module handle rather than by name, as shown in *Listing 4*:

```
function MakeResourceURL(const Module: HMODULE; const ResName: PChar;
  const ResType: PChar = nil): string; overload;
begin
  Result := MakeResourceURL(GetModuleName(Module), ResName, ResType);
end;
```

Listing 4

This function simply gets the name of the module by calling the *SysUtils* unit's *GetModuleName* function. It then calls the other overloaded function, passing it the module name.

Defining RT_HTML

RT_HTML is a constant that denotes an HTML resource. Unlike other *RT_** constants it is not defined in Delphi 7. If needed, the constant should be defined as follows:

```
const
  RT_HTML = MakeIntResource(23);
```

Listing 5

How to create HTML resources

Creating resource file source code

Although you can embed and access HTML and related files as RCDATA or any other user-defined resource type, it is recommended that they are stored as resource type #23 (*RT_HTML*). Any type of file that can be displayed in a browser can legitimately be included under this resource type. If you have access to a resource compiler, the easiest way to create a resource file containing HTML resources is as follows:

1. **Assemble the required assets**

   Gather together all the files you wish to include in the resource file in a convenient location.

2. **Create a resource source file**

   Next we create the resource source (`.rc`) file that, when compiled, embeds the specified files in the binary resource file. Here is an example source file:

   ```
   /* standard resource # for HTML */
   #define HTML 23

   /* include all resource from external files */
   TIPS_HTML       HTML    "tips.html"
   HELP_HTML       HTML    "help.html"
   LEFTARROW_GIF   HTML    "left-arrow.gif"
   RIGHTARROW_GIF  HTML    "right-arrow.gif"
   FILL_GIF        HTML    "fill.gif"
   HELP_GIF        HTML    "help.gif"
   ```

```
TIPS_JS         HTML    "tips.js"
STYLE_CSS       HTML    "style.css"
```

Listing 6

3. **Compile the source file**

   Use a resource compiler such as Borland's BRCC32 to compile the source file. The following command line instructs BRCC32 to compile a source file, `HTML.rc`, into a binary resource file named `HTML.res`:

   ```
   BRCC32 -fo"HTML.res" HTML.rc
   ```

   We assume that the source file is in the current directory and the compiler is on the path. If you don't have a suitable resource compiler then `.res` files can be generated programmatically – *article #2* explains – or you can use my open source *HTML resource file compiler*.

4. **Link the resource file into your application or DLL.**

   In Delphi we link the resource by adding the following line to a project file or to a convenient unit. For the example given above we would use:

   ```
   {$R HTML.res}
   ```

   Listing 7

Using better resource names

You may find it helpful to name resources with file-like names. Doing this will make it easier to migrate an existing series of interlinked HTML files to resources (providing the files are all in the same directory). Why? because the existing local links in the file will still work after the move!

Windows does it!

Several HTML resources in Windows DLLs are named in this way.

For example, suppose you have an application that displays local files from a single directory in a *TWebBrowser* control. Assume the directory contains the following files:

- `index.html`
- `page1.html`
- `page2.html`
- `logo.gif`

Each HTML file displays the logo and so contains an `<img>` tag such as:

```
...
<img src="logo.gif" ... />
...
```

Listing 8

Additionally, `index.html` contains a relative link to both `page1.html` and `page2.html` thus:

```
...
<a href="page1.html">Page 1</a><br />
<a href="page2.html">Page 2</a>
...
```

Listing 9

You now decide to store the pages within your program's HTML resources and display them using the `res://` protocol. Suppose you decided to give an arbitrary name to each resource as follows:

```
INDEX_PAGE  23   "index.html"
PAGE_1      23   "page1.html"
PAGE_2      23   "page2.html"
LOGO_GIF    23   "logo.gif"
```

Listing 10

To get your program working again you will need to track down every link and change it to reference the new resource name rather than the old file name. For example, each of the logo `<img>` tags will have to be changed to:

```
...
<img src="LOGO_GIF" ... />
...
```

Listing 11

This is obviously prone to error. A far better way would be to name the resources with the same name as the original file, i.e.:

```
index.html  23   "index.html"
page1.html  23   "page1.html"
page2.html  23   "page2.html"
logo.gif    23   "logo.gif"
```

Listing 12

Using this scheme all the original links will continue to work unchanged. Of course this technique only works if all the files are in the same directory.

Problems with BRCC32

There's always a gotcha somewhere isn't there? The BRCC32 resource compiler that ships with Delphi refuses to compile any resource file that has dots (i.e. full stops, periods) in any of its resource names. This means that the naming scheme proposed above is useless for those of us who use BRCC32.

To get round the above problem I have written *HTML resource compiler*, a command line application that can compile HTML resource files. This compiler automatically names each resource after its associated file.

I have never used the Microsoft® RC compiler so can't comment on whether or not it will work in this situation. If you know, please *let me know* too!

Conclusion

In this article we have investigated how to use the Internet Explorer (and hence *TWebBrowser*) specific `res://` protocol to view HTML content stored in a module's resources. We first looked at how to access a HTML resource from Internet Explorer. We then examined the component parts of a `res://` protocol URL before showing how to generate such URLs from Delphi code. The article ended by demonstrating how to create HTML resource files and observed that naming HTML resources after the original file can at times be useful.

I hope you found this article useful. If you have any comments or if you've found an error, please *contact me*.

Demo programs

A demo program that can be used to test and exercise the code presented here is available for download. It stores a series of HTML and other files in its resources and displays them using an embedded *TWebBrowser* control. The URLs the program is using to load each page are also displayed.

Download the demo code

If you need further demo code, the *HTML Resource Compiler* comes with sample HTML files and a Delphi project to create a DLL containing only *RT_HTML* resources. These resources can be displayed in Internet Explorer by using the `res://` protocol.

Bibliography

Various resources on the *Microsoft Developer Network* were used in researching this article. Unfortunately the original topics have now been removed so no references can be given.

The following articles on *DelphiDabbler.com* deal with resources in general:

- *How to store files inside an executable program*.
- *How to read data embedded in your program's resources*.