# An Introduction to Outlook 2007 Programming

Outlook stores items such as mail, appointment, task, and contact items in tables in a hierarchically structured database. This is unlike the underlying document object model that most other Office applications use and requires a change of orientation for programmers experienced in programming applications, such as Word or Excel. For this reason, this chapter explains Outlook's data model and introduces Outlook's data storage and data presentation models.

In this chapter, you first open the Outlook VBA project and set it up for use in creating and running macros and prototype code. Using the Outlook VBA project is often the easiest way to quickly test and prototype your Outlook code. Next, you discover the concept of a NameSpace and how to access Outlook data in folders and in individual Outlook items. This is the basis of all Outlook data access and is a building block for all Outlook programming. You next see how Outlook stores its data and how to access that data. In Outlook 2007, you can now access data either with the traditional Folders and Items collections or with the new Stores collection and Table object. The new members of the Outlook object model are explained in Chapter 2. Finally, this chapter discusses Inspectors—the windows that display items such as emails or appointments, as well as Explorers—the windows that display folders. Working with these collections is critical for any Outlook program that works with the Outlook display.

# Setting Up Outlook VBA

Outlook VBA is a very convenient way of testing and prototyping your code. You can quickly write procedures to test various elements in the Outlook object model, and in VBA code you have an intrinsic Outlook Application object that makes it easy to work with Outlook items. Instead of writing your code in a Component Object Model (COM) addin to test something, it's far easier to quickly test in a macro and then place the tested code in your COM addin. In fact, it's even easier to test Outlook forms code, which uses VBScript, in Outlook VBA and then convert the VBA code to VBScript than it is to test the code in an Outlook form.

# Setting Up Macro Security

The first thing when using Outlook VBA is to set up your macro security so that your code will run. By default, Outlook's macro security is set to high, which doesn't run code not signed with a code-signing certificate and which warns you when signed code is present. At a minimum, you should set your security to prompt you when macro code is present.

Never set your security to perform no security checks; that's a very dangerous setting that leaves you with no security at all.

To set security, do the following:

**1.** Select Tools ⇔ Macro ⇔ Security to open the Security dialog. The Security dialog, shown in Figure 1-1, appears.

Trust Center		? ×
Trust Center Trusted Publishers Add-ins Privacy Options E-mail Security Attachment Handling Automatic Download Macro Security Programmatic Access	Macro Security          No warnings and disable all macros         Warnings for signed macros; all unsigned macros are disabled         Warnings for all macros         No security check for macros (Not recommended)	
	ОК Са	ncel

Figure 1-1

- **2.** Set your security:
  - **Warnings for all macros:** Use this if you don't intend to sign your Outlook VBA project.
  - □ **Warnings for signed macros; all unsigned macros are disabled:** If you intend to sign your VBA project, it's recommended that you use this security setting.
- **3.** Click the OK button to save your changes to the macro security and then exit and restart Outlook to apply the new security setting.

After setting up your macro security, you can open the Outlook VBA project by using the keyboard shortcut Alt+F11.

2

# Creating a Code-Signing Certificate

Unlike other Office applications such as Word or Excel, Outlook stores its macros globally and only has one VBA project that you can use. Outlook's VBA project is always named VBAProject.OTM, and signing this code project provides a higher degree of security than just enabling warnings for all macros.

Even if you don't have a standard code-signing certificate from a trusted authority, such as VeriSign or Thawte, you can use the certificate-generating software included with Office 2007, Selfcert.exe, to create a personal signing certificate. You shouldn't use this certificate to distribute code, because it doesn't derive from a trusted authority, but you can use it for your own code on your development computer.

To create a code-signing certificate, follow these steps:

**1.** Navigate to the folder where your Office 2007 applications are installed, usually C:\Program Files\Microsoft Office\Office12 for Office 2007, and run the Selfcert.exe program. Running this application results in the dialog box shown in Figure 1-2.

🖼 Create Digital Certificate	×		
This program creates a self-signed digital certificate that bears the name you type below. This type of certificate does not verify your identity.			
Since a self-signed digital certificate might be a forgery, users will receive a security warning when they open a file that contains a macro project with a self-signed signature.			
Office will only allow you to trust a self-signed certificate on the machine on which it was created.			
A self-signed certificate is only for personal use. If you need an authenticated code signing certificate for signing commercial or broadly distributed macros, you wil need to contact a certification authority.			
Click here for a list of commercial certificate authorities			
Your certificate's name:			
OK Cancel			

Figure 1.2

- **2.** Enter a name for your certificate—just use your own name—and press Enter to create your personal code-signing certificate.
- **3.** Now open the Outlook VBA project using Alt+F11 in Outlook and select Tools  $\Rightarrow$  Digital Signature to open the dialog shown in Figure 1-3. In this dialog, click the Choose button and select the certificate you created using Selfcert.exe, then click OK twice to choose your certificate and sign your code project with that certificate.

Digital Signature	2 🗙
The VBA project is currently signed Certificate name: [No certificate	as]
Sign as Certificate name: Ken Slovak	
Remove	Choose
ОК	Cancel

Figure 1-3

- **4.** Now select File  $\Rightarrow$  Save VBAProject.OTM to save your VBA project; then exit and restart Outlook. If you are prompted again to save your VBA project when Outlook is closing, save the project again.
- **5.** When you open your VBA project using Alt+F11, you are prompted to enable your macros in the dialog shown in Figure 1-4. You can use this dialog to trust your certificate, which prevents this dialog from appearing again, or you can just enable the macros for that Outlook session. It's recommended to trust your certificate so you don't get prompted to trust your VBA code every time you want to run a macro or go into the Outlook VBA project.



Figure 1-4

**6.** Click the Trust All Documents from This Publisher button to trust your code-signing certificate and add it to the trusted publishers list. If you open the Trust Center dialog shown in Figure 1-1 again and click the Trusted Publishers area, you will now see your code-signing certificate listed as a trusted publisher. Now you won't get the security prompts, and your VBA code will run without problems.

# **Reviewing the VBA Editor Interface and Options**

Take a few minutes to explore the menus in the Outlook VBA interface and click Tools  $\Rightarrow$  Options to set up the VBA editor options so that the settings match what you want:

- □ The Require Variable Declaration option: Checking this checkbox is recommended. It inserts an Option Explicit statement in every module you add to your project. Declaring your variables before using them is a valuable tool in making sure that you are typing your variables and objects correctly and is good programming practice.
- □ The Object Browser: Before leaving the Outlook VBA project use the keyboard shortcut F2 to open the Object Browser, and spend a few minutes reviewing the properties, methods and events in the Outlook object model. The Object Browser is the most useful tool you have when writing Outlook code. It shows what is available in the Outlook object model and provides prototypes of all the Outlook properties, methods, and events that show how they are used. Clicking F1 to call Help when any entry in the Object Browser is selected brings up context-sensitive Help about that Outlook object, usually with some sample code to show how to use and access that object.

# **Understanding Outlook**

When you start Outlook, you are logged into an *Outlook session*. An Outlook session starts when Outlook is started and ends when you close Outlook. The data and properties that you have available to you in this session are set by the *Outlook profile* that logged into that Outlook session. Each Outlook profile consists of a data store, email accounts, and other settings that define the Outlook environment. Most users have only one default Outlook profile, but Outlook allows you to set up as many profiles as you want to define different environments in which to run Outlook.

Outlook can only have one session open at a time and can use the properties from only one Outlook profile at a time. To switch to a different Outlook profile, you must close and reopen Outlook.

Figure 1-5 shows the hierarchical structure used to organize access to Outlook data. The Outlook Application opens one session, also known as a NameSpace. The session has a Stores collection, consisting of all open stores in that session. One or more stores may be opened in a session. Each store has a collection of folders, and each folder has a collection of items that make up the data in that folder.



Figure 1-5

# **Outlook Profiles and Data Store**

Each Outlook profile has access to selected data stores, which can be PST files, Exchange mailboxes, Exchange public folders, or special stores known as custom store providers. A profile also has certain email accounts set up, and send/receive groups, which control how certain Outlook features behave, such as sending emails on a deferred basis or sending meeting invitations. Regardless of the profile or profile properties, the Outlook session always opens into a NameSpace, which is always "MAPI":

```
Dim oNS As Outlook.NameSpace
Set oNS = Application.GetNameSpace("MAPI")
```

All of the code snippets in this chapter use VBA, Outlook's macro language. VB.NET and C# code samples in the book show similar code in those languages.

MAPI stands for *Messaging Application Programming Interface*, and Outlook uses MAPI protocols for all of its data accesses, and the Outlook data stores are designed around the MAPI storage interface. Outlook wraps most of the MAPI protocols and data storage in the Outlook object model, so most of the time you don't have to deal directly with the complexities of MAPI.

## **Outlook Sessions and Data Stores**

In any Outlook session, the available data stores, such as PST files, are all contained in the Stores collection of the NameSpace object. Each Store object in the Stores collection provides access to the top level of that data store. You can also access these stores at a lower level from the NameSpace object's Folders collection. Stores are new to the Outlook object model in Outlook 2007.

Each store has its own Folders collection, and each Folder object in those collections can contain its own Folders collections (subfolders). The user interface represents these collections and folders as Outlook folders and subfolders.

#### The Evolution of the MAPIFolder object

In previous versions of the Outlook object model, the Folder object was called the MAPIFolder object. This alias is still accepted in Outlook code, so the forward compatibility of existing code isn't compromised.

The new properties and methods for the Folder object, such as the GetTable() method used to get a Table object for a folder, are also available for the MAPIFolder object, although only for Outlook 2007.

The new events for the Folder object are not added to the MAPIFolder object. If you enhance existing code to use the new Folder events, make sure to convert all declarations of MAPIFolder objects into Folder declarations.

## **Outlook Folders**

Each folder in Outlook is dedicated to containing certain types of items, such as emails, tasks, contacts, or appointments. Folders contain Items collections to hold items that are compatible with the type of

folder (refer to Figure 1-5). There are also special folders such as Inbox and Deleted Items that can hold items of many different types.

The following VBA code snippet illustrates how to use code to access one item from the Outlook Sent Items folder using Outlook VBA code:

```
Dim oNS As Outlook.NameSpace
Dim oFolder As Outlook.Folder 'note: this is new for Outlook 2007
Dim colltems As Outlook.Items
Dim oMail As Outlook.MailItem
Set oNS = Application.GetNameSpace("MAPI")
Set oFolder = oNS.GetDefaultFolder(olFolderSentMail)
Set colltems = oFolder.Items
Set oMail = colltems.Item(1)
```

This code uses the intrinsic Application object that always represents Outlook.Application when running in the Outlook VBA project, or in custom form code. In other contexts, such as VBA code running in a Word macro, an Outlook.Application object needs to be declared and instantiated:

```
Dim oOL As Outlook.Application
```

```
Set oOL = CreateObject("Outlook.Application")
```

Please note the following regarding this code:

- The code instantiates a NameSpace object and uses the GetDefaultFolder() method to retrieve a reference to the Sent Items folder. This folder is declared as a Folder object, new for Outlook 2007.
- □ The Folder object is derived from the MAPIFolder object and adds additional methods, properties and events to that object.
- □ The code then gets the Items collection of the Sent Items folder and returns a reference to one item from the Items collection.

This code is typical of how you access an Outlook item from an Outlook data store, navigating the storage and interface hierarchy to access that item. This code assumes that the first item in Sent Items is an email item, which it may not be because email folders can also hold Post items.

Many Outlook objects such as Application, Inspector, and Explorer have a Session property that returns the NameSpace object and exposes all the properties, methods and events of the NameSpace object. The Session object can be used as an alias for the NameSpace object.

If you use the Session object make sure you set a NameSpace object using the GetNameSpace (MAPI) method once in your code before using the Session object. The Session object isn't fully instantiated until you log in to the NameSpace object at least once, particularly when you start Outlook using automation code.

#### **Outlook Items**

Each Outlook item contains many properties that define the data for that item. Default properties are present in every item and are always defined for an item, such as Subject or Body (the note text in an item). Users can also add properties to items either from code or the user interface. These UserProperties are also called *named properties* and play a prominent role in most advanced Outlook development, often being used to store application-specific data in items.

The following example uses the oMail object instantiated in the previous example:

```
Dim colProps As Outlook.UserProperties
Dim oProp As Outlook.UserProperty
Dim sSubject As String
sSubject = oMail.Subject
If InStr(1, sSubject, "support", vbTextCompare) > 0 Then
Set colProps = oMail.UserProperties
Set oProp = colProps.Item("Processed Date")
If oProp Is Nothing Then 'if Nothing add the user property
Set oProp = colProps.Add("Processed Date", olDateTime)
End If
oProp.Value = #April 21, 2007# 'set a date this item was processed
oMail.Save 'save the change to the MailItem
End If
```

This example tests to see if the Subject of a mail item contains the case-insensitive string "support". If so, it tries to get an existing UserProperty named "Processed Date" from a mail item. If that property doesn't exist, the code adds it to the item. The Value property of "Processed Date" is set, and then the mail item is saved. You can add data in a UserProperty to a view of an Outlook folder, so it's visible to users.

# **Outlook Data Storage**

Programming Outlook is different from programming most other Microsoft Office applications in that it's not centered on documents but rather on individual items stored in databases. Each Outlook item, whether it's an appointment or an email, is really a row in a database with the database columns being individual properties, such as Subject. Outlook disguises the underlying data storage with collections of folders and items, but you still access a database when you work with Outlook data.

Outlook data is stored in PST (Personal Storage) files for users not working with Exchange server, and for archive and offloaded data even for Exchange server users. Other Outlook data stores are OST (Offline Storage) files used in offline and cached mode by Exchange server users and Exchange mailboxes. What these data stores have in common are storage organization and the use of a derivative of the JET database used for Access. Using Open DataBase Connectivity (ODBC) database drivers Outlook items can be accessed as recordsets, although only a subset of the properties in an item is available as columns in the recordset.

Outlook data is organized into Stores, Folders, and Items. Each store, such as a PST file, has a one-tomany relationship to folders, including default folders such as Inbox and Calendar, and nondefault folders

that the user creates. Each Folder object has a Folders collection, which can contain many subfolders, with each of those subfolders able to contain subfolders of its own. Folders have one-to-many relationships with items, such as emails and contacts. Each item has many different properties stored in its row in the folder item table, including properties common to all items, such as Subject, and special properties related to certain items such as the start date and time of an appointment.

# Accessing Data Stores Using NameSpace

When you access an Outlook data store, you do so in one of two ways. The first and familiar way for Outlook programmers is from the NameSpace.Folders collection, the other, new way is to use the NameSpace.Stores collection. The difference is in the returned object and where you end up in the store hierarchy.

When you use the NameSpace.Folders collection, or one of the methods that retrieves a folder from the NameSpace object, such *as* GetDefaultFolder() or GetFolderFromID(), you get a Folder (MAPIFolder) object. If you use NameSpace.Folders.Item(1), you get a Folder that is called the RootFolder or Top of Information Store. This is usually shown as something like "Personal Folders" or "Mailbox - Ken Slovak."

The following VBA examples show how to retrieve a Folder object using some of the various NameSpace methods:

```
Dim oNS As Outlook.NameSpace
Dim oFolder As Outlook.Folder 'note: this is new for Outlook 2007
Set oNS = Application.GetNameSpace("MAPI")
'Get a default Outlook folder
Set oFolder = oNS.GetDefaultFolder(olFolderCalendar) 'get Calendar
'Another way:
Set oFolder = oNS.GetFolderFromID(strFolderID) 'using a string EntryID
'Another way:
Set oFolder = oNS.Folders.Item("Personal Folders") 'get RootFolder
```

The disadvantage of these methods is they can only take you so high in the Outlook folders hierarchy, to the Top of Information Store or RootFolder. There are other interesting Outlook folders, however, that aren't visible to the user but are important to programmers, such as the Reminders and Common Views folders, as well as Search Folders and the new To-Do Bar task list folder. Using the new Stores collection, you can access these hidden folders, as well as properties of the store itself, such as the Out of Office state and the support mask that indicates what types of operations the store supports.

## Accessing Data with Tables

Some of the new features in the Outlook 2007 object model relate to tables, which provide direct readonly access to an Outlook table, such as an Items collection of all items in a folder. These tables provide much faster access to your Outlook data than using other methods, such as a iterating through a folder's Items collection or using a filter on an Items collection. The disadvantage of table operations is that the data is read-only, so if you need to change your returned data, you must include columns with information that lets you retrieve individual items corresponding to the rows that need to be changed.

To retrieve individual items from the Outlook database, whether from tables or Items collections, you make use of one or both of the GUIDs (Globally Unique Identifiers) that uniquely identify all Outlook items in the Outlook data storage.

The first and most important GUID is named EntryID. This GUID uniquely identifies any Outlook item in an Outlook data store, such as a PST file or Exchange mailbox. The other important GUID is StoreID, which is unique to any Outlook data store and which you can use in combination with EntryID to retrieve any Outlook item stored in any opened Outlook data store:

```
Set oMail = oNS.GetItemFromID(strEntryID, strStoreID)
```

where strEntryID and strStoreID are string variables that are storing specific GUIDs for an Outlook item and data store.

# **Outlook Data Display**

Now that you know how Outlook stores and accesses data, you need to understand how Outlook displays its data. Outlook has two primary display collections, the Explorers and Inspectors collections. Outlook also has other user interface objects, such as the familiar menus and toolbars that are composed of the CommandBars collection, the new Ribbon displayed in open items such as emails, and various Panes, such as the Navigation and Reading (preview) Panes. You learn how to use all those interface elements and to create or modify them in Chapter 7, but for now let's concentrate on the Explorers and Inspectors collections.

- □ The Explorers collection contains all open folder views, such as a view of the Inbox or Calendar. Each open folder view is an Explorer object and has a CurrentFolder object, which is the folder whose data is currently being displayed in the Explorer. The currently active Explorer object is also available as the ActiveExplorer object. When you open Outlook as a user, one Explorer object is opened. You can add additional Explorers to the Explorers collection by right-clicking a folder in the Navigation Pane and selecting Open in New Window. When you open Outlook using code, there are no open Explorers unless you explicitly open an Explorer to display the contents of a folder.
- The Inspectors collection contains all open Outlook items. Each time you open an Outlook item, such as an email, a contact, or an appointment item, you are adding an Inspector to the Inspectors collection. Logically, the currently active Inspector object is available as the ActiveInspector object.

The distinction between Inspectors and Explorers is fundamental to Outlook programming, and they are often confused by new Outlook programmers. If you want to access a folder being displayed in Outlook, use the Explorers collection. If you want to access open Outlook items, use the Inspectors collection. You cannot access the display of a folder without using an Explorer, and you can't access any open Outlook items without using an Inspector.

To change the subject of a newly opened email, use the following code to access the ActiveInspector and the item being displayed in the Inspector:

Dim oMail As Outlook.MailItem Dim oInsp As Outlook.Inspector

```
Set oInsp = Application.ActiveInspector
If oInsp.CurrentItem.Class = olMail Then ' check for mail item first
Set oMail = oInsp.CurrentItem
oMail.Subject = "Test Message"
oMail.Save 'save the change
Else
MsgBox "This is not a mail item"
End If
```

This code first checks the ActiveInspector to make sure that the currently displayed item in the Inspector is a mail item, and if it is it changes the Subject to "Test Message." If the item is not a mail item, it displays a message box describing the problem.

The following code shows how to use the current ActiveExplorer object to first display the default Contacts folder and then change the view of that folder to the Business Cards view:

```
Dim oNS As Outlook.NameSpace
Dim oExpl As Outlook.Explorer
Dim oFolder As Outlook.Folder
Dim oView As Outlook.View
Set oNS = Application.GetNamespace("MAPI")
Set oExpl = Application.ActiveExplorer
Set oFolder = oNS.GetDefaultFolder(olFolderContacts)
Set oExpl.CurrentFolder = oFolder
oExpl.CurrentView = "Business Cards"
```

The code gets the default Contacts folder using the GetDefaultFolder method, using the olFolderContacts constant, and then sets the CurrentFolder property object of the Explorer to the Contacts folder. The Explorer's CurrentView property is then set to "Business Cards."

# Summary

In this chapter, you learned how to configure the Outlook VBA project, so you can use it without warning prompts and so your VBA code will run. You also learned:

- How to access Outlook items using the Outlook object model.
- How Outlook organizes its data storage.
- □ The differences between Explorers and Inspectors and how they are used to display Outlook objects.

In the next chapter, you learn about some of the most important new features of the Outlook object model and how they are used to display Outlook objects.

49945c01.qxd:WroxPro 8/27/07 4:53 PM Page 12

Æ

Æ