Integrating With Outlook And Exchange

by Berend de Boer

In this article I investigate how you can write programs with Delphi 3 which integrate with Microsoft Outlook and Exchange Client. The focus will be on Outlook. When something doesn't work with Exchange Client it is explicitly mentioned.

There are four different types of Exchange/Outlook client extensions. Firstly there are *command extensions*: these add new commands to Outlook's menubar or toolbar. Secondly, *event extensions* enable developers to add to or override behaviour, such as the arrival of new messages, reading and writing messages, sending messages, reading and writing attached files and tracking selection changes in a window. Next we have *property sheet extensions*: useful for displaying custom form property sheets. Finally, *Advanced criteria extensions* extend Outlook's search capabilities.

In this article I'll cover the first client extension type. I'll show you how to add your menu items to Outlooks menubar, and how to add buttons to its toolbar. I'll also show something of Outlook's programming model. With Outlook it's quite easy to get information, like which folders exist, the subjects or receive dates of messages and so on. And best of all, without having to resort to MAPI.

These days, everything is COM in the Microsoft World and that's true for Outlook too. So to integrate with Outlook we need to implement a COM interface, the `IExchExt` interface. As you see from the naming, this interface is inherited from the Exchange Client days so a lot will work under Exchange too. Outlook has one big advantage over Exchange: accessing folders and mail, subject, recipients, etc, is a lot easier than under Exchange.

```
function TExchExt.Install(eecb: IEXCHEXTCALLBACK; mecontext: ULONG; ulFlags:
  ULONG): HResult;
begin
  case mecontext of
    EECONTEXT_VIEWER : Result := S_OK;
  else
    Result := S_FALSE;
  end; { of case }
end;
```

➤ *Listing 1*

First Example

Let's first implement the famous *Hello World* example. We'll add a menu item to Exchange and when you choose it, you will get a *Hello World* dialog box. This example does work with both Outlook and Exchange, so I'll use the word Exchange in this example and mean both Microsoft MAPI clients. There is a minor difference between Outlook and Exchange in this respect. Exchange loads extensions at startup, Outlook can load them when required. This will not affect the examples I present here.

Exchange client extensions live in DLLs. Each DLL needs to have an entry point, a procedure which is exported with the ordinal 1. Exchange calls this procedure when it loads the DLL. The name of this entry point is not important, only its ordinal. I've called it `ExchEntryPoint`. `ExchEntryPoint` needs to return the COM object which implements the extensions. This COM object should implement the `IExchExt` interface (see Table 1). The most simple entry point looks like this:

```
function ExchEntryPoint:
  IExchExt; cdecl;
begin
  Result := TExchExt.Create;
end;
```

Unfortunately, it's not quite as simple as this. Due to a bug in Delphi's compiler we need to write more complicated code, as you can see in the examples.

The `IExchExt` interface has just one method: `Install`. This method returns `True` (or in COM terms `S_OK`) when this extension runs in a certain context. Examples of contexts are the viewer context (that is, Exchange's main window), the remote viewer context, the address book context and the property sheet window context. So if you want to add a menu item you should return `S_OK` when Exchange asks you if you run in the `EECONTEXT_VIEWER` context. Every context has a unique number, assigned by Microsoft, prefixed by `EECONTEXT_`. A simple implementation of `Install` which wants to be active in Exchange main window looks like Listing 1.

So now you have informed Exchange that you can do something in the context viewer window. To actually do something there, you have to implement another interface, the `IExchExtCommands` interface (see Table 2). This interface has some more methods, seven in total. In this first

➤ *Table 1*

IExchExt	
Install	Enables an extension object to determine the context into which it is being loaded, along with information about that context.

example, we only need to implement two of them. The first is `InstallCommands`. In `InstallCommands` you can add menu items to Exchange's or Outlook's menubar. The code looks like that shown in Listing 2.

In the first line we use the Exchange callback interface to retrieve the handle to the `Tools` submenu. The Exchange callback interface can be used to retrieve information about Exchange's current state: see Table 3 for its interface. In the next lines we append the menu item `Hello World 1` to this menu. You can use the standard Win32 API command `AppendMenu` to accomplish this. You may not choose any command number you want. Exchange passes the first command you want to use in `cmdidBase`. If you have added a menu item you must increment it, so Exchange knows that you did indeed add something. Outlook even deletes menu items you have added, if you didn't increment `cmdidBase` properly! I store the command number my menu item is bound to in `MyCommandNum`. We will need it later.

The menu item is now shown, but nothing happens when a user selects it. When a user chooses a menu item (or toolbar button), `DoCommand` is called with the command number. You need to implement `DoCommand` to add functionality to your menu items. An example `DoCommand` implementation is shown in Listing 3.

As you see I've used the command number I'd saved in `MyCommandNum` to see if it really is my command. You need to return `S_OK` if you did handle a command, else return `S_FALSE`.

For this first example I've implemented two more methods: `QueryHelpText` and `Help`. `QueryHelpText` shows status line help in the Exchange Client (not within Outlook) and `Help` is called when a user chooses `What's this` and selects your menu item or toolbar button. Both get the command which is selected and you use an If statement similar to the one shown in `DoCommand` to see if it really is your command.

```
function TExchExt.InstallCommands(
  eecb: IEXCHEXTCALLBACK;
  hwnd: HWND;
  hmenu: HMENU;
  var cmdidBase: UINT;
  lptbeArray: LPTBENTRY;
  ctbe: UINT;
  ulFlags: ULONG): HResult;
var
  r: HResult;
  hMenuTools: Windows.HMENU;
begin
  r := eecb.GetMenuPos(EECMDID_ToolsOptions, hMenuTools, nil, nil, 0);
  // add our extension command
  MyCommandNum := cmdidBase;
  AppendMenu(
    hMenuTools,
    MF_BYPOSITION and MF_STRING,
    MyCommandNum,
    'Hello World 1');
  Inc(cmdidBase);
  Result := S_OK;
end;
```

➤ *Listing 2*

IExchExtCommands	
InstallCommands	Enables an extension to install its menu commands or toolbar buttons.
InitMenu	Enables an extension to update its menu items when the user begins using the menus. You could for example enable/disable menu items.
DoMenu	Carries out a menu or toolbar command chosen by the user.
Help	Provides user help for a command. It's called when a user chooses What's this from the Help menu and clicks the menu item or toolbar button.
QueryHelpText	Provides status bar or tool tip Help text for a command. Only visible with the Exchange Client, not with Outlook.
QueryButtonInfo	Provides information about the extension's toolbar buttons.
ResetToolbar	Enables an extension to restore its toolbar buttons to their default positions.

➤ *Table 2*

```
function TExchExt.DoCommand(eecb: IEXCHEXTCALLBACK; cmdid: UINT): HResult;
begin
  if cmdid = MyCommandNum then begin
    ShowMessage('Hello World 1!');
    Result := S_OK;
  end else
    Result := S_FALSE;
end;
```

➤ *Listing 3*

The final step is to register your DLL within Exchange Client or Outlook. Make sure there is a registry entry

```
HKEY_LOCAL_MACHINE\SOFTWARE\
  Microsoft\Exchange\Client\
  Extensions
```

Sometimes the `Extensions` key is not present, so you need to create it. Next add a registry value to the `Extensions` key. The name of the value is your description. The value itself looks like this:

```
4.0;e:\winnt\system32\
  helloworld1.DLL;1;010000;1000000
```

The value is a semicolon separated

IExchExtCallback	
GetVersion	Returns the version number of the Microsoft Exchange application.
GetWindow	Returns a window handle corresponding to the specified flag.
GetMenu	Returns the Microsoft Exchange menu handle for the current window.
GetToolbar	Returns a toolbar's window handle.
GetSession	Returns an interface to the current open MAPI session and associated address book.
GetObject	Returns an interface and store for a particular object.
GetSelectionCount	Returns the number of objects selected in the window.
GetSelectionItem	Returns the entry identifier of a selected item in a Microsoft Exchange window.
GetMenuPos	Returns the position of a command or set of commands on the Microsoft Exchange menu.
GetSharedExtsDir	Returns the Microsoft Exchange shared-extensions directory.
GetRecipients	Returns a pointer to the recipient list of the currently selected item.
SetRecipients	Sets the recipient list for the currently selected item.
GetNewMessageSite	Returns interface pointers to the message site and view context of the selected message.
RegisterModeless	Enables extension objects that display modeless windows to coordinate with windows displayed by the Microsoft Exchange client.
ChooseFolder	Displays a dialog box that enables users to choose a specific message store and folder.

➤ *Table 3*

file. The first value is the Exchange Client level you want to be active in. The value can be either 4.0 or 5.0. Next comes your DLL location. The third field contains the ordinal of the exported function which Exchange should call. This value can be left empty, it defaults to 1. This is the ordinal of our `ExchEntryPoint` function.

The bit values in position 4 and 5 are optional. Field 4 is called the `ContextMap`, field 5 the `InterfaceMAP`. The `ContextMAP` tells Exchange in which contexts your DLL is active, so Exchange can load the DLL only when needed, see Table 4. The `InterfaceMAP` tells Exchange which interfaces you have implemented, see Table 5. Exchange is able to find out all this, but telling it beforehand could improve performance.

Outlook has an additional way to register extensions. They are called Extension Configuration files (.ECF). I suggest you stay away from them as they are not well documented and do not really add anything important. But if you want to give it a try, take a look at

```
www.microsoft.com/msdn/news/
  feature/032598/ecf.htm.
```

To summarize: the steps you need to take to implement an Exchange Client extension are as follows. First, create a DLL with the `ExchEntryPoint`; this DLL should return a pointer to the `IExchExt` interface. Secondly, implement the `IExchtExt` interface. Thirdly, implement the `IExchExtCommands` interface. Lastly, register your DLL so Outlook or Exchange Client knows about it.

You can find the complete implementation on the companion disk in project `HelloWorld1`, our DLL, and `ComHelloWorld1`, the implementation of `IExchExt` and `IExchExtCommands`. The implementation could be quite simple because Delphi allows us to implement more than one interface in an object (no need to implement `QueryInterface` for example).

Second Example

In the second example I will expand a bit on the previous example. I'll show you how to add a button to the toolbar and how to add a menu item in a different context. The most interesting thing, however, is that I will give you a glimpse of how you can program Outlook. This example, except adding the button and menu items,

➤ *Table 4: Context Map Bit Positions*

Position	Context	Position	Context
1	EECONTEXT_SESSION	8	EECONTEXT_SENDPOSTMESSAGE
2	EECONTEXT_VIEWER	9	EECONTEXT_READPOSTMESSAGE
3	EECONTEXT_REMOTEVIEWER	10	EECONTEXT_READREPORTMESSAGE
4	EECONTEXT_SEARCHVIEWER	11	EECONTEXT_SENDRESENDMESSAGE
5	EECONTEXT_ADDRBOOK	12	EECONTEXT_PROPERTYSHEETS
6	EECONTEXT_SENDNOTEMESSAGE	13	EECONTEXT_ADVANCEDCRITERIA
7	EECONTEXT_READNOTEMESSAGE	14	EECONTEXT_TASK

is specific for Outlook, you cannot use it with Exchange.

Outlook's COM interface is quite easy to use. You can find it in the type library MSOUTL8.OLB. On the disk with this issue you will find it as the file Outlook_TLB.pas. Every item in Outlook is covered by a separate object. The application as a whole is covered by the `Application` object. A mail message is covered by the `MailItem` object, a contact is covered by the `ContactItem` object. Outlook objects have visual and non-visual methods. Many Outlook objects, for example `MailItem` and `NoteItem` have a `Display` method to display a certain message of their class.

You can completely program Outlook not only from within your Exchange Extension, but also from any OLE Automation controller. This is quite useful to debug parts of your Exchange Extension as it is easier to step through code within a Delphi form than using the external debugger to step through an Exchange Extension.

As an example, let's create a new entry in the `Contacts` folder (see Listing 4).

Let's take a look at the source code for the second example. Example2.dpr is the DLL, and `ComHelloWorld2` is the implementation of the `IExchExt` and `IExchExtCommands`. Because this extension should run only within Outlook, `IExchExt.Install` has a check to see if it's called from Outlook.

Adding menu items is done as in the first example. One thing is different: I add a menu item only to the `New Message` window, not to Outlook's main window. To add a menu item (and toolbar button) only in the `New Message` window, and not in Outlook's main window, I store the current context within `IExchExt.Install`. When `IExchExtCommands.InstallCommands` is called next, I use it to decide if I have to add the menu item or not.

Adding a button to the toolbar is a two step process. First make the button image available, and second install the button. Making the button image available is done when `IExchExtCommands.InstallCommands` is called. In the main resource file of example 2, Example2.RES, I've added two button images with the names `101` and `102`. Within `InstallCommands` I use these names to add the proper image to the toolbar list of images. Reserving a command number, incrementing the `cmdidBase` parameter, is done within `InstallCommands`.

After that `IExchExtCommands.QueryButtonInfo` is called with various parameters, the most important is `ptbb`, a pointer to a `PTBButton` structure. The properties of `ptbb` have to be set to display the button.

`IExchExtCommands.DoCommand` contains the actual implementation of manipulating Outlook. I've written three examples. In Outlook's main toolbar a new button has appeared with the number 1 on it. If you click it, the subjects of all messages in the current folder are written to the field dump.txt in your temporary directory. In the new message window a new button has appeared with the number 2 on it. If you click this one a journal item of type `E-mail message` is created and displayed. You can save or cancel it. As the last item under the `Tools` menu of the new message window you find the `Create a contact` item. If you select it, a new contact is created and saved immediately. Go to your `Contacts` folder to see it.

Registering example2.dll is done similarly to the first example. The correct registration key is:

```
4.0;c:\winnt\system32\
  Example2.DLL;1;010001;1000000
```

➤ *Table 5: Interface Map Bit Positions*

Position	Interface
1	IExchExtCommands
2	IExchExtUserEvents
3	IExchExtSessionEvents
4	IExchExtMessageEvents
5	IExchExtAttachedFileEvents
6	IExchExtPropertySheets
7	IExchExtAdvancedCriteria

```
var
  app: Application;
  ci: ContactItem;
begin
  app := CoApplication.Create;
  ci := app.CreateItem(olContactItem) as _DContactItem;
  ci.FullName := 'Berend de Boer';
  ci.CompanyName := 'NederWare';
  ci.Save;
end;
```

➤ *Listing 4*

Compiling The examples

To be able to compile the examples, you need some libraries on your search path. For the standard Exchange Extensions you need access to the MAPI headers. Inprise didn't translate them, but luckily a good guy named Alexander Staubo provided a very complete translation. You can find his translation at

```
Www.mop.no/~alex/
  technical_delphitrans.html
```

This translation is included on the companion disk (in the mapi subdirectory of the directory for this article).

You also need the header for the Exchange Extension interface, which I've partly translated. You find it also in the mapi subdirectory as ExchExt.pas.

You can translation the outlook object library from MSOUTL8.OLB, included with Office 97, but it's also included with the source for this article as Outlook_TLB.pas in the Outlook subdirectory. This directory also contains two other files you need MSForms_TLB.pas and Office_TLB.pas

The examples are in the Example1 and Example2 subdirectory. Both directories have a

compileit.bat file and a dcc32.cfg file. This should make compiling the sources a breeze.

Conclusion

I hope to have given you a good start in writing Exchange Extensions. We did cover a lot, but there is also a lot we didn't cover. Like writing Event Extensions. In the following resource section some pointers are given which give more information about the subjects we discussed, but also to subjects we didn't.

Information Resources

The main guide for the Exchange Extension programmer is the *Extending the Microsoft Exchange Client* guide. You can find it on the MSDN CD. It's also available online at the time of writing, at

```
Http://premium.microsoft.com/isapi/
  Devonly/prodinfo/msdnprod/
  msdnlib.idc?theURL=/msdn/library/
  Sdkdoc/cx-01_6s1f.htm
```

Also you can find a good FAQ regarding MAPI and Exchange Extension programming at

```
Www.angrygraycat.com/goetter/
  mdevfaq.htm
```

You can get a good understanding of how to things ought to be done when programming Outlook at

```
www.microsoft.com/OutlookDev/
  Articles/Outprog.htm
```

From browsing through Outlook_TLB.pas you get a good understanding of things which are possible with Outlook. There is also a help file, vbaoutl.hlp, which covers all objects and properties. This helpfile can be found on the Office 97 CD in the /VALUPACK/MOREHELP directory.

Berend de Boer is President of Nederware, a software engineering firm in the Netherlands and can be contacted by email at berend@pobox.com