Beating the System: Add Extra Richness To RichEdit Controls!

by Dave Jewell

Isn't it funny how one thing always leads to another? No, the wife hasn't got me decorating again, I'm talking about how this month's column evolved.

My original intention was to put together a grab-bag of assorted Delphi programming hints and tips. However, part way through this I discovered something rather interesting that related to the rich text edit control. I couldn't resist pursuing this further and one thing just led to another...

Introducing RichEdit Version 2.0

If you've done much Delphi programming, you'll probably have come across the rich edit (`TRichEdit`) control. This component lives on the `Win32` page of the component palette in Delphi 3 and it is effectively a VCL wrapper around the underlying Windows rich text control. Using this control, you can create your own mini word-processor, with different parts of the text being shown with different fonts, different attributes (bold, italic, etc), different colours and so forth. It isn't limited to 64Kb of text and, overall, the capabilities of the rich text control are quite extensive. It's used as the basis for the WordPad application which ships with Windows 95.

However, having said all this, the fact is that Delphi programmers are being short changed in terms of what we get with Delphi 3. Let me explain. When Microsoft introduced NT 4.0, they brought out a new enhanced implementation of the rich edit control, called version 2.0. Even though Delphi 3 came out some considerable time after rich edit 2.0, support for the new rich edit code was never incorporated into `TRichEdit`. Perhaps one of the reasons for this is the uncertainty over whether or not it's installed on a particular machine. Like I said, version 2.0 of the control ships with NT 4.0 and will also be included with NT 5.0. It was never officially included with Windows 95, but will undoubtedly form a part of the Windows 98 shrink-wrap. The rich edit 2.0 code *might* be present on a Windows 95 system if some application that uses it has already been installed.

From this, you can see that we can't make any assumptions about whether or not rich edit 2.0 support is present. If you develop a Delphi application which makes use of the component described in this month's column, or is otherwise dependent on rich edit 2.0 functionality, then you must include the necessary redistributable Microsoft DLL along with your program.

Specifically what files am I talking about? If you look in your \WINDOWS\SYSTEM directory, you should find a file called RICHED.DLL. This is the old 16-bit implementation of rich edit, for our purposes it can be ignored. You should also see a file called RICHED32.DLL. This is the 32-bit version 1.0 implementation and it should be present on all 32-bit platforms. If you've got rich edit 2.0 installed on your system you will also find a file called RICHED20.DLL. This is the new, 32-bit 2.0 code and it's this file (305Kb under Windows 98 beta 3) that you will need to distribute when shipping an application that requires rich edit 2.0 functionality.

See the notes at the end about distribution issues.

What's New In Version 2.0?

So what exactly does rich edit 2.0 bring to the party? What am I making all the fuss about? Rich edit 2.0 includes a number of interesting new features such as a multi-level undo/redo buffer, full support for Unicode and Far Eastern languages and a number of assorted user interface improvements.

For example, if you use Office 97, you've probably noticed that, in Word 97, you can type a URL into a document and Word will automatically recognise it as such. Word turns it into a hyperlink, displaying it underlined in blue. When you click on the hyperlink, Internet Explorer is automatically invoked with the target URL. Similar things happen in other Microsoft applications such as Outlook Express. As far as I know, Word 97 isn't actually based on the rich edit control, but rich edit 2.0 does offer very similar functionality in terms of automatic URL detection and highlighting. This is obviously a nice feature to build into your own applications, particularly with today's heavy emphasis on internet interoperability.

So how do we get a rich edit 2.0 control into a Delphi application? This is where the fun starts! My hope was that Borland's VCL wrapper code would automatically detect the presence of the version 2.0 DLL and use it if present. This turns out not to be the case. My next approach was to try renaming the RICHED20.DLL file to RICHED32.DLL, thus forcing the VCL library to use the newer control. Again, this didn't work and it wasn't a terribly good idea anyway, since it might cause side effects with other software. In an attempt to get to the bottom of the problem, I examined the source code for Delphi's `TRichEdit` control and I also disassembled the new RICHED20.DLL *[we didn't hear you say that, Dave... Ed]*. Lastly, I checked through the MSDN programming information relating to the version 2.0 control which, after something over two years in the field, is still marked as 'preliminary'.

Microsoft Strikes Again... Groan....

It wasn't difficult to see what the problem was. Aside from the change to the name of the DLL, which I've already mentioned, Microsoft have also changed the underlying API-level window class name that's associated with the rich edit control.

You'll probably appreciate that VCL push button controls, for example, map down onto a Windows `BUTTON` control, Delphi listboxes map down onto a `LISTBOX` control, and so on. The Delphi implementation of `TRichEdit` maps down onto a control called `RICHEDIT`. However, with the release of rich edit version 2.0, Microsoft changed the underlying class name to `RICHEDIT20A`. Well thank you, Microsoft...

If you're getting fed up of me having a dig at Microsoft on a monthly basis, please be assured that I'm even more irritated by it than you are. You might be forgiven for thinking that I deliberately set out to find some nook or cranny of Windows where Microsoft have committed some almighty gaff. Please be assured that this isn't the case. It's just that, no matter where I turn, I invariably seem to end up putting my foot in something that I really wish hadn't ended up on the sole of my shoe!

So what's my grouch this time? Simply that the new version 2.0 control should have been implemented as a super set of the existing control. Rather than introducing a new DLL and a new class name, the new version 2.0 functionality should have been seamlessly rolled into the existing `RICHED32.DLL` library and `RICHEDIT` class name. Any new functionality should have been implemented purely through new `EM_XXXX` edit messages, new notification messages and new style bits when the window is created. By doing things in this way, older client applications would have been able to use the new control without any coding changes while newer client software could have used a special message as a version number check to determine what flavour of the control they were dealing with. Sometimes, I really do wonder what passes for grey matter in Redmond...

Anyway, for better or worse, we're saddled with the current state of play. If you open the VCL source file COMCTRLS.PAS, and search for the `TCustomRichEdit.CreateParams` routine, you'll appreciate the problem. Firstly, Delphi's code tries to load the `RICHED32` library. The assumption is that, by loading the library into memory, it will, in turn, register the `RICHEDIT` control class. Next, the `CreateSubClass` routine is called, which causes the `TCustomRichEdit` control

➤ *Listing 1*

```
unit RichEdit2;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ComCtrls, RichEdit;
type
  TRichEdit2 = class (TCustomRichEdit)
  private
    fLibHandle: THandle;
    procedure WMNCDestroy (var Message: TWMNCDestroy);
      message wm_NCDestroy;
  protected
    procedure CreateParams (var Params: TCreateParams);
      override;
  public
  published
    property Align;
    property Alignment;
    property BorderStyle;
    property Color;
    property Ctl3D;
    property DragCursor;
    property DragMode;
    property Enabled;
    property Font;
    property HideSelection;
    property HideScrollBars;
    property ImeMode;
    property ImeName;
    property Lines;
    property MaxLength;
    property ParentColor;
    property ParentCtl3D;
    property ParentFont;
    property ParentShowHint;
    property PlainText;
    property PopupMenu;
    property ReadOnly;
    property ScrollBars;
    property ShowHint;
    property TabOrder;
    property TabStop default True;
    property Visible;
    property WantTabs;
    property WantReturns;
    property WordWrap;
    property OnChange;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnEnter;
    property OnExit;
    property OnKeyDown;
    property OnKeyPress;
    property OnKeyUp;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnResizeRequest;
    property OnSelectionChange;
    property OnStartDrag;
    property OnProtectChange;
    property OnSaveClipboard;
  end;
procedure Register;
implementation
{$R *.DCR}
procedure TRichEdit2.CreateParams(
  var Params: TCreateParams);
const
  HideScrollBars: array[Boolean] of Longint =
    (ES_DISABLENOSCROLL, 0);
  HideSelections: array[Boolean] of Longint =
    (ES_NOHIDESEL, 0);
var
  OldError: Longint;
begin
  OldError := SetErrorMode (sem_NoOpenFileErrorBox);
  fLibHandle := LoadLibrary ('RICHED20.DLL');
  SetErrorMode (OldError);
  if (fLibHandle > 0) and
    (fLibHandle < hInstance_Error) then
    fLibHandle := 0;
  inherited CreateParams (Params);
  if fLibHandle <> 0 then
    CreateSubClass (Params, 'RICHEDIT20A')
  else
    CreateSubClass (Params, 'RICHEDIT');
  with Params do begin
    Style := Style or
      HideScrollBars[Inherited HideScrollBars] or
      HideSelections[HideSelection];
    WindowClass.style :=
      WindowClass.style and not (cs_HRedraw or cs_VRedraw);
  end;
end;
procedure TRichEdit2.WMNCDestroy(var Message: TWMNCDestroy);
begin
  Inherited;
  if fLibHandle <> 0 then
    FreeLibrary (fLibHandle);
end;
procedure Register;
begin
  RegisterComponents('XFactor', [TRichEdit2]);
end;
end.
```

to be based on the underlying (and now registered) RICHEDIT class.

If we were to force the CreateParams code to use the RICHED20 library, it wouldn't make a blind bit of difference because the CreateSubClass routine would still search for an API-level class called RICHEDIT20A and use this as a basis for the new control. It's for this reason that simply renaming the name of the DLL won't work.

Delphi To The Rescue...

In this sort of situation, the Delphi approach is to derive a new control and modify the descendant's behaviour as required. That's exactly what I've done in Listing 1. Here, you can see that I've derived a new control, TRichEdit2, from the existing TCustomRichEdit control. The new class overrides two methods of its parent class, CreateParams and WMNCDestroy. The CreateParams code attempts to load the newer, RICHED20 library and stores the library handle in fLibHandle, a private variable which shouldn't be confused with the variable of the same name in the parent class. If it successfully loaded the library, then it calls CreateSubClass with a class name of RICHEDIT20A, causing the new control to subclass the new version 2.0 rich edit control. If the library wasn't loaded for any reason, then it defaults to using the RICHEDIT class name.

It's worth pointing out that because the Inherited CreateParams handler is called, the RICHED32 library will be loaded into memory even if we only make use of the new RICHED20 library. In practice, this isn't likely to cause any sort of problem. The only reason that both CreateParams handlers call LoadLibrary is to guarantee that the target class name is registered before the call to CreateSubClass.

Introducing URL Automation!

OK, so we've got a shiny new version of the rich edit control. What can we do with it? As I mentioned earlier, one of the cute capabilities of rich edit 2.0 is the ability to automatically recognise URL hyperlinks in text. According to the (still!) preliminary documentation, all the URL formats shown in Listing 2 are auto detected.

As you can see, this isn't *quite* as nice as the URL recognition that's built into Word 97. With Word, you can type something like mailto:Humpty@Dumpty.com and it will be converted into a clickable link. You can likewise type something like www.wombat.com and the initial http:// part of the URL will be assumed. Neither of these examples would be recognised by the auto URL detection built into rich edit 2.0. You're undoubtedly hoping that the back-room boys at Microsoft have provided a facility for defining your own auto URL recognition templates? That would be nice, but no such luck. Maybe in version 3.0.

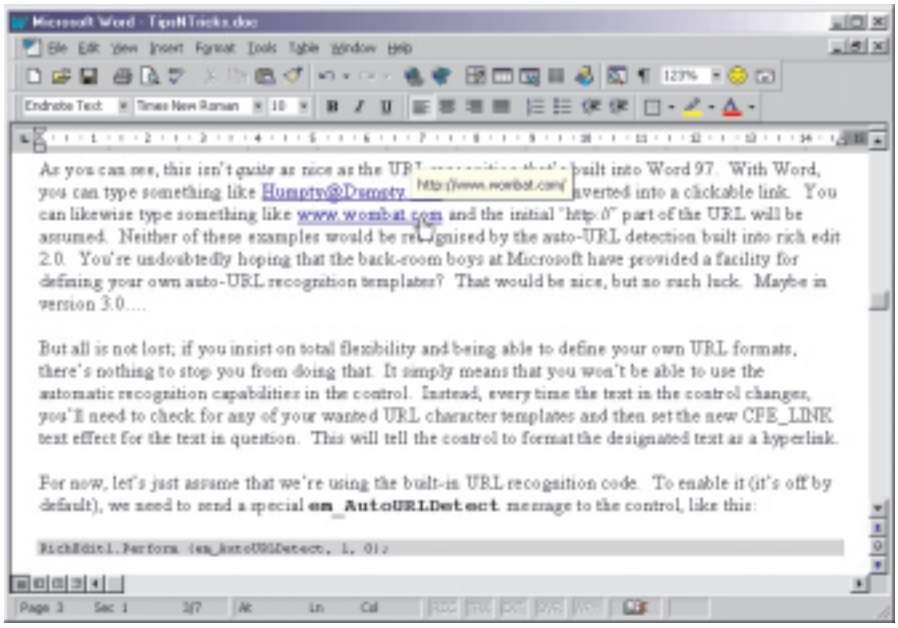
But all is not lost: if you insist on total flexibility and being able to define your own URL formats, there's nothing to stop you from doing that. It simply means that you won't be able to use the automatic recognition capabilities in the control. Instead, every time the text in the control changes, you'll need to check for any of your wanted URL character templates and then set the new CFE_LINK text effect for the text in question. This will tell the control to format the designated text as a hyperlink.

For now, let's just assume that we're using the built-in URL recognition code. To enable it (it's off by default), we need to send a special em_AutoURLDetect message to the control, like this:

```
RichEdit1.Perform(
  em_AutoURLDetect, 1, 0);
```

Once done, you'll find that entering a URL which matches the above format will cause the text to be displayed in blue and underlined, just as in Word. However, moving the mouse cursor over the text is ignored and, when the highlighted text is clicked, nothing special happens. In order to fully exploit the new URL capabilities of the rich

➤ *Figure 1: As you will probably know, Word 97 (and other Office applications) can automatically detect URL references and convert them into hyperlinks. In this month's column, Dave shows you how to achieve the same effect using Version 2.0 of Microsoft's rich text edit control*




```
http:<text with no whitespace>
file:<text with no whitespace>
mailto:<text with no whitespace>
ftp:<text with no whitespace>
https:<text with no whitespace>
gopher:<text with no whitespace>
nntp:<text with no whitespace>
prospero:<text with no whitespace>
telnet:<text with no whitespace>
news:<text with no whitespace>
wais:<text with no whitespace>
```

➤ *Listing 2*

```
procedure TForm1.WMNotify (var Message: TWMNotify);
begin
  if Message.NMHdr^.hwndFrom = RichEdit1.Handle then case Message.NMHdr^.code of
    en_Link : URLLinkNotification (Message.NMHdr);
    // Add other notification types here....
  end;
end;
procedure TForm1.URLLinkNotification (Link: Pointer);
type
  // Need to redefine this - RICHTEXT.PAS gets it wrong!
  TTextRange = record
    chrg: TCharRange;
    lpstrText: PAnsiChar;
  end;
var
  sz: String;
  TextRange: TTextRange;
  pENLink: ^TENLink absolute Link;
begin
  with pENLink^ do begin
    SetLength (sz, chrg.cpMax - chrg.cpMin);
    TextRange.chrg := chrg;
    TextRange.lpstrText := Pointer (sz);
    RichEdit1.Perform (em_GetTextRange, 0, Integer (@TextRange));
    if Msg = wm_MouseMove then
      RzStatusPane1.Caption := sz
    else if Msg = wm_LButtonDown then
      ShellExecute (Handle, 'open', PChar (sz), Nil, Nil, sw_Show);
  end;
end;
```

➤ *Listing 3*

edit 2.0 control, we have to enable link notification messages. That's done like this:

```
mask := RichEdit1.Perform(
  em_GetEventMask, 0, 0);
mask := mask or enm_Link;
RichEdit1.Perform(
  em_SetEventMask, 0, mask);
```

Here, `mask` is a scratch variable that's used to get the current event mask bits, add in the `enm_Link` flag and then set this as the new event mask. Once link notifications are enabled, the control will start sending `wm_Notify` messages to the parent window (in our case, the Delphi form) any time that the mouse moves over a hyperlink text area, is clicked on a hyperlink text area, or whatever. To receive those `wm_Notify` messages, we need to add a message handler to the class declaration:

```
procedure WMNotify(
  var Message: TWMNotify);
  message wm_Notify;
```

You'll also notice that once link notifications are enabled, the rich edit 2.0 control automatically displays a 'hand' cursor as the mouse moves over a hyperlink. I expected to have to implement the cursor-changing code myself but the control will take care of it for you. Wow, maybe there is intelligent life in Redmond after all!

The code fragment in Listing 3 shows how to make use of these link notification messages. This code is taken from a working program, the complete source code to which is included on this month's cover disk. Firstly, all `wm_Notify` messages received by the form arrive at the `WMNotify` method. In a complex program, you'll typically have several different types of notification arriving from different controls. You may even have more than one rich edit control on a single form. Consequently, the first thing the routine does is check that the notification messages are coming from the required control. It then branches according to the actual type of notification that's received. In this case, we're interested in `en_Link` notifications. Every time one of these comes in, we call `URLLinkNotification`.

Unfortunately, even Inprise *[Doesn't trip off the tongue like 'Borland' does it? Ed]* screw up once in a while and here we have an example. The RICHEDIT.PAS file that comes with Delphi 3 includes all the constants, messages and data structures that apply to rich edit 2.0, even though Inprise haven't yet implemented support for version 2.0. If you compare the Microsoft documentation with Inprise's definition of `TTextRange`, you'll see that the second field in the data structure, `lpstrText`, is defined as type `AnsiChar` when it should actually be a pointer to an `AnsiChar`. I don't like fixing the Inprise VCL sources in case they make the same mistake in Delphi 4, which would effectively unfix my fix! In such cases, the simplest approach is to copy the wanted structure definition into the code that needs it, and implement the fix there. That's what I've done.

Inside the `URLLinkNotification` code, the received message includes a data structure that gives us the beginning and ending character position of the text which constitutes the hypertext link. By subtracting one from t'other, we know the character length of the link information, and this is used to initialise the length of a string. Next, the character position information and a pointer to the start of the string is copied into the `TextRange` variable and a `em_GetTextRange` message is issued to retrieve just the specified amount of text from the rich edit control.

At this point, the `sz` string contains the text of the hypertext link. As you'll see from the code, the 'parent message' is included as part of the notification so that we know whether the mouse was clicked, double-clicked, just moved over the link or whatever. If you've got a status bar in your application, a nice touch is to display the text of the link in the status bar as the mouse moves over it. This provides visual feedback to the user, indicating what's going to happen if the link is clicked. To do this job properly, you'd also need to implement code to clear the status bar information when the mouse isn't over a hypertext link. This is left as an exercise to the reader, but you can email me if you want some ideas!

In this particular case, left-clicking on a link causes the `ShellExecute` routine to be called with the link text as an argument. Thus, if you click on a `mailto` link, your email program (on my system, Out-Look Express) will be launched; click on a `http` link and Internet Explorer will start, click on a `Telnet` link to start the Telnet client and so

on. If you've never used `ShellExecute` before, it's deeply cool. I have got no complaints in this area, Microsoft!

Time For The Caveats...

On this month's disk I've included a stripped-down program which illustrates how to use a 'richer' text control along the lines described in this article.

Please understand that this program is most definitely not a functional word processor, it is simply a test bed to demonstrate what I've described here. Full source code is included, but do bear in mind that the program uses the Raize components version 1.6 to implement the toolbar and the status window. If you don't have Raize components, you won't be able to rebuild the code.

You should appreciate that the idea here isn't for you to rebuild my sample application, but for you to understand how to incorporate version 2.0 of the rich edit control into applications *of your own*. If you really must rebuild the program, then you can always download the shareware version of Raize Components from Ray Konopka's website at www.raize.com.

Also, I have not linked up `Save`, `Save As` or `Font` dialogs: I figured that you were all grown up boys and girls and well able to do this yourselves!

There's one important issue you should be aware of regarding the `TRichEdit2` control. If you create a new instance of this control on a form, it will, by default, have one line inserted into the `Lines` property which matches the component name. In other words, if the component name is `RichEdit21`, then the text `RichEdit21` will be inserted into the control when it's first created. Whatever you do, *do not save the form* with the control in this state. If you do, you won't be able to reload the form, the IDE will generate an exception. I haven't had time to investigate the problem thoroughly, but I believe that, in some small ways, the Version 2.0 control is behaving differently to the Version 1.0 control and this is causing the VCL code to fail.

If you want to pursue this further, the exception is generated in `TRichEditStrings.Insert` (in the file COMCTRLS.PAS), you'll see that the last line of this procedure raises an exception, and this happens when the form is loaded into memory.

Fortunately, the work-around is very straightforward, just be sure to clear the `Lines` property before saving the form! In next month's column, I'll continue the development of this control by examining how to implement a multiple undo/redo facility, and we'll also encapsulate the URL hyperlink functionality into custom properties, events and methods of the control.

Rich Edit 2.0 Distribution

We tried to clarify the position on whether Microsoft allow the RICHED20.DLL file to be distributed with third party software. The best information we have at the moment is that they don't. But, there is a better than even chance your users will already have it.

Dave Jewell is a freelance consultant/programmer and technical journalist specialising in system-level Windows and DOS work. He is Technical Editor of *Developers Review* which is also published by iTec. You can contact Dave at Dave@HexManiac.com

➤ *Figure 2: Here, you can see our own little test-bed program running. OK, so it's not quite as fancy as Word 97, but great oaks from little acorns do grow! As you can see, the cursor changes to a hand over a hyperlink, and the current hyperlink text is being displayed in the Raize status bar. Clicking the link will launch Internet Explorer*