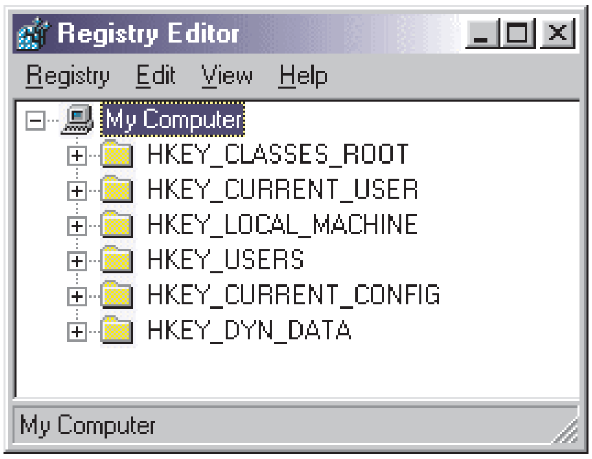
Beating The System: A Windows 98 File Association Un-Mangler

by Dave Jewell

For a while now, I've been planning to devote some column space to the development of a utility which will allow you to set up the file associations on your machine and, just as importantly, give you some way of easily restoring file associations that have been trampled on by other software. I refer largely (though by no means exclusively) to Internet Explorer and Windows itself.

For some time, Microsoft have taken the rather narrow view that the only program in the universe capable of opening GIF files (for example) is Internet Explorer. Every time you install Internet Explorer onto your system, Microsoft thoughtfully modify your file associations so that clicking a graphic file will almost always invoke their web browser. I've lost track of the number of times that I've had to work my way through the system registry, putting everything back the way it was. Recently, I was foolish enough to install the shrink-wrap version of Windows 98. Once again, after the installation smoke had cleared, I found that assorted file associations were again conforming to Microsoft's view of reality, and I was forced to fire up the ever-faithful REGEDIT utility, cursing under my breath as I did so. The real reason why I was cursing under my breath, by the way, is because Windows 98 resolutely refuses to recognise my 56K baud modem, but that's another story, and not one for the faint hearted...

It's hard to figure out why Microsoft do this. If an install routine detects that Internet Explorer is already installed, then it's not unreasonable to assume that the file associations are the way they are because the user wants them that way. But Microsoft can't refrain from imposing their own world view onto everybody else's desktop. I'm a great fan of the shareware Paint Shop Pro application (now up to version 5) and not only does Paint Shop Pro load and display graphic files a lot faster than Internet Explorer, it also does it a good deal more reliably, not to put too fine a point on it.



➤ *Figure 1: Under Windows 98, there are no less than six different 'hives' or master keys in the system registry. The total number of hives may vary somewhat across Windows 95, Windows 98 and different flavours of NT.*

Well, this time round, I finally snapped, and *Windows 98 File Association UnMangler* is the fruit of my labours. This program will allow you to take a snapshot of your file associations and then restore them at a later date, preferably after each installation of a Microsoft product! The functionality to do that isn't included in this month's code, but will be added next month. This time, I'll introduce the program, explain what it needs to do with respect to the structure of the system registry and describe how this month's code works.

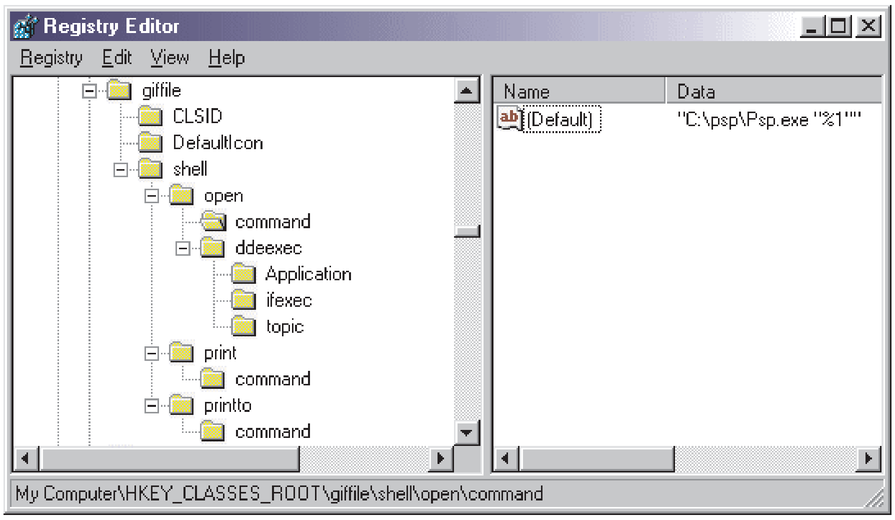
An Introduction To File Associations

In order to modify the file associations in the registry, we obviously need to know where this information is stored. Under Windows 98, the system registry now contains a total of six different master keys or 'hives' as they're often called. You can see all six hives in Figure 1. The `HKEY_CLASSES_ROOT` hive is the one that we're interested in, this contains all the file association information in addition to all the COM-based GUID class associations that we've looked at in the *Delphi Meets COM* series.

For every known file association, there's an entry directly under the `HKEY_CLASSES_ROOT` master key. The name of this entry corresponds to the extension of the file type. Thus, if you use `REGEDIT` to explore the registry, you'll find (for example) that there's a key with the name:

```
HKEY_CLASSES_ROOT\.dpr
```

This key provides the first-level file association for Delphi project files. If you look at the value part of the key, you'll find that it has a value of `DelphiProject`. If we now take this value and look up the key of the same name (still directly under the `HKEY_CLASSES_ROOT` master key) then we'll find the *real* second-level, file association information that relates to .DPR files. It's tempting to ask why Microsoft did things in this slightly convoluted manner. Why didn't they just put all the file association information directly under the file extension key rather than going through another level of registry lookup?



(Default) "C:\psp\Psp.exe "%1""

My Computer\HKEY_CLASSES_ROOT\giffile\shell\open\command

➤ *Figure 2: This screenshot shows the parts of the registry that relate to a single file association. This is a relatively simple one, but some are much more complex: take a look at the entry for .HTM files on a system that has Internet Explorer installed.*

The short answer is flexibility. By implementing things in this way, it's possible to have many different file extensions, all of which reference the same second-level registry information. A common example of this is the JPEG graphics file format. In my system registry, the file extensions .JPG, .JPE and .JPEG all have the same key value of `JPEGFILE`, and they therefore all point to the same set of file association information. This scheme means that more efficient use is made of the registry, but equally, it goes without saying that if we modify the second-level registry information for one of these file types then we're effectively modifying it for all of them.

So what does the second-level file association information look like? On the basis that a picture is worth a thousand words, take a look at Figure 2 which shows the hierarchical registry data relating to the .GIF file extension. As you can see, there are three sub-keys, `open`, `print` and `printto`, which descend from a `shell` sub-key. For our purposes, all we're interested in is the `open` branch, since this is what the Explorer refers to when you double-click a file. However, if you wanted, you could extend my little UnMangler program to operate on the other two branches as well. In order to restore the status quo for GIF files (I'm using this file type as an example throughout this discussion, but the same general comments apply to other file types as well) it's necessary to change the value of the `\shell\open\command` key value so that it references the application you wish to associate with this file type.

You'll also notice that there's a `ddeexec` sub-tree containing `Application`, `ifexec` and `topic` keys. These keys correspond to the DDE-related associations which you can set up so that (for example) an already running application is notified via DDE that a file has been opened. If you're familiar with Delphi 3, you'll know that double-clicking a Delphi-related file from Windows Explorer while the IDE is running will start a second instance of the IDE. Using the DDE mechanism, this behaviour can be avoided and in fact Inprise seem to have improved the situation under Delphi 4. For example, double-click a project (.DPR) file while Delphi is running and the IDE will automatically close the current project and swap over to the newly launched project.

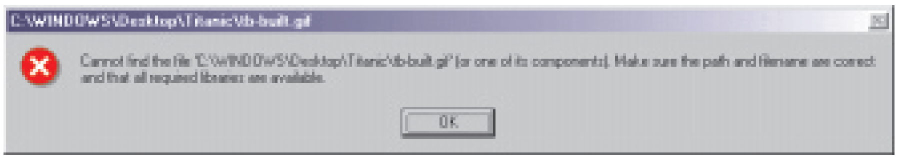
However, the presence of the `ddeexec` sub-tree isn't always benign. For example, if you come along after a Windows 98 or Internet Explorer installation and reset the `\shell\open\command` key value to point to Paint Shop Pro, double-clicking a GIF file will certainly launch PSP, but it will also display an error message like that shown in Figure 3. In order to fully restore the status quo, it's necessary to delete the `ddeexec` sub-tree and things will then work as in days of old.

The UnMangler Application

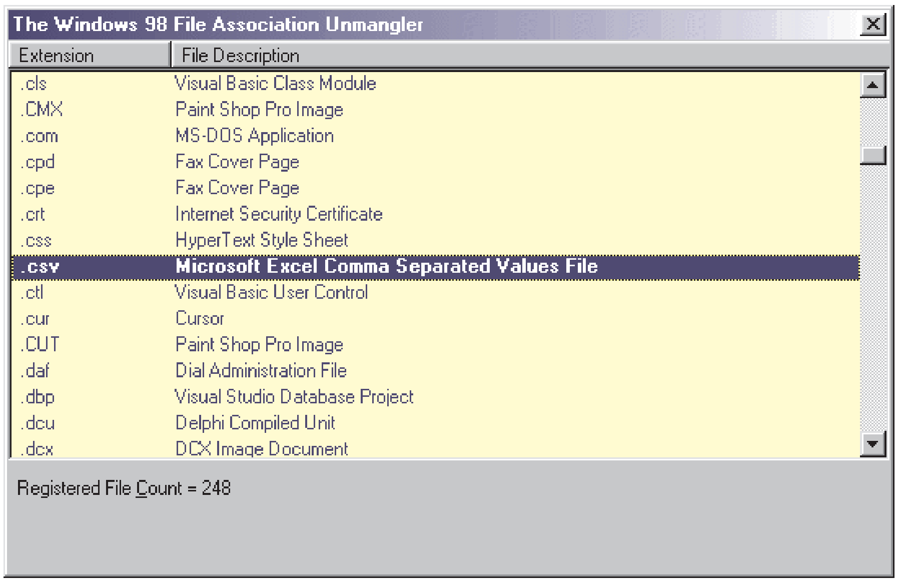
You can see my little UnMangler application running in Figure 4. Developing this application gave me an excuse to play with the `THeaderControl` component which forms part of the Win32 common controls library. The application shows a listbox containing all the file associations found in the system registry. On the left-hand side of the listbox you can see the different file extensions, sorted alphabetically. On the right-hand side are the description strings (also pulled out of the registry, as we'll see later) which are used by the Windows Explorer to fill in the `Type` column when looking at a directory using the `Details` view. However, if you click the right-hand section of the header control, the right-hand section of the listbox will change to display the actual file associations as shown in Figure 5.

So how does it work? Well, as I pointed out, the actual code for saving and restoring a file association snapshot isn't there yet, I'll be developing it in next month's column. Nevertheless, as you can see from Listing 1, a fair chunk of

➤ *Figure 3: If you don't delete the ddeexec sub-tree which was put there by Microsoft, applications such as Paint Shop Pro will get rather upset when you double-click GIF files, for example.*



Cannot find the file 'C:\WINDOWS\Desktop\Titanic\tb-built.gif' (or one of its components). Make sure the path and filename are correct and that all required libraries are available.



The Windows 98 File Association Unmangler

Extension	File Description
.cls	Visual Basic Class Module
.CMX	Paint Shop Pro Image
.com	MS-DOS Application
.cpd	Fax Cover Page
.cpe	Fax Cover Page
.crt	Internet Security Certificate
.css	HyperText Style Sheet
.csv	Microsoft Excel Comma Separated Values File
.ctl	Visual Basic User Control
.cur	Cursor
.CUT	Paint Shop Pro Image
.daf	Dial Administration File
.dbp	Visual Studio Database Project
.dcu	Delphi Compiled Unit
.dcx	DCX Image Document

Registered File Count = 248

➤ *Figure 4: Here's the UnMangler application displaying file description information obtained from the registry. These are the same file type descriptions that the Explorer uses when displaying a directory view in 'Details' mode.*

code is needed just to initialise the listbox contents. This month's code will also allow you to totally obliterate existing file associations, about which more later: please read the important caveats before hitting the `Del` key!

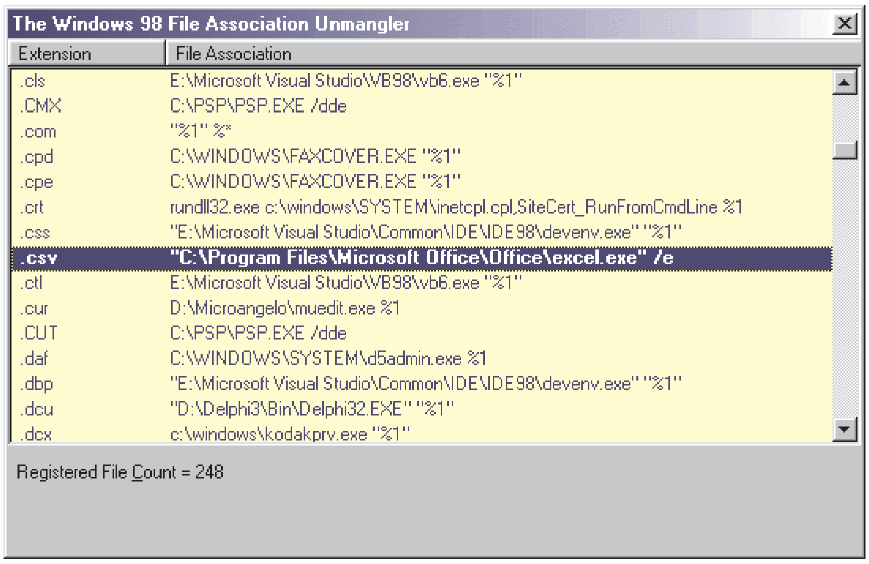
The biggest routine in Listing 1 is the `FormCreate` handler, which parses the system registry and sets up a list of file associations. I don't know about you, but when working with the registry I find that the `TRegistry` class is generally too much like hard work because of the repeated need to keep opening and closing file keys. I've therefore used the `TRegIniFile` class for registry access, which makes the job a lot more straightforward. Additionally, this class uses buffered writes (so-called 'lazy' writes) so you get good performance even when a large number of registry changes need to be applied in one go.

As a quick aside, Delphi 4 provides a new class called `TMemIniFile` which is primarily for manipulating .INI files (not the registry) under Windows NT. NT doesn't cache .INI file writes which can result in exceptionally poor performance where a large number of file updates are involved. The `TMemIniFile` class was developed to address this problem by internally caching write operations. In fact, it's not a new class at all, it was internally used by the Delphi 3 IDE before being made available for public consumption.

Having created a `TRegIniFile` object, the `FormCreate` code opens the `HKEY_CLASSES_ROOT` 'hive', creates a `TStringList` object and calls the deeply wonderful `ReadSections` routine to read all the top-level key values from the hive into the string list. This is a surprisingly fast operation even when there are a large number of keys involved. Of course, at this point many of the entries in the string list aren't actually file extensions: there will be all sorts of other stuff in there as well including the second-level key names. In order to whittle things down to the required file associations, the code first checks that an entry begins with a period *[Ahem, 'full stop' in Britain... Ed]*. If it doesn't, then it's effectively discarded. You might be forgiven for thinking that another good test would be to make sure that there are no more than three characters following the period, but I've already shown that to be a false assumption, as in the case of the .JPEG file extension. In fact, my registry even contains an entry for files with the extension '.properties'. Thus, we can't make any assumptions about the maximum length of a file extension.

For each file extension found, the code checks that there is a corresponding second-level entry in the registry. With .GIF files, for example, it checks that there's a key within the hive called `giffile`. If there is, then the default value of this key contains the human-readable description of this file type. We read it into the `Desc` string. Finally, the file association itself (ie, the pathname of the program that's associated with this

➤ *Figure 5: And here's the UnMangler again, this time showing the file associations alongside each file extension. In next month's column, I'll add the code needed to take a snapshot of this information and restore it to the registry on demand.*



The Windows 98 File Association Unmangler

Extension	File Association
.cls	E:\Microsoft Visual Studio\VB98\vb6.exe "%1"
.CMX	C:\PSP\PSP.EXE /dde
.com	"%1" %*
.cpd	C:\WINDOWS\FAXCOVER.EXE "%1"
.cpe	C:\WINDOWS\FAXCOVER.EXE "%1"
.crt	rundll32.exe c:\windows\SYSTEM\inetcpl.cpl,SiteCert_RunFromCmdLine %1
.css	"E:\Microsoft Visual Studio\Common\IDE\IDE98\devenv.exe" "%1"
.csv	"C:\Program Files\Microsoft Office\Office\excel.exe" /e
.ctl	E:\Microsoft Visual Studio\VB98\vb6.exe "%1"
.cur	D:\Microangelo\muedit.exe %1
.CUT	C:\PSP\PSP.EXE /dde
.daf	C:\WINDOWS\SYSTEM\d5admin.exe %1
.dbp	"E:\Microsoft Visual Studio\Common\IDE\IDE98\devenv.exe" "%1"
.dcu	"D:\Delphi3\Bin\Delphi32.EXE" "%1"
.dcx	c:\windows\kodakprv.exe "%1"

Registered File Count = 248

```
unit RFMain;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Registry, ComCtrls;
type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    FileTypesLabel: TLabel;
    HeaderControl1: THeaderControl;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure ListBox1DrawItem(Control: TWinControl; Index:
      Integer; Rect: TRect; State: TOwnerDrawState);
    procedure HeaderControl1SectionTrack(HeaderControl:
      THeaderControl; Section: THeaderSection; Width:
      Integer; State: TSectionTrackState);
    procedure ListBox1DblClick(Sender: TObject);
    procedure HeaderControl1SectionClick(HeaderControl:
      THeaderControl; Section: THeaderSection);
    procedure ListBox1KeyDown(Sender: TObject; var Key:
      Word; Shift: TShiftState);
  private
    SysReg: TRegIniFile;  { For accessing system registry }
    { True for descriptions, False for  associations }
    ShowDesc: Boolean;
    { A little hackette for on-the-fly header resizing }
    HeaderZeroSize: Integer;
    function GetStr (S: String; Idx: Integer): String;
    procedure DeleteItem (const ItemString: String);
  public
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
function TForm1.GetStr (S: String; Idx: Integer): String;
var IdxPos: Integer;
begin
  while Idx <> 0 do begin
    IdxPos := Pos (Chr (9), S);
    S := Copy (S, IdxPos + 1, MaxInt);
    Dec (Idx);
  end;
  IdxPos := Pos (Chr (9), S);
  if IdxPos = 0 then
    IdxPos := MaxInt;
  Result := Copy (S, 1, IdxPos - 1);
end;
procedure TForm1.FormCreate(Sender: TObject);
var
  Idx: Integer;
  Desc, Str, CurSubKeyName: String;
  SubKeys, FileExts: TStringList;
begin
  { Open registry and access the hKey_Classes_Root hive }
  SysReg := TRegIniFile.Create ('');
  SysReg.RootKey := hKey_Classes_Root;
  SysReg.OpenKey ('', False);
  { Create temporary stringlist to hold raw subkey names }
  SubKeys := TStringList.Create;
  { And another for holding tab-delimited file extensions }
  FileExts := TStringList.Create;
  try
    SysReg.ReadSections (SubKeys);
    for Idx := SubKeys.Count - 1 downto 0 do begin
      CurSubKeyName := SubKeys [Idx];
      if CurSubKeyName [1] = '.' then begin
        Str := SysReg.ReadString (CurSubKeyName, '', '');
        if Str <> '' then begin
          Desc := SysReg.ReadString (Str, '', '');
          if Desc <> '' then begin
            Str := SysReg.ReadString (Str +
              '\shell\open\command', '', '');
            if Str <> '' then
              FileExts.Add (CurSubKeyName +
                Chr(9) + Desc + Chr(9) + Str);
          end;
        end;
      end;
    end;
    ListBox1.Items.Assign (FileExts);
    ListBox1.ItemIndex := 0;
    FileTypesLabel.Caption := Format(
     'Registered File &Count = %d', [ListBox1.Items.Count]);
  finally
    SubKeys.Free;
    FileExts.Free;
  end;
  ShowDesc := True;
  HeaderZeroSize := HeaderControl1.Sections [0].Width;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  SysReg.Destroy;
end;
procedure TForm1.ListBox1DrawItem (Control: TWinControl;
    Index: Integer; Rect: TRect; State: TOwnerDrawState);
var
  Idx: Integer;
  ItemString: String;
begin
  with ListBox1.Canvas do begin
    FillRect (Rect);
    if odSelected in State then
      Font.Style := Font.Style + [fsBold];
    ItemString := ListBox1.Items [Index];
    TextOut(Rect.Left+5, Rect.Top, GetStr(ItemString,0));
    if ShowDesc then Idx := 1 else Idx := 2;
    TextOut(HeaderZeroSize, Rect.Top,
      GetStr(ItemString,Idx));
  end;
end;
procedure TForm1.HeaderControl1SectionTrack(HeaderControl:
  THeaderControl; Section: THeaderSection; Width:
  Integer; State: TSectionTrackState);
begin
  if State = tsTrackMove then begin
    HeaderZeroSize := Width;
    ListBox1.Invalidate;
  end;
end;
procedure TForm1.HeaderControl1SectionClick (HeaderControl:
  THeaderControl; Section: THeaderSection);
begin
  if Section = HeaderControl1.Sections [1] then begin
    ShowDesc := not ShowDesc;
    if ShowDesc then
      Section.Text := 'File Description'
    else
      Section.Text := 'File Association';
    ListBox1.Invalidate;
  end;
end;
procedure TForm1.DeleteItem (const ItemString: String);
begin
  with ListBox1 do begin
    Items.Delete (ItemIndex);
    ItemIndex := 0;
    FileTypesLabel.Caption := Format(
      'Registered File &Count = %d',
      [ListBox1.Items.Count]);
    { Now delete the registry stuff too }
    SysReg.EraseSection(SysReg.ReadString(
      GetStr(ItemString, 0), '', ''));
    SysReg.EraseSection (GetStr (ItemString, 0));
  end;
end;
procedure TForm1.ListBox1KeyDown (Sender: TObject; var Key:
  Word; Shift: TShiftState);
var
  ItemString: String;
begin
  if Key = vk_Delete then with ListBox1 do begin
    ItemString := Items [ItemIndex];
    if MessageDlg(Format(
      'Remove all registry entries for ''%s''?',
      [GetStr (ItemString, 0)]), mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then
      DeleteItem (ItemString);
  end;
end;
end.
```

➤ *Listing 1*

file type) is read from the \shell\open\command sub-tree as previously described. All three fields (file extension, description and application pathname) are added to the FileExts string list, using tabs to separate each field from the next.

Finally, once every entry has been validated in this way, the resulting string list is assigned to the Items property of the listbox and the code exits. Before it does so, however, it sets up a couple of member variables, ShowDesc and HeaderZeroSize for use by the owner-draw code which displays the listbox items.

The ListBox1DrawItem code takes care of drawing individual items in the listbox. As you can see, I've arranged things so that the highlighted entry is displayed in bold for an extra bit of visual emphasis. The main part of the code obtains the file extension part of the string list item through a call to GetStr and displays it indented slightly from the left-hand edge of the listbox. Next, it obtains either the description information or the pathname (depending on the value of the ShowDesc boolean variable) and displays this at a horizontal location that's consistent with the current setting of the header control. The GetStr routine is a simple little function that takes a string comprising an arbitrary number of tab-delimited fields and returns the field indicated by the Idx parameter. Centralising this logic in a separate function makes the ListBox1DrawItem code very much simpler and more readable than it would otherwise be.

Fun With Header Controls

From the program description I've given so far, the eagle-eyed will be asking why I needed to declare a separate variable HeaderZeroSize and initialise it in the FormCreate routine? Why didn't I just pick up the current section width from inside the ListBox1DrawItem routine?

Well, as I said earlier, part of the reason for this program was to serve as an excuse for playing with the THeaderControl component! When working with multiple columns of data in a listbox component, I like to provide immediate visual feedback as the user changes the width of individual columns, in other words I like the affected column(s) to move during the resizing operation. The VCL 'wrapper' around the underlying Windows common control provides two distinct events that relate to the resizing of section widths. These are OnSectionResize and OnSectionTrack.

The OnSectionResize event occurs immediately *after* a section has been resized and it's therefore useless for our purposes. By the time this event fires, the user has already released the mouse button and the section has snapped to its new width. This leaves us with the OnSectionTrack event, a much better bet as it fires every time the mouse moves during a section resize operation. Inside the HeaderControl1SectionTrack event handler, I update the HeaderZeroSize variable according to the passed Width parameter and then immediately invalidate the listbox contents, causing it to be redrawn. With this in view, you can see why I need the HeaderZeroSize variable. It allows us to get the current 'on-the--fly' value of a section width during a resize operation.

In a similar fashion the HeaderControl1SectionClick routine responds to mouse clicks on the header control. If the section being clicked is the second section then the ShowDesc boolean variable is toggled, the text of the section is updated according to whether we're displaying descriptions or file associations and, once again, the listbox contents are invalidated to force a redraw.

Oops, I Didn't Mean To Do That...!

A surprising number of (allegedly) commercial quality packages leave junk in the system registry, and this is as true of file associations as it is of COM registration, miscellaneous application settings and so on. To give just one example, if you walk into your local branch of W H Smith's, you can buy the new *Maplin Electronics* catalogue in the form of a CD ROM. This application is based around the Fourth Dimension database engine. When you uninstall it ('nuff said!) it leaves some detritus in the system registry, including file associations for the file types used by Fourth Dimension itself.

If you switch the UnMangler application so as to display file associations rather than descriptions, it becomes very easy to just scan through the list and look for information that's referencing directories that no longer exist. If you wanted to extend my program, you could obviously add code to step through the file association list, specifically looking for references to non-existent directories and highlighting them in some way, or even automatically deleting them.

For my own use, I've added a simple delete facility to the UnMangler app. You need to appreciate that this has got relatively little to do with file associations, it simply deletes any reference to a given file type including the second-level registry information. Understand that you use it at your own risk!

The ListBox1KeyDown routine simply checks for a press of the Del key, displays a confirmation dialog asking if you want to delete the highlighted item and then calls the DeleteItem routine to delete the first- and second-level registry information (all of it!) relating to this file type. The displayed file count information is then updated.

You'll recall that earlier I mentioned the possibility of multiple file extensions mapping to a single set of second-level file association information, as in the case of .JPG, .JPE and .JPEG files. Because of such possibilities, you need to be especially careful when deleting file associations. If what you're deleting points to a non-existent directory, then you can delete the association with impunity but otherwise, be careful! If you wanted to write a more sophisticated version of my UnMangler, then you could store the second-level key name (eg giffile) as part of the tab-delimited data that's stored in the main listbox. Then, whenever an attempt is made to delete a file association, it would be a simple matter to scan the list looking for any other file extensions which share the same association data. You could then put up a suitable warning and optionally delete all other affected file types at the same time.

Well, that's it for this month. Next time round, I'll add the code to take file association snapshots and then re-impose a snapshot back on the registry at a later time. See you then!

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

Windows 98: What's In It For You?

For some time, the Editor has been nagging me to do a piece on the new features in Windows 98 and NT 5. Although I was initially enthusiastic about this idea, my enthusiasm waned somewhat when it became apparent that there wasn't really all that much to write about from a programmer's perspective. Ok, Windows 98 has got Active Desktop, USB support and a number of other cute user interface enhancements: I especially like the Quick Launch area on the Windows taskbar and ability to rearrange Start Menu items by dragging them around. It's also nice being able to view folders as if they were web pages and generally apply the same access paradigms to both the desktop and the internet. Nevertheless, the amount of new API-level functionality available to programmers is relatively small. Yes, Windows 98 now has a new 32-bit device driver model, but how often is this particular feature going to be relevant during ordinary application development?

It isn't just me that feels this way by any means. Being a confirmed anorak, I subscribe to a number of American programmer's magazines in addition to the excellent publication which you are now reading *[This man will go far! Ed]*. In the countdown to Windows 95, one of these, *Microsoft Systems Journal*, devoted a great deal of space to all the new programming goodies that were available such as shell programming with the Explorer, long filename support, how to use the new common controls, common dialogs and so forth. This time round, Windows 98 has barely rated a mention.

So where does this leave us? In my opinion, it's strictly business as usual. From what I've seen of Windows 98 so far, there are relatively few situations (in terms of day-to-day application development) where you're likely to care much about differentiating between Windows 98 and Windows 95. The only real exception to this is the new multi-monitor support. With Windows 98, it's possible to plug more than one display card (each with its own monitor) into your PC and have the system allocate the logical desktop area across all installed monitors. This makes it possible (for example) to drag windows from one monitor and put them on another monitor.

So why would you want to do this? Good question. If multi-monitor support had been unveiled four or five years ago when we were all working with 14 inch screens, then there might have been some point to it. Nowadays, 17 inch screens are pretty much the norm, and 19 inch screens (like the one sat in front of me) are eminently affordable. With a reasonable resolution (I use 1152 by 864 pixels), screen clutter really isn't an issue. Why should I clutter my desktop (the wooden one, not the Windows one!) in an effort to reduce screen clutter that I don't have?

OK, I accept that for some people multi-monitor support is the best thing since sliced bread and I can certainly appreciate that it's useful during debugging. But I reckon that multi-monitor set-ups will always be the exception rather than the rule, and you'd be daft if you wrote a mass-market application that depended upon this feature. The good news, for Delphi 4 programmers, is that multi-monitor support is now directly accessible through enhancements to the VCL library. Forms now have a `DefaultMonitor` property which can be used to specify which monitor you want the form to appear on, and the Screen object has a new `Monitors` array property which can be used to enumerate all the monitors on a system, enumerate their size, relative position and so forth. Other than that, happy Delphi programming!

Dave Jewell