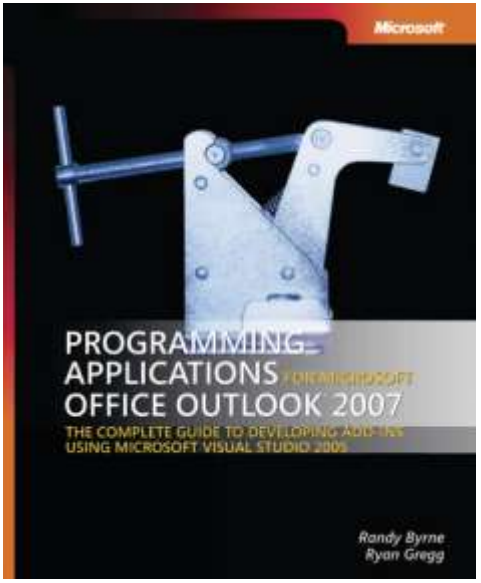

Programming Applications for Microsoft® Office Outlook® 2007

Randy Byrne; Ryan Gregg

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/9179.aspx

9780735622494
Publication Date: February 2007

Microsoft Press

Table of Contents

Foreword xxi
Acknowledgments xxv
Introduction xxvii
Why We Wrote This Book xxvii
Who This Book Is For xxviii
How This Book Is Organized xxviii
Part I: Introducing Microsoft Office Outlook 2007 xxviii
Part II: Quick Guide to Building Solutions xxix
Part III: Working with Outlook Data xxix
Part IV: Providing a User Interface for Your Solution xxix
Part V: Advanced Topics xxx
Sample Code on the Web xxx
Code Snippets xxxiii
Building the Sample Add-Ins xxxiii
System Requirements xxxv
Support for This Book xxxv

Part I **Introducing Microsoft Office Outlook 2007**

1 What's New in Microsoft Office Outlook 2007 3
Form Regions 4
Security 6
Table Object 7
Improved Search 8
Enhanced Events 9
AddressEntry Enhancements 11
SelectNamesDialog Object 11
ExchangeUser and *ExchangeDistributionList* Objects 12

Sharing Objects 12
Rules Objects 13
PropertyAccessor Object 14
PropertyAccessor Sample Code 14
Developer Reference 16
Summary 17

2 Outlook as a Platform 19

Why Integrate with Outlook? 19
Different Types of Outlook Integration 21
Data Integration 22
Functional Integration 24
Integration Guidelines 26
Data Integration 26
Business Logic 29
User Interface Integration and Data Presentation 30
InfoPath Forms 38
APIs 40
Architecture 40
Outlook Object Model 41
Form Regions 43
MAPI as a Platform Component 45
Outlook 2007 Integration API Reference 51
Simple MAPI 51
Deemphasized and Phased-Out Components 52
Development Tools 53
Visual Basic for Applications 53
Visual Studio Tools for Office 54
Managed Versus Native Code 55
Add-In Model 56
Summary 57

Part II Quick Guide to Building Solutions

3 Writing Your First Outlook Add-in Using Visual Basic .NET 61

Introducing the Instant Search Add-In 61
Install the Outlook Add-in Templates 62

Creating the Instant Search Add-In 63
Writing Code 65
The *InitializeAddin* Method 66
Turn *Option Strict* On 67
Adding Instance Variables 67
Hooking Up Events in Visual Basic 68
ItemContextMenuDisplay Event 68
ContextMenuClose Event 70
The *DisplayInstantSearchExplorer* Method 71
Writing Code for Submenu *Click* Events 72
Building the Add-in Project 73
Creating a Shim Project 74
Creating a Setup Project 78
Building the Setup Project 81
Installing the Instant Search Add-In 81
Testing the Instant Search Add-in Solution 82
What to Expect 82
Troubleshooting 82
Debug Mode 83
Debugging Code 84
Summary 85

4 Writing Your First Outlook Add-in Using C# 87

Introducing the Instant Search Add-In 87
Install the Outlook Add-in Templates 88
Creating the Instant Search Add-In 89
Writing Code 91
InitializeAddin Method 92
Adding Instance Variables 93
Hooking Up Events in Visual C# 94
ItemContextMenuDisplay Event 94
ContextMenuClose Event 97
Cleaning Up Event Handlers 98
DisplayInstantSearchExplorer Method 98
Writing Code for Submenu *Click* Events 99
Building the Add-in Project 101
Creating a Shim Project 101

Creating a Setup Project . . . 105
Building the Setup Project . . . 108
Installing the Instant Search Add-In . . . 108
Testing the Instant Search Add-in Solution . . . 109
What to Expect . . . 109
Troubleshooting . . . 109
Debug Mode . . . 110
Debugging Code . . . 111
Summary . . . 112

Part III **Working with Outlook Data**

5 Built-in Item Types . . . 115

Introduction to Built-in and Custom Item Types . . . 115
Understanding *MessageClass* . . . 117
Built-in vs. Custom Types . . . 118
Creating an Item . . . 118
MailItem, *PostItem*, and *SharingItem* Objects . . . 122
Appropriate Uses of *MailItem* and *PostItem* . . . 123
Compose *MailItem* . . . 123
Read *MailItem* . . . 132
Adding an Electronic Business Card . . . 136
Create a To-Do Item . . . 137
PostItem Object . . . 139
Creating a *PostItem* . . . 140
Responding to a *PostItem* . . . 140
AppointmentItem Object . . . 140
Appropriate Uses of *AppointmentItem* . . . 141
One-Time Appointments . . . 141
All-Day Events . . . 143
Appointment Attendees . . . 144
Recurring Appointments . . . 146
MeetingItem Object . . . 154
ContactItem Object . . . 158
Appropriate Uses of *ContactItem* . . . 158
Working with Contact Properties . . . 158
Electronic Business Cards . . . 160

TaskItem Object 162
Appropriate Uses of *TaskItem* 162
Creating a Recurring Task 162
Delegating a Task 163
TaskRequestItem Object 163
Working with Task Requests 164
Other Item Types 166
DistListItem Object 166
JournalItem Object 167
NoteItem Object 167
StorageItem Object 168
Summary 170

6 Accessing Outlook Data 171

An Overview of Outlook Data Storage 171
Exchange Server 171
Personal Folder Files (.pst) 173
Custom Store Providers 173
Accounts Collection and *Account* Object 173
Stores Collection and *Store* Object 174
Stores Collection 174
Adding or Removing a *Store* Programmatically 175
Working with the *Store* Object 176
Folders Collection and *Folder* Objects 178
An Overview of Folder Types 178
Folders Collection 180
Folder Object 182
Working with the *Folder* Object 183
Folder Properties and Methods 187
Folder Permissions 191
Assigning Folder Permissions 192
Assigning Roles 193
Using the *SharingItem* Object to Assign Folder Permissions 194
Accessing Items in a Folder 194
Performance Considerations 194
OutlookItem Helper Class 195
Items Collection 196

Table Object 201
Summary 214

7 Address Books and Recipients 215

An Overview of Outlook Address Books 215
Exchange Global Address List 215
Exchange Containers 216
Offline Address Book 216
Outlook Address Book 217
Other Address Book Providers 217
The *Recipients* Collection and *Recipient* Objects 218
Outlook Object Model Guard Considerations 218
The *CreateRecipient* Method 218
Working with the *Recipients* Collection Object 220
Obtaining the SMTP Address of a Recipient 223
The *AddressLists* Collection and *AddressList* Objects 224
Enumerating *AddressList* Objects 224
The *AddressListType* Property 224
Determining Resolution Order of Address Lists 225
Finding a Specific *AddressList* Object 225
Determining the Contacts Folder for a Contacts Address Book 226
The *AddressEntries* Collection and *AddressEntry* Object 227
The *AddressEntryUserType* Property 228
Finding a Specific *AddressEntry* Object 229
The *GetAddressEntryFromID* Method 229
Displaying *AddressEntry* Details 231
Getting Availability Information for a User 232
The *ExchangeUser* Object 234
Working with *ExchangeUser* Properties 234
Obtaining an *ExchangeUser* Object from an *AddressEntry* Object 235
The *GetExchangeUserManager* Method 236
The *GetDirectReports* Method 236
The *GetMemberOfList* Method 237
Obtaining Proxy Addresses for an *ExchangeUser* Object 238
The *ExchangeDistributionList* Object 238
The *GetExchangeDistributionListMembers* Method 239
The *GetMemberOfList* Method 240

The *GetOwners* Method 240
The *SelectNamesDialog* Object 240
Using the *SetDefaultDisplayMode* Method 241
Dialog Caption and Recipient Selectors 242
Setting the *InitialAddressList* Property 243
Displaying the Select Names Dialog Box 245
Using *SelectNamesDialog.Recipients* 245
Summary 246

8 Responding to Events 247

Writing Event Handlers in Managed Code 247
Hooking Up Events in Visual Basic .NET 249
Hooking Up Events in C# 251
Outlook 2007 Events 254
Application Object Events 254
Explorers Collection Event 259
Explorer Object Events 262
Folders Collection Events 264
Folder Object Events 264
FormRegion Object 265
Inspectors Collection Event 265
Inspector Object Events 268
Items Collection Events 269
Item-Level Events 270
Namespace Object Events 274
NavigationGroups Collection Events 275
NavigationPane Object Event 275
OutlookBarPane Object Events 275
OutlookBarGroup Object Events 276
OutlookBarShortcut Object Events 276
Stores Collection Events 277
SyncObject Object Events 278
Reminders Collection Events 278
Views Collection Events 279
Summary 280

9 Sharing Information with Other Users 281

Outlook and Shared Data 281

Sharing in iCalendar Format 281
Sharing a Calendar Through E-Mail 282
Saving a Calendar to Disk. 283
Saving an Appointment to Disk 284
Opening an iCalendar File 285
Subscribing to Shared Folders 286
RSS Feeds 286
SharePoint Folders. 287
Internet Calendars 289
Using the *SharingItem* Object 290
SharingItem Types 291
Sharing a Folder with a Sharing Invitation 291
Requesting Folder Access with a Sharing Request. 292
Processing a Sharing Item 293
Summary 295

10 Organizing Outlook Data. 297

How Outlook 2007 Helps to Organize Information 297
The *Categories* Collection and *Category* Objects 297
Creating a Category 299
Assigning One or More Categories to an Item 299
Displaying the Categories Dialog Box 300
Task Flagging 301
Controlling Visibility of the To-Do Bar 301
Creating To-Do Items That Appear in the To-Do Bar. 302
The *Rules* Collection and *Rule* Objects 303
Overview of Rules Programming 303
Rules Collection 306
The *Rule* Object 310
The *RuleActions* Collection. 312
The *RuleConditions* Collection. 314
Get or Set Action or Condition Properties with an Array 317
Rules Sample Add-In. 318
Search Folders 319
When to Use a Search Folder. 319
Enumerating Search Folders 320
Creating a Search Folder Programmatically 321

Outlook Views . . . 325
Objects That Derive from the *View* Object . . . 325
Adding or Removing a View Programmatically . . . 326
Customizing Your View . . . 327
Specifying Fields in a View . . . 327
Filtering Items in the *View* Object . . . 329
Sorting Items in a View . . . 329
The *AutoFormatRules* Collection . . . 330
Summary . . . 334

11 Searching Outlook Data . . . 335

Overview of Searching Data . . . 335
Outlook Query Languages . . . 335
AQS . . . 337
DASL . . . 342
Date-Time Comparisons . . . 354
Filtering Recurring Items in the Calendar Folder . . . 354
Date-Time Format of Comparison Strings . . . 355
Time Zones Used in Comparison . . . 356
Conversion to UTC for DASL Queries . . . 357
Integer Comparisons . . . 358
Invalid Properties . . . 359
Jet . . . 359
DASL . . . 359
Comparison and Logical Operators . . . 360
Comparison Operators . . . 360
Logical Operators . . . 360
Null Comparisons . . . 361
Search Entry Points . . . 361
Search Considerations . . . 364
Performance . . . 364
Read-Only vs. Read/Write . . . 365
Searching Subfolders . . . 366
Windows Desktop Search . . . 366
Summary . . . 367

Part IV Providing a User Interface for Your Solution

12 Introducing the Outlook User Interface 371

Decoding the User Interface 371
The Explorer Window (The *Explorer* Object) 372
Programming the *Explorer* Object. 373
The *Explorers* Collection 373
The Inspector Window (The *Inspector* Object) 377
Programming the *Inspectors* Collection and *Inspector* Object 378
The *Inspectors* Collection 378
Working with the Navigation Pane. 380
Making the Most of Navigation Modules 380
Adding Structure with Navigation Groups 382
Removing Folders 384
Folder Views 385
The Reading Pane 385
Customizing the Reading Pane 385
The To-Do Bar. 386
Command Bars. 386
Context Menus. 386
Folder Home Pages 389
Summary 390

13 Creating Form Regions. 391

Introduction to Form Regions 391
Form Pages Compared with Form Regions 392
Form Region Types 392
Standard Form Types. 395
Anatomy of a Form Region Solution 396
Becoming Familiar with Form Region Design 396
Designing a Form Region. 397
Adding Controls. 399
Working with Fields. 403
Polishing Your Form Region. 406
Form Region End to End. 411
Step 1: Creating a Form Region. 411
Step 2: Writing Business Logic. 415

Step 3: Registering the Form Region 423
Advanced Form Region Methods 433
Summary 434

14 Form Region Controls 435

Standard Controls 435
The Outlook Check Box 435
The Outlook Combo Box 435
The Outlook Command Button 436
The Outlook Label Control 437
The Outlook List Box 437
The Outlook Option Button 437
The Outlook Text Box 438
Outlook-Specific Controls 438
The Outlook Body Control 438
The Outlook Business Card Control 438
The Outlook Category Control 439
The Outlook Contact Photo Control 440
The Outlook Date Control 441
The Outlook Frame Header Control 441
The Outlook InfoBar Control 442
The Outlook Page Control 443
The Outlook Recipient Control 444
The Outlook Sender Photo Control 444
The Outlook Time Zone Control 445
The Outlook Time Control 446
The Outlook View Control 447
Using Form Region Controls 447
Adding Controls to the Control Toolbox 447
Adding Controls Programmatically 448
Programmatic Access to Controls 450
Summary 452

15 Extending the Ribbon 453

Introducing Ribbon Extensibility 453
What Happens with Existing Code 454
Outlook RibbonX Sample Add-In 458
Installation Instructions 458

Running the Sample Add-In 459
Modifying Your Code to Use RibbonX 459
Authoring Ribbon XML 461
IRibbonExtensibility Interface 462
Detecting Errors 465
NewInspector Event 466
OutlookInspector Class 467
IRibbonUI Object 468
IRibbonControl Object 468
Summary 470

16 Completing Your User Interface 471

Custom Task Panes 471
When to Use a Custom Task Pane 472
Implementing a Custom Task Pane 472
Adding a Custom Task Pane in an Add-In 475
Custom Property Pages 478
Designing a Custom Property Page 479
Summary 486

Part V Advanced Topics

17 Using the *PropertyAccessor* Object 489

Scenarios for *PropertyAccessor* 489
Objects That Implement *PropertyAccessor* 490
PropertyAccessor Namespaces 491
Obtaining a Specific *SchemaName* String 491
Type Specifiers 492
The *Proptag* Namespace 492
Named Property *ID* Namespace 493
Named Property *String* Namespace 494
Office Namespaces 495
DAV Namespaces 496
The *PropertyAccessor* Object 497
The *GetProperty* Method 497
The *SetProperty* Method 498
The *GetProperties* Method 499
The *SetProperties* Method 500

The *DeleteProperty* Method 501
The *DeleteProperties* Method 501
Date-Time Properties 502
Multivalued Properties 502
Helper Methods 503
Detecting and Reporting Error Conditions 505
Property Size Limitations 506
Summary 507

18 Add-in Setup and Deployment 509

Creating a Setup Project 509
Writing Required Keys to the Windows Registry 510
Installing to HKEY_CURRENT_USER 510
Installing to HKEY_LOCAL_MACHINE 510
Registry Keys Required for an Add-In 510
Registry Keys Required for a Form Region 512
Required Installation Components 512
.NET Framework Version 2.0 512
Visual Studio Tools for Office Runtime 513
Primary Interop Assemblies 514
Add-in Assembly and Other Required Components 516
Using a COM Shim 516
Writing Custom Actions 516
Deploying to Users Who Are Not Administrators 517
Summary 517

19 Trust and Security 519

Code Security for Outlook 2007 519
Guard Principles 522
Security Warning Types 523
Detecting Trusted State 525
Trapping Errors 526
Restricted Properties and Methods 526
Trusting Managed Code 531
Trustable Shared Add-Ins 531
Trust Center 532
Administrative Options 535
Group Policy Security for COM Add-Ins 535

Exchange-Brokered Security for COM Add-Ins 536
Configuring a Security Policy. 536
Trusting an Add-In. 537
Form Region Policy 540
Folder Home Page Policy 541
Summary 542

Index. 543

Chapter 2

Outlook as a Platform

The platform is an essential cornerstone that contributes to the success of Microsoft Office Outlook 2007. Outlook won't ever have every single feature that customers ask for. The platform allows independent software vendors (ISVs) to leverage this business opportunity by complementing Outlook functionality not available in the shipped Outlook product. The platform also allows organizations to tailor Outlook to meet their specific needs and implement custom business logic or requirements within Outlook.

The goal of this chapter is to provide an overview of the Outlook platform capabilities with high-level guidance of when to integrate with Outlook and how to accomplish this integration. This chapter contains no code samples and focuses on a top-level view of the Outlook platform. If you are not familiar with the Outlook platform, consider reading this chapter to gain a sense of the Outlook development landscape. If you are familiar with the Outlook platform and want to dive directly into the Outlook object model, add-in construction, and Microsoft Visual Basic .NET or C# sample code, you can skip this chapter and proceed directly to Chapter 3, "Writing Your First Outlook Add-In Using Visual Basic .NET," or Chapter 4, "Writing Your First Outlook Add-In Using C#."

Why Integrate with Outlook?

For users around the globe, Outlook is the information hub they depend on for messaging, time, contact, and information management. Outlook allows the user to prioritize, organize, and search information. Users especially value Outlook's offline capabilities, which allow them to remain connected with work when they're on the go. They often spend many hours daily in Outlook. From an application perspective, Outlook is where users live. The Outlook platform provides ISVs and organizations with the opportunity to extend, enrich, and customize the Outlook experience for these users.

The Outlook platform allows the introduction of entirely new features or adjustment of Outlook's built-in functionality to meet specific needs. Outlook's solution landscape is vast, and here is a quick, although incomplete, glance at solutions that target Outlook:

- Device synchronization applications
- Antivirus, phishing, and spam solutions
- Integration of customer relationship management (CRM) systems, line of business (LOB) applications, workflow, archiving, and document management

- Mail utilities that include attachment compression, content encryption, thread compression, and productivity solutions
- Unified messaging and other forms of communications (fax, voice messaging, video)

One interesting characteristic of a seamlessly integrated Outlook solution is that the customer perceives it as being a native Outlook feature. This user perception is desirable because it helps make Outlook predictable, consistent, and easy to use. Deep integration can be accomplished by adopting Outlook's user experience metaphors and paying attention to customer expectations. For example, it's important for a solution to support offline capabilities, meaning the customer can get work done even when there's no network or limited connectivity. If the solution introduces form customizations, they show up in the Reading Pane and also when the form is opened in a separate window. The associated functionality for these custom forms is exposed in context menus, Ribbon, and command bars. Users can search, sort, filter, categorize, drag, or create rules for the data that the solution introduces in the same way they would if they were working with Outlook's built-in data. The solution offers a consistent experience across different computers by roaming preferences (just like Outlook roams views), categories, and rules in Microsoft Exchange mailboxes.

A successful Outlook solution accomplishes more than just introducing a new feature; it actually solves a customer problem in a holistic manner by guiding the user end-to-end to get his or her job done. Many such Outlook solutions leverage the user's familiarity with Outlook by building on Outlook's extensive usability experience, therefore reducing or eliminating the need for additional user training.

The contrast to a seamless Outlook solution is a superficial integration that takes advantage of users living in Outlook. Outlook is not designed to be a generic shell. Such integrations often cause user confusion because they don't work like the rest of Outlook and introduce functionality that does not really belong to Outlook. These inconsistencies can have an impact beyond the solution on all of Outlook because the user does not know what constitutes the boundary between Outlook and the solution.

The prime goal of the Outlook platform is to facilitate integration of your solution into Outlook. The Outlook platform is not about offering a palette of reusable controls that can be used outside the context of Outlook or supporting server scenarios like running the Outlook object model in Microsoft ASP.NET applications, automating mailboxes of an organization, or installing the Outlook implementation of the Messaging Application Programming Interface (MAPI) on a server.

Customers expect Outlook solutions to build on Outlook strengths such as cached Exchange mode. Outlook offers rich functionality that is also available when the customer works offline or without network connectivity. Critical information is stored in a user's mailbox and is easily accessible thanks to integrated Instant Search, a new feature in Outlook 2007. Outlook's cached Exchange capability means that customers also expect data associated with Outlook solutions such as a CRM integration to work offline and be easily searchable. Customers also

expect that basic Outlook functionality like categories, rules, flagging for follow-up, or responding to mail works consistently for all data showing up in Outlook.

Outlook's core functionality revolves around Mail, Calendaring, Contacts, and Tasks. These types are prominently exposed in the Navigation Pane and have dedicated forms, views, and appropriate actions such as Send, Reply, Reply to All, and Forward. For example, media types (pictures, music, or videos) are not native Outlook types and are less deeply integrated into Outlook; they show up as attachments in the Reading Pane and on an Inspector. Outlook 2007 offers a richer preview experience for attachments, but otherwise Outlook is not optimized for these types. If the solution you're about to write is not about extending Outlook's core types, then integrating into Outlook might not be the right answer for your customers. In this case, you should investigate delivering the functionality of your solution as part of a separate standalone application.

For those solutions that follow the guidelines established in this chapter and deeply integrate with Outlook, the Outlook 2007 platform enhancements enable your solution in the following ways:

- The consolidated Outlook 2007 object model is sufficient for most solutions and replaces deprecated application programming interfaces (APIs) such as Collaboration Data Objects (CDO) 1.21. The Outlook object model is fully supported for Microsoft .NET Framework development.
- Form regions allow you to integrate your user interface with Outlook's built-in forms and provide a rich palette of controls to ensure that you can clone existing Outlook forms.
- Ribbon extensibility (referred to as RibbonX) and custom task panes ensure that your solution is integrated with the discoverability and usability improvements in the 2007 Microsoft Office system.

These enhancements are not only geared toward external developers; many Outlook integrations developed within Microsoft rely heavily on these Outlook 2007 platform enhancements: Microsoft Business Contact Manager, Microsoft Exchange Unified Messenger Add-in, and Microsoft Outlook Mobile Service as well as other Microsoft add-ins that ship with Office 2007. This internal Microsoft usage helps to ensure that the platform is stable, performant, and offers a comprehensive set of functionality.

Different Types of Outlook Integration

There are two different categories of Outlook integration from a platform perspective: data integration and functional integration. Many real-world Outlook solutions utilize both types of integration. The following discussion introduces the two types of Outlook integration, cites specific implementations that ship with Outlook 2007, and then proposes guidelines for achieving data and functional integration with your solution and Outlook.

Data Integration

This model is used by solutions that want to either simply access Outlook data or bring their data into Outlook and let Outlook manage the data from that point on.

Examples of Data Integration

The following list provides some general categories of data integration and also focuses on specific data integration solutions:

- **Synchronization of Contacts, Calendar, and Mail** These solutions enable two-way synchronization of Outlook folders with miscellaneous portable devices and mobile phones.
- **Connector/provider applications** These applications bring data into Outlook from other sources like messaging and collaboration back ends or CRM systems.
- **Online Meeting integration** Adds Online Meeting to the Outlook Calendar and introduces additional information for a meeting like dial-in number, access code, and URLs to resources. Shows these fields on meeting requests.
- **Calendar Gadget for Microsoft Windows SideShow** This add-in shipping with Office 2007 obtains one week of appointments and meetings from the default Outlook Calendar and passes this data to Windows SideShow gadgets.

How Did Microsoft Office Outlook 2007 Calendar Gadget for Windows SideShow Do It?

The Microsoft Office Outlook 2007 Calendar Gadget for Windows SideShow is an in-process Component Object Model (COM) add-in for Outlook. It relies primarily on the Outlook object model. The add-in allows users to view their calendar on Windows SideShow-compatible devices. Windows SideShow, shown in Figure 2-1, is new for Microsoft Windows Vista and enables developers to write gadgets or mini-applications to send data from the computer to devices connected to the computer. A Windows SideShow-compatible device can take several forms, such as a display attached to the lid of a laptop, a front-panel display on a desktop or server, or a small display in a keyboard. It can also be part of an existing device such as a cell phone, portable media player, or digital picture frame.

Figure 2-1 Windows SideShow uses the Outlook object model to gather its data.

SideShow Implementation Windows SideShow is implemented as an Outlook add-in. The add-in enumerates calendar items on the user's default calendar, and sends about a week's worth of calendar data to the Windows SideShow platform. The add-in only reads calendar data; there is no integration into the Outlook user experience. Writing this functionality as a trusted COM add-in ensures that the code by default won't trigger any Outlook security prompts. The default date range is 2 days previous and 5 days ahead. The information is updated every day, as well as when calendar changes are detected. The primary data format used is *iCalendar*. The add-in generates its own *iCalendar* representations of the appointments on a background thread for performance reasons, and sends those directly down to the devices. For Windows SideShow-compatible devices that do not support *iCalendar*, the add-in provides a simple text-only version of the content that shows the user's next five appointments within the next 24 hours. This is very useful information on small text displays, such as those embedded in keyboards, as it gives the user at-a-glance access to his or her important upcoming appointments. This data is refreshed at a regular interval. When there are no Windows SideShow–compatible devices connected to the computer, the add-in does nothing except wait for devices to be connected to minimize the performance impact for Outlook.

Paying Attention to Performance The first performance goal is to ensure that the add-in is only retrieving calendar data when there's actually a SideShow device present and active. Only doing work when required is an important guideline applying to all add-ins. The second goal is to ensure that users won't notice when the SideShow add-in accesses Outlook data. Any operation has to be less than 250 milliseconds in duration for users to not perceive the operation as a hang. This requirement is especially important because the user did not initiate the sync to the SideShow device, so the user would not understand why Outlook suddenly appears to hang.

Minimizing the performance impact to Outlook represents the main design challenge when utilizing the Outlook object model for add-ins without a user interface (UI). All calls into the object model occur on Outlook's main UI thread; thus they have the ability to negatively affect

the user experience by causing sporadic hangs, stutters, and feelings of sluggishness. To address these issues, the Calendar Gadget performs as little work as necessary on the Outlook UI thread, reserving the bulk of the heavy lifting for a background thread. To minimize the cross-thread marshaling, all calls into the object model occur on the main thread. When processing the calendar, the add-in handles one appointment at a time, and then sets a timer before processing the next item to allow the message pump to process other messages. For each calendar appointment, it extracts the important properties, stores them in a temporary object, and queues it for the worker thread. Generating the *iCal* items on the worker thread is more efficient than utilizing Outlook's save as *iCalendar* functionality for single appointments. Bulk exporting *iCalendar* items through the *CalendarSharing* object is not an option because SideShow gadgets don't support *iCal* recurrence.

Functional Integration

Functional integration is the model that's followed by solutions that want to introduce new functionality into the Outlook user experience through customizing command bars for Outlook's main Explorer window, introducing custom forms with RibbonX, or providing custom task panes for Outlook Explorer or Inspector windows.

Examples of Functional Integration

There are many ways to categorize functional integration. The following list provides some examples of functional integration.

- **Mail applications** Compression of attachments, archiving mail, adding disclaimers to mail messages, encryption, and content security
- **Corporate compliance** Restricting Reply All, ensuring that messages are addressed to correct recipients
- **E-mail protection** Spam blocking and antivirus
- **Utilities** Printing of Calendars or Contacts
- **Productivity tools** Helping a user become more productive with Outlook by enhancing search, prioritization, and organizational schemes
- **Unified messaging** Directing voice mails and other communications to a user's Inbox

How Did Unified Messaging Add-In for Outlook 2007 Do It?

The Unified Messaging add-in for Exchange provides integrated support for listening to voice mails within Outlook. Voice mails can be viewed using a form region that includes an inline Media Player and a private field for taking notes (see Figure 2-2). A form region is a new technology in Outlook 2007 that allows you to replace or integrate with Outlook's built-in forms. The Unified Messaging add-in also provides the ability to have the Exchange server make an outbound phone call to a specified number and play the voice mail. This feature is useful

when privacy is a concern, or when the computer does not have speakers. The add-in introduces an additional Tools Options page for viewing and editing Unified Messaging preferences. A user can reset his or her PIN, update his or her voice mail greetings, and define the default folder when accessing the Exchange mailbox with a phone.

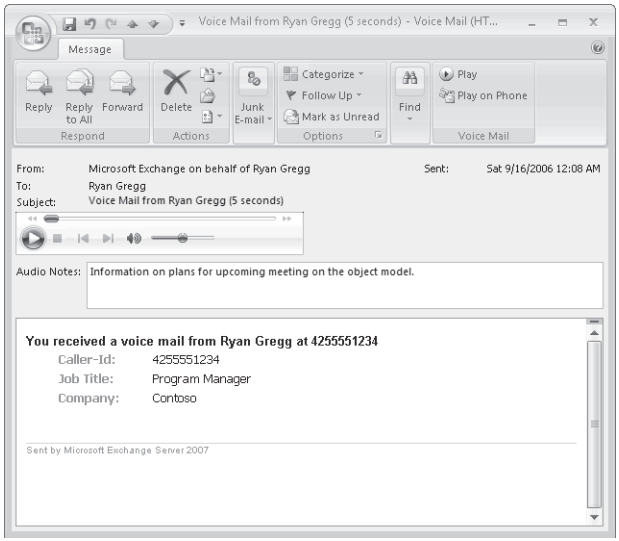

Figure 2-2 The Unified Messaging add-in uses Outlook 2007 form regions.

Unified Messaging Implementation The Unified Messaging add-in relies on the Outlook object model, form regions, and RibbonX. The add-in introduces a custom form region with its own custom message class for Exchange voice messages. The form region renders in the Reading Pane and in a separate Inspector window. It hosts a Windows Media Player control and extends the default Ribbon by adding controls for playing the voice mail and initiating Play on Phone. A custom string property is used to store the voice mail annotations. The add-in uses property pages to add a new tab to Outlook's Tools Options dialog box. It also utilizes two Exchange Web services. The Auto-Discover Web service is used to locate the appropriate Exchange Server that provides Unified Messaging support, and the Unified Messaging Web service provides advanced features specific to Unified Messenger for voice mails, such as Play on Phone.

Paying Attention to Discovering Unified Messenger Availability The add-in is scoped to voice messages with a certain message class and offers its functionality once it discovers that Exchange Unified Messaging is available. This implementation decision has been made to minimize the performance impact for users without Unified Messenger capabilities and to limit the attack surface. This is not at all to say that the Unified Messenger add-in is insecure; it actually went through an in-depth security analysis. It's simply following a good security practice, which is to only enable functionality when required. The add-in attempts to locate the Exchange Unified Messaging Server over different network connections using corporate local area network (LAN), Internet, virtual private network (VPN), or remote procedure call

(RPC) over Secure Hypertext Transfer Protocol (HTTPS). If it fails to find the server, it will disable the Tools Options tab and *Play on Phone* command.

Integration Guidelines

The goal for the Outlook 2007 platform enhancements is to enable developers to build rich solutions by relying on only the Outlook object model, form regions, RibbonX, custom task panes, and also, in some cases, Extended MAPI.

Data Integration

There's a wide range of data integration scenarios and there are many options available to accomplish those scenarios. This breadth of options makes it important to define clear goals first so that the appropriate data integration path can be identified. The purpose of the following discussion is to identify the data integration scenarios. After the scenarios have been identified, the focus will be on how the data integration can be accomplished.

Integrating with Data That Is Already in Outlook

Many customers have one single Outlook MAPI profile that often contains only one store containing all Outlook data (mail, contacts, calendar, and tasks). Examples are customers with an Exchange account or multiple Post Office Protocol (POP) accounts. If there's more than one store associated with the Outlook profile, then it's likely to be either one or more proxy Personal Folders File (.pst) stores used for Hotmail and Internet Message Access Protocol (IMAP) e-mail replication or an archive .pst file.

One example of such a data integration scenario would be synchronizing calendar and contact data from Outlook to a device. When configured to run against Exchange, Outlook would replicate these appointment and contact items to the default folders for these items while this Outlook sync integration would then sync the data from and to the device. Because the Outlook platform does not offer a sync API, it's up to this sync integration to keep track of replication state and perform conflict resolution. For example, an item might have changed both in Outlook and on the device. The sync integration code has to detect that the items have changed and attempt a conflict resolution. Ideally, conflict resolution does not involve the user and is silent. If a user has added a contact phone number on the device and changed the mailing address in Outlook, the sync code can silently merge these two changes.

The two APIs that accomplish data integration are the Outlook object model and Extended MAPI.

Bringing Data into Outlook

Since the introduction of cached Exchange mode in Microsoft Office Outlook 2003, customers expect data to be available locally when they're on the go without network connectivity or

connected over a high-latency/low-bandwidth network connection. This means that customers also expect that the data associated with solutions is available offline. Often the back-end data repository contains much more data than what's suitable to cache locally, asking for a model allowing to cache only the critical subset and offering online access to the rest.

The data can be locally cached in a dedicated Outlook .pst-based store, meaning that there's a separate store offering its own folder hierarchy exposed in the Outlook Navigation Pane. There are two options to accomplish the replication: the solution can replicate the data into the .pst either by writing its own replication algorithm using a combination of MAPI and the Outlook object model, or by relying on the Replication API. The Replication API provides the functionality for a MAPI Transport provider to synchronize Outlook items between a server and a private, .pst-based local store created for that provider.

Another option is to introduce either a custom MAPI store provider that offers online-only access to the back end or a store provider that locally caches the data. One more alternative is a hybrid architecture, which locally caches the most critical data and offers online access to the rest.

Customizing Outlook Items

Besides identifying in what store the data should be persisted, a solution also needs to determine if the data can be stored in the properties already defined by Outlook or if new custom properties have to be introduced. MAPI provides the foundation for Outlook data storage. MAPI predates Extensible Markup Language (XML), which means that there's no schema associated with MAPI items. Outlook items represent simple property bags that are preserved when items are moved or copied within and across stores. If an Outlook item requires additional properties and these properties need to be visible within Outlook, then the custom properties must be created through the Outlook object model's *UserProperties* object. In this case, the properties would appear in the Outlook Field Chooser so they can be added or removed from Views or displayed in an Outlook Inspector window. Creating properties directly through MAPI won't do the job, but once the property is created through the object model it can be accessed with MAPI.

Another question that you must ask is whether or not a custom item type identified by a custom message class is required. A custom message class is typically introduced if the item is rendered by a custom form, which is covered later in the section "User Interface Integration and Data Presentation."

Data Integration with Outlook Object Model

The Outlook object model is by far the most comprehensive and powerful API for programming Outlook, but performance considerations have to be factored in. The Outlook object model runs on Outlook's foreground thread, meaning performance is critical especially if your application requires that you synchronize a significant amount of data. Performance will

be likely the most demanding aspect for data integration. Performance considerations include the following:

- Using appropriate object model members, including the new *Table* object and Instant Search queries.
- Data throttling, where you fetch data when the machine is in an idle state.
- Data granularity, so that you read and write data in small chunks.

It's recommended to couple the solution lifetime to Outlook and only run when Outlook does, including running against the same MAPI profile that Outlook is logged into.

Data Integration with Extended MAPI

Extended MAPI allows data access and runs on a separate thread. The tricky part is that MAPI sits below Outlook's business logic, which means that writing data is complex and opaque for certain scenarios. If the data access scenario involves mail or contact items, then using MAPI is acceptable. If the scenario involves appointment or task items, MAPI is problematic. The business logic of Outlook's Calendar is complicated, and many properties such as the recurrence pattern are stored as an opaque binary blob. In addition, all meeting actions have many side effects, including meeting deletion, sending a meeting cancellation, or changing properties such as location or time and then sending a meeting update. Task assignments and recurrence are opaque in a manner similar to that of appointment items. If possible, it's recommended that the MAPI integration only be executed when Outlook is running and against the same Outlook profile the user is logged into.

Data Integration with MAPI Store or Address Book Providers

Although MAPI providers offer a rich model for creating a store or Address Book provider, you should be aware that writing such a provider, especially a MAPI store provider, is a complex task requiring developers with extensive unmanaged C++ and MAPI coding skills. If you are writing a custom MAPI store provider, your store will not be indexed by the Instant Search engine. If you rely on folder home pages for your custom store, those pages are disabled by default in a nondefault store as a security mitigation. Keep these limitations in mind before you decide to invest your resources in writing a MAPI store or Address Book provider.

Data Integration with Replication API

The Replication API, documented as part of the Outlook 2007 Integration API Reference, offers another option for replicating items from a back-end data repository into an Outlook .pst-based store. The Replication API is used for replicating the data into a dedicated .pst-based store and keeping track of the synchronization state. Because the business logic for appointment, contact, and task items is only exposed in the Outlook object model, this API is not ideal for these types. The positive aspect of this approach is that it does not require the introduction of a custom MAPI store provider, which is complex to write and maintain. Criti-

cal Outlook 2007 functionality such as Instant Search will work without modifications on your part provided that Windows Desktop Search is installed and enabled for your store.

Business Logic

Outlook implements business logic for both built-in and custom items. Events in the Outlook object model allow you to write code that overrides or modifies that business logic.

Business Logic for Built-In Items

Integrating into Outlook through the Outlook object model ensures that the Outlook solution also benefits from Outlook's rich business logic for different Outlook item types. When an item is accessed through MAPI, the Outlook business logic is bypassed.

Outlook's business logic is invoked whenever an item of a certain type is loaded. For example, if an item is programmatically opened without a UI or a user opens a contact item (*MessageClass* is *IPM.Contact*) or a customized contact rendered with a form region (*MessageClass* is *IPM.Contact.OwnVersion*), then Outlook's contact business logic is invoked. For a discussion of how *MessageClass* relates to built-in and custom types, see Chapter 5, "Built-In Item Types." Calendaring involves considerable business logic, and the following examples provide you with an overview:

- Appointment items ensure that the start time of the meeting is before the end time.
- Appointments also support recurrence, including exceptions; for example, a weekly meeting on Wednesday at 5 P.M. except for this week, when the meeting is on Thursday at 2 P.M.
- The recurrence also accommodates non-Gregorian calendars, like birthdays based on lunar calendars.
- A cancellation is sent when the meeting is deleted from the calendar or an attendee is removed.
- Meeting requests are processed when they arrive in the mailbox and get tentatively added to the calendar.
- When the user accepts or declines a meeting, the user is asked to send a response to the organizer.
- Meeting responses from meeting attendees are automatically added to the meeting's Tracking tab in the organizer's calendar.

For contact items, the business logic primarily revolves around keeping related fields in sync, for example the *FullName* property with *FirstName*, *MiddleName*, *LastName*, *Suffix*, and *Prefix* or *FileAs* with *CompanyName* or the name fields. Addresses (Home, Business, Other) are also stored as both individual fields (city, country, postal code, state, street) and as free-form multiline fields with custom formatting.

Custom Business Logic

Outlook does not allow turning off or overwriting the built-in business logic for the different item types. In other words, if a solution introduces an item with a *MessageClass* of *IPM.Contact.OwnVersion*, Outlook's business logic for contacts will kick in, and if the *FirstName* field changes it will also update the *FullName* field. Although the Outlook business logic cannot be overwritten, Outlook's rich event model provides developers with a way to customize and refine Outlook's built-in behavior. This customization can be accomplished by writing an Outlook add-in with an event handler that gets called when Outlook saves an item, a built-in or custom property is changed, a file gets attached, or an e-mail is sent. This rich event model allows Outlook solutions to extend built-in actions and perform additional data validation.

Another option is to customize built-in actions or introduce new custom actions as part of introducing a form region to render a custom item type. If the solution intends to introduce its own item type and control the business logic associated with the item, you should consider basing the custom item on a Post item. The Post item is the closest item type to a "start-from-scratch" item, as this is the item type with the least built-in Outlook business logic. Custom properties can be added to this custom Post item using the *UserProperties* object, and your add-in can control custom property business logic by implementing event handlers hooked up to item or form control change events.

User Interface Integration and Data Presentation

This section enumerates the different Outlook UI extensibility mechanisms and provides guidelines for how to integrate your UI with Outlook. The goal of UI integration is that customers don't perceive your UI as different from the Outlook UI. The entry points of your UI should be parallel with entry points for the Outlook UI.

Outlook Explorer Window

The Outlook Explorer window shown in Figure 2-3 is the main Outlook application window and displays folder contents. A word of caution is in order before discussing the Explorer window. Although Outlook offers the ability to extend its UI, you should consider that the surface area for customization is limited in the Explorer window. This constraint means that it's often desirable to integrate with the existing UI rather than layering your custom UI on top of the Outlook UI. It's quite common for a user to have a number of add-ins installed. If each one of them introduces a new top-level menu and toolbar for the Outlook Explorer, the UI will become busy to a point that Outlook usability as a whole suffers.

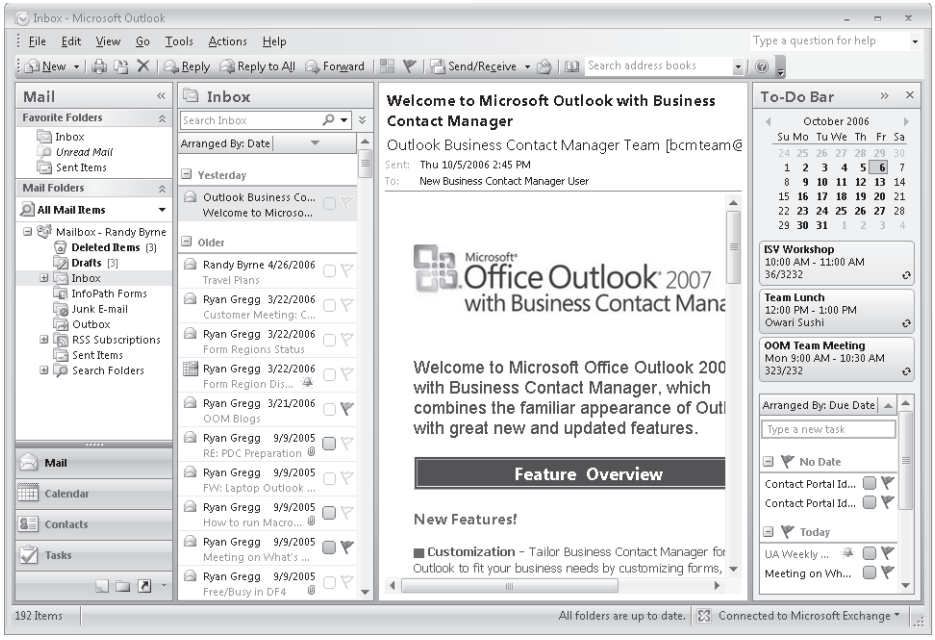

Figure 2-3 The Outlook Explorer window.

Instead of introducing a new menu or toolbar, consider merging custom commands with existing menus. Add-in preferences can be exposed under an additional tab in the Outlook Tools Options dialog box. Another solution is to make the functionality available only when the user actually needs it. In short, your UI should be context-sensitive. If an Outlook solution has its own store, then a custom UI can be available when the user works in this store and be hidden otherwise.

Introducing a custom task pane for Outlook Explorer is challenging because Outlook 2007 also introduces the To-Do Bar, meaning that many users will run Outlook with four panes (the Navigation Pane, the View Pane, the Reading Pane, and the To-Do Bar). Introducing a fifth vertical pane is not desirable unless you locate a custom task pane in a horizontal location at the bottom of the Explorer window. To avoid user confusion, add-ins should not automatically hide or collapse panes without user interaction.

Command Bars

The menu bar, standard, and advanced toolbars that appear at the top of the Outlook Explorer window are built using Office command bars. Add-ins can introduce a custom toolbar or add, remove, or hide commands in built-in menu bars and toolbars. Command bars are an Office extensibility mechanism shared by different Office applications. With Office 2007, Microsoft Word, Excel, and PowerPoint use the Ribbon exclusively, whereas Outlook 2007 is a hybrid, relying on command bars for Outlook Explorer windows and the Ribbon for Outlook Inspector windows.

Navigation Pane

The Navigation Pane shown in Figure 2-4 appears on the left side of the Explorer window and allows the user to select different Outlook modules, such as Mail or Calendar. Additionally, the Navigation Pane displays a list of folders for each module. The object model includes support for switching modules, controlling which modules are displayed, and modifying the grouping of folders in modules that have folder groups. The Outlook platform does not allow you to add a new module to the Navigation Pane.

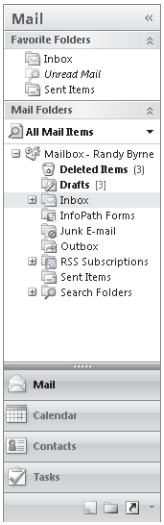

Figure 2-4 Outlook Navigation Pane.

View Pane

The View Pane shown in Figure 2-5 typically renders the contents of a folder with a view optimized for the item types stored in the folder. For example, mail items are displayed in a table view, meetings and appointments in a calendar view, and contacts as business cards. The Outlook 2007 object model allows fully dynamic view customization in the View Pane. You can add or modify folder views programmatically. View fields can be added or removed, Group By fields can be added or removed, a filter can be applied, and almost all aspects of the view can be customized programmatically.

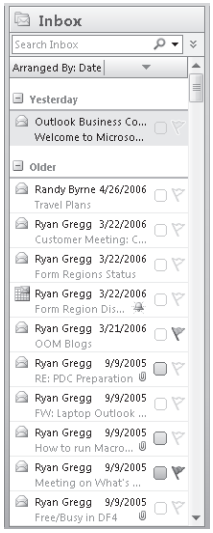

Figure 2-5 Outlook View Pane.

Reading Pane

The Reading Pane shown in Figure 2-6 displays the currently selected item or attachment. Outlook 2007 provides the ability to customize the look of the Reading Pane for both items and attachments. Form regions can be used to extend or replace how an item is rendered in the Reading Pane. A custom preview handler also can be registered to control the way an attachment is previewed in the Reading Pane.

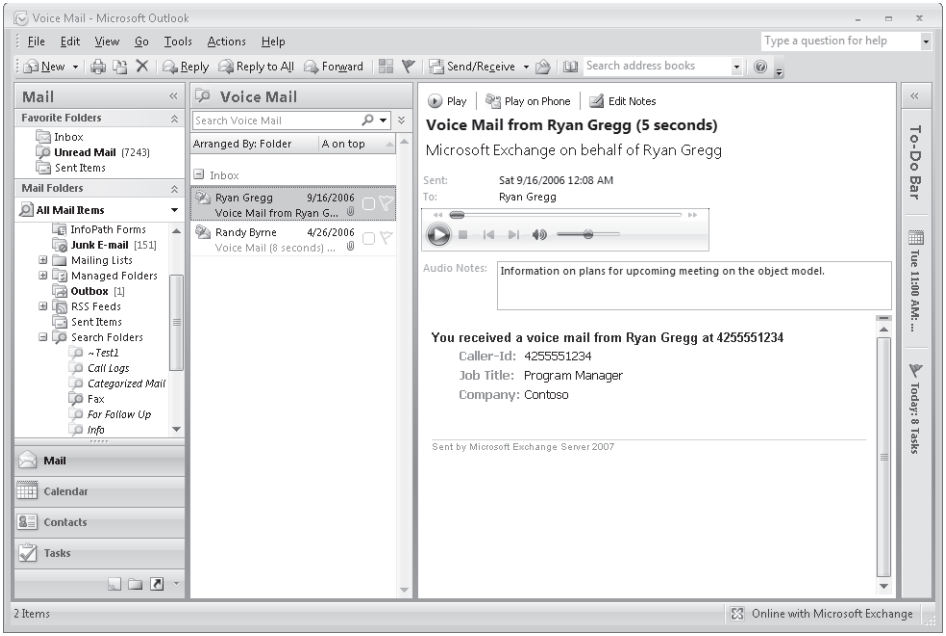

Figure 2-6 Reading Pane with form region customization.

To-Do Bar

The To-Do Bar displayed in Figure 2-7 provides a quick summary of upcoming appointments and tasks. Items are added to the To-Do Bar by creating a new task item or by flagging mail or contact items for follow-up.

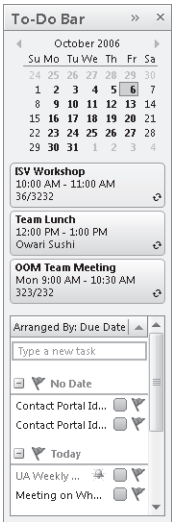

Figure 2-7 Outlook To-Do Bar.

Context Menus

The Explorer window contains a number of context menus that can be customized. The Navigation Pane offers the store and folder context menus, the View Pane offers the items and views context menus, the Reading Pane offers the attachment context menu, and the To-Do Bar also supports the item context menu.

Property Pages

Configuration options for your solution can be integrated into the Outlook Tools Options dialog box by using a property page similar to the one shown in Figure 2-8. Property pages can also be used to extend the Folder Properties dialog box.

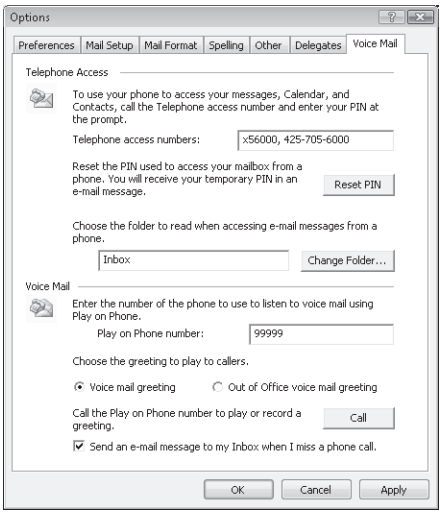

Figure 2-8 Unified Messaging property page in the Tools Options dialog box.

Outlook Inspector Window

New to Outlook 2007, form regions are the centerpiece for customizing the Inspector window and the Reading Pane. Form regions allow for an additive UI by introducing an adjoining form region that shows up on the bottom of the first tab of a custom or built-in Outlook Inspector window. Figure 2-9 illustrates an adjoining form region on an Outlook Inspector window. Separate regions provide more control. They can be either added to or replace one or all of the tabs of an existing form. If, for example, a Contact form needs to be completely customized, the add-in can introduce a new form region that is used whenever an *IPM.Contact.MyCustomer* item is displayed. The add-in can also register *IPM.Contact.MyCustomer* as the default form used when a user creates a new Contact in the MyCustomer folder. Outlook's built-in Inspectors can essentially be cloned due to the introduction of a number of additional controls for use in form regions. Controls that ship with Outlook 2007 include simple label, edit, list box, and combo box controls, as well as more complex controls to duplicate Outlook's date/time

picker, category strip, and scheduling controls. All controls support data binding to built-in or custom properties. Unlike controls in previous versions of Outlook, both the control hosting surface and the controls placed on that surface use Windows themes.

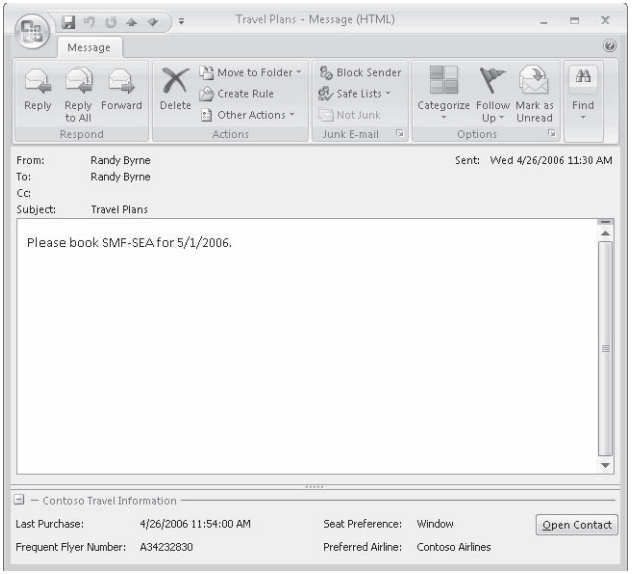

Figure 2-9 The Outlook Inspector Window with adjoining form region.

Although not an immediate component of the Outlook platform, Ribbon extensibility (known as RibbonX) allows you to add custom groups and commands to the Ribbon for a given Inspector type. Figure 2-10 illustrates a custom group added to the Ribbon on the Inspector for an appointment item. You can also repurpose built-in commands, hide built-in groups and commands, and insert your custom commands into built-in groups. RibbonX offers a superior control palette to Office command bars. You can leverage new picture galleries in your solution or utilize a host of controls that had no equivalent in the Office command bars object model.

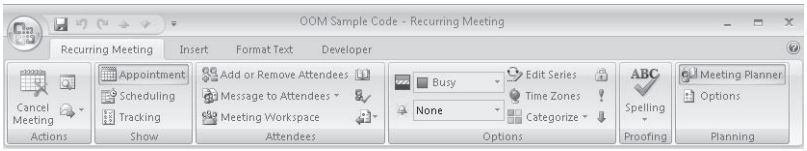

Figure 2-10 Prepare for Meeting sample add-in customizes the Ribbon for an Outlook appointment item.

Custom task panes provide another customization option for the Outlook Inspector window. Whereas form regions target the extension and customization of the Outlook Inspector itself by typically displaying new user properties, custom task panes facilitate bringing related data

into Outlook just like a built-in task pane such as the Research Pane. Figure 2-11 shows the Prepare for Meeting custom task pane in an Appointment Inspector window.

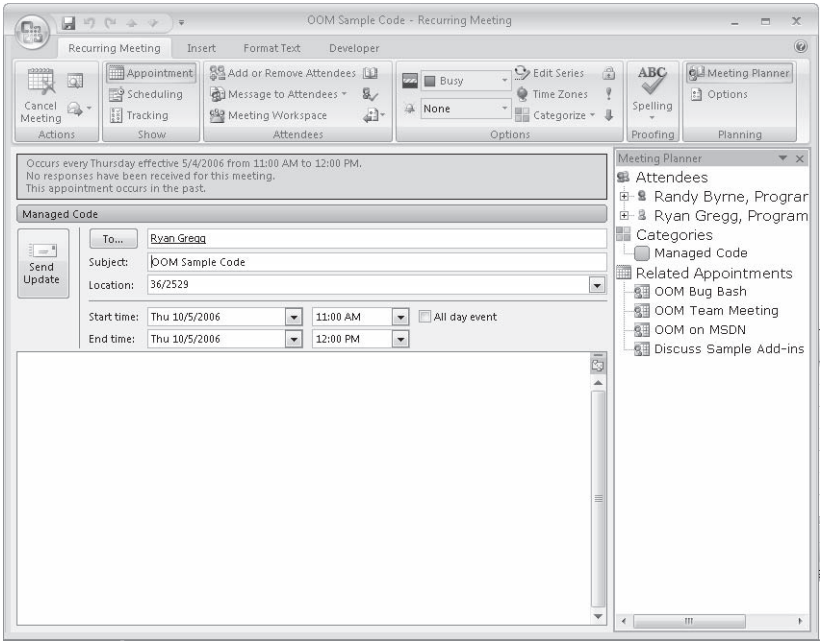

Figure 2-11 Custom task pane in an Appointment Inspector window.

Use Interface Integration Example

Let's look at an add-in with the goal of customizing Outlook Contacts as customers of a shoe store. Shoe Size and Customer ID are two additional properties that need to be tracked for each customer. The add-in would introduce an adjoining form region for Contacts containing these two properties and store the data as user properties in the backing Contact item. The add-in would rely on the Outlook built-in Contact Inspector to render the rest of the form. RibbonX would be used to add a new verb to the Ribbon, allowing the user to check store inventory for shoes in a corresponding size and then show the results in a custom task pane docked to the Outlook Contact Inspector. This "Check for shoes" verb could also be added to the item context menu so the clerk could enumerate the available shoes in a custom task pane docked to the Outlook Explorer window. The Views object model would be used to create a new custom List View for the Customer Contact folder including these two new properties. The add-in could also introduce a custom toolbar that shows up when this Customer Contact folder is selected and allows the store clerk to restrict the Customer list and only display customers with a specified shoe size.

InfoPath Forms

The following discussion concentrates on the use of InfoPath forms. Although not strictly a component of the Outlook platform, InfoPath e-mail forms do provide a compelling way to collect survey data from messaging recipients. This section helps you understand the purpose and design of InfoPath forms. You'll learn when InfoPath forms are appropriate in comparison to Outlook forms. Although there is no direct link between the Outlook object model and the InfoPath development environment, an InfoPath e-mail form uses aspects of the platform such as MessageClass and form-based rules that will help you understand the platform as a whole.

Microsoft Office InfoPath 2007 is a forms application that provides users with a way to gather structured information. Because InfoPath uses XML standards, data collected in InfoPath forms can be integrated directly into existing business processes such as databases, Web services, or workflows. Alternatively, collected data can be saved as individual files on collaborative sites such as a Microsoft Office SharePoint Server document library. Integration with other Office applications, such as Outlook 2007 and Microsoft Office Excel 2007, allows the forms experience to reach more users and provide easier data analysis. Use of InfoPath Forms Services even allows users to fill out InfoPath forms in the browser.

When to Use InfoPath Forms

InfoPath forms are best suited to collecting data by integrating into the e-mail functionality of Outlook. These forms can target data collection in an ad hoc manner, such as status reports and surveys, or be designed to integrate tightly with existing business processes and LOB applications using databases, Web services, and workflows. Forms designed to target people, calendar, or task information are better suited to using Outlook forms. Such forms can add new capabilities on top of Outlook, such as a customer relations form, or can extend certain information on existing Outlook forms, such as adding an employee number to a contact item.

An example of a common InfoPath form scenario is a weekly status report. Although certain information in the form could be derived from Outlook, the majority of the information pertains to business processes. In this case, an InfoPath status report form can be published to all team members as an InfoPath e-mail form. Each team member completes and submits the form, which is submitted back to the manager. Individual reports can be merged together to form a single report, which can be again submitted to the manager's manager, and so on.

Alternatively, the status report could be part of an LOB application. For example, if the status report tracks sales numbers, then data could be submitted directly to the back-end CRM system using InfoPath's built-in support for XML Web services. Subsequent workflow operations could send updated sales numbers, reports, and tasks to other members of the sales team.

Creating and Deploying InfoPath Forms

InfoPath forms are created using the design environment inside InfoPath 2007. The InfoPath design environment allows form designers to drag and drop controls to quickly build the form.

When a form designer creates a new InfoPath form, he or she actually creates what is known as a form template. A form template defines the data structure, appearance, and behavior of the forms that users fill out. Think of a form template as a blueprint—the starting point that enables users to create new forms that use and store data in the same way. Because a form template must be available before you can fill out a form, form templates must be deployed to a location where users can access them. Form templates are commonly deployed to locations on a company network, such as shared folders, Web servers, or libraries on Microsoft Windows SharePoint Services version 3 sites. Forms can also be deployed via installable packages (.msi or .js).

If a user has permission to access the location where a form template is stored, he or she can fill out a form based on that template by using InfoPath, a Web browser, a mobile device, or Outlook 2007. Whether a form is filled out by using InfoPath or one of the other methods depends on several factors, including how a form template is designed and the technology available when the form is deployed. For example, to fill out a form in Outlook 2007, the form must be published to a list of e-mail recipients.

Using InfoPath E-Mail Forms in Outlook 2007

You can use InfoPath forms in Outlook 2007 to help streamline the processes you use to collaborate and share data. That's because you can open, fill out, and submit InfoPath e-mail forms with Outlook 2007. If you receive an InfoPath e-mail form, you can reply to it, forward it, and store it just as you would with other items in Outlook 2007.

InfoPath e-mail forms also allow added analysis features. By storing collections of related e-mail forms in InfoPath Forms folders in Outlook 2007, you can organize and review data easily. For example, if you collect status report forms from your team, you can store the completed forms in an InfoPath Forms folder. Besides keeping all related forms in one place, you can also choose to show data from each form in columns in a custom view for that folder thanks to Outlook 2007 read-only promotion of a subset of XML properties for InfoPath forms. This allows for quickly grouping, filtering, and sorting data from multiple forms. The InfoPath form is stored as an XML attachment of an item with message class set to *IPM.InfoPath.FormID*, meaning the XML payload (data, schema) is opaque to Outlook. Outlook defers rendering, editing, and actions (for example, responding to these forms) to InfoPath. Because InfoPath is built on XML standards, information can be quickly merged into a single InfoPath form or exported for more detailed analysis in Excel.

APIs

The following section discusses the architecture and APIs that serve as the foundation for Outlook and any third-party address book, store, or transport provider used to extend Outlook.

Architecture

To understand the role of the different APIs, it's best to take a quick look at the overall Outlook architecture illustrated in Figure 2-12. Significant parts of Outlook are built on top of its own implementation of MAPI.

The three MAPI pillars used by Outlook are as follows:

- MAPI Address Book providers
- MAPI store providers
- Outlook Transport is partially built as a MAPI Transport infrastructure

Figure 2-12 provides a simplified perspective of the Outlook architecture. MAPI is the common foundation for both Outlook and CDO 1.2.1, which implement their own separate business logic. One goal for the Outlook 2007 platform enhancements was to unify these APIs all under the Outlook object model. This unification means that now all applications built on top of the Outlook object model will go through the very same business logic that Outlook relies on internally for its Personal Information Management (PIM) data types.

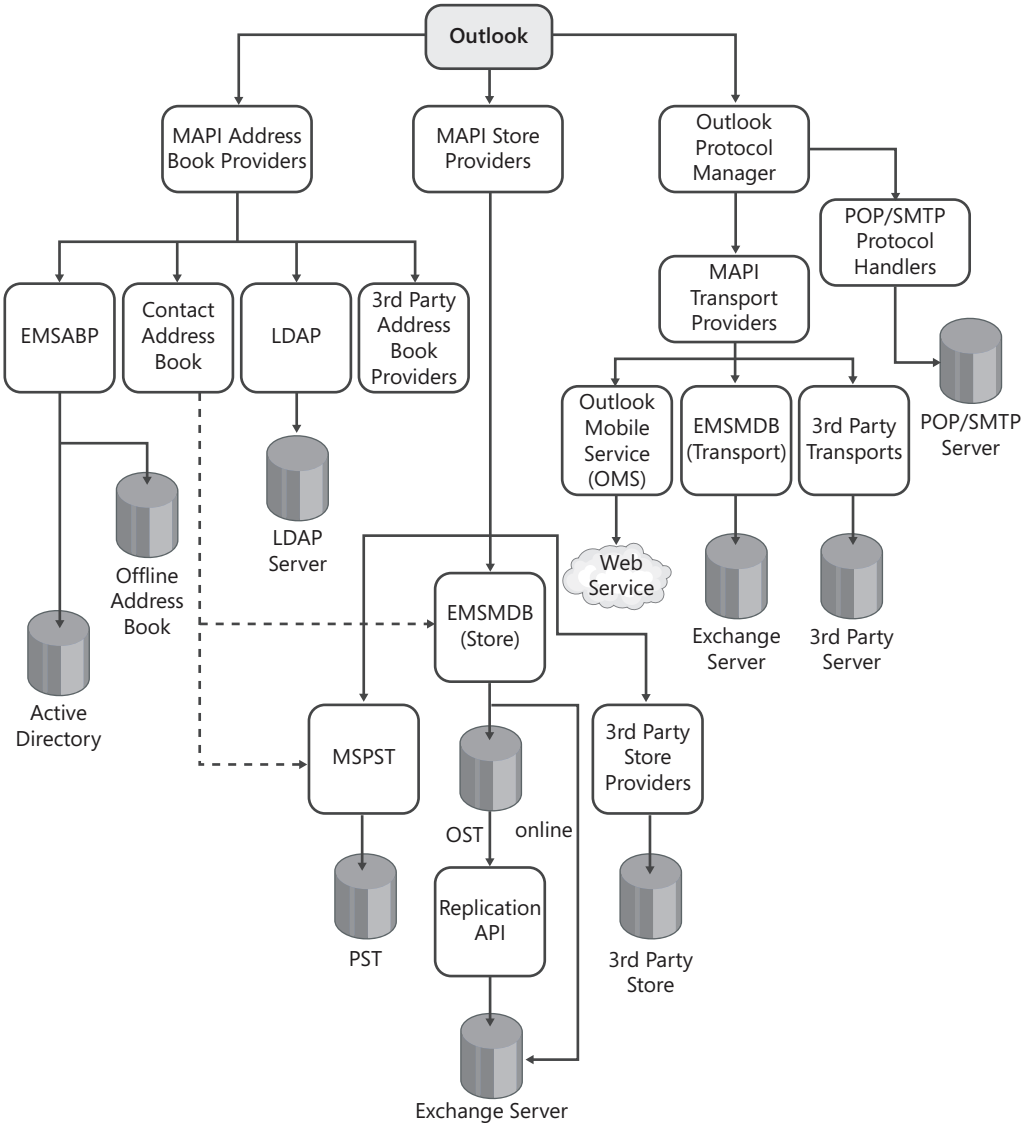

Figure 2-12 Outlook 2007 architecture.

Outlook Object Model

The object model constitutes the heart of the Outlook platform. The goal for the Outlook 2007 platform enhancements is to enable developers to build rich solutions by relying on the Outlook object model. The following areas provide the pillars of the Outlook 2007 platform.

Unification

The Outlook object model is the central piece of the Outlook platform. One of the prime goals of the Outlook 2007 platform enhancements was to unify existing APIs such as CDO, Exchange Client Extensions (ECEs), and a subset of Extended MAPI into the Outlook object model. As part of the unification, many events were added to the Outlook object model to accomplish parity with ECEs. The Outlook object model now provides equivalent objects for CDO's *AddressBook*, *InfoStores*, *Fields*, and *HiddenMessages* objects. The new *PropertyAccessor* and *Table* objects offer alternatives to the *IMAPIProp* and *IMAPITable* interfaces in Extended MAPI.

Unification means that the Outlook object model will be the sole API developers will rely on when writing a tightly integrated Outlook solution. Solution developers are no longer shut out by entry barriers caused by fragmented APIs. Unification reduces the cost of writing Outlook solutions because developers have to familiarize themselves with only one API. If possible, you should avoid relying on CDO and ECE when writing new Outlook solutions. These APIs are primarily supported to ensure compatibility with existing solutions. Application compatibility represents another important goal. The Outlook 2007 object model is compatible with previous Outlook versions; newly added functionality enhances the object model without removing or altering existing objects or methods. Unlike CDO, ECE, and MAPI, the Outlook object model is fully supported for managed code development.

Performance

Performance is another critical attribute for writing successful Outlook integrations. Our customers spend much time in Outlook and expect it to remain responsive at all times. The new *Table* object provides lightweight read-only row items for performant enumeration, sort, and search of Outlook data. Enumerating items with the *Table* object is approximately an order of magnitude faster than the enumeration of the *Items* collection without calling the *SetColumns* method. Making the Outlook object model performant is also essential to ensure that developers won't have to fall back to Extended MAPI or CDO when writing their solutions. The Outlook object model also allows developers to leverage the Outlook 2007 integrated Instant Search infrastructure by using the content indexer for prefix and substring matching for searching Outlook items and attachments.

Security and Trustworthiness

Outlook 2007 has improved the object model guard that warns the user when a program attempts to send e-mail messages or get address book information, making it easier for Outlook developers to write solutions that do not trigger Outlook security warnings. At the same time, Outlook remains secure and trustworthy out of the box and allows administrators control over which solutions should run in the enterprise. By default, all installed add-ins have trusted access to the Outlook object model, meaning they won't trigger any security prompts. If antivirus software is installed, these prompts also don't show up when an

external application accesses the object model from a different process. IT administrators can define security preferences through the existing Outlook security form stored in Exchange Public Folders or using Windows Group Policies. Group Policy administration of Outlook security settings is new to Outlook 2007.

Innovation

Outlook 2007 introduces a wide range of new features that can be controlled programmatically:

- Sharing protocols (webcal://, feed://, stssync://)
- Calendar improvements (side-by-side, overlay, *ICal* sharing)
- Electronic business cards
- Task flagging
- Color categories
- Time zones for appointments
- Navigation groups and folder in the Navigation Pane

Besides providing the object model for these new features, Outlook 2007 programmatically exposes rules and views. The chapters in this book provide detailed information and sample code on using the objects that represent these features in your solution.

Form Regions

Outlook solutions can rely on Outlook custom forms with form pages or the new Outlook 2007 form regions to customize the forms associated with Outlook. The new separate and adjoining form regions in Outlook 2007 provide an additive or replacement UI to custom and built-in forms. Outlook 2007 form regions are the preferred form of customization technology because they have the following advantages over Outlook custom forms with form pagess:

- Form regions are supported in the Reading Pane.
- An additive UI can be provided through adjoining form regions. One item type can support multiple adjoining form regions from different add-ins.
- Outlook 2007 also introduces a full palette of Outlook form controls for form regions, allowing developers to clone the look and functionality of Outlook built-in forms:
    - Category Strip and Button
    - Contact Photo
    - Electronic Business Card Preview
    - Scheduling Free/Busy
    - InfoBar

  - Date Picker, Time Picker, and TimeZone
  - Sender Contact Photo

- Form regions and controls support Windows themes, meaning they have the same appearance as Outlook built-in forms.
- Form regions are installed locally as part of the Outlook solution and no longer rely on the Outlook forms cache. Form regions are no longer deployed through the Exchange Public Folder organizational forms library.
- The business logic for a form region is implemented in an add-in and no longer in VBScript behind an Outlook form. The form and add-in make up an Outlook solution that is installed and updated at the same time.
- Form region controls support additional events besides the *Click* event.
- Compared to forms in earlier versions of Outlook, form regions are easily localized without creating a separate form definition for each language supported for your solution.
- Form regions support auto-layout, allowing the developer to define placement and resizing rules at design time that Outlook obeys at run time when drawing the forms.

Form regions are based on earlier Outlook forms technology. You design form regions in the Outlook Forms Designer shown in Figure 2-13.

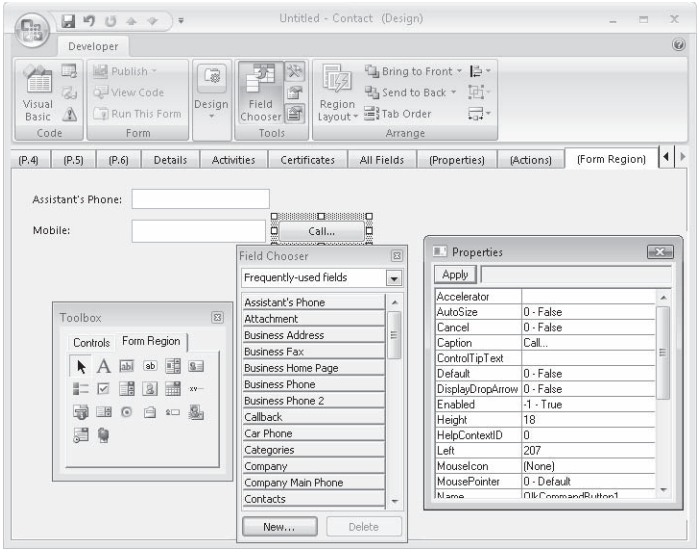

Figure 2-13 Designing form regions in the Outlook Forms Designer.

Form regions can provide an additional UI to built-in forms or completely customize the forms area, which involves the introduction of a custom item type with a customized message class such as *IPM.Contact.MyItem*.

The following list describes the available types of form regions for built-in or custom forms:

- **Adjoining** An adjoining form region allows showing an additive UI on the bottom of the main tab of a built-in or custom form. Many custom form scenarios involve adding a small number of fields and don't require a change to the rest of the form. An adjoining form region addresses this scenario. More than one adjoining form region is supported per form and each form region is expandable and collapsible. They enable lightweight extension of forms without having to redesign the entire form and also facilitate more than one solution extending the Outlook form.
- **Separate** A separate form region allows adding an additional tab to a built-in or custom form.

MAPI as a Platform Component

MAPI was introduced in the early 1990s as a specification for a messaging subsystem and all the components that interact with it. MAPI was created to provide a standardized application-level interface that allowed messaging components to communicate with widely incompatible messaging systems. The interfaces between components are not APIs, but rather COM interfaces. MAPI is not a library, meaning there are different implementations of MAPI developed by Microsoft and other ISVs.

In the context of this chapter, MAPI refers to the Outlook implementation of MAPI that gets installed as part of Outlook. The Outlook implementation of MAPI is a cornerstone of Outlook's own architecture and enables ISVs to tightly integrate into Outlook. MAPI differentiates between consumers (client applications) and producers (access providers) of messaging services. Outlook does both; it is a rich application offering a consistent user experience across different back ends and also ships a number of MAPI providers enabling connectivity to a wide range of back-end servers. The MAPI solution landscape for Outlook also covers both aspects: from MAPI clients such as antivirus software accessing e-mail messages to MAPI providers adding additional address book, store, and transport capabilities to Outlook. From a MAPI perspective, Outlook is a MAPI client with its own MAPI subsystem implementation and a number of MAPI service providers.

The main programming interface for MAPI is an object-based interface known as the MAPI programming interface. Based on COM, the MAPI programming interface is used by the MAPI subsystem and by messaging-based client applications and service providers written in unmanaged C or C++. Writing managed MAPI code is not supported.

What happened in the last decade since MAPI was introduced? The Exchange Client with separate Personal Address Book (PAB) and Calendaring (Schedule+) components became Outlook with tightly integrated Calendaring, Contacts, Tasks, Notes, and Journaling. Outlook as a PIM is implemented on top of the MAPI 1.0 specifications.

In other words, the additional functionality is not directly exposed through MAPI but through other APIs sitting on top of the MAPI infrastructure (Outlook object model, Outlook integration APIs, CDO). MAPI itself knows about e-mail items and address book entries but is agnostic to other PIM types like Contact, Appointment, Task, and Journal items. For MAPI these items are simply mail items with extra properties. MAPI does not know the meaning of these properties, nor is it aware of the business logic. Recurrence for an appointment item is a binary blob, and start time and end time are two individual properties not coupled with each other. This means MAPI APIs are great for mail and address book functionality and to some degree for also reading other PIM items, but MAPI is largely not suitable to create or update them, as it would be up to the caller to be intimately familiar with item properties and business logic.

The recommendation is to use higher-level APIs like the Outlook object model to create or modify Outlook items. Using the object model ensures that the items are created in a consistent manner so that the proper side effects are triggered. For example, when a meeting organizer changes the meeting time, an update needs to be sent to all attendees; simply changing the property would leave the meeting in an inconsistent state.

One significant difference between MAPI and the Outlook object model is that MAPI runs in the caller's process, whereas the Outlook object model runs on Outlook's foreground thread, making it very critical to write performant code. MAPI does not specify the protocol used between client and server. This architecture allowed Outlook to add support for the popular Internet protocols (POP, Simple Mail Transfer Protocol [SMTP], IMAP, Lightweight Directory Access Protocol [LDAP]) and formats (Multipurpose Internet Mail Extensions [MIME], iCal, vCard) that emerged in the mid-1990s on top of its MAPI-based infrastructure. This enabled ISVs to either rely on these standards to integrate with Outlook or use MAPI to implement a custom solution. Outlook has positioned itself as the premier offline client. One significant enhancement is cached Exchange built on top of a refined synchronization protocol when running against Exchange. ISVs can benefit from these capabilities by also making their solutions work seamlessly whether the user is online or offline.

Over time, the MAPI spooler was replaced in Outlook by the MAPI protocol handler, and the MAPI form servers infrastructure by Outlook forms and form regions in Outlook 2007. For additional information on development tasks using MAPI, please search for the MAPI Software Development Kit on MSDN.

MAPI Profiles

MAPI profiles allow storing different configurations for a specific user. A profile defines what stores, address books, and accounts, including preferences associated with them, should be loaded when Outlook is booted into this profile. MAPI profiles are stored in the registry under HKEY_CURRENT_USER (HKCU). For example, the user can have a profile configured with

a business Exchange account and another profile with private POP and Hotmail accounts. MAPI profiles predate Windows user profiles. One scenario for which they were introduced was multiple users sharing the same computer, but with Windows supporting fast user switching this is no longer a driving factor. When multiple profiles are set up, Outlook is typically configured to prompt at boot to allow the user to select which profile Outlook should load. Currently most users have only one single profile and add all the services they need to this single profile. A large percentage of Outlook users keep Outlook running throughout the entire day; closing Outlook and restarting the application is often too cumbersome to get to the mail sent to a different account. Solutions integrating with Outlook should be able to handle multiple profiles, but the majority of users run with only one profile.

MAPI Service Providers

MAPI service providers come in three varieties: transport providers, message store providers, and Address Book providers. Providers are dynamic-link libararies (DLLs) that implement a specific pseudo-COM API, such as IMessageStore, and the underlying required interfaces, such as IMAPIProp. MAPI providers are registered in Mapisvc.inf so the Outlook implementation of MAPI can discover these new services and make them available under Account Settings on the Tools menu. The user can configure these services for each Outlook profile. Administrators can also preconfigure Outlook profiles by creating and deploying Outlook profile file (PRF) files.

Note Outlook 2007 no longer relies on Mapisvc.inf to discover its own built-in MAPI providers.

The Outlook 2007 Integration API Reference available on MSDN includes updated sample code for different MAPI providers.

MAPI Message Store Provider A message store provider supports creating, submitting, and storing messages. Outlook relies on these MAPI messages to store Mail, Contacts, Appointments, and Tasks. Additionally, other Outlook components such as views or rules are stored in hidden messages. The stores associated with MAPI store providers show up in the Outlook folder list and in Outlook's Navigation Pane, shown in Figure 2-14.

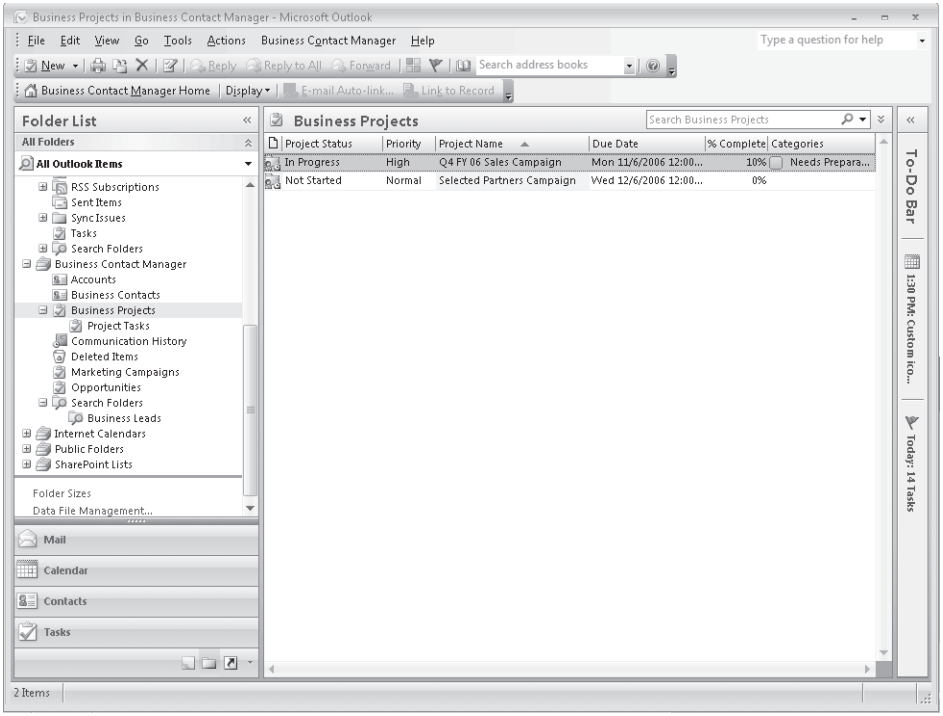

Figure 2-14 Business Contact Manager folder hierarchy implemented as MAPI store provider.

Outlook's Message Store Provider Outlook 2007 ships with the following MAPI message store providers:

- Mspst32.dll contains the MAPI provider for Outlook Personal Folders files (.pst). Offline Folder files (.ost) are also .pst-based and therefore also rely on this provider.
- Emsmdb32.dll is the Electronic Messaging System Microsoft Database provider. EMSMDB implements both a transport and a message store, and as such is a dual provider. The transport provides the ability to submit messages to Exchange Server. This provider also enables reading and writing messages to an Exchange store.
- Bcmms32.dll implements the MAPI store provider for the Business Contact Manager functionality of Outlook 2007. This storage is SQL Express and contains Contact-based items (Account and Business Contacts), Task-based items (Opportunity, Business Projects, Project Task, Marketing Campaign) and Journal-based items (Business Notes and Phone logs) but no Mail, Calendar, and Notes items or folders.

Custom Message Store Provider Writing a MAPI store provider allows a solution to integrate data from another source like a local database, server back end for messaging, CRM, or LOB application into Outlook. Writing a message store provider is complex and requires unmanaged C++ and MAPI development proficiency. Before you consider writing a custom

message store provider, it's recommended that you also evaluate other alternatives. An alternative to a MAPI custom store provider is to cache the data in a .pst-based store by using the Outlook Replication API documented in the Outlook 2007 Integration API Reference.

MAPI Transport Providers The purpose of a MAPI transport provider is to facilitate the transmission of messages over different protocols. An example would be sending e-mail messages over a Web service or using a separate transport to communicate with a back-end service. Microsoft LiveMeeting uses a separate transport provider to communicate with LiveMeeting server.

Outlook's MAPI Transport Providers Outlook 2007 ships with the following MAPI transport providers:

- Omsxp.dll implements the Outlook Mobile Service (OMS) transport provider for sending Short Message Service (SMS) and Multimedia Messaging Service (MMS) text and multimedia messages through the Outlook Mobile Service Web service to a cellular carrier that routes these messages to mobile phones. The OMS provider delivers to recipients with e-mail type MOBILE. The OMS provider also converts the MAPI message to an XML payload as part of submitting the message.
- Emsmdb32.dll is the Electronic Messaging System Microsoft Database provider. As mentioned earlier, EMSMDB implements both a transport and a message store, and as such is a dual tightly-coupled provider.

Note Outlook does not internally rely on a physical MAPI transport provider for submitting messages through SMTP (POP or IMAP configurations), HTTP (Hotmail), or Exchange (cached Exchange mode).

Custom MAPI Transport Providers Outlook Mobile Service offers the canonical example of such a provider. New to Outlook 2007, Outlook Mobile Service enables delivery to recipients with a custom e-mail type such as MYADRTYPE. When the user sends a message to such a recipient, MAPI will ask each registered transport if it knows how to deliver to recipients with type MYADRTYPE. Only the custom MAPI transport provider knows how to transport this item and therefore gets to deliver the message. The transport provider can serialize the MAPI message into another format such as XML or MIME during the submission process.

MAPI Address Book Provider An Address Book provider enables recipient e-mail lookup and directory browsing. Recipients can be either single users or distribution lists. An Address Book provider introduces a new list of entries into the Address Book. As shown in Figure 2-15, the OMS address list shows up in the Address List drop-down box along with Global Address List and Outlook Address Book in the Outlook Address Book. This address list will also be searched during name resolution when a user clicks the Check Names button on the Ribbon or during automatic background name resolution.

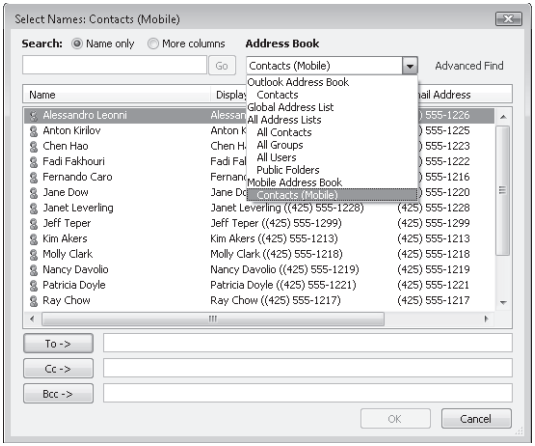

Figure 2-15 Outlook Mobile Service Address Book provider.

Outlook's MAPI Address Book Provider Outlook 2007 ships four different MAPI Address Book providers:

- Emsabp32.Dll (Electronic Messaging System Address Book provider) implements the Address Book provider used when Exchange accounts are configured. It implements *IAddrBook* and supports both online mode against Microsoft Active Directory directory service via Name Service Provider Interface (NSPI) and offline mode against Offline Address Book (OAB), which is either downloaded from an Exchange Public Folder or with Background Intelligent Transfer Service (BITS) from a Web server.
- Contab32.dll contains the Outlook Address Book, also known as the Contact Address Book provider. This provider exposes Personal Contacts Personal Distribution Lists stored in Contacts folders in the Address Book.
- Emablt32.dll is the Outlook LDAP provider enabling Outlook to connect to a variety of different LDAP servers.
- Omsxp.dll implements the Outlook Mobile Service Address Book provider, which allows addressing of SMS messages. It's a sibling of the Outlook Address Book. It shows the mobile number instead of the e-mail address for Outlook Contacts. When the user picks a user from this provider, a recipient with type MOBILE is created. This provider is configured once the user adds an Outlook Mobile Service account to an Outlook profile.

Custom MAPI Address Book Provider An Address Book provider allows integrating another source of contact information into Outlook, typically for addressing e-mail messages. Writing such a provider is difficult, so alternatives should be considered. If possible, rely on the built-in LDAP provider or, if the data set is relatively small, replicate data into an Outlook Contacts folder and rely on the Outlook Address Book provider.

Outlook 2007 Integration API Reference

The Microsoft Office Outlook 2007 Integration API Reference provides a set of complimentary APIs that allow developers to create a tight integration with Outlook. The Replication API is worth looking at especially for data integration scenarios. These APIs are version specific; they were introduced for Outlook 2003 (meaning they're not supported for versions before Outlook 2003) and were enhanced and slightly modified for Outlook 2007. Although it's a platform goal to maintain compatibility for these APIs, it should be assumed that solutions relying on these APIs need to be revised for every major version of Office. Because these APIs offer deep Outlook integration, they have to be adjusted when the internal Outlook architecture evolves. Fortunately, the transition from Outlook 2003 to Outlook 2007 involves very little work.

The components of the Outlook 2007 Integration API Reference are as follows:

- The Account Management API provides access to user account information and notifications of account changes.
- The Offline State API supports Outlook callbacks, notifying clients of changes in a user's connection state in Outlook—for example, from being online to being offline in Outlook.
- The Data Degradation Layer API enables data access to stores, items, and folders in either Unicode/Wide or ANSI (Windows System Codepage) for both Unicode and ANSI stores. For ANSI stores the data is downgraded (degraded) from Unicode to ANSI.
- The Free/Busy API provides free/busy status information about specific user accounts within a specific time range.
- The MAPI–MIME Conversion API supports conversion between MIME objects and MAPI messages.
- The Replication API supports synchronizing Outlook folders and Outlook items between a local store and a server. The most powerful API documented in this reference, it allows .ost-based offline storage and replication. A CRM solution can, for example, use these APIs to replicate data from the CRM database back end.
- The Store API provides miscellaneous store functionality.

Simple MAPI

Simple MAPI is a very limited library of only 12 functions. Originally, Simple MAPI was developed to enable the Microsoft Mail client to communicate with Microsoft Mail post offices. Extended MAPI completely supersedes the older version. These simple APIs continue to offer applications an easy way to offer e-mail capabilities by integrating into the default e-mail program. For example, Office applications or the Windows Shell use Simple MAPI for Send To functionality. This API is not recommended for further use.

Deemphasized and Phased-Out Components

Outlook solutions should no longer rely on the following technologies and APIs. Although most of these components continue to be supported in Outlook 2007, you should consider updating your code to use the new technologies introduced in this version of Outlook. If you are creating a new solution that only targets Outlook 2007, use of these deprecated technologies is not recommended.

Collaboration Data Objects 1.2.1

CDO 1.2.1 is an API that sits on top of MAPI for creating, sending, and receiving e-mail as well as calendaring and public folder applications. Thanks to the unification effort for Outlook 2007, all critical CDO functionality is now incorporated in the Outlook 2007 object model. If you use CDO, you should consider removing your solution's dependency on CDO. CDO used to be an optionally installed component, but with Office 2007 it's only available as a Web download. Existing versions of CDO will be removed by Office 2007 Setup.

Exchange Client Extensions

As the name indicates, this extensibility technology was introduced for the Microsoft Exchange 4.0 client to customize menus and toolbars, preprocess outgoing and incoming messages, add property sheets, and provide MAPI forms integration. Outlook, the successor of the Exchange client, later added support for ECEs. Outlook 2007 added a large number of new events to the Outlook object model, enabling developers to phase out their code relying on ECEs. ECEs also enable the customization of command bars for Outlook items; these customizations manifest themselves on the Add-ins tab of the Ribbon.

Outlook Custom Forms with Form Pages

Outlook 2007 continues to support Outlook custom forms with form pages, except for one-off form definitions, which were deprecated for security and reliability reasons. In one-off forms, the form is embedded in the message and contains definitions for custom properties and VBScript. One-off forms no longer run script and do not propagate custom properties to a recipient. A one-off form will render in a built-in form instead of the custom form definition stored in the form. Although Outlook custom forms with form pages continue to be supported in Outlook 2007, solutions that target Outlook 2007 should use form regions in place of these legacy forms.

MAPI Form Server (IMAPIForm)

MAPI Form Server is a pluggable forms infrastructure introduced with MAPI 1.0. Outlook 2007 continues to support this technology, but it's expected to be deprecated in a future Office release.

Exchange Forms Designer Forms

Exchange Forms Designer forms were included with Exchange 4.0 and 5.0 and were designed for the pre-Outlook Exchange client. Outlook 97 also included the Exchange Forms Designer. This form development package has its own design environment based on 16-bit Microsoft Visual Basic 4.0. Outlook has supported using these forms, but since Outlook 2002 you must download the necessary runtime files for these forms to work. Although they are considered obsolete, Outlook 2007 continues to support these forms if you have the runtime installed.

Electronic Forms Designer (Microsoft Mail 3.0) Forms

Electronic Forms Designer forms are based on 16-bit Microsoft Mail 3.0 technology. These forms work in Outlook 97 through Outlook 2002 and have a dependency on the Microsoft Mail 3.0 client extension. Outlook 2003 and Outlook 2007 do not include this extension and therefore Electronic Forms Designer forms are not supported in Outlook 2003 and later versions.

Development Tools

The following section discusses development tools that you can use to create an Outlook solution. The development tool that you select to create your solution depends in part on your scenario and also on the resources within your company. If you have a large number of developers who have moved from Microsoft Visual Basic 6.0 to Microsoft Visual Basic .NET, then Visual Basic will be a natural choice for your development tool. If you have trained C# developers, then C# would be your preference. Although the sample code in this book has been written using C#, the code is also provided online in Visual Basic versions. The Microsoft Visual Basic .NET version of the sample code is not listed directly in the book because of space considerations rather than as a matter of preference. Visual Basic .NET sample code is available on the Web site for this book.

Visual Basic for Applications

Microsoft Visual Basic for Applications (VBA) is the development environment that ships with Outlook 2007. However, unlike other Office applications like Word and Excel, Outlook VBA is a prototyping tool only. Although Outlook VBA can be used to create personal productivity macros, it is not suitable for the development and deployment of a professional solution. The limitations of Outlook VBA are as follows:

- The Outlook VBA project is contained in a single file named VBAProject.otm. Only one Outlook VBA project can be deployed in a given user profile on a machine. This architecture means that only one VBAProject.otm can be deployed for a user. If another solution overwrites the VBAProject.otm file, the customizations in the original file are lost.

- Unlike other Office applications, Outlook VBA cannot be attached to a document so that customization travels with the document or template.
- Custom task panes are not supported in Outlook VBA.
- You cannot customize the Ribbon using Outlook VBA. Only command bars can be customized in Outlook VBA.
- You cannot record macros in Outlook VBA.

Visual Studio Tools for Office

Microsoft Visual Studio 2005 Tools for the 2007 Microsoft Office System (VSTO) is a recommended development environment for Outlook add-ins. As shown in Figure 2-16, VSTO provides an abundance of benefits to the Outlook developer. Due to scheduling considerations, VSTO is not used for the sample add-ins in this book. The sample add-ins use a COM shim architecture and do not utilize VSTO features. Both MSDN and third-party publications will provide you with plenty of information to get you started using VSTO. A short list of VSTO benefits includes the following:

- Use of the Common Language Runtime (CLR) 2.0. The CLR offers type safety, memory management and garbage collection, and a host of useful classes and features that are well documented in the extensive literature available on MSDN.
- A professional development environment with integrated debugger.
- Templates that provide the framework for creating a managed Outlook add-in using either C# or Visual Basic .NET.
- AppDomain isolation. For an Outlook developer, AppDomain isolation prevents an add-in from crashing due to an unhandled exception in another add-in. It also prevents an add-in from crashing other add-ins in a shared AppDomain.
- Support for new 2007 Office system extensibility features, including:
  - Outlook form regions
  - RibbonX
  - Custom task panes

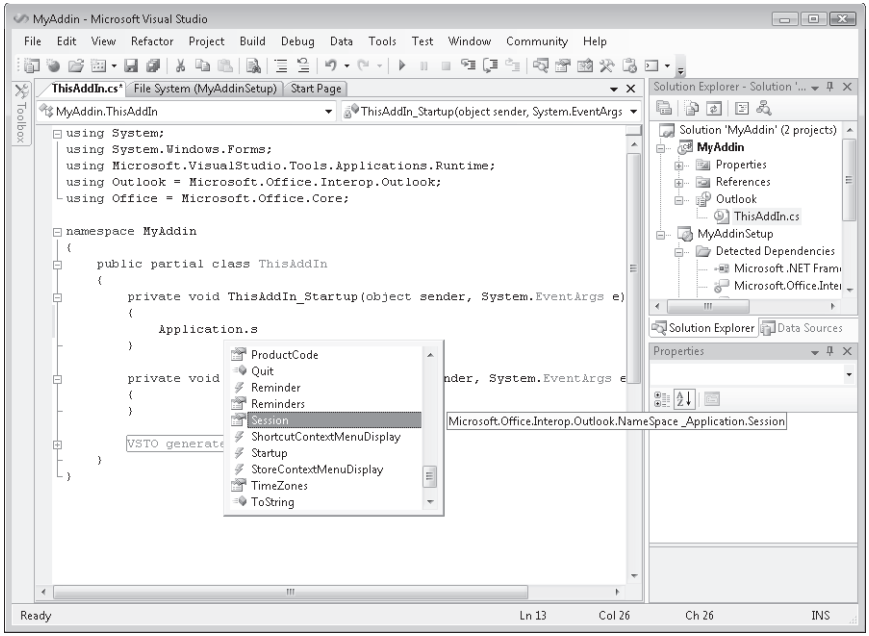

Figure 2-16 Using VSTO to develop an add-in for Outlook 2007.

Managed Versus Native Code

Microsoft Visual Studio represents the strategic development platform for Microsoft, and the recommendation of Visual Studio 2005 or VSTO for Outlook add-in development follows that strategic direction. If your team is trained in either Visual Basic .NET or C#, they will be able to develop professional Outlook solutions using the guidance provided in this book. However, you might wonder if managed code is suitable for all Outlook solutions. The answer, of course, depends on your scenario. If your solution requires development of any of the following platform components, you should consider using a native code development environment such as Microsoft C++:

- Transport, Address Book, or custom store providers
- Synchronization using Replication API
- Extended MAPI functionality not available in the Outlook 2007 object model

Another indication that you should consider native code rather than managed code is performance. Although Outlook 2007 has targeted performance improvements over previous versions, you should understand that managed code requires the Outlook Primary Interop Assemblies (PIAs) to provide the Interop layer between .NET Framework assemblies and the COM-based Outlook object model. This Interop layer does add a performance consideration to using managed code. Although this performance layer is not an obstacle for the vast majority

of Outlook solutions, it is a gating factor if your solution repeatedly enumerates large numbers of Outlook items, for example. Remember that the Outlook object model operates on the foreground thread, so operations that poll the Outlook data store using the object model and the foreground thread can degrade the performance of Outlook as a whole. When you have to satisfy stringent performance requirements, you should consider a native development environment such as C++. If you must write native code, you should also weigh the additional cost in time and resources required by C++ development against going with a solution built using C# or Visual Basic .NET. The final decision is yours, and the choice of development tool is dictated first by your scenario, and second by your resources and budget.

Add-In Model

The primary customization technology for the Outlook 2007 platform is a COM add-in. Of course, you can create a managed add-in using VSTO or Visual Studio 2005. Managed add-ins use an Interop layer to connect to the Outlook host application, but they still use the basic COM add-in technology. In Outlook 2007, add-ins have achieved new prominence because an add-in is the only way that you can create Outlook 2007 form regions, Ribbon customizations for an Outlook Inspector, and Office custom task panes. Add-ins are also the preferred method for creating event listeners that implement the business logic of your solution. Your add-in can leverage the improved event model in Outlook 2007 to determine when an item is being saved or sent, an attachment is being added or removed from an item, an item is being deleted from a folder, or a custom or built-in item property is being changed. And those are just a few of the event listeners that are possible in Outlook.

Outlook has supported COM add-ins since Microsoft Office 2000, which is when the COM add-in architecture was first introduced in Office. The core architecture involved in creating a COM add-in has not changed in terms of how the Office application communicates with COM add-ins. Office 2007 continues to use the same architecture that was first implemented in Office 2000.

Overall, unless there is a specific reason that dictates otherwise, COM add-ins are the recommended way for developers to create a custom solution that integrates with Outlook. Of course, there are many factors to take into account when designing a custom solution, but key benefits of using a COM add-in in Outlook include the following:

- COM add-ins are the supported way to create an Outlook-based solution that can be deployed (as opposed to Outlook VBA). Deployment can occur through a standard Windows installer (.msi) or through a push technology such as Microsoft Systems Management Server.
- COM add-ins allow a solution to be trusted so that Object Model Guard security prompts are not generated when the object model is used to access the address book or recipient information, or to programmatically send e-mail. In Outlook 2007, all add-ins are trusted by default from the perspective of the Outlook object model guard. In locked-

down environments, IT administrators can use the Outlook security form in Exchange public folders or use group policy objects to control a list of trusted add-ins.

- Even if you are developing a standalone application, you could also develop a proxy add-in and have the standalone program interact with the COM add-in by using public methods and properties exposed by the add-in. In this way, any functionality that would typically generate a security prompt can be done from within the add-in. Outlook 2007 also provides the ability to turn off security prompts when antivirus software is installed and current. This ability can also be controlled via group policy.
- COM add-ins are the replacement technology for ECEs. Previous versions of the Outlook object model did not provide some key functionality that you could achieve in a client extension, so many developers still developed extensions in C++ because they provided more options. However, due to the many object model improvements in Outlook 2007, the COM add-in architecture is even more viable for many developers.

There are some scenarios where an add-in is not recommended for an Outlook solution. It's worthwhile to point these out so that you don't attempt to create a solution with the wrong technology. Add-ins and the Outlook object model are not recommended if you need to create any of the following components:-

- **Windows service application** Because the Outlook object model can display UI elements and operates on Outlook's foreground thread, the Outlook object model and add-in technology are not suitable for use in a Windows service application. For a service application, you can use CDO or Extended MAPI. Because CDO is being deprecated in this release, it is suggested that you use Extended MAPI for a Windows service application.
- **Web application** The Outlook object model is not suitable for use in a Web application. Exchange 2007 offers Web services for use in a Web application. Previous versions of Exchange offer the Web Distributed Authoring and Versioning (WebDAV) protocol or the Exchange OLE DB (ExOLEDB) provider for use in Web applications.

Summary

This chapter has taken you on a tour of the Outlook platform. Along the way, you've learned how some solutions that ship with the 2007 Office system have taken advantage of the Outlook 2007 object model and form regions. In brief, this chapter has described the pillars of the Outlook 2007 platform enhancements. You've also been exposed to how Outlook relies on MAPI as the cornerstone of its internal architecture. During this discussion, prescriptive guidance has been provided to offer suggestions about how to build solutions on top of Outlook. Although prescriptive guidance must always be tied to practical scenarios, it's hoped that you will follow these guidelines to make your solution integrate seamlessly with Outlook 2007.