Under Construction:
Delphi 2.0 AddIn Experts

by Bob Swart

Delphi 1.0 has three kinds of experts: project, form and standard. The first two can be found in the `Options | Gallery` (or `Repository` in Delphi 2.0) dialog, while standard experts (like the Database Form Expert) are in the `Help` menu. Delphi 2.0 added a fourth kind of expert to this collection: the special *AddIn Experts*, which have to deal with the interface to Delphi all by themselves. This article will focus on writing AddIn Experts for Delphi 2.0.

The major reason why everyone thinks experts are difficult is because they are not documented. Not in the manuals or on-line Help, that is (there have been some magazine articles and even a chapter in one particular book on this topic, but that's it). The main source of information is the source code itself! If you take a look at the documentation and source code on your hard disk, you'll find some important files and even two example experts that are installed automatically by Delphi. The example files can be found in the SOURCE\TOOLSAPI subdirectory (for Delphi 2.0), and are EXPTINTF.PAS and TOOLINTF.PAS. The first one shows how to derive and register your own expert, while the second one shows how to use Delphi's tool services to make the integration complete.

First of all, for all Delphi experts, we need to have a look at the abstract base class interface definition in EXPTINTF.PAS located in directory SOURCE\TOOLSAPI\, which is shown in Listing 1.

If we want to derive our own AddIn expert, we have to derive it from the abstract base class `TIExpert`. Since it is an *abstract* base class, it would seem that we need to override each function. However, we don't need `GetMenuText` and `GetState` (which are only used by standard experts) nor do we need `GetGlyph`, `GetPage` and `GetComment` (which are only used by project and form experts). Finally, there is no need to have an `Execute` method since we have to find another way inside Delphi to activate our AddIn expert.

This actually leaves us with four interface methods: `GetIDString`, which needs to return a unique ID string for every expert we register; `GetStyle`, which needs to return the `esAddIn` style; `GetName`, which is needed (otherwise you'll get an access violation if you try to load your expert – I tried!) and finally `GetAuthor` which is not really needed (only for form and project experts) but I decided to add this one for completeness. Overriding these four functions, we can write our first non-functional AddIn expert for Delphi 2 (Listing 2).

Since we don't have an `Execute` method (for AddIn experts `Execute` is never called), we seem to have missed something. Our expert skeleton is complete, but we still have to find a way to get inside

➤ *Listing 1*

```
Type
  TExpertStyle = (esStandard, esForm, esProject, esAddIn);
  TIExpert = class(TInterface)
  public
    { Expert UI strings }
    function GetName: string; virtual; stdcall; abstract;
    function GetAuthor: string; virtual; stdcall; abstract;
    function GetComment: string; virtual; stdcall; abstract;
    function GetPage: string; virtual; stdcall; abstract;
    function GetGlyph: HICON; virtual; stdcall; abstract;
    function GetStyle: TExpertStyle; virtual; stdcall; abstract;
    function GetState: TExpertState; virtual; stdcall; abstract;
    function GetIDString: string; virtual; stdcall; abstract;
    function GetMenuText: string; virtual; stdcall; abstract;
    { Launch the Expert }
    procedure Execute; virtual; stdcall; abstract;
  end;
```

➤ *Listing 2*

```
Type
  TBAddInExpert = class(TIExpert)
    function GetStyle: TExpertStyle; override;
    function GetIDString: String; override;
    function GetName: String; override;
    function GetAuthor: String; override;
  end {TBAddInExpert};
function TBAddInExpert.GetStyle: TExpertStyle;
begin
  Result := esAddIn
end {GetStyle};
function TBAddInExpert.GetIDString: String;
begin
  Result := 'DrBob.AddIn.Expert'
end {GetIDString};
function TBAddInExpert.GetName: String;
begin
  Result := 'DrBob.AddIn.Expert'
end {GetName};
function TBAddInExpert.GetAuthor: String;
begin
  Result := 'Bob.Swart'
end {GetAuthor};
```

Delphi and make sure we can get some action somehow. This is where the Menu Interface classes from TOOLINTF.PAS come into play. See Listings 3 and 4.

The TIMainMenuIntf class represents the Delphi main menu. We can actually get a list of menu items by calling GetMenuItems (which returns the top level menus) and we can search for a specific menu item with FindMenuItem, as long as we know the exact VCL component name of the particular item (ie not the name of the menu as it appears in the menu bar, but the name of the menu item component itself).

Once we have a list of menuitems, or we have found one particular menu item, we have a much more powerful component in our hands: the TIMenuItemIntf, an expert's interface to menu items, with which we can add our own menu item(s) into the Delphi menu system! See Listing 4.

The TIMenuItemIntf class is created by Delphi. This is simply a virtual interface to an actual menu item found in the IDE. It is the responsibility of the client to destroy all menu items which it created. Failure to properly clean up will result in unpredictable behaviour, according to the comments in the source code of the class TIMenuItemIntf.

Functions that return a TIMenuItemIntf should be used with care. Unless created by a particular add-in tool, we should not keep the menu items for long, since the underlying actual VCL TMenuItem may be destroyed without any notification (in which case we're holding a pointer to nowhere). In practice, this only pertains to Delphi created menus or menus created by other add-in tools. It is also the responsibility of the user to free these interfaces when finished. Any attempt to modify a menu item that was created by Delphi will fail.

The most important functions are DestroyMenuItem, which needs to be called whenever we get a menu item from Delphi (that is, allocated by Delphi, in, for example, the GetParent or GetItem functions).

Any menu item can have submenus. The function GetItemCount returns the number of submenus. Using GetItem we can walk through the list of menu items (GetItem is zero-based, so start by counting from 0 to GetItemCount-1, otherwise you'll get an *Index out of bounds* exception). All TIMenuItemIntfs we get from GetItem must be freed by calling DestroyMenuItem on them.

To get the true VCL component name of a menu item, we need to call the GetName method. This function is important, since we need the actual name to be able to search for menu items in the main menu (with the FindMenuItem function). Actually, it seems that we would need a list of names first, before we can actually search for a unique one.

Any menu item has a menu parent, and we can get the parent menu item by calling GetParent (obviously). A parent menu is important, since we must use the parent to be able to install a menu item next to another (in practice this means that the parent gets another child).

Using the GetCaption and SetCaption methods we can get and set the actual captions of the menu items. This may be useful, but can be very confusing (although we can only modify menu captions that are not part of the Delphi IDE skeleton – ie we can modify the text for the Database Expert, but we cannot modify the File menus). Using GetShortCut and SetShortCut we can get and set the shortcuts for

➤ *Listing 3*

```
TIMainMenuIntf = class(TInterface)
public
  function GetMenuItems: TIMenuItemIntf; virtual; stdcall; abstract;
  function FindMenuItem(const Name: string): TIMenuItemIntf; virtual;
    stdcall; abstract;
end;
```

➤ *Listing 4*

```
Type
  TIMenuFlag = (mfInvalid, mfEnabled, mfVisible, mfChecked, mfBreak,
                mfBarBreak, mfRadioItem);
  TIMenuFlags = set of TIMenuFlag;
  TIMenuClickEvent = procedure (Sender: TIMenuItemIntf) of object;
  TIMenuItemIntf = class(TInterface)
  public
    function DestroyMenuItem: Boolean; virtual; stdcall; abstract;
    function GetIndex: Integer; virtual; stdcall; abstract;
    function GetItemCount: Integer; virtual; stdcall; abstract;
    function GetItem(Index: Integer): TIMenuItemIntf; virtual; stdcall;
      abstract;
    function GetName: string; virtual; stdcall; abstract;
    function GetParent: TIMenuItemIntf; virtual; stdcall; abstract;
    function GetCaption: string; virtual; stdcall; abstract;
    function SetCaption(const Caption: string): Boolean; virtual; stdcall;
      abstract;
    function GetShortCut: Integer; virtual; stdcall; abstract;
    function SetShortCut(ShortCut: Integer): Boolean; virtual; stdcall;
      abstract;
    function GetFlags: TIMenuFlags; virtual; stdcall; abstract;
    function SetFlags(Mask, Flags: TIMenuFlags): Boolean; virtual; stdcall;
      abstract;
    function GetGroupIndex: Integer; virtual; stdcall; abstract;
    function SetGroupIndex(GroupIndex: Integer): Boolean; virtual; stdcall;
      abstract;
    function GetHint: string; virtual; stdcall; abstract;
    function SetHint(Hint: string): Boolean; virtual; stdcall; abstract;
    function GetContext: Integer; virtual; stdcall; abstract;
    function SetContext(Context: Integer): Boolean; virtual; stdcall;
      abstract;
    function GetOnClick: TIMenuClickEvent; virtual; stdcall; abstract;
    function SetOnClick(Click: TIMenuClickEvent): Boolean; virtual;
      stdcall; abstract;
    function InsertItem(Index: Integer; Caption, Name, Hint: string;
      ShortCut, Context, GroupIndex: Integer; Flags: IMenuFlags;
      EventHandler: TIMenuClickEvent): TIMenuItemIntf; virtual; stdcall;
      abstract;
  end;
```

the individual menu items. Again, we cannot really modify the pre-existing Delphi IDE menu shortcuts, but only the added ones. Other functions include GetFlags and SetFlags, to get and set the state of the menu item, GetGroupIndex and SetGroupIndex, to get and set the GroupIndex property of a TMenuItem (useful for specifying values for grouped radio menus), GetContext and SetContext to get and set the help context ID of a TMenuItem, and finally GetHint and SetHint that do not work at all (the IDE seems to simply ignore them).

There is one more really important method left: InsertItem. This is the API that creates and inserts our new sub menu item into the menu of the Delphi IDE. The function takes a lot of arguments, so let's have another look (Listing 5).

Index is the place where the new menu item should be placed (in the list of the Parent's menu items). If the index is less than zero or equal or greater than GetItemCount, then the new menu item is actually appended to the bottom of the list (since the list is zero-based).

The Caption is the text that we'll see in the menu, something like '&Dr.Bob''s Expert'. The Name is the name of the VCL menu item component. It's not clear whether or not this name should be actually the same as the component name that holds the menu item that we've just created: in that case, we probably need to use some unique component name as well. For now, I've used DrBobItem, which seems pretty unique (so far). The final string is a Hint, which is not used at this time it seems, so I leave it blank for now. Then, we can add a ShortCut key, a help Context and GroupIndex. As MenuFlags I always use enabled and visible, but we can pick from a set of them (see Listing 4). Finally, we need to assign an OnClick event that gets fired when the menu item for our expert is selected. This is the place where for other expert types our Execute method would kick in. Now we need a method of type TIMenuClickEvent (Listing 5). So, an actual call to InsertItem could be:

```
DrBobItem := Tools.InsertItem(
  ToolsTools.GetIndex+1,
  '&Dr.Bob''s Expert',
  'DrBobItem','',
  ShortCut(Ord('D'), [ssCtrl]),0,0,
  [mfEnabled, mfVisible], OnClick);
```

The last two methods of TIMenuItemIntf are GetOnClick and SetOnClick, which can be used to get and set the OnClick method (useful in case we want to do something else based on a special condition).

With this additional information, it's time to add code to our AddIn expert. What would be the best place to add it to the Delphi IDE menu system? Well, a constructor would seem a fine place to me. But

➤ *Listing 5*

```
function InsertItem(Index: Integer; Caption, Name, Hint: String; ShortCut,
   Context, GroupIndex: Integer; Flags: TIMenuFlags; EventHandler:
   TIMenuClickEvent): TIMenuItemIntf; virtual; stdcall ;abstract;
```

➤ *Listing 6*

```
Type
  TBAddInExpert = class(TIExpert)
  public
    constructor Create; virtual;
    destructor Destroy; override;
    function GetStyle: TExpertStyle; override;
    function GetIDString: String; override;
    function GetName: String; override;
    function GetAuthor: String; override;
  protected
    procedure OnClick(Sender: TIMenuItemIntf); virtual;
  private
    MenuItem: TIMenuItemIntf;
  end {TBAddInExpert};
constructor TBAddInExpert.Create;
var Main: TIMainMenuIntf;
    ToolsTools: TIMenuItemIntf;
    Tools: TIMenuItemIntf;
begin
  inherited Create;
  MenuItem := nil;
  if ToolServices <> nil then begin
    Main := ToolServices.GetMainMenu;
    if Main <> nil then { we've got the main menu }
    try
      ToolsTools := Main.FindMenuItem('ViewPrjMgrItem');
      if ToolsTools <> nil then { we've got the " Tools | Tools" item }
      try
        Tools := ToolsTools.GetParent;
        if Tools <> nil then { we've got the Tools menu }
        try
          MenuItem := Tools.InsertItem(ToolsTools.GetIndex+1,
            '&Dr.Bob''s Expert', 'DrBob','', ShortCut(Ord('D'),
            [ssCtrl]),0,0, [mfEnabled, mfVisible], OnClick)
        finally
          Tools.DestroyMenuItem
        end
      finally
        ToolsTools.DestroyMenuItem
      end
    finally
      Main.Free
    end
  end
end {Create};
destructor TBAddInExpert.Destroy;
begin
  if MenuItem <> nil then MenuItem.DestroyMenuItem;
  inherited Destroy
end {Destroy};
procedure TBAddInExpert.OnClick(Sender: TIMenuItemIntf);
begin
  ShowMessage('Dr.Bob Says: Hello, World!'#10#10+
              'Thank you for reading my'#10+
              'Under Construction column'#10+
              'in The Delphi Magazine!')
end {OnClick};
```

TIExpert doesn't have a constructor! OK, so let's define one! And while we're at it, let's override the destructor as well to make sure we clean up the MenuItem that we'll create in the constructor in the first place. See Listing 6.

Note that I've looked for the menu item called 'ViewPrjMgrItem', which is a rather funny looking name. How did I know what to look for in the first place? Well, sit tight, because we're about to find out the names of all Delphi's menu items.

After we've installed our first AddIn expert (more about this later), we can start Delphi up again and see it as part of the View menu (see Figure 1).

Menu Names

We've seen a generic but pretty useless expert so far. In order to write truly useful experts, we need to do something special inside the OnClick method, like show an interesting form in which a lot of things can happen. But first, let's dig a little bit deeper in the main menu of the Delphi IDE. Now that we have the power, let's use it to get a list of VCL names for the individual menu items, so we don't need to look for one if we need it. To do this, I've made a new MenuList Expert which just contains a modified Create constructor, to walk through the main menu items and write their names to a file (Listing 7).

The resulting list is pretty impressive and gives us a good idea of which VCL menu item component names are used (and can be used as arguments to the FindMenuItem function of the MainMenu). A list is included on this month's disk, but you can create it yourself with the TBMenuList expert.

The third and last expert we will create is one that I'd like to add to the Tools menu. Here's the list of menu items that the MenuList expert found for the ToolsMenu:

```
ToolsMenu
  ToolsOptionsItem
  ToolsGalleryItem
  ToolsToolsItem
```

At first sight there's nothing special, right? Well, it turns out that we somehow cannot get a correct handle to the ToolsToolsItem. Probably because this menu is created dynamically by Delphi and possibly modified after our expert has "nested" itself in the menu.

Installing ourselves between the ToolsGalleryItem (actually the repository) and the ToolsToolsItem does work without problem as expected. That's the place where we'll install our third expert: the Error Report Expert.

Error Report Tool

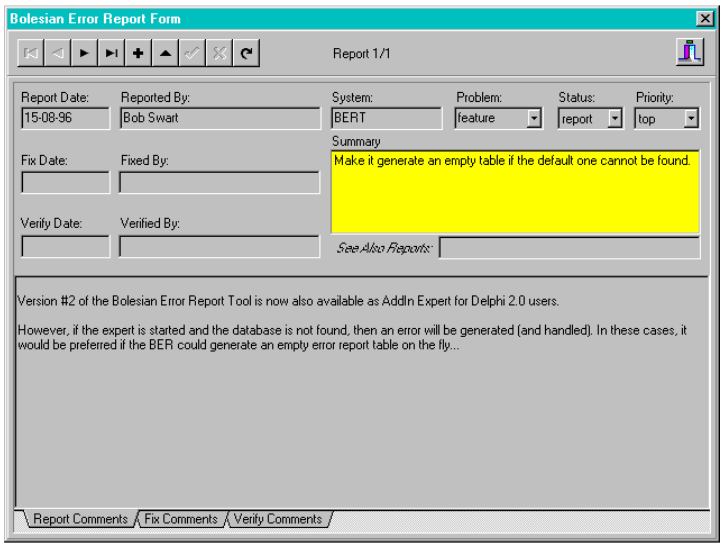
This is basically the same as the first two, except for the fact that in the OnClick even we'll do a ShowModal of a form with a connection to a special database. In this case, a database with error reports, fixes and verification reports on these fixes. Figure 2 shows the expert in action and an example database is included on the disk. With the shortcut Ctrl+E any user of Delphi 2 can activate this AddIn expert and have a look at the error report database (which could be shared on a network).

As I've said, the expert skeleton is basically the same. The only

➤ *Figure 1*

➤ *Listing 7*

```
constructor TBListMenuExpert.Create;
var Main: TIMainMenuIntf;
    MenuItems: TIMenuItemIntf;
    ToolsTools: TIMenuItemIntf;
    Tools: TIMenuItemIntf;
    i,j: Integer;
    f: System.Text;
begin
  inherited Create;
  if ToolServices <> nil then
  try
    Main := ToolServices.GetMainMenu;
    if Main <> nil then { we've got the main menu }
    try
      MenuItems := Main.GetMenuItems;
      if MenuItems <> nil then
      try
        System.Assign(f,'C:\MENU');
        System.Rewrite(f);
        writeln(f,MenuItems.GetName,' -',MenuItems.GetItemCount);
        for i:=0 to Pred(MenuItems.GetItemCount) do begin
          Tools := MenuItems.GetItem(i);
          if Tools <> nil then { we've got a sub-menu }
          try
            writeln(f,'  ',Tools.GetName);
            for j:=0 to Pred(Tools.GetItemCount) do begin
              ToolsTools := Tools.GetItem(j);
              if ToolsTools <> nil then { sub-sub-menu }
              try
                writeln(f,'    ',ToolsTools.GetName);
              finally
                ToolsTools.DestroyMenuItem
              end
            end
          finally
            Tools.DestroyMenuItem
          end
        end
      finally
        System.Close(f);
        MenuItems.DestroyMenuItem
      end
    finally
      Main.Free
    end
  except
    HandleException
  end
end {Create};
```



➤ *Figure 2*

routines needing modification are `Create` and `OnClick`, in which we show a form of type `TReportForm`, for which you can find full source code on this issue's disk (Listing 8).

Note the use of exceptions in the `OnClick` event. Actually, things get even more complicated, since all three AddIn experts that I've shown so far are in fact DLL experts and are installed using the registry instead of CMPLIB32.DCL.

DLL Experts

The easiest way to write an expert is to treat and install it as a regular component: inside CMPLIB32.DCL (see my expert article in Issue 3). However, Delphi also allows DLL experts. As an example, you should check out EXPTDEMO.DLL which contains the Dialog and Application experts that are shipped with the standard version of Delphi.

When writing DLL experts for Delphi, it is very important to make sure that all exceptions that may be raised inside the DLL are also handled within the same DLL (they have no meaning outside the DLL, as the DLL may be servicing other non-Delphi applications and even if another Delphi application will be calling the DLL, the code/data segment would be set all wrong to be able to handle the exception correctly. An access violation would result). So, we need to be sure to use a `try-except` block around all our code in the DLL (Listing 9).

Two new functions also need to be defined in the DLL versions of

```
procedure HandleException;
begin
  if Assigned(ToolServices) then
    ToolServices.RaiseException(ReleaseException)
end {HandleException};
{ the code inside the DLL will look as follows: }
try
  { do the old stuff... }
except
  HandleException
end
```

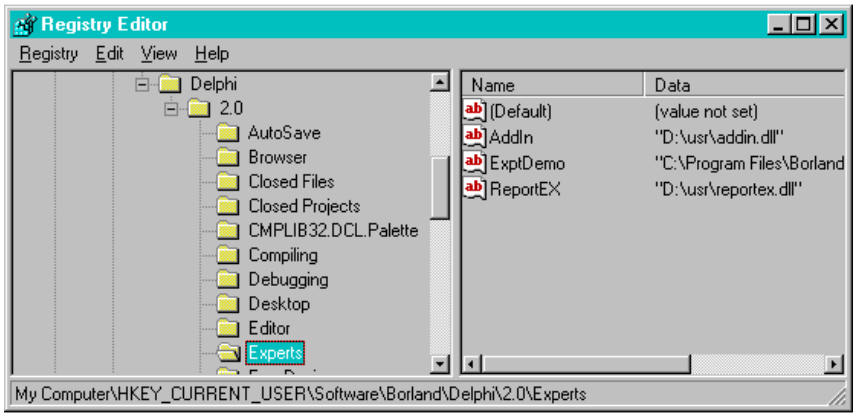
➤ *Above: Listing 9*

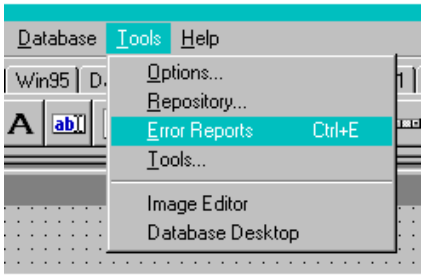
➤ *Below: Listing 10*

```
procedure DoneExpert; export;
begin
  { WEP && cleanup }
end;
function InitExpert(ToolServices: TIToolServices; RegisterProc:
  TExpertRegisterProc;
var Terminate: TExpertTerminateProc):
    Boolean; stdcall;
begin
  ExptIntf.ToolServices := ToolServices; { Save! }
  if ToolServices <> nil then
    Application.Handle :=
      ToolServices.GetParentHandle;
  Terminate := DoneExpert;
  if (@RegisterProc <> nil) then
    Result := RegisterProc(TBReportExpert.Create)
end {InitExpert};
exports
  InitExpert name ExpertEntryPoint;
```

➤ *Below: Listing 8*

```
constructor TBReportExpert.Create;
var Main: TIMainMenuIntf;
    ToolsTools: TIMenuItemIntf;
    Tools: TIMenuItemIntf;
begin
  inherited Create;
  MenuItem := nil;
  if ToolServices <> nil then begin
    Main := ToolServices.GetMainMenu;
    if Main <> nil then { we've got the main menu }
    try
      ToolsTools :=
        Main.FindMenuItem('ToolsGalleryItem');
      if ToolsTools <> nil then
      { we've got the " Tools | Tools" item }
      try
        Tools := ToolsTools.GetParent;
        if Tools <> nil then { we've got the Tools menu }
        try
          MenuItem := Tools.InsertItem(
            ToolsTools.GetIndex+1, '&Error Reports',
            'BERT','', ShortCut(Ord('E'),[ssCtrl]),0,0,
            [mfEnabled, mfVisible], OnClick)
        finally
          Tools.DestroyMenuItem
        end
      finally
        ToolsTools.DestroyMenuItem
      end
    finally
      Main.Free
    end
  end
end {Create};
procedure TBReportExpert.OnClick(Sender: TIMenuItemIntf);
begin
  with TReportForm.Create(Application) do
  try
    ShowModal
  finally
    Free
  end
end {OnClick};
```





➤ *Left: Figure 3*

➤ *Above: Figure 4*

our AddIn experts: `InitExpert` and `DoneExpert`, defined as in Listing 10.

Installation

To install DLL experts you need to use the Windows 95 Registry. Simply start up REGEDIT and go to the

```
MyComputer\HKEY_CURRENT_USER\Software\
Borland\Delphi\2.0\Experts
```

section where you need to add a new key with any name you wish and as the value the place where to find the DLL expert.

After we've installed the AddIn experts, we can start Delphi 2 up again and check the Delphi IDE Menu. Check the Tools menu for the Error Report expert and indeed it's there (Figure 4). But of course, you're now itching to start writing your own AddIn experts, which is why I should stop writing right now and give you time to experiment...

Conclusions

For more information on writing Delphi Experts, I suggest the book *Revolutionary Guide to Delphi 2.0*, published by WROX Press which contains a rather big chapter on the Delphi 2.0 Open Tools API in general and experts in particular.

[Now I wonder who wrote that chapter...?! Editor]

Next time, we'll get back to component building, testing and debugging, looking at some really interesting techniques and tools!

Bob Swart (http://www.pi.net/~drbob/) is a full-time professional knowledge engineer for Bolesian in The Netherlands and a freelance technical author. In his spare time, Bob likes to watch video tapes of Star Trek Voyager and Deep Space Nine with his 2.5-year old son Erik Mark Pascal.