Tips & Tricks

Edited by Mike Orriss. Please send your Tips direct to Mike at mjo@compuserve.com

IDAPI32.CFG

Have you ever found that all your aliases have suddenly disappeared? It happened to me when trying out a shareware component that had a very badly behaved install that created a new version of IDAPI32.CFG instead of merging into the current file. The aliases are not lost but the Registry no longer points to them.

The easiest way of making the correct CFG current that I have found so far with Delphi 3 is to run BDEAdmin.EXE and use File|Open to open your CFG file. When you close BDEAdmin, you will be asked if you want this to be your current config and it will update the Registry for you.

Contributed by Mike Orriss, mjo@compuserve.com

Changing The Month

There are occasions where you wish to allow the user to specify a new date by adding or subtracting a given number of months. This is a fairly simple operation, but it does have its pitfalls: try subtracting one month from the 30th March for example.

There is a simple trick to obtain the number of days in a given month. The integer part of a TDateTime variable represents a number of days so we can just take the date of the first of the following month and subtract one. Supposing that you know the month and year, then the date of the last day of the month is given by:

```
Date1 := EncodeDate(yyyy,mm + 1,1) - 1;
```

The number of days in the month can then be extracted via the DecodeDate procedure:

```
DecodeDate(Date1,yyyy,mm,dd);
```

where yyyy,mm,dd are Word variables that you supply.

Listing 1 contains a generic function for adding/subtracting month(s) which avoids the problem above.

Contributed by Mike Orriss, mjo@compuserve.com

➤ *Listing 1*

```
function(date: TDateTime, months): TDateTime;
var
  yyyy,mm,dd,ddmax: word;
  date2: TDateTime;
begin
  DecodeDate(date,yyyy,mm,dd);
  mm := mm + months;  // months can be negative
  while mm > 12 do begin dec(mm,12); inc(yyyy); end;
  while mm < 1  do begin inc(mm,12); dec(yyyy); end;
  ddmax := DecodeDate(EncodeDate(yyyy, mm+1, 1)-1,
    yyyy, mm, ddmax);
  if ddmax < dd then dd := ddmax;
  Result := EncodeDate(yyyy,mm,dd);
end;
```

Plugging In A Class

ClassPlug provides a way to implement VCL changes without modifying (or even having) the VCL source code, but it should be used with great care! It will even allow you to modify the affect of compiled units and even works at design time. Imagine the class:

```
TPanel = class(TWinControl);
```

Suppose you want to change the message handling of WM_PAINT and that that change is to be used on every Panel available? Normally you derive a new class:

```
TMyPanel = class(TPanel)
```

and replace all the used panels with TMyPanel. But what happens if somebody derives a new component from TPanel? Your change to TMyPanel won't stick to that control and you have to add your code to it. This is (read was) a major problem, but I have a solution: you can use ReplaceParentClass to stick a class in between TWinControl and TPanel to hook in the WM_PAINT procedure.

How? Derive a new control from TWinControl which is used to hook WM_PAINT and then temporarily replace the ParentClass in TPanel with THookControl (using Initialization and Finalization). See Listings 2 and 3.

Oh, in case you were wondering, to make the change occur at design time you can register a component with the initialization and finalization part in the unit, to make the change when the package is loaded. There is an example project on the disk (CLASSPLUG.ZIP) which demonstrates how display a caption on a panel as it is re-sized.

Contributed by Jonas W Nordlöf, jonas.nordlof@autodiagnos.se

➤ *Listing 2*

```
unit Test;
interface
uses
  Windows, SysUtils, Classes, Messages,Menus, Graphics,
  Controls, Forms, Consts, Dialogs, ClassPlug;
THookControl = class( TWinControl);
  protected
    Procedure WMPaint(Var Message: TWMPAINT);
      message WM_PAINT;
end;
Implementation
Procedure THookControl.WMPaint(Var Message: TWMPAINT);
begin
  // Do your thing....
  inherited; // Important!! Ok?
end;
Initialization // Make the change
  ReplaceClassParent( TPanel, TWinControl, THookControl);
Finalization // Restore to be clean...
  ReplaceClassParent( TPanel, THookControl, TWinControl);
end.
```

Notebook Menu Merging

Contrary to Delphi's help system, `TMainMenu` components do not have to be on separate forms to work. I wanted to merge different menus when a different notebook page is selected. I put a menu for each page onto my form and in the `TTabSet.OnChange` event I call `MainMenu1.Merge(Page1Menu)` etc.

This works well if you don't use any other modeless forms. If you have to use modeless forms you need to keep a pointer to the current menu and remerge the menu when the form closes.

The end result is really nice. I like it so much I would like to see this technique adopted as a standard interface. Try the demo (on the disk as NOTEMENU.ZIP) and let me know what you think.

Contributed by Paul Warren,
hg_soft@uniserve.com

Version Information

As you know, Delphi 3 allows us to include version information in our project. When you select `Project|Options|Version info`, it gives you the chance to specify items such as major/minor version numbers, release and build numbers, as well as module attributes and a series of Key/Value pairs describing specific information related to the application you are building. You can even instruct Delphi to auto-increment the Build Number, a useful feature that allows you to keep track of how many times your app has been compiled.

All this information is stored inside the EXE file in the form of a special resource (of type `RT_VERSION`, as defined by Microsoft). The question is, how can we access the internals of this resource and obtain this information, let's say, in order to show it in an About box? The answer lies in using the Win32 API function `VerQueryValue`, that returns the contents of the specified section of the `VersionInfo` resource.

As a complete description of the structure of the `VersionInfo` resources and the `VerQueryValue` function could take several pages but it is well covered in the Win32 API Help. I will show the way to obtain just the version/release/build information, which are the most useful items (in my opinion) since the product name can always be taken from somewhere and the name of my company is already "wired" into the About Box repository template I use in all my projects.

In order to call `VerQueryValue`, you must specify the address of your `VersionInfo` resource (there will be typically only one in a project, named #1), the key that identifies the element you need, the address of the place where you want a pointer to the information to be stored, and the address of an integer variable, where Windows will put the real length of the item for which you asked.

In order to access version/release/build numbers, you must supply \ (for the root block) as the key to be found. In this case, Windows returns us a pointer to a (fixed length) structure of the Delphi type `TVSVersionFileInfo` (defined, of course, in Windows.PAS). Listing 4 shows how to prepare the call to `VerQueryValue` and after that unpack the desired values.

In the example project included (VINFO.ZIP on the disk), this code is attached to the constructor of an About Box form.

Contributed by Octavio "Dave" Hernandez,
cppbdany@danysoft.com

Starting Projects Via DPR

This tip makes life a lot easier when you have to work with Delphi 1 on a system that has Delphi 2 and/or 3 installed. Since I installed Delphi 2 and Delphi 3, double-clicking the .DPR file for any project brings up the last version of Delphi that I installed. However, I have some projects that I still need to work on in Delphi

➤ *Listing 3*

```
unit ClassPlug;
interface
procedure ReplaceParentClass(DClass, OldParent,
  NewParent: TClass);
implementation
uses Windows;
type
  PP = ^Pointer; // Pointer of Pointer of Parent...
procedure ReplaceParentClass( DClass, OldParent,
  NewParent: TClass);
var
  a: ^Byte;
  p1,p2: PP;
  prot: Longint;
begin
  // Simple dummy check..
  if ( NewParent = nil ) or ( DClass = nil ) then Exit;
  // Find the class Parent pointer of AClass
  while ( DClass.ClassParent <> OldParent ) do begin
    if DClass.ClassParent = nil then Exit;
    if DClass.ClassParent = NewParent then Exit;
    DClass := DClass.ClassParent;
  end;
  a := Pointer(DClass);
  inc(a,vmtParent);
  p1 := Pointer(a);
  { Find the class Self pointer of NewParent, wich will be
    used to fill the place of the ParentClass Pointer }
  a := Pointer(NewParent);
  inc(a,vmtSelfPtr);
  p2 := Pointer(a);
  { Big THANX to Cyril Jandia for the next 3 steps, taken
    from the Issue 24 class traps example.I had it all
    worked out, except for the VirtualProtect thingy }
  VirtualProtect(p1, SizeOf(Pointer), PAGE_READWRITE,
    @prot); // let's be brave
  p1^ := p2; // let's be yet more brave
  // time to be clean: not necessary but easy, then...
  VirtualProtect(P1, SizeOf(Pointer), prot, @prot);
  // use the next line to visualize the change...
  // TClass(PP(P1)^^).className
end;
end.
```

➤ *Listing 4*

```
var
  h: HRSRC;
  ptrBlock, verinfo: Pointer;
  verInfoSize: integer;
begin
  // ...  h := FindResource(HInstance, '#1', RT_VERSION);
  if h <> 0 then begin
    ptrBlock := LockResource(LoadResource(HInstance, h));
    VerQueryValue(ptrBlock, '\', verInfo, verInfoSize);
    with PVSFixedFileInfo(verInfo)^ do
      Version.Caption := ' Version ' +
        IntToStr(dwProductVersionMS div 65536) + '.' +
        // major version
        IntToStr(dwProductVersionMS mod 65536) + '.' +
        // minor version
        IntToStr(dwProductVersionLS div 65536) + '.' +
        // release
        IntToStr(dwProductVersionLS mod 65536);
        // build number
  end;
  // ...
end;
```

1. So I created a shortcut and put it in the same directory as the project. The target line of the shortcut invokes Delphi 1 passing the name of the .DPR file. The start in directory is set to the directory containing the project. This makes it a lot easier than starting Delphi 1 from the start menu and then navigating to the project.

Contributed by Mark Erbaugh, 71370.1475@compuserve.com

Code Completion

Inside a form's method, type self. and the components, properties and methods of the form itself will be listed in the code completion combobox.

Contributed by Stephane Grobety, grobety@capp.ch

Navigation With Cursor Keys

A few days ago, one of our clients asked us to implement navigation through the edit controls on a form using the cursor keys instead of the usual Windows Tab and Shift-Tab keystrokes.

A common request from our clients who are used to old DOS user interfaces was always that pressing the Enter key should move focus to the next control in the tab order, and we got used to implementing the feature by means of a well-known trick: setting the form's KeyPreview property to True (so that keystrokes go first to the form and only after that to the active control), and then programming the form's OnKeyPress event so that the call:

```
Perform(WM_NEXTCTLDLG, 0, 0);
```

is made when the Enter key is received. But this time we needed more: not only to navigate the form's controls forwards, but also backwards (and interestingly there's no WM_PREVCTLDLG message in Windows!).

The solution we found is based on the use of a Win32 API function, Keybd_Event, which is capable of generating keystrokes, ie synthesising a WM_KEYUP or WM_KEYDOWN message given a virtual key code. As before, we set form's KeyPreview to true, but instead of supplying a handler for OnKeyPress, we use the lower level OnKeyDown event (Listing 5).

➤ *Listing 5*

```
procedure TForm1.FormKeyDown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  case Key of
    VK_DOWN,VK_RETURN :
      // Enter OR Arrow Down - same as Tab
      begin
        Keybd_Event(VK_TAB,0,0,0);
      end;
    VK_UP:
      // on receiving ArrowUp, simulate Shift-Tab
      begin
        // simulate Shift key pressed
        Keybd_Event(VK_SHIFT,0,0,0);
        Keybd_Event(VK_TAB,0,0,0);
        // Shift key released
        Keybd_Event(VK_SHIFT,0,KEYEVENTF_KEYUP,0);
    end;
  end;
end;
```

Whenever our event handler receives a VK_UP virtual key code, stating that the ArrowUp key has been pressed, we activate a series of three calls to Keybd_Event that simulate the pushing of the Shift-Tab combination. For the Shift key, we must make two calls that correspond to pressing and releasing (the reader is invited to check for him/herself that the second call is just indispensable!).

Be aware that this solution works as shown only for forms that do not include controls for which up and down arrow keys have a concrete meaning, as is the case for list boxes, combo boxes, or multi-line edit controls. In such cases, additional programming will be needed in order to ensure the arrow keys work as expected when such a control has the focus.

In any case, Keybd_Event seems to be a very handy function that can be applied to many different situations. The Win32 API Help shows how an application can use it to produce a full-screen or window snapshot and save it to the clipboard, which could be useful, for instance, for debugging purposes.

Contributed by Octavio "Dave" Hernandez, cppbdany@danysoft.com

Update: More Delimiter Tools, Issue 24

In case anyone got confused by the last two paragraphs on page 63, issue 24, the word "Next" in the penultimate paragraph referred to the next function to be discussed, Get_nth_word, not to some procedure or function named Next. I probably should have made that clearer originally, but things do get past us once in a while. In the last paragraph I probably should also have made an explicit mention of Get_nthWord, instead of just the place where I got the inspiration from, the REXX function Word().

Brandon Smith, http://members.aol.com/synature

Update: Just The Extension Please

In Issues 21 and 24 Tom Corcoran and Pete Frizell deal with the problems of extracting a filename without the extension and changing the extension. Try these:

```
shortFileName := ExtractFileName(
  ChangeFileExt(longFileName, ''));
newFileName := ChangeFileExt(fileName,
  newExtension);
```

ChangeFileExt has existed since Delphi 1, so I believe Tom and Pete missed a trick here!

Contributed by Thale Hadderingh, thale@pi.net