Tips & Tricks

Controlling Internet Explorer Via OLE Automation

I have just started a small project to control Microsoft Internet Explorer 3.0 (IE3) using OLE automation and thought I would share my experiences. To add the IE3 web browser control to your component library, follow these steps:

➢ Select component/install from the menu.
➢ Press the OCX button.
➢ Select `Microsoft Internet Controls` from the listbox and press OK.
➢ Press OK to rebuild the library.

A new component (`TWebBrowser`) will be added to your OCX tab and this component can now be added to any project as per normal.

Documentation for the web browser control is hard to find, but there is a small SDK-type document available on the Microsoft web site. Unfortunately, when you start using the control, you will find that you can set properties and use methods successfully, but you cannot capture the browser events.

To make events work successfully you need to make some changes to the VCL. In method `TOleInPlaceSite.GetWindow`, replace the existing code with that shown in Listing 1.

Then in method `TOleClientSite.GetContainer`, change the function result assignment to return `E_NOINTERFACE`. Whenever changing the VCL source I always take a local copy and amend that; I then add the amended source to the required project, so that the local copy (in other words the copy in my project directory, not in Delphi's main source code directory) gets linked in. This prevents me from messing up the true VCL files and is a handy "quick and dirty" way of making VCL changes.

Contributed by Alistair Moffatt,
alistair@phrenetic.co.uk

➤ *Listing 1*

```
begin
  if FControl.Parent <> nil then begin
    wnd := FControl.Parent.Handle;
    Result := S_OK;
  end else if (FControl.Owner <> nil) and
    (FControl.Owner is TWinControl) then begin
    wnd := TWinControl(FControl.Owner).Handle;
    Result := S_OK;
  end else begin
    wnd := 0;
    Result := E_FAIL;
  end;
end;
```

QuickReport Exceptions

When running QuickReports, once an exception is raised the default preview window refuses to show anything! This should clear the error:

```
try
  if chkPreview.Checked then
    { check box to switch preview mode on/off  }
    QuickReport1.Preview
  else
    QuickReport1.Print;
finally
  QRPrinter.EndDoc;
  QRPrinter.Cleanup;
end;
```

Contributed by Richard Smith,
European_SD@compuserve.com

Vertical Fonts

I had to create a vertical font the other day to output into QuickReport. Fortunately for me it needed to appear at the same place on each page of the report, rather than moving around (which would have been more tricky). Here's how I did it. In the `private` section of the form class definition add:

```
hdlNewFnt : integer;  //handle of the new font
hdlCurrFnt : integer; //handle of the holding font
NewFont : TFont;
OldFont : TFont;
```

and in the `FormCreate` event add the code in Listing 2. This creates a font about 8 point. It seems that if you

➤ *Listing 2*

```
var
  fntName : string;
begin
  fntName := 'Vertical';
  hdlNewFnt := CreateFont(-5, -5, 900 ,900, FW_NORMAL, 0,
    0, 0, DEFAULT_CHARSET, OUT_DEVICE_PRECIS,
    CLIP_DEFAULT_PRECIS, PROOF_QUALITY,
    VARIABLE_PITCH or FF_SWISS, PChar(fntName));
  NewFont := TFont.create;
  NewFont.Handle := hdlNewFnt;
  OldFont := TFont.Create;
  ...
```

➤ *Listing 3*

```
...
OldFont.Assign(qrprinter.canvas.font); //store current font
qrprinter.canvas.font.assign(NewFont); //assign new font
{ describe a rectangle that will fit the vertical font,
  QRShape is an object at the top of the available area on
  the band }
Rect.top := QRShape1.Top + QRShape1.height + 5;
{ calculate the left position based on a component placed
  at the leftmost position - this is my calculation as to
  where the left most pos should be }
Rect.Left := Plan1.Left + (32*(iThisTime-1)) + 10;
//pgHdr is the band it is being written into
Rect.Bottom := pgHdr.Height - 5;
Rect.Right := Rect.Left + 10;
qrprinter.Canvas.textrect(Rect, Rect.Left, Rect.Bottom,
  qryReport.fieldbyname('Description').AsString);
qrprinter.canvas.font.assign(OldFont);  //reassign old font
...
```

specify 8 in the first two parameters of the CreateFont call then it is OK on NT4 but not on Windows 95. Can anyone explain this? In the BeforePrint event of the relevant band place the code in Listing 3 and hey presto!

Contributed by Richard Smith, European_SD@compuserve.com

Assigning Accelerator Keys At Runtime

Consider a form with a main menu and a notebook containing a number of pages. Depending on certain conditions, the objective is to set some of these page tabs at run-time. The tabs on the notebook are to have hotkeys, so the problem is how to assign hotkeys to these tabs without using already existing hotkeys.

In this example I assume that there are no controls on the form that are not a child of the notebook, so hotkeys that cannot be used are any in the main menu, in the notebook tabs, in the notebook popup menu and then any hotkeys on each respective page where the tab hotkey need to be assigned.

Looking at the popup menu the only potential hotkey conflict is any menu items with a shortcut beginning with Alt (see the procedure GetHotKeysInShortCuts in Listing 4). I use a TStringList to hold the hotkeys found, you could also easily use a TStrings so the result would be assignable from the Items properties of TListBox, TComboBox, etc. A TStringList can also be passed to hold any duplicates found, so if so desired these routines could be used in testing to ensure on complex forms that there was no hotkey duplication.

The GetHotKeysInWinControl procedure uses recursion to get the hotkeys for a specified control and its children. If you pass a form to this routine it will get all the hotkeys in the form.

Much of the credit for this work should go to Dean Thompson of Classic Software (100033.1230@compuserve.com). He gave me the HasNamedProperty function that uses the very cool undocumented GetPropInfo and also explained the Controls versus Components distinction to me, as well as providing many suggestions, including using recursion. We use his excellent Classic Notebook a lot and you can find the hotkeys on a Classic page by typecasting the page object as a TCSPage instead of a TTabPages as described above.

The example in Listing 4 shows how I assign the hotkeys for the notebook tabs at runtime, it assumes that Form1 contains a TTabbedNotebook (called nbk), a TMainMenu (called mnuMain) and a TPopUpMenu (called pop). Using a combination of the routines many other scenarios can be dealt with.

Contributed by Tom Corcoran, tomc@unitime.com

Disk Space *Really* Free

The DiskFree and DiskSpace functions in Delphi are designed around what used to be quite reasonable limits, but times have changed...

Since they return integer (LongInt in Delphi 1) byte-count values, they are limited to a return value of MaxLongInt: roughly 2 billion. Thus, run it against a drive

➤ *Listing 4*

```
procedure SetHeadings;
var
  slstHotKeys,
  slstHotKeysSave: TStringList;
  procedure SetTab(pageIndex: integer; header: string);
  begin
    { hot keys that can’t be used are <slstHotKeysSave> =
      main menu, notebook tabs, toolbar, popup. This list
      must be added to for each each successive page }
    slstHotKeys.Assign(slstHotKeysSave);
    GetHotkeysInWinControl(TTabPages(
      nbk.Pages.Objects[pageIndex]), slstHotKeys, nil);
    SetAcceleratorKey(slstHotKeys, header);
    nbk.TabCaption[pageIndex] := header;
    slstHotKeys.Clear;
  end;
begin
  slstHotKeys := TStringList.Create;
  slstHotKeysSave := TStringList.Create;
  try
    { pass nil as confident that form contains no duplicates}
    GetHotKeysInMainMenu(form1.mnuMain,
      slstHotkeysSave, nil);
    GetHotKeysInNbkTabs(nbk, slstHotKeysSave, nil);
    GetHotKeysInShortCuts(pop, slstHotKeysSave, nil);
    { tab names could be obtained at run-time }
    SetHeading(1, ’First’);
    SetHeading(2, ’Second’);
    SetHeading(3, ’Third’);
    SetHeading(4, ’Fourth’);
    SetHeading(5, ’Fifth’);
  finally
    slstHotKeys.Free;
    slstHotKeysSave.Free;
  end;
end;
procedure GetHotKeysInWinControl(ctrl: TWinControl;
  slstHotkeys,slstDuplicates: TStringList);
  procedure ChkValidAndAdd(ctrl: TControl);
  var
    hotKey: string;
  begin
    if HasNamedProperty(ctrl, ’Caption’) then begin
      { can use anything to typecast as long as it has a
        caption property }
      hotKey := GetHotKey(TLabel(ctrl).Caption, False);
      AddHotKey(slstHotkeys, slstDuplicates, hotKey);
    end;
  end;
  procedure CycleControls(ctrl: TWinControl);
  var
    i: integer;
  begin
    { use recursion to check for hotkeys on nested
     TWinControls }
    if ctrl.ControlCount > 0 then
      for i := 0 to ctrl.ControlCount - 1 do begin
        if ctrl.Controls[i] is TWinControl then
          CycleControls(TWinControl(ctrl.Controls[i]));
        ChkValidAndAdd(ctrl.Controls[i]);
      end;
  end;
begin
  CycleControls(ctrl);
end;
procedure AddHotKey(slstHotkeys, slstDuplicates: TStringList;
  hotKey: string);
begin
  if hotkey <> ’’ then
    { returns -1 if check not in list, otherwise returns index }
    if slstHotKeys.IndexOf(hotKey) = -1 then
      slstHotKeys.Add(hotkey)
    else
      if Assigned(slstDuplicates) then
        slstDuplicates.Add(hotKey);
end;
procedure SetAcceleratorKey(slstHotKeys: TStringList;
  var toSet: string);
var
  j: integer;
  inList, found : boolean;
  ch : string;
begin
  j := 1;
  found := false;
  while not found do begin
    ch := UpperCase( toSet[j] );
    inList := (slstHotKeys.IndexOf(ch) <> -1);
                                    { ** CONTINUED ON NEXT PAGE }
```

with 3Gb available, and DiskFree will return “2 billion bytes free boss”. Not a problem in most circumstances, you might think? Well, it can be if you have clever little routines for detecting free space before copying or creating files: you know, the ones you originally wrote to make sure you weren’t about to overfill a floppy, or a cluttered hard disk. Something like:

“I can write this file provided its size is *less* than the total space available, defined as the disk free space PLUS the size of any existing copy of the file.”

It’s that final addition that will do your program in unless you are careful. In pseudocode:

```
FreeSpaceOnTargetDrive = DiskFree
if TargetFileExists then
   Inc(FreeSpaceOnTargetDrive,
     SizeOfTargetFile) { bzzzt, danger! }
```

What’s the problem? Take MaxLongInt. Add a number to it. Oh dear, a supremely negative result. Suddenly our clever little routine, working on a drive with simply gigabytes to spare, thinks the free space is, well, less than zero...

The workaround you choose will depend on your need, but the key lesson is: *don’t* do the addition if it will (or might ever) cause an overflow. Because in the late 1990s, even in a 32-bit compiler, such things are entirely possible once again.

Contributed by Peter Hyde, author of TCompress, who is now busily re-writing his installation programs and can be reached as peter@spis.co.nz

No Extension

Have you found yourself needing a file name without the extension? This simple routine is a variation on Delphi’s ExtractFileName:

```
function ExtractFileNameNoExt(
  fileName: string): string;
var lengthFileName: byte absolute fileName;
begin
   Result := Copy(ExtractFileName(fileName),
     1, lengthFileName - 4);
end;
```

Contributed by Tom Corcoran, tomc@unitime.com

Thanks for the Tips, everyone – keep them coming in! Just drop an email to the Editor on DelphiMagazine@compuserve.com

➤ *Listing 4 (continued)*

```
{ ** CONTINUED FROM PREVIOUS PAGE }
    if (not inList) and (ch <> #32) then begin
      slstHotKeys.Add(ch);
      toSet := Copy( toSet, 1, j-1 ) + '&' +
        Copy( toSet, j, Length(toSet) );
      found := true;
    end else begin
      inc(j);
      found := (j > Length(toSet));
    end
  end;
end;
function HasNamedProperty(AComponent: TComponent;
  const propertyName: string): boolean;
var
  propInfo: PPropInfo;
begin
  propInfo := GetPropInfo(AComponent.ClassInfo,
    propertyName);
  Result := (propInfo <> nil);
end;
procedure GetHotKeysInNbkTabs(ctrl: TWinControl;
  slstHotkeys, slstDuplicates: TStringList);
var
  i: integer;
  str : TStrings;
  hotKey: string;
begin
  { setup for 2 notebooks }
  if ctrl is TTabbedNotebook then
    str := (ctrl as TTabbedNotebook).Pages
  else if ctrl is TcsNotebook then
    { Classic notebook }
    str := (ctrl as TCsNotebook).Pages
  else
    Exit;
  { add hotkeys on notebook tabs }
  for i := 0 to str.Count - 1 do begin
    hotKey := GetHotKey(str.Strings[i], False);
    AddHotKey(slstHotkeys, slstDuplicates, hotKey);
  end;
end;
procedure GetHotkeysInMainMenu(mnu: TMainMenu;
  slstHotkeys, slstDuplicates: TStringList);
var
  i: integer;
  hotKey: string;
begin
  for i := 0 to mnu.Items.Count - 1 do begin
    hotKey := GetHotKey(mnu.Items[i].Caption, True);
    AddHotKey(slstHotkeys, slstDuplicates, hotKey);
  end;
end;
procedure GetHotKeysInShortCuts(mnu: TMenu;
  slstHotkeys, slstDuplicates: TStringList);
  { use TMenu arguement so can pass TMainMenu and TPopupMenu}
  procedure CycleMenu(itm: TMenuItem);
  var
    i: integer;
    hotKey: string;
    function GetHotKeyInShortCut(
      strShortCut: string): string;
      { will only conflict with hotkey if shortcut in Alt+? format }
    begin
      Result := '';
      if Copy(strShortCut, 1, 3) = 'Alt' then
        Result := Copy(strShortCut, 5, 1);
    end;
  begin
    { use recursion to check for hotkeys in nested
      TMenuItems }
    if itm.Count > 0 then
      for i := 0 to itm.Count - 1 do begin
        CycleMenu(itm.Items[i]);
        if itm[i].ShortCut <> 0 then begin
          hotKey := GetHotKeyInShortCut(
            ShortCutToText(itm[i].ShortCut));
          AddHotKey(slstHotkeys, slstDuplicates, hotKey);
        end;
      end;
  end;
begin
  CycleMenu(mnu.Items);
end;
function GetHotKey(str: string; msg: boolean): string;
var
  i: integer;
  length: byte absolute str;
  nextChar: string;
begin
  i := 1;
  nextChar := Copy(str, i, 1);
  while ((nextChar <> '&') and (i <= length)) do begin
    Inc(i);
    nextChar := Copy(str, i, 1);
  end;
  if (i = length) then begin
    { nextChar could be & or not }
    if msg then
     { self check: should never go in here (for TMenuItems) }
     MsgError(Format('%s has no hotkey', [str]));
    Result := '';
  end else
    Result :=  Copy(str, i + 1, 1);
end;
```