How to get notified when the content of the clipboard changes

Why do it?

It is often useful to know when the clipboard's contents have changed. For example you may wish to enable or disable a *Paste* button on a toolbar according to whether the clipboard contains data in a format supported by your application or not. Alternatively you may wish to display details of the items on the clipboard.

How it's done

Overview

Windows provides two different methods that can be used to get notifications when the clipboard contents change.

The preferred method is to use the "listener" API. However this is not supported on all versions of Windows - support started with Windows Vista. On older OSs the older clipboard viewer chain API needs to be used. This older API is unreliable since it depends on each application that uses it to maintain the "chain"e; of viewers. All it needs is one badly behaved (or crashed) application to corrupt the chain. The newer API is managed by Windows and doesn't suffer from this major design headache.

Using the clipboard listener API

This API is quite simple to use and if your app is running on Windows Vista or later, it's the one you *should* use. You need to register one of the application's windows to receive a notification from Windows when the clipboard format changes. That window receives a *WM_CLIPBOARDUPDATE* message when the content changes.

To register window to use to receive notification you call the *AddClipboardFormatListener* API function, passing it the handle of the window. Before the application terminates you must also call the *RemoveClipboardFormatListener* function, passing the same window handle.

The last step is to handle the *WM_CLIPBOARDUPDATE* in the registered window's message loop and take whatever action you need to take when the clipboard changes.

Assuming you have a standard Delphi VCL application and you want to use the main form's window to receive change notifications, you can use code like the following.

To register the main form window insert the following code into your main form's *OnCreate* event handler:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  // ... other code
  AddClipboardFormatListener(Handle);
  // ... other code
end;
```

Listing 1

To un-register the window insert the following code into the form's *OnDestroy* handler:

```
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  // ... other code
  RemoveClipboardFormatListener(Handle);
  // ... other code
end;
```

Listing 2

Finally you need to handle the *WM_CLIPBOARDUPDATE* message. To do this create a new message handler in you main form class like this:

```
// ...

interface

// ...

type
  TMainForm = class(TForm)
  private
    // ...
    procedure WMClipboardUpdate(var Msg: TMessage); message WM_CLIPBOARDUPDATE;
    // ...
  end;

// ...

implementation

// ...

procedure TMainForm.WMClipboardUpdate(var Msg: TMessage);
begin
  // Do require clipboard change processing here
end;

// ...
```

Listing 3

Using the clipboard viewer chain

When using this API you also need to register a window to receive notifications. The API is more convoluted than the "listener" API discussed above.

The registered window is known as a clipboard viewer. Windows maintains a linked list of clipboard viewers – the clipboard chain. Each viewer window is responsible for passing on notifications to any window that follows it in the chain. Since Windows relies on the cooperation of viewer applications, a badly behaved application can bring down the whole notification system. It is therefore important to play by the rules. The viewer window should also be un-registered before the associated application terminates.

When the content of the clipboard changes, Windows notifies them first window in the chain by sending it a *WM_DRAWCLIPBOARD* message. This window is responsible for passing the message to the next registered viewer window, and so on down the list. Consequently it can be seen that each clipboard viewer needs to record the identity of the next window in the chain.

Windows also notifies the windows in the clipboard chain when clipboard viewers are removed from the list. This is done by passing a *WM_CHANGECBCHAIN* message to the first window in the chain. The parameters of *WM_CHANGECBCHAIN* identify the window being removed along with the window that follows it in the chain. The message is passed along the chain until the window preceding that being removed is found. That window then updates its record of the next window in the chain.

In summary, here are the key steps in creating, managing and deleting a clipboard viewer:

- Register the viewer window by calling the *SetClipboardViewer* API function. This function returns the handle of the next window in the chain (or 0 if no such window).
- Keep track of viewer windows that are removed by processing the *WM_CHANGECBCHAIN* message and passing it along the chain or updating the record of the next window in the chain as necessary.
- Respond to clipboard changes by handling the *WM_DRAWCLIPBOARD* message and passing the message along the chain.
- Before closing the application remove the viewer window from the clipboard chain by calling the *ChangeClipboardChain* API function.

As before we assume that we have a Delphi VCL application that will use it's main form to receive notifications of clipboard changes. Here's an outline of the required code.

To register the main form window as a clipboard viewer we call the *SetClipboardViewer* API function in the application's *FormCreate* event handler. *SetClipboardViewer* returns the handle of the next window in the viewer chain which we must in a form field for future reference (this handle may be 0 if we are the first viewer in the chain):

```
// ...

interface

// ...

type
  TMainForm = class(TForm)
  private
    // ...
    fNextCBViewWnd: HWND;
    // ...
  end;

// ...

implementation

// ...

procedure TMainForm.FormCreate(Sender: TObject);
begin
  // ... other code
  fNextCBViewWnd := SetClipboardViewer(Handle);
  // ... other code
end;
```

Listing 4

We need to un-register our viewer window when the application is closing. In the *FormDestroy* event handler we add the required call to *ChangeClipboardChain*.

```
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  // ... other code
  ChangeClipboardChain(Handle, fNextCBViewWnd);
  // ... other code
end;
```

Listing 5

Finally we need to handle the *WM_CHANGECBCHAIN* and *WM_DRAWCLIPBOARD* messages. We declare a message handler method for each in the form's interface and implement the methods as follows:

Every clipboard viewer must respond to the two clipboard messages discussed above. The window procedure of the viewer window must handle the messages as follows (where *Msg* is the message number and *LParam* and *WParam* are the parameters passed with the message):

```
// ...

interface

// ...

type
  TMainForm = class(TForm)
  private
    // ...
    procedure WMChangeCBChain(var Msg: TMessage); message WM_CHANGECBCHAIN;
    procedure WMDrawClipboard(var Msg: TMessage); message WM_DRAWCLIPBOARD;
    // ...
  end;

// ...
```

```
implementation

// ...

procedure TMainForm.WMChangeCBChain(var Msg: TMessage);
begin
  // A window has been detached from clipboard chain
  if HWND(Msg.WParam) = fNextCBViewWnd then
    // our next window has changed: record it
    // we don't pass the message on since we handled it
    fNextCBViewWnd := HWND(Msg.LParam)
  else
    // our next window has not changed: pass message on
    // if next window exists (i.e. handle <> 0)
    if fNextCBViewWnd <> 0 then
      SendMessage(fNextCBViewWnd, WM_CHANGECBCHAIN, Msg.WParam, Msg.LParam);
end;

procedure TMainForm.WMDrawClipboard(var Msg: TMessage);
begin
  // Clipboard content has changed
  //
  // do something here to respond to change in clipboard
  //
  // now pass message along to next window in chain, if it exists
  if fNextCBViewWnd <> 0 then
    SendMessage(fNextCBViewWnd, WM_DRAWCLIPBOARD, Msg.WParam, Msg.LParam);
end;

// ...
```

Listing 6

The *WParam* value of *WM_CHANGECBCHAIN* is the handle of the window to be removed while the *LParam* value is the handle of the window that follows it in the chain (or 0 if there is no following window). If the window that follows ours in the chain is the one to be removed we replace our record of its handle with the handle of the following window. In this case the message has been handled and there is no need to pass it further down the chain. However, if the window being removed is not the next window in the chain we simply forward the message on to the next window (if it exists).

When a *WM_DRAWCLIPBOARD* message is received we first take some suitable action in response to the change and then pass the message and parameters on to the next window (if any) in the chain.

Example project

An example project is presented below. This project implements a very basic text editor that has buttons to cut, copy and paste text. The *Paste* button is enabled only when there is text on the clipboard. In order to implement the *Paste* button functionality, the program's main window is registered as a clipboard viewer. The *Copy* and *Cut* buttons are enabled only when some text is selected in the edit control. These buttons are presented only for completeness and their workings are not relevant to the purpose of this article.

Project file

The demo's project file – `CBEdit.dpr` – is defined below:

```
program CBEdit;

uses
  Forms,
  FmEditor in 'FmEditor.pas' {EditorForm};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TEditorForm, EditorForm);
  Application.Run;
end.
```

Listing 7

Form Unit

The form is defined in a file named `FmEditor.dfm` as follows (non-essential items have been removed):

```
object EditorForm: TEditorForm
  Left = 190
  Top = 107
  Width = 696
  Height = 480
  Caption = 'EditorForm'
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  object Panel1: TPanel
    Align = alTop
    object btnCut: TButton
      Left = 8
      Top = 8
      Caption = 'Cut'
      OnClick = btnCutClick
    end
    object btnCopy: TButton
      Left = 88
      Top = 8
      Caption = 'Copy'
      OnClick = btnCopyClick
    end
    object btnPaste: TButton
      Left = 168
      Top = 8
      Caption = 'Paste'
      OnClick = btnPasteClick
    end
  end
  object RichEdit1: TRichEdit
    Align = alClient
    OnSelectionChange = RichEdit1SelectionChange
  end
end
```

Listing 8`

Finally, the code associated with the form is stored in `FmEditor.pas` and is as follows:

```
unit FmEditor;

interface

uses
  Forms, StdCtrls, ComCtrls, Classes, Controls, Windows, Messages;

const
  // WM_CLIPBOARDUPDATE is not defined in the Messages unit of all supported
  // versions of Delphi, so we defined it here for safety.
  WM_CLIPBOARDUPDATE  = $031D;

type
  TEditorForm = class(TForm)
    btnCut: TButton;
    btnCopy: TButton;
    btnPaste: TButton;
    RichEdit1: TRichEdit;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure RichEdit1SelectionChange(Sender: TObject);
    procedure btnCutClick(Sender: TObject);
    procedure btnCopyClick(Sender: TObject);
    procedure btnPasteClick(Sender: TObject);
  private
```

```
    // Flag indicating if the new style clipboard format listener API is
    // available on the current OS.
    fUseNewAPI: Boolean;
    // Handle of next clipboard viewer handle in chain. Used only when old
    // clipboard viewer API is in use, i.e. when fUseNewAPI is False.
    fNextCBViewWnd: HWND;
    // References to AddClipboardFormatListener and
    // RemoveClipboardFormatListenerAPI functions. These references are nil if
    // the functions are not supported by the OS, i.e. if fUseNewAPI is False.
    fAddClipboardFormatListener: function(hwnd: HWND): BOOL; stdcall;
    fRemoveClipboardFormatListener: function(hwnd: HWND): BOOL; stdcall;
    // References to SetClipboardViewer and ChangeClipboardChain API functions.
    // These references is are if the newer clipboard format listener API is
    // available, i.e. if fUseNewAPI is True.
    fSetClipboardViewer: function (hWndNewViewer: HWND): HWND; stdcall;
    fChangeClipboardChain: function(hWndRemove, hWndNewNext: HWND): BOOL;
      stdcall;
    // Message handlers
    procedure WMClipboardUpdate(var Msg: TMessage); message WM_CLIPBOARDUPDATE;
    procedure WMChangeCBChain(var Msg: TMessage); message WM_CHANGECBCHAIN;
    procedure WMDrawClipboard(var Msg: TMessage); message WM_DRAWCLIPBOARD;
  end;

var
  EditorForm: TEditorForm;

implementation

uses
  SysUtils, Clipbrd;

{$R *.dfm}

procedure TEditorForm.btnCopyClick(Sender: TObject);
begin
  RichEdit1.CopyToClipboard;
end;

procedure TEditorForm.btnCutClick(Sender: TObject);
begin
  RichEdit1.CutToClipboard;
end;

procedure TEditorForm.btnPasteClick(Sender: TObject);
begin
  RichEdit1.PasteFromClipboard;
end;

procedure TEditorForm.FormCreate(Sender: TObject);
const
  cUserKernelLib = 'user32.dll';
begin
  // Enable / disable buttons at start-up
  btnPaste.Enabled := Clipboard.HasFormat(CF_TEXT);
  btnCut.Enabled := RichEdit1.SelLength > 0;
  btnCopy.Enabled := btnCut.Enabled;
  // Load required API functions: 1st try to load modern clipboard listener API
  // functions. If that fails try to load old-style clipboard viewer API
  // functions. This should never fail, but we raise an exception if the
  // impossible happens!
  fAddClipboardFormatListener := GetProcAddress(
    GetModuleHandle(cUserKernelLib), 'AddClipboardFormatListener'
  );
  fRemoveClipboardFormatListener := GetProcAddress(
    GetModuleHandle(cUserKernelLib), 'RemoveClipboardFormatListener'
  );
  fUseNewAPI := Assigned(fAddClipboardFormatListener)
    and Assigned(fRemoveClipboardFormatListener);
  if not fUseNewAPI then
  begin
    fSetClipboardViewer := GetProcAddress(
      GetModuleHandle(cUserKernelLib), 'SetClipboardViewer'
    );
    fChangeClipboardChain := GetProcAddress(
      GetModuleHandle(cUserKernelLib), 'ChangeClipboardChain'
    );
    Assert(Assigned(fSetClipboardViewer) and Assigned(fChangeClipboardChain));
  end;
  if fUseNewAPI then
  begin
    // Register window as clipboard listener
    if not fAddClipboardFormatListener(Self.Handle) then
      RaiseLastOSError; // On early Delphis use RaiseLastWin32Error instead
  end
  else
  begin
    // Register window as clipboard viewer, storing handle of next window in
    // chain
    fNextCBViewWnd := fSetClipboardViewer(Self.Handle);
  end;
end;

procedure TEditorForm.FormDestroy(Sender: TObject);
begin
  // Remove clipboard listener or viewer
  if fUseNewAPI then
    fRemoveClipboardFormatListener(Self.Handle)
  else
    fChangeClipboardChain(Self.Handle, fNextCBViewWnd);
end;

procedure TEditorForm.RichEdit1SelectionChange(Sender: TObject);
begin
  btnCut.Enabled := RichEdit1.SelLength > 0;
  btnCopy.Enabled := btnCut.Enabled;
end;

procedure TEditorForm.WMChangeCBChain(var Msg: TMessage);
begin
  Assert(not fUseNewAPI);
  // Windows is detaching a clipboard viewer
  if HWND(Msg.WParam) = fNextCBViewWnd then
    // window being detached is next one: record new "next" window
    fNextCBViewWnd := HWND(Msg.LParam)
  else if fNextCBViewWnd <> 0 then
    // window being detached is not next: pass message along
    SendMessage(fNextCBViewWnd, Msg.Msg, Msg.WParam, Msg.LParam);
end;

procedure TEditorForm.WMClipboardUpdate(var Msg: TMessage);
begin
  // Clipboard content changed: enable paste button if text is on clipboard
  btnPaste.Enabled := Clipboard.HasFormat(CF_TEXT);
end;

procedure TEditorForm.WMDrawClipboard(var Msg: TMessage);
begin
  Assert(not fUseNewAPI);
  // Clipboard content changed
  // enable paste button if text on clipboard
  btnPaste.Enabled := Clipboard.HasFormat(CF_TEXT);
  // pass on message to any next window in viewer chain
  if fNextCBViewWnd <> 0 then
    SendMessage(fNextCBViewWnd, Msg.Msg, Msg.WParam, Msg.LParam);
end;

end.
```

Listing 9

This application first tries to load the clipboard listener API functions from the OS kernel. If this fails we are running on an OS that does not support this API so we fall back to using the old clipboard viewer chain API and load the required functions from the kernel. We use an assertion to check that the old API has been found as exepected.

The *fUseNewAPI* field is used to inform the app which API is being used. If we're using the new API we add our forms window handle as a listener, otherwise we hook into the clipboard viewer chain. The necessary tidying up is done when the form is destroyed.

The remainder of the code needs little further explanation. We enable the cut and copy buttons only when some text is selected in the rich edit control. The paste button is enabled or disabled each time the clipboard contents change, depending on whether the *CF_TEXT* format is available on the clipboard. The code in three custom message handlers was explained earlier in the article.

A demo project, based on *Listings 7 to 9* is available in the ***delphidabbler/article-9-demo*** Git repository on *BitBucket*.

Component

A *clipboard viewer component* that triggers an event whenever the clipboard changes is available from this site. This component uses a hidden window which is registered as the clipboard viewer with Windows. The use of such a hidden window is described in *article #1*.