Designed for Microsoft® Windows NT® Windows® 98

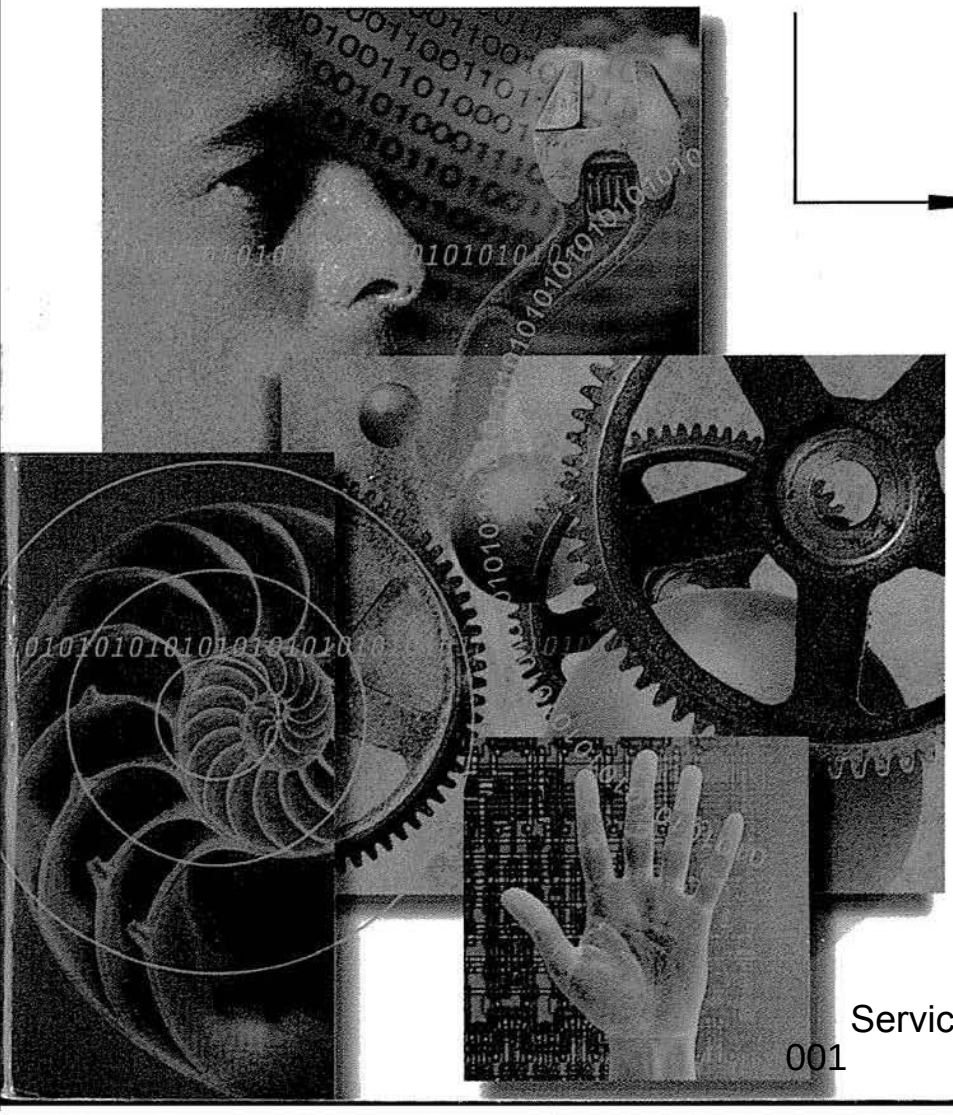
FOR ENTERPRISE DEVELOPERS

MICROSOFT® PROGRAMMING SERIES

CD-ROM Included

Microsoft Press

Programming Microsoft® Outlook® and Microsoft Exchange



Build collaborative business solutions with Microsoft Outlook 98/2000 and Microsoft Exchange Server 5.5

Thomas Rizzo

Programming Microsoft® Outlook® and Microsoft Exchange

Thomas Rizzo

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Rizzo, Thomas, 1972-
Programming Microsoft Outlook and Microsoft Exchange / Thomas Rizzo.
p. cm.
ISBN 0-7356-0509-2
1. Application software--Development. 2. Microsoft Outlook. 3. Microsoft Exchange. I. Title.
QA76.76.A65R59 1999
005.369--dc21 99-13555
CIP

Printed and bound in the United States of America.

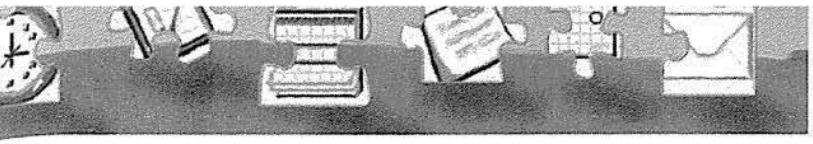
3 4 5 6 7 8 9 QMQM 4 3 2 1 0 9

Distributed in Canada by Penguin Books Canada Limited.

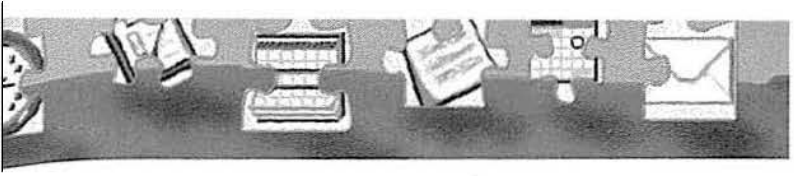
A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at mspress.microsoft.com.

Acquisitions Editor: Eric Stroo
Project Editor: Victoria Thulman
Technical Editor: Robert Lyon

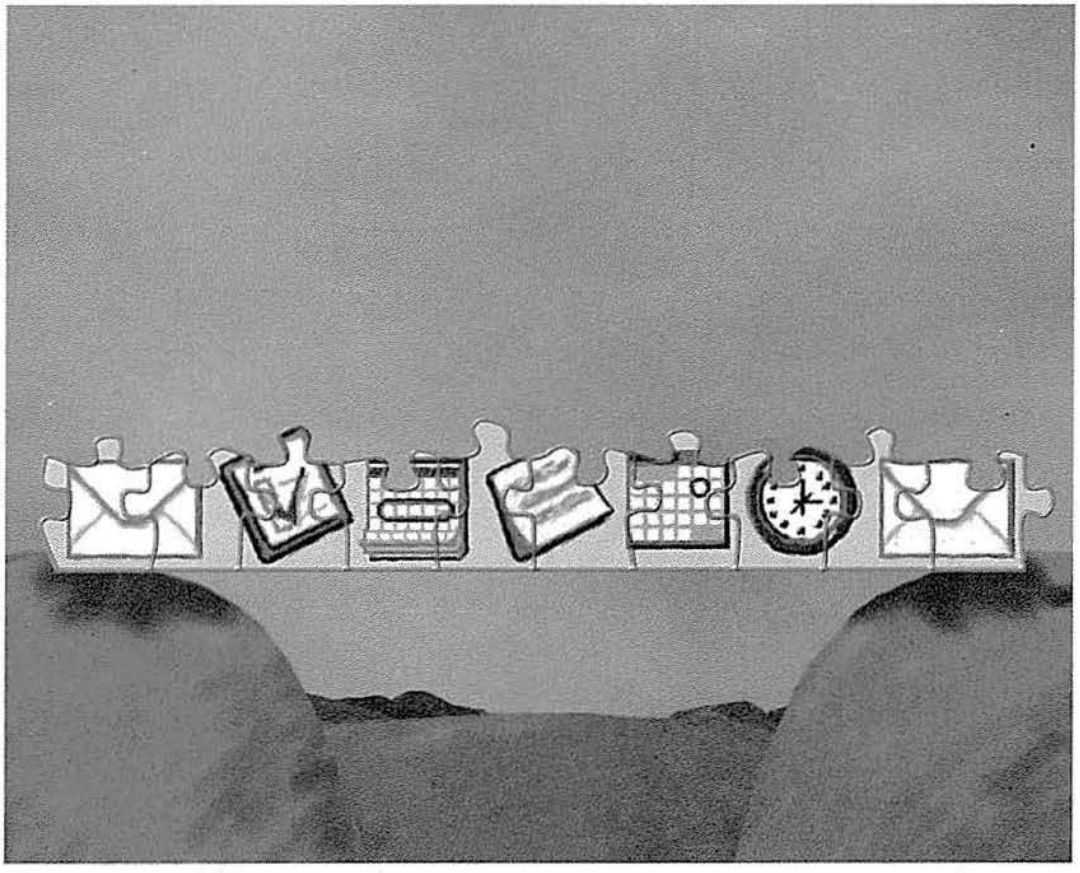


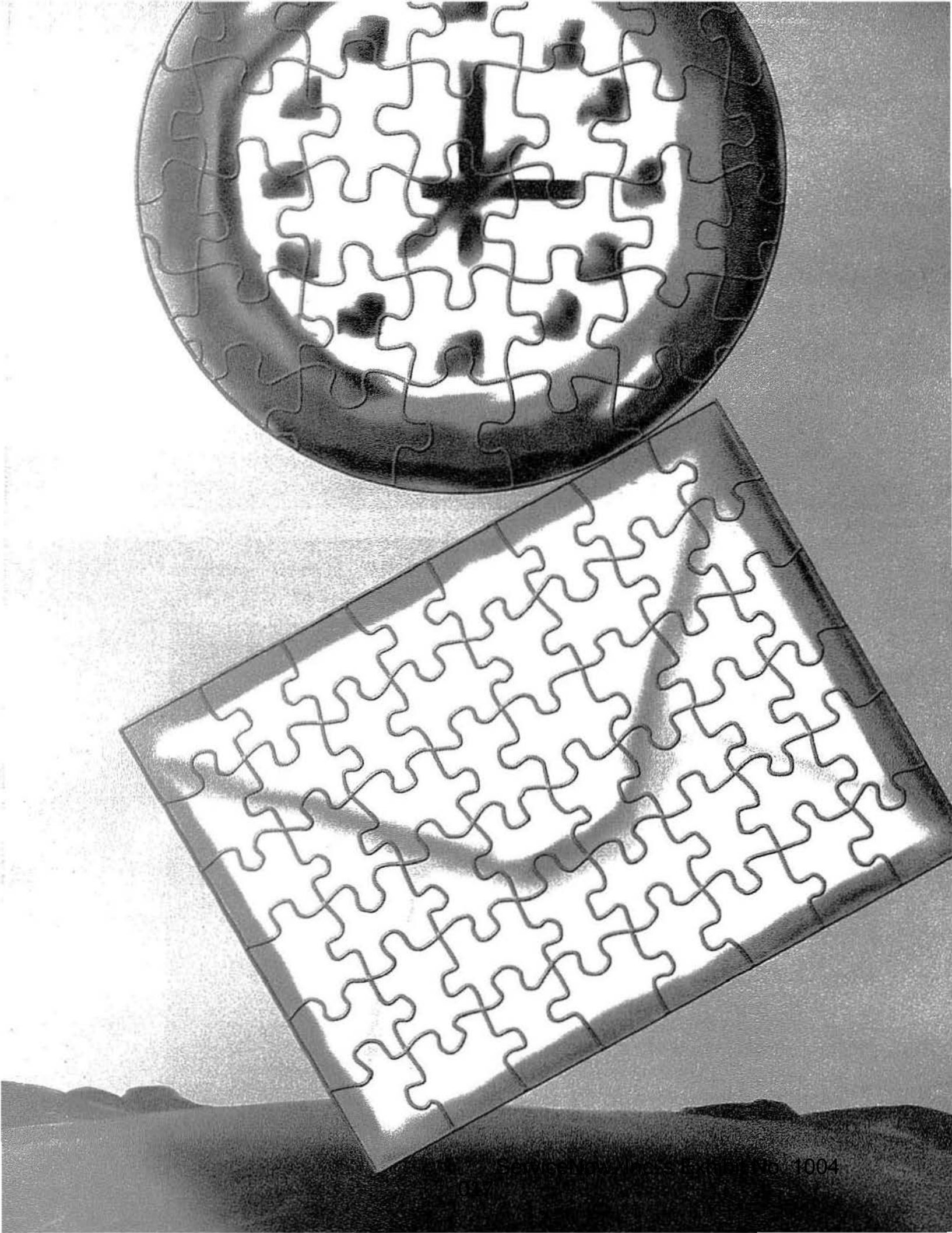
CONTENTS

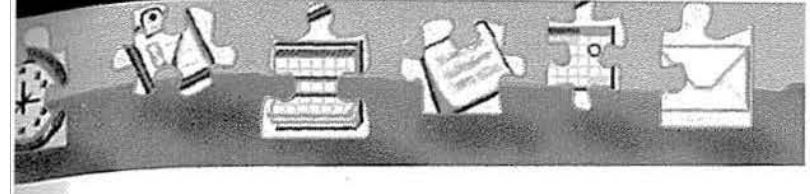


PART I

INTRODUCTION TO COLLABORATIVE SYSTEMS







CHAPTER ONE

A Broader Definition of Collaboration

If you asked ten different people to define collaboration in a computer environment, you would receive ten different answers. Some would say collaboration is e-mail. Others would mention video teleconferencing or the World Wide Web. You might even hear Internet chat as an answer. People struggle to define collaboration because there are so many technologies and its definition today is broad. Really, all of these answers are correct. Collaboration—at least in part—is the integration of many different technologies into a single application or environment to facilitate information sharing and information management.

Integrated technology, however, is only one aspect of collaboration as we're defining it. Timing is another. We're all familiar with real-time collaboration in which you work with others at the same moment, taking turns communicating ideas. But new technology offers you an entirely different way to collaborate—asynchronous collaboration—in which you don't have to be present to participate. Asynchronous collaboration allows you, at your convenience, to collaborate with other people, at their convenience. E-mail, public databases, the Internet, and intranets are all forms of asynchronous communication.

Collaborative technology provides these key benefits to businesses:

- *Extensive, secure communication.* Collaborative technologies enable extensive communication through many different mediums and secure communication through encryption and digital signature technology, which is critical as businesses increase their use of the Internet.
- *Storage of information in a central location.* Information is placed in a central repository, or database, so that individuals inside and outside a corporation can access it. If shown in a threaded view, the history of the information is accessible and new information can be added to it.

- *Ability to extend existing technologies with new functionality and bridge islands of information.* Collaborative systems connect disparate systems and facilitate finding and sharing information stored in existing technologies. Essentially, they bridge islands of information.

How does a collaborative system provide these benefits to corporations? In terms of its architecture, a collaborative system must have several characteristics. First, it must have a robust, replicated object database that can store many different types of information such as web pages, office documents, and e-mail messages, and it must support replication both from server to server and from server to client. This replication allows geographically dispersed individuals to access diverse information. To work with the data, the database needs to allow many different clients, ranging from web browsers to e-mail clients.

Second, it must support the Internet and industry standards. The days of stovepipe computing are over. New technologies are connecting disparate networks to form one global, cohesive network. A collaborative system must be able to interoperate with these networks over the Internet, and it must follow industry standards to allow openness to a large number of external systems as well as guarantee the integrity of the data.

Third, a collaborative system must offer powerful, easy-to-use development tools and technologies. The environment must be open so that developers can use any tool to develop solutions and users can access and customize the user interface.

Tools for Building Collaborative Systems

Microsoft offers a number of products and tools that are designed to help you leverage a company's current technology investments and extend them with new functionality. These tools, which fall under three key product types, are listed here:

- *Client products.* Tools include Microsoft Outlook and Microsoft Internet Explorer.
- *Server products.* Tools include Microsoft Exchange Server, Microsoft SQL Server, Microsoft Internet Information Services and Microsoft Site Server.
- *Development products.* Tools include Microsoft Visual Studio, which encompasses Microsoft Visual Basic and Microsoft Visual InterDev.

The two main tools you will want to learn are Microsoft Outlook and Microsoft Exchange Server. Both provide a robust infrastructure with which

corporations can run mission critical services. Combine this infrastructure with the rich development tools provided by both products and you have a powerful platform on which you can write solutions. The type and complexity of these solutions can range from simple forms to complex applications. The next few sections briefly describe some products and tools available from Microsoft for building collaborative solutions.

Microsoft Outlook

Outlook supports the ability to manage information (e-mail messages, appointments, contacts, and tasks) and share it throughout an organization. Outlook also includes a development environment that allows you to write collaborative applications quickly. Part II of this book is dedicated to Outlook and its development environment. Chapters 9 and 10 in particular discuss and illustrate the features of Outlook 2000 that will enable you to further extend Outlook.

Microsoft Internet Explorer

With the ubiquity of the Internet, browser technology is becoming increasingly important for user collaboration. Internet Explorer, with its support for dynamic HTML, scripting, and security, is an ideal client interface for your applications. In Chapters 11, 12, and 13, you will see how to take advantage of Internet Explorer using both Outlook and Exchange Server.

Microsoft Exchange Server

Exchange Server, which is part of the Microsoft BackOffice suite of products, is a linchpin for any collaborative system because it supports communication, information sharing, and workflow services that use Internet standards and protocols. Chapter 3 of this book provides an introduction to Exchange Server.

Microsoft SQL Server

SQL Server is a relational database system that offers easy storage and retrieval of relational information. Its built-in data replication, powerful management tools, Internet integration, and open system architecture allow you to integrate SQL Server into existing environments cost-effectively.

Microsoft Internet Information Server

Internet Information Server is a free web server available for Microsoft Windows NT Server. It provides an easy way to publish and share information securely over corporate intranets and the Internet through HTML documents. The power of IIS is demonstrated when web applications are written using its

built-in server-side script technology called Microsoft Active Server Pages (ASP). ASP allows developers to write applications by using any ActiveX scripting language, such as JScript and Microsoft Visual Basic Scripting Edition (VBScript). These scripts execute on IIS and can access different data such as that provided by Exchange Server or SQL Server. The information returned from the server-side script is in HTML, making these applications compatible with any standard HTML web browser such as Internet Explorer. Chapter 8 introduces Active Server Pages and its programming model.

Microsoft Site Server

Site Server is a web publishing, analysis, and search tool. Because Site Server is integrated with Windows NT Server and IIS, you can easily set up and deploy intranets. Site Server helps corporations get the most from their intranets by implementing best practices for publishing and staging intranet content.

Site Server can also implement content tagging. Content tagging is a structured, site vocabulary that authors use to classify the web content they create. When used in conjunction with Site Server's integrated search and knowledge management capabilities, these tags enable users to more easily find information. Plus, Site Server integrates and manages the information from other BackOffice products through full-text indexing of these different data sources.

Microsoft Visual Studio

Visual Studio is an integrated and comprehensive suite of development tools for building web-based or Microsoft Windows–based applications. You can quickly build collaborative solutions that take advantage of the BackOffice family of products because Visual Studio and BackOffice are integrated. Throughout this book, you will see examples of collaborative solutions that use Visual Studio tools.

Microsoft Visual Basic

Visual Basic is an effective and easy-to-use tool for creating high-performance windows applications. It includes a rapid development environment with graphical layout tools and great performance because of native code compilation. Visual Basic also creates open, industry-standard ActiveX components. These components can provide functionality to other applications whether they are web-based or Windows-based.

Microsoft Visual InterDev

Visual InterDev empowers web application developers to rapidly build fully interactive, dynamic web sites. With visual development features and powerful database tools, Visual InterDev provides the most complete and technically

advanced development system for building both intranet and Internet applications. Through the use of Visual InterDev Design–time controls and wizards, you can add collaborative technologies to your web applications.

Examples of Collaborative Solutions

Now that you have a better understanding of collaboration and collaborative technology, let's look briefly at the systems you can create. With Exchange Server, you can build many different types of open and extensible applications, all of which can take advantage of information stored inside and outside of Exchange Server. You can leverage other data sources in your organization, such as SQL databases. This openness to other data sources allows you to pick the best database for storing the application's information without compromising the user interface consistency.

The types of applications you can build can be broken down into five categories: messaging, tracking, workflow, real-time, and knowledge management. None of these application categories are mutually exclusive—for example, a workflow application can take advantage of messaging services. Rather, these categories define the primary function of a particular application. Throughout this book, we'll explore sample applications that fall into these five categories.

Messaging Applications

Messaging applications use primarily the messaging infrastructure of Exchange Server. E-mail is the most well-known of these, but you can build many other types, such as discussion group applications. Exchange Server supports threaded discussions; any folder in Exchange Server can be a threaded discussion folder by changing the view of the messages inside that folder. These discussions can be replicated to and from Internet newsgroups and can be moderated for the appropriate content.

Another example of a messaging-based application is a mailbox agent. A mailbox agent can perform many different types of functions based on how it is programmed. For example, suppose a sales force needs the ability to run certain queries against a database of sales information. Although you could write a Microsoft Access application that queries the database and returns the results, a salesperson wouldn't be able to work on other items until the database processed the request and returned the data set in the Access user interface. This means that users would have to check the Access application continually to see whether the data was available. However, with a mailbox agent, a salesperson could use a form to specify the type of information she needed and then e-mail the form to the agent. The agent would process the form and run the query on her behalf.

Once the database was finished processing the query, the agent would e-mail the data set to the salesperson. Eventually, she receives notification containing the requested data set.

A mailing list server is a messaging agent that forwards all mail it receives to its registered recipients and allows users to add and remove themselves from the recipient's list via e-mail.

A document library is another example of a messaging application. Users can submit documents to a library by dragging and dropping them, e-mailing them, or sending them through a web browser. Because these libraries are stored in a central location, many users have access to the documents. Intelligence can be added to the library by creating a mailbox agent that notifies users when new documents are available. Custom views are available on a folder so that users can quickly find desired documents. Comments about the documents can be placed in the folder, and interested users can gauge their relative value.

One popular example of a document library is a library of web favorites. A user can set up a document library to store a corporation's favorites in a central location. By dragging and dropping Internet shortcuts into this library, a user's personal favorites become corporate favorites. Plus, users get the benefit of being able to create custom fields and views that describe and categorize the favorites in the folder.

NOTE: To see a mailing list application in action, sign up for the Microsoft Exchange Server mailing list at *http://www.msexchange.org*.

Tracking Applications

Tracking applications manage and track information, such as a list of contacts, from its creation to its deletion or "completion." Tracking applications usually require the integration of many different data sources because the information needing to be tracked typically resides in more than one location.

One example of a tracking application is a job candidate tracking application, which enables a human resource department and other employees to track a prospective employee from the moment he submits a resume through the interview process and finally to the decision to hire or reject. The candidate's status is always available for review. Figure 1-1 shows a hypothetical example of a job candidate tracking application that uses Outlook and Exchange Server to track prospective candidates.

You could also create an account tracking application, which includes tracking for contacts, revenue, and tasks. Figure 1-2 shows the Account Tracking application we'll build in Chapter 7. In Chapter 10, we'll enhance this application for Outlook 2000.

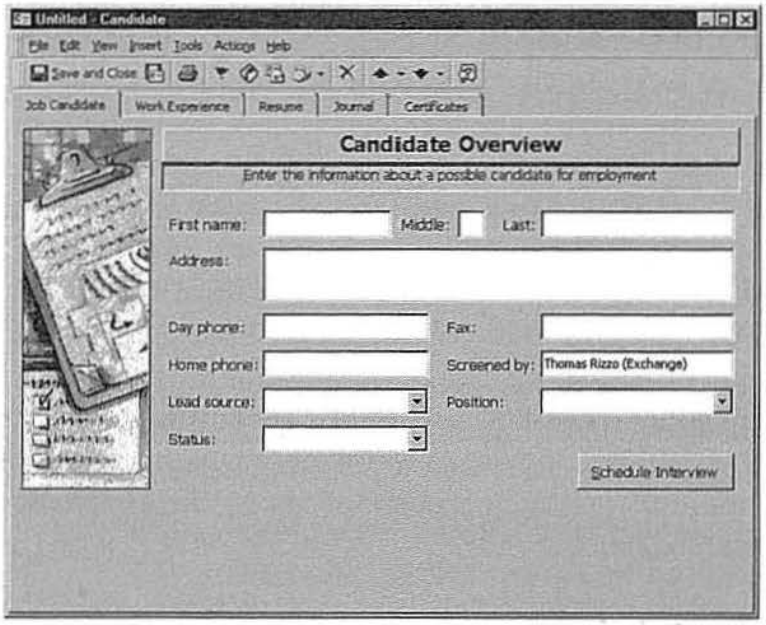


Figure 1-1
A hypothetical job candidate tracking application in Outlook.

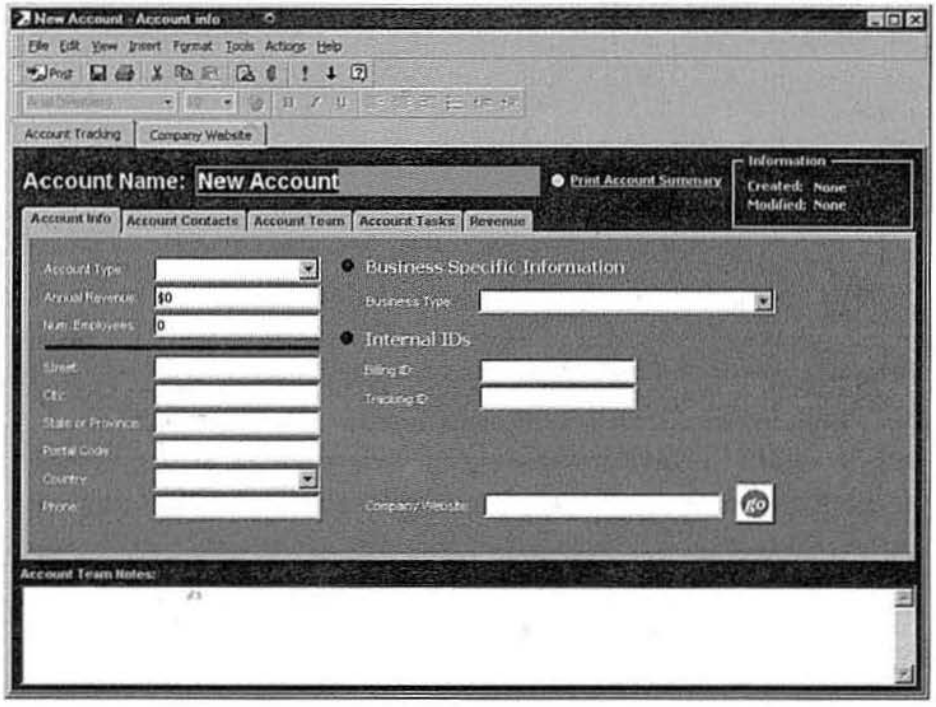


Figure 1-2
The Account Tracking application from Chapter 8.

Helpdesks are also tracking applications. In a helpdesk application, trouble tickets are submitted to the help desk by users specifying technical problems. Problems are assigned to technicians based on the ticket type. Audit trails are established for each ticket so that the technicians have historical information that helps them work on the problems. After fixing a problem, the technician adds the ticket and its resolution to a log of frequently asked questions, which users can query.

A helpdesk might include other tracking applications as well, such as inventory management. For example, if the technician had to request a new machine for the user, an inventory management program informs the technician whether a new machine is in stock. By adding a workflow application to the help desk application, the technician could obtain approval for the machine from the user's manager and the help desk manager. Figure 1-3 shows the Helpdesk application we will build in Chapter 11.

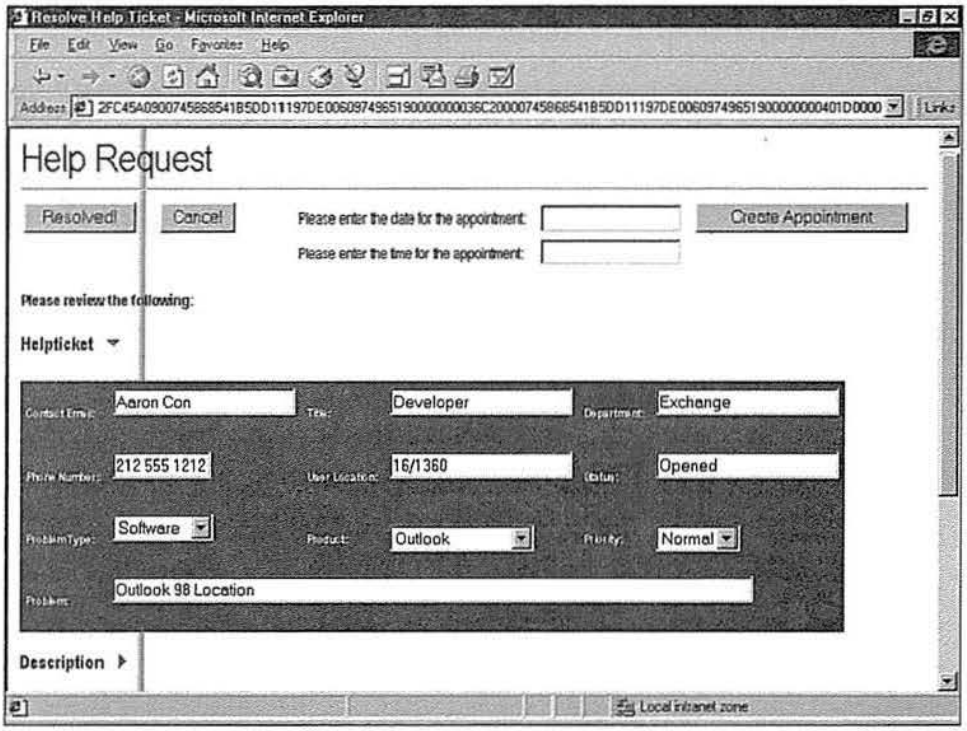


Figure 1-3
The web-based Helpdesk application from Chapter 11.

One last example of a tracking application you might build is a class registration application, which tracks information about a class and its participants. It informs users when desired classes become available, reminds them of which classes they are registered for at least one day in advance, and notifies them of any updated materials made available by the teacher. When the class is completed, class notes and a survey can be distributed to class members.

Workflow Applications

Workflow applications are primarily constructed around three concepts, which are known as the three Rs—Roles, Routes, and Rules:

- *Roles.* A role is the logical representation of a person or an application in a workflow process—for example, an expense report approver. Roles can change dynamically depending on who is involved in the particular workflow process. They allow you to easily abstract the different functions people perform in a workflow process.
- *Routes.* A route defines what information will route and who will receive it. Routes can be sequential, parallel, conditional, or any combination of these. Figure 1-4 illustrates three types of routes.
- *Rules.* A rule is conditional logic that assesses the status of the workflow process and determines the next steps. Here's an example of a rule: *if the manager approves the expense report, route the report to accounting, or else send the expense report back to the submitter.* A rule can be based on the properties of a message or on some other data source.

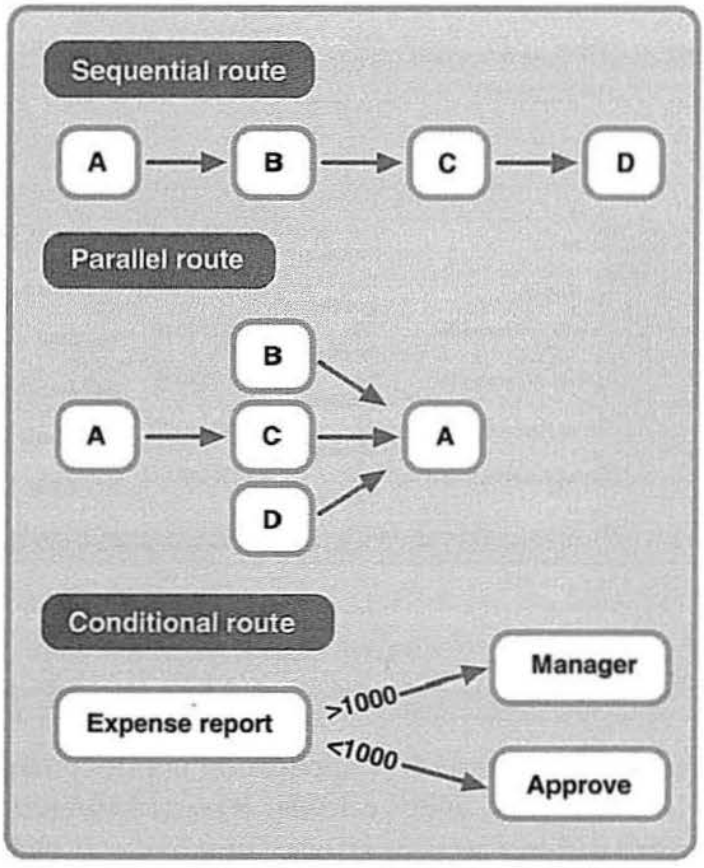


Figure 1-4
Sequential, parallel, and conditional routing types.

Let's take a brief look at a few workflow samples. The Expense Report application, which is discussed in detail in Chapters 12 and 13, is one example of a workflow application that you can build with Exchange Server. Here's how such a workflow application might function: a user submits expense reports from a web application, and based on the total amount of the expense reports, a particular workflow process is started. If the expense is under $5,000, the expense report is automatically approved; if the expense is over $5,000, the report is routed to the user's manager for approval. The manager either approves or rejects the expense report, and based on his decision, another workflow process is initiated to either pay the expense report or inform the user that the expense report has been denied. Finally, if the manager does not approve or reject the expense report in a certain period of time, the workflow application reroutes the expense report to the manager's manager for approval. Figure 1-5 shows an example of the Expense Report application in action.

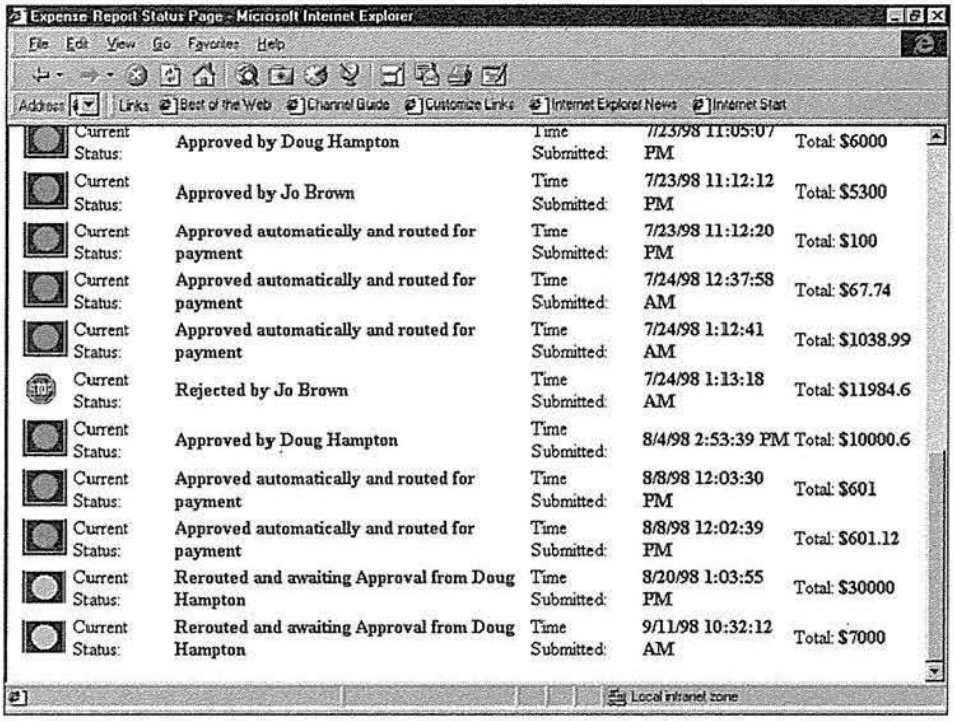


Figure 1-5
The web-based Exchange Server Expense Report application from Chapters 12 and 13.

Another example of a workflow application is a document routing application, in which a document to be reviewed is routed to users in parallel, user feedback is collected within a certain period of time and consolidated into a single

message, and the consolidated message is sent to the originator of the workflow application. Chapter 13 will show you how to create an application like this using Microsoft Exchange Server Routing Objects.

Real-Time Applications

Real-time collaborative applications are the newest category of Exchange Server applications. Real-time applications have the potential to enable instantaneous collaboration (as compared to the "delayed" collaboration of messaging-based applications). The challenge, of course, is to connect geographically dispersed users in real time. When you combine real-time and messaging technologies, you can build applications that leverage the strengths of both.

One example of a real-time application that you can build with Exchange Server is a class registration system that schedules virtual classes by sending Microsoft NetMeeting requests. NetMeeting allows individuals to collaborate over the Internet using video, audio, whiteboards, and application-sharing technology, as shown in Figure 1-6. In Chapter 7, you'll examine an Account Tracking sample that demonstrates how to integrate NetMeeting into your own application.

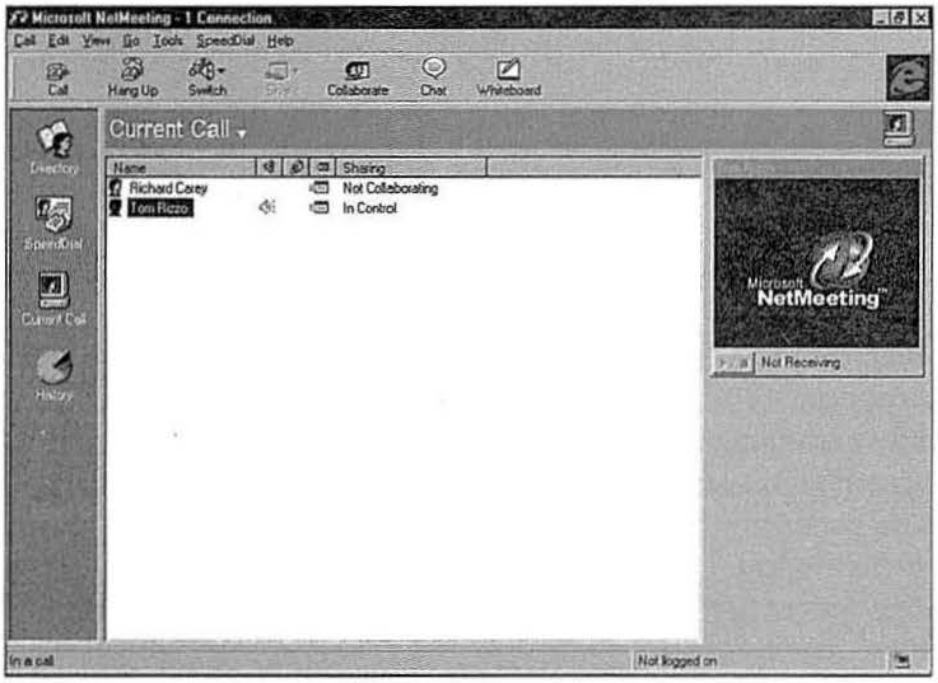


Figure 1-6
NetMeeting allows you to collaborate with other people in real time.

Another real-time application that will interest you as an Exchange Server developer is a new technology called Instant Messaging. Instant Messaging allows users to monitor when other users are online so that they can collaborate with one another. It allows two different organizations to create virtual "buddies," or business partners. Instant Messaging is like a virtual water cooler!

Chat Enables Real-Time Collaboration

Chat—a popular service on the Internet today—is one example of a real-time application. Chat enables real-time conversation by allowing a participant to type in messages that appear instantly on another participant's computer. When added to collaborative applications, chat can greatly enhance functionality for users. For example, you can extend a help desk application with chat services so that help desk technicians can hold "office" hours during which they conduct real-time question-and-answer sessions. Those chat transcripts can be posted to a discussion group so that other users can troubleshoot questions based on the transcript.

Knowledge Management Applications

"Knowledge management" is the latest buzzword in the computer industry. It refers to the use of collaborative technology to implement structured processes for finding and gathering information—in other words, it is a strategy for moving information from the individual to the larger group or corporation. At a time when corporations want to leverage the information that their intellectual assets—people—possess, knowledge management applications are critical. They make available all kinds of information, from individual experiences to best practices to detailed technical data. The effective sharing of knowledge brings to a company three primary advantages: more effective use of existing intellectual assets; competitive advantage through the pooling of resources and greater accessibility of important information; and new opportunities and more focused innovation. Although knowledge management is a new term and a new strategy for mining and sharing information, it uses technology that has been available since Exchange Server first shipped. Applications based on this strategy are called knowledge management applications.

Implementing a Strategy to Manage Knowledge

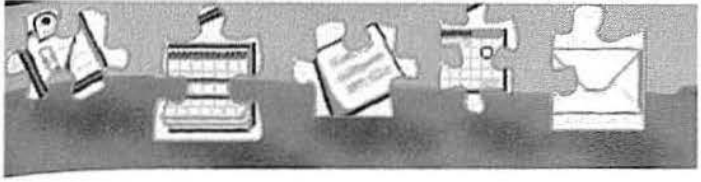
You can use collaborative technology such as Exchange Server to employ a knowledge management strategy, but collaborative technology is not synonymous with knowledge management. That is, corporations must establish processes

that will not only collaborate but also gather and make accessible information that is current, relevant, and tested.

You might be wondering what types of applications you can build with Exchange Server to implement the concept of knowledge management. One type is a search application in which you can search discussion groups and contacts in Exchange Server, as well as search in SQL databases and web sites. This search capability is a very powerful tool. One important benefit of universal search engines is that users do not have to change the way they collaborate because the search engine crawls the necessary data sources to retrieve the relevant information.

Another type of application that facilitates knowledge management is a knowledge base. By developing knowledge bases with Exchange Server, you can enhance conventional collaborative methods. Typically, knowledge bases are used by corporate users who post free-form, unmoderated messages to a common folder. Users who want specific information, such as text in a message, query the knowledge base in a general way and then cull all the returned information that meets their criteria. Because of the general nature of the queries and unstructured way information is posted, many of the results are irrelevant or invalid.

Imagine how a more structured method of entering and searching for information in a knowledge base could facilitate collaboration and knowledge management. Suppose users who were posting information to a knowledge base had to fill out a form that asked them to categorize their information, indicate how long it would be valid, and rate its usefulness if it originated from external sources. Users would be able to query on categories and ratings and receive only current and relevant information. By supplying just a little extra information, users make the data stored in the knowledge base infinitely more useful. And if you added smart agent technology to the application, you could program the knowledge base to e-mail links to relevant information that meets users' predefined criteria.



CHAPTER THREE

Exchange Server as a Platform for Collaboration

A builder is only as good as his tools: This adage still holds true for developers building successful software applications. As a developer, you require solid tools and technologies, and Microsoft Exchange Server is one of those tools. It provides a number of core capabilities—such as robust messaging functionality, an industrial-strength object database, Internet protocols, and an open directory structure—that make it an ideal platform for your collaborative solutions.

Robust Messaging Infrastructure

Exchange Server provides an infrastructure with certain core services that enable you to focus on building value-added services rather than on re-creating existing services. This infrastructure complements current network topologies and protocols and, as you will see, guarantees that every message gets through to its destination. The following sections discuss some of the advantages of the Exchange Server messaging infrastructure.

Least-Cost Routing, Load Balancing, and Failover

Exchange Server provides technologies in its messaging engine that allow organizations to define different routes of communications between Exchange Servers. Costs can be assigned to these different routes, and the least costly route is always attempted first by the Exchange Server. If this route is unavailable, the Exchange Server will failover to the next least costly route. If you assign the same cost to two different routes, the Exchange Server will distribute the communications traffic evenly over both routes, thereby load balancing the connections.

Let's look at an example. Imagine there are three routes between an Exchange Server in New York and an Exchange Server in California, and the routes consist of one route over the Wide Area Network, another over a dial-up 28.8 modem, and the third over a satellite link. The administrator of the Exchange Server system can assign costs to each of these routes: the WAN route is assigned a cost of 20, the modem route is assigned a cost of 50, and the satellite route is assigned a cost of 70. Based on the cost of the routes, for communications, the Exchange Server would always attempt the WAN route first. If this route was down, the Exchange Server would failover to the next least costly route (the modem), and if that route was unavailable, it would attempt to connect over the satellite.

Now this is a simple example, but Exchange Server supports the building of very complex routing tables with associated costs that it automatically calculates. For example, consider a message that has to be routed through seven different Exchange Servers until it reaches its final destination. Each Exchange Server has three unique routes to the next server. Exchange Server would automatically find the least costly route of all of the supplied routes.

Delivery and Read Receipts

Exchange Server supports both delivery and read receipts when delivering information through the Exchange Server system. Delivery receipts are returned to an individual user or an application when an item has been delivered to its final destination. This destination can be another Exchange Server or messaging server over the Internet. Delivery receipts also report the time and date that an item was received by a particular system. You can take advantage of delivery receipts in your application by using them to trigger events when they are returned. For example, a workflow application can consolidate delivery receipts to track the status of message delivery to workflow participants. Figure 3-1 shows an example of a delivery receipt.

Read receipts are similar to delivery receipts, except that read receipts are sent to a user or application when the recipient actually opens the item, and delivery receipts are sent as soon as the item is delivered to the destination server. You might want to use read receipts in your application for time-sensitive items sent through the Exchange Server system. The application could track when the item is read, and if no action is taken after a certain amount of time, it could reroute the item to a different user or application. Figure 3-2 shows an example of a read receipt.

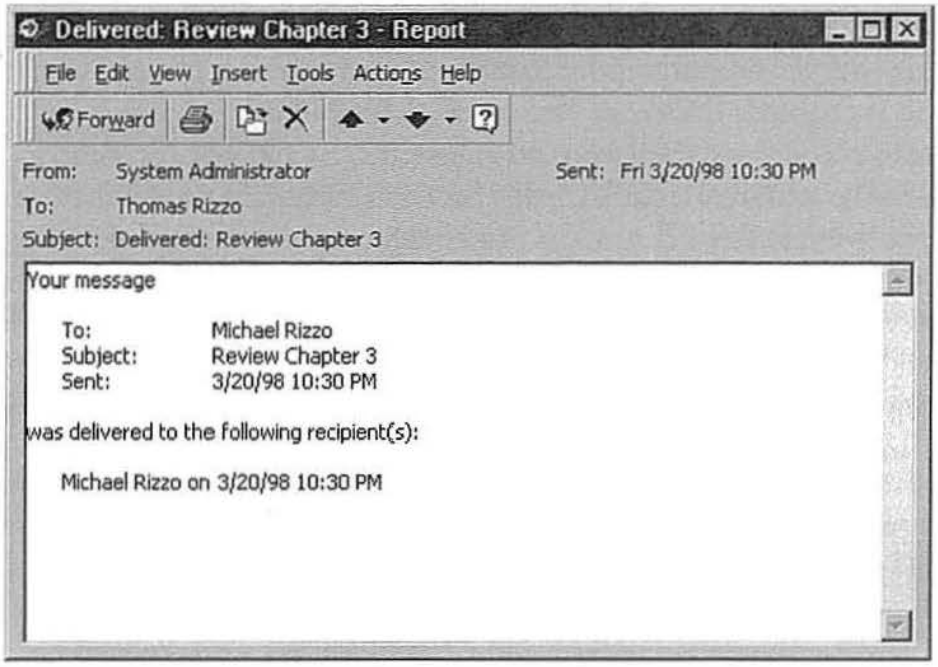


Figure 3-1
A delivery receipt sent back to a user looks like this. Applications can also send back delivery receipts.

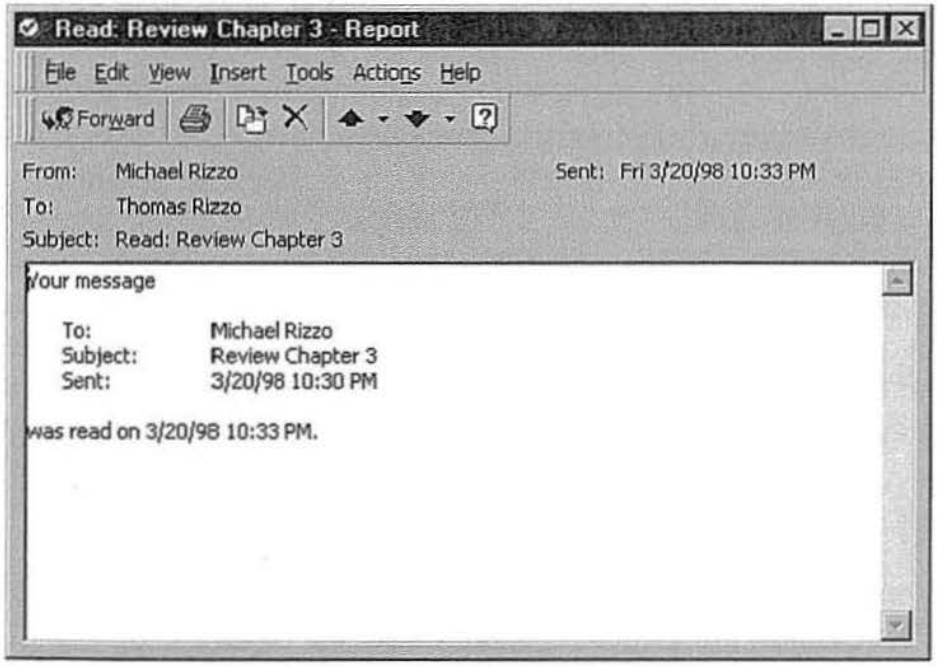


Figure 3-2
Applications can use read receipts like this to track when users open items sent by the application.

Message Tracking

Exchange Server supports more than delivery and read receipts. When message tracking is enabled, Exchange Server keeps logs of the items that have entered the Exchange Server system from other systems. Exchange Server also logs where items were routed to, which Exchange Server components routed them, and when the items were delivered to their final destinations. Message tracking enables you to find an item's route based on specific criteria such as the sender of the item, the intended recipient, or even the component of Exchange Server that handled the message. This powerful tool allows you to trace any item in your application and determine whether or not it reached its destination. Figure 3-3 shows an example of tracing an item in the Exchange Server system.

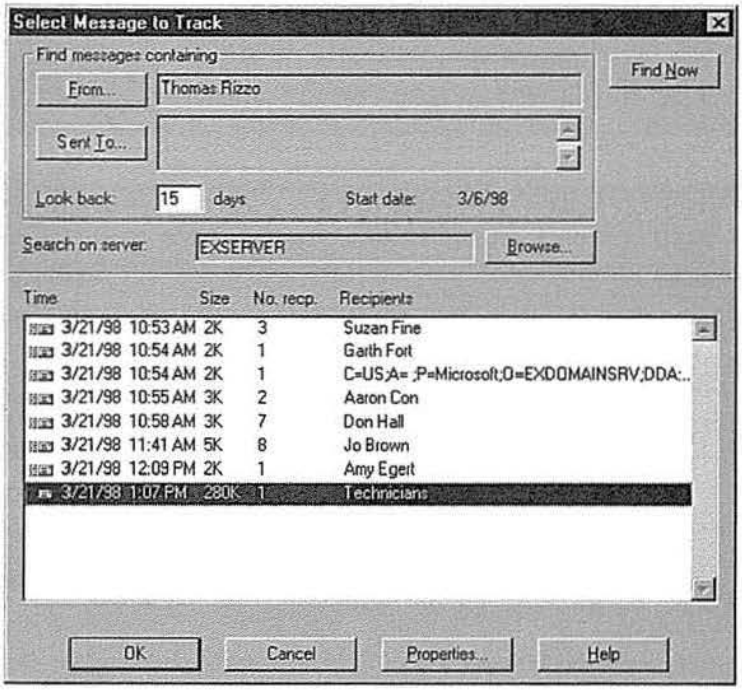


Figure 3-3
Tracking items from Thomas Rizzo across the Exchange Server system.

Industrial-Strength Object Database

At its core, Exchange Server is an object database. This object database is highly scalable, replicated, built for 24-hour availability 7 days a week, and can hold many different types of objects, including messages, Microsoft Office documents, video files, voice mail, faxes, hyperlinks, text documents, custom forms, and applications such as executable files. You can store all of your application's data in the database while replicating the information to other locations, so the

application and its relevant information are available anytime, anywhere. Core features of the Exchange Server database are discussed in the following sections.

Huge Storage Capacity

Many collaborative applications require large amounts of data to be available anytime. Exchange Server makes an excellent repository for this data because it can handle large amounts of information and ensure the reliability and availability of that information. Exchange Server supports very large databases—up to 16 terabytes (16,000 gigabytes) of information. That's pretty big considering that if you compiled every Wall Street transaction in history, you'd have only a little more than 1 terabyte. Really, the only factor limiting the size of your database is the hardware you run Exchange Server on. The database can run continuously because it has online defragmentation and allows backup programs to work with the database, even when users are logged on.

Exchange Server can store many different types of objects and their associated data in the same database. These objects can even be in the same table, or folder (as tables are called in Exchange Server). Users simply drag and drop different types of objects into these folders, and Exchange Server adds them to the database. This flexibility gives you a distinct advantage when developing applications. Figure 3-4 shows a folder in Outlook with many different types of objects.

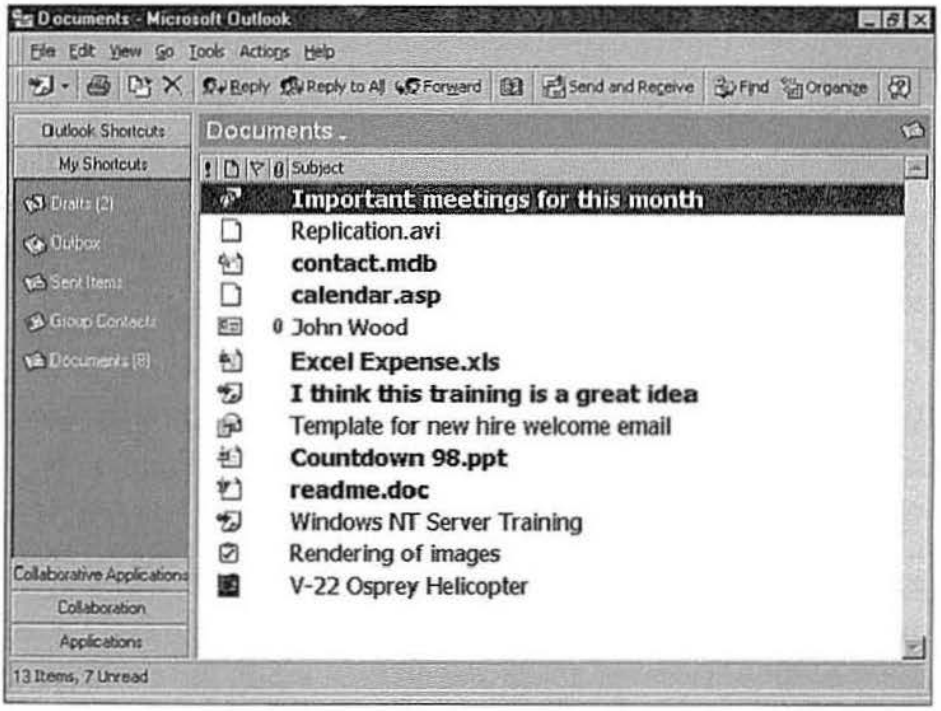


Figure 3-4
The Exchange Server database is an object database and can hold many types of objects in a single folder.

Multiple Views

The Exchange Server database not only supports multiple objects in a single folder but multiple views of those objects. You can customize views of the objects stored in the folder by sorting, grouping, and filtering the objects using any combination of their properties. For example, you can customize the view of an Exchange Server folder containing Office documents by specifying Office properties, as shown in Figure 3-5. Even custom properties can be columns in a view, and you can use them to sort and group items in the view.

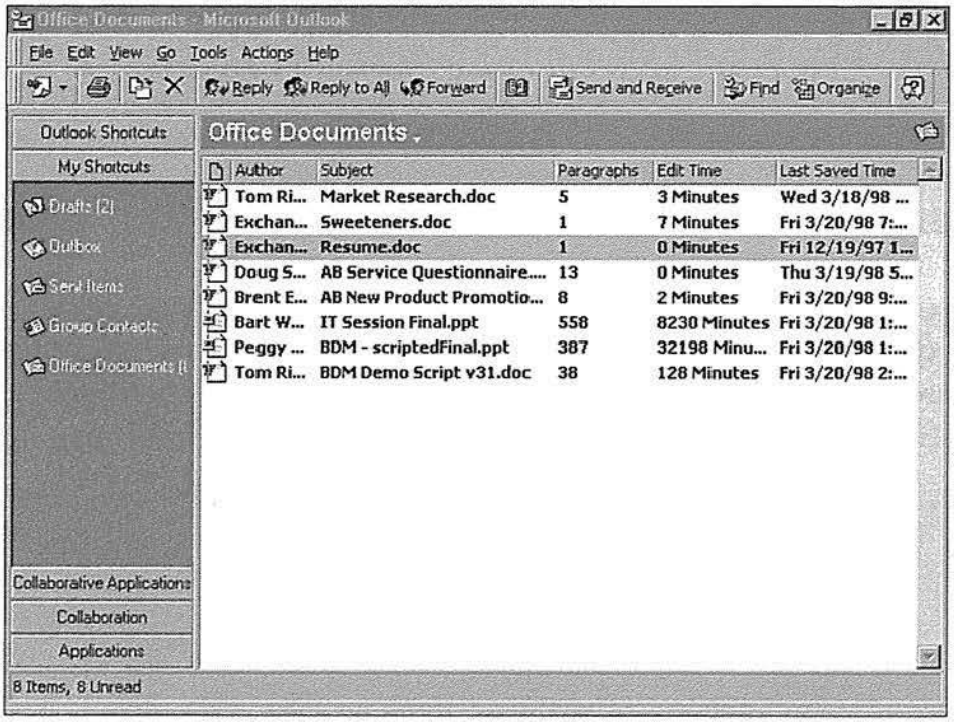


Figure 3-5
Views support using properties from Office documents. Your applications can use these properties for sorting, grouping, and filtering.

Exchange Server also supports "per-user" views that allow individual users to create custom views. Exchange Server actually maintains for each user the initial view of the folder, the status of read and unread items, and whether a particular grouping is expanded in the view. Figure 3-6 shows a single view of an Exchange Server folder but for different users. Notice how different the views look. Views can also be replicated offline by using Exchange Server's built-in replication features.

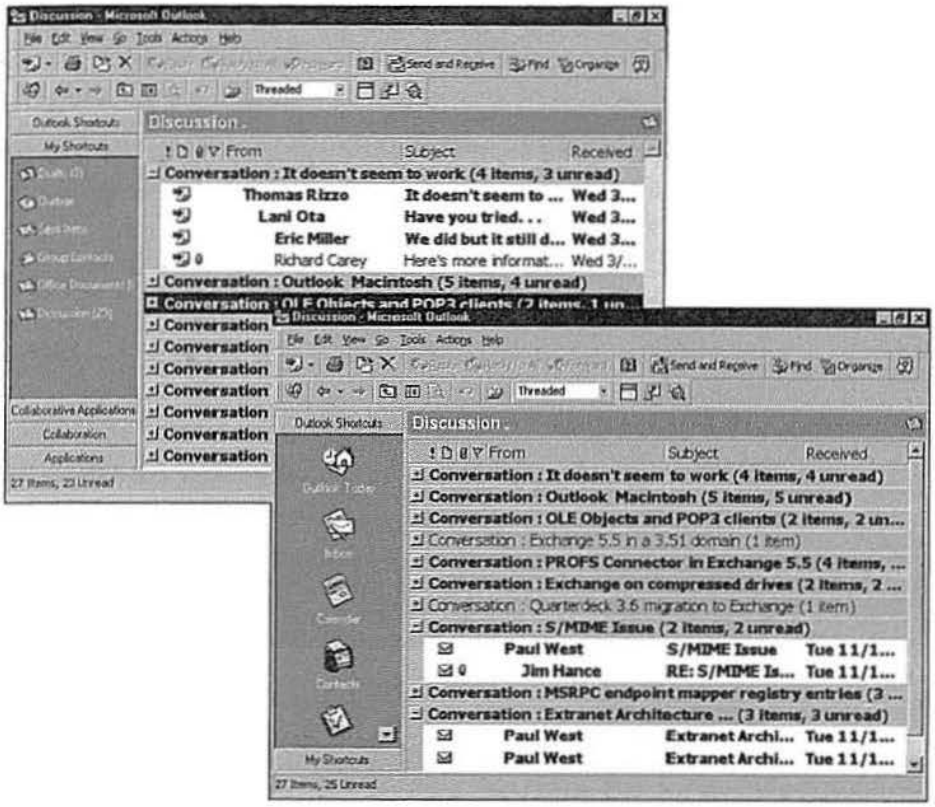


Figure 3-6
The initial view of the same discussion folder for two different users. Notice that certain groups are expanded and certain items are marked as read.

Built-In Replication

The Exchange Server database is a replicated database, enabling replication from Exchange Server to Exchange Server and from Exchange Server to Outlook on the client machine. Exchange Server even supports filtered replication between the server and the client.

Replication in Exchange is not the same as simple duplication. Exchange Server replication is more similar to the concept of synchronization in that only the changes are sent to replicas in the system. Sending only the changes, as opposed to copying the entire folder for each replication cycle, saves time and network bandwidth.

Setting up server-to-server replication with Exchange Server is easy. All the administrator has to do is select the folder to be replicated and then select the server to replicate the folder to. The settings that enable server-to-server replication

in the Microsoft Exchange Administrator program are shown in Figure 3-7. Once these settings are in place, the actual replication messages are sent over the Exchange Server messaging infrastructure. This allows the replication messages to leverage Exchange Server's load balancing, least-cost routing, and failover capabilities. Exchange Server also supports setting the time and size limits of the replication messages.

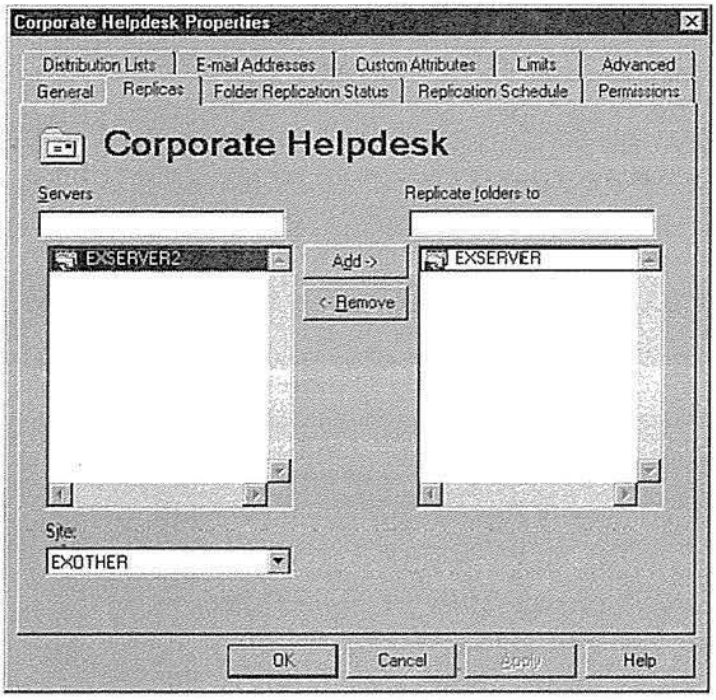


Figure 3-7
Setting up server-to-server replication for your applications in Exchange Server is as easy as pointing and clicking in the Exchange Administrator program.

The Exchange Server replication feature has built-in conflict management capabilities that enable users to edit the same information at the same time in the same folder or even in replicas of a folder in different locations. To determine which item to accept as the newest, Exchange Server implements "last saved wins," the process of querying the time an item was saved and retaining the most recently saved item. You can also set an option that alerts users via e-mail when items are in conflict. Both versions of the item are sent to these users, and they can decide which item is the most up-to-date. Exchange Server will keep the item they select.

For server-to-client replication, Exchange Server and Outlook support bidirectional synchronization of changes to information in Exchange Server folders. This synchronization occurs in Outlook as a background process, so users can continue working in Outlook. The synchronization can be scheduled so that it happens at certain intervals. For example, a user can configure Outlook replication so that every 30 minutes the Outlook client synchronizes its local database with new information from the Exchange Server.

Outlook also supports filtered replication, in which only a subset of information is synchronized to the local database. Filtered replication is most useful to users when large amounts of data are available in the Exchange Server database but users want to take only a subset of that data offline. For example, imagine an Exchange Server folder with 50,000 sales contacts. A typical user wouldn't be able to accommodate the entire folder on her local hard disk, so she could set the replication criterion to only those contacts for whom she is the sales representative. Instead of 50,000 contacts, the filtered subset is 1,000 contacts. Figure 3-8 shows the interface in Outlook where users can set the criteria for filtered replication.

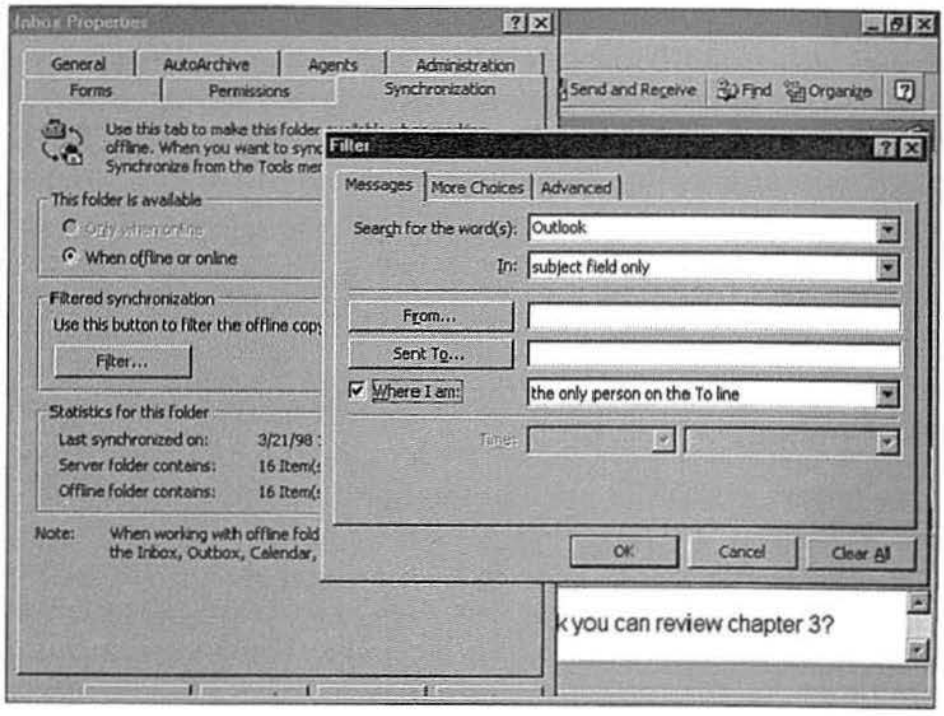


Figure 3-8
Setting up filtered replication in Outlook is easy for users of your application.

Schema Flexibility

Typically, when you begin work on an application that deals with a database, you are forced to plan your schema for the database before you start writing your application. If the application requirements change and a new field has to be added to the database, the schema might not be flexible enough to support the addition, and you might have to drop the present database and create a new one.

With the Exchange Server database, however, you can add new fields at any point in development, which allows you to accommodate the changing requirements of an application. New fields are automatically available to users, so users can create custom views using them.

Transaction Logging

A transaction is a unit of work, such as adding an item to an Exchange Server database. Before any item is committed to the Exchange Server database, the transaction is written to a transaction log file and then to the database. This process is called write-ahead transaction logging, and it guarantees that no item will be lost.

Transaction logs allow the Exchange Server to recover the database after some form of failure, such as a power loss. In this type of scenario, after power is restored and the server is rebooted, the Exchange Server automatically recovers the database. Using checkpoints in the transaction logs, the Exchange Server replays any transactions that were not committed to the database before the power failure.

The transaction log is an inherent feature of the Exchange Server database, so any application you develop on Exchange Server can take advantage of it. Any items your application sends or stores in the Exchange Server system will be delivered or committed, even in the event of certain failures in the computer system or network.

Exchange Server Directory

To collaborate effectively, users must be able to find other users and information easily. Exchange Server provides a hierarchical directory for this purpose. This directory holds the critical information of an organization, and it can meet the needs of both large and small organizations because it's scalable and easy to manage. Some of the most important features of the Exchange Server directory are described in the following sections.

Reliable Database Engine

The Exchange Server directory is implemented using the same database technology as the Exchange Server messaging infrastructure, so the database engine's reliability is high. This reliability guarantees that the directory will always be available to your applications.

Multimaster and Replication Capabilities

The Exchange Server directory is a multimaster, replicated directory. A multimaster directory allows an administrator to make changes to it on any Exchange Server in the organization, changes that Exchange Server then propagates to other servers through replication. A replicated directory is implemented over the messaging infrastructure of Exchange Server, so directory-replicated messages can take advantage of the least-cost routing, failover, and load-balancing features of Exchange Server.

Directory replication in the Exchange Server system is not limited only to server-to-server replication. Exchange Server also supports server-to-client directory replication. By using a feature called the Offline Address Book, Outlook can replicate the Exchange Server directory, or a subset of it, to a user's local machine. This allows a user of your application to address items to other users and to look up detailed directory information, even when the user is working offline.

Customizable Attributes and "White Pages"

Exchange Server exposes a number of attributes in the directory that you can customize and. replicate. For example, you could customize the Exchange Server directory with a field named "cost center," and set up a supplies requisition program that dynamically queried the directory for users ordering supplies. Based on what information users entered in the cost center field, the application would update an accounting system so that the cost of supplies are automatically deducted from the cost center. Figure 3-9 on the following page shows where you can customize the Exchange Server directory.

The directory has some additional built-in features that you can take advantage of, such as its ability to store all types of information about an organization, including users' office locations, phone numbers, department names, titles—even a user's manager and direct reports. Exchange Server is an ideal "white pages."

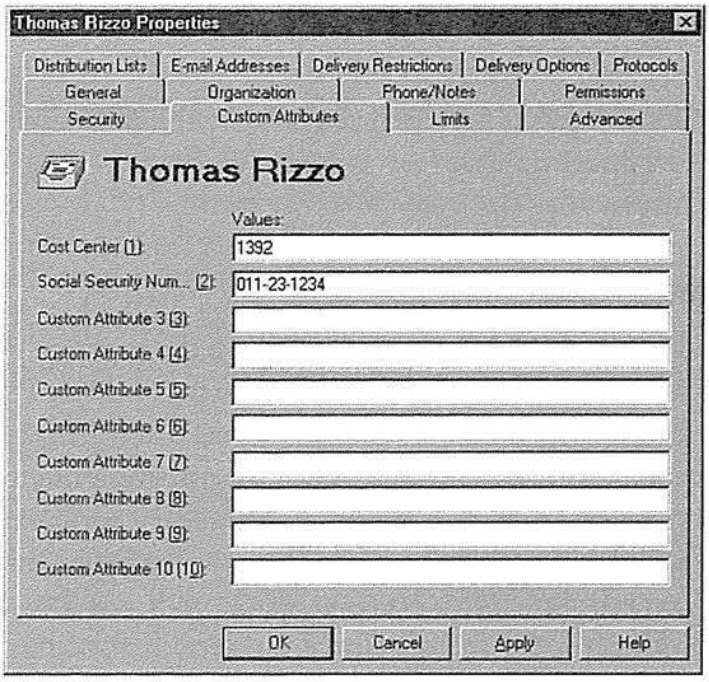


Figure 3-9
Customizing attributes in the directory. Your applications can take advantage of these customized attributes.

For workflow applications, a central, hierarchical directory of this kind is crucial. Workflow applications must be able to route items based on an organization's staff structure, which is dynamic. If names of individuals were hard-coded in an application, staffing changes would require the application to be rewritten. With the Exchange Server directory, you can query and dynamically generate employee information.

Extensibility and Security

The Exchange Server directory is not limited to storing information for only one organization. Through the use of custom recipients, the Exchange Server directory can also hold address and organizational information for users from other organizations. The Exchange Server directory exposes the same functionality to these types of directory objects as it does to the standard directory objects. Figure 3-10 shows an example of a custom recipient in the Exchange Server directory.

Any directory object in the Exchange Server system can be secured by using access permissions, which determine who can see particular objects in the directory. For example, an administrator can set the access permissions on the business partner directory entries so that certain workers are denied access. These permissions can be set either per user or per group.

Figure 3-10

A custom recipient in the Exchange Server directory. Recipients can hold organizational information for users outside your current organization.

Internet and Industry Standards Support

The Exchange Server directory supports Internet standards such as LDAP version 3. LDAP, which stands for Lightweight Directory Access Protocol, is an adapted subset of the X.500 standard that specifies a common protocol for directory access over TCP/IP. The key benefit of LDAP support in Exchange Server is that any LDAP-compliant client or application can query the Exchange Server directory. LDAP version 3 as implemented in Exchange Server enables you to chain directories together through a feature called referrals. Referrals tell the Exchange Server directory where to look for information that a user is querying for when the directory does not currently possess it. For an application, referrals are crucial since one directory might not contain all the needed information about users and services. Rather, many different directories, which could be hosted on servers in different locations and even in different organizations, might contain pieces of this information.

The Exchange Server directory supports ADSI (Active Directory Services Interface). ADSI is an application programming interface that enables you to modify many different directories using standard protocols. The different directories that ADSI supports are the Active Directory in Microsoft Windows 2000, the Microsoft Windows NT version 4 domain-based directory, any

LDAP-compliant directory such as Exchange Server directory, Novell NetWare's NDS Directory, and Novell NetWare Bindery. The ADSI interface abstracts the low-level functions of these directories and exposes a number of objects with which you can write applications. Because ADSI provides COM interfaces that give every directory element a common set of properties, the application can use the same programming interface to connect to directory elements in several directory services. Figure 3-11 shows a diagram of ADSI and the directory services it can access. ADSI is an important technology to learn since it ties all of these disparate directories together with a common programming model, and it is Microsoft's strategic directory programming interface. Chapter 14 demonstrates how to program to an Exchange Server directory using ADSI.

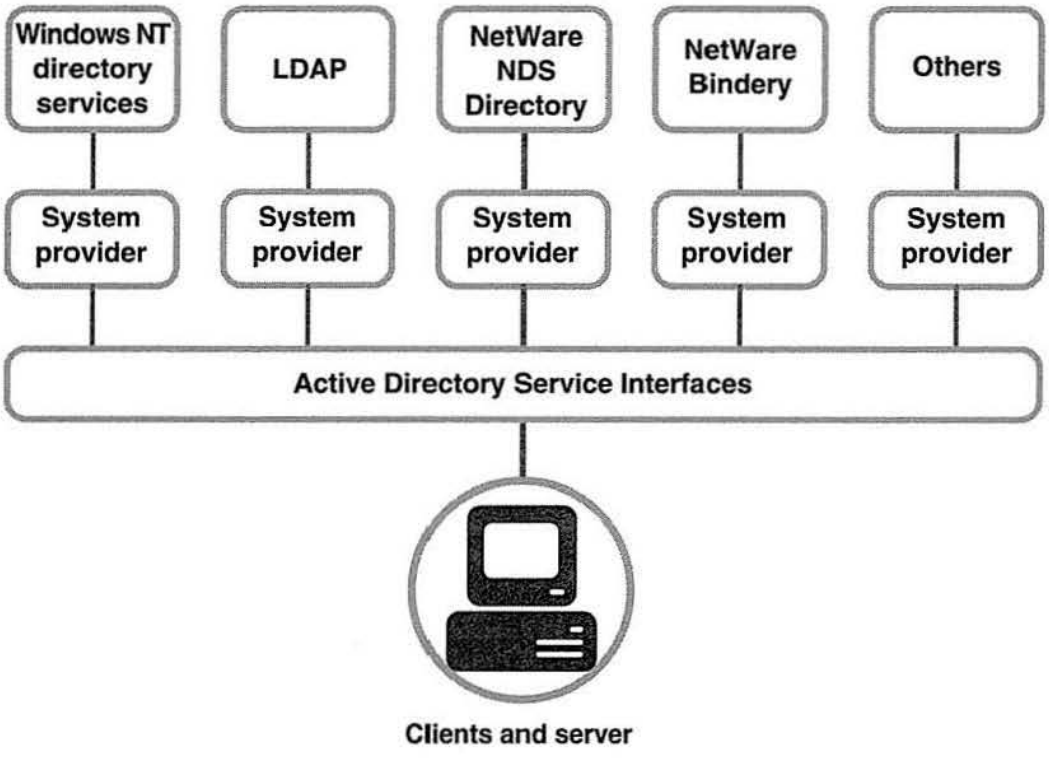


Figure 3-11
ADSI allows you to talk to many different directories, including Exchange Server, using the same interfaces. This access is provided through the different system providers (SPs) in ADSI.

Public Folders

The core of Exchange Server's collaborative technologies is a feature called public folders. Public folders are repositories for all kinds of information users will share. They can be accessed by many types of clients, using various protocols to

communicate. Public folders can contain custom forms for contributing or reviewing information in the folder, and users can create custom views for organizing and filtering the information in the folder. The main features of public folders for developers are described in the following sections.

Public folders, just like the Exchange Server directory, are built on the Exchange Server object database, so they take advantage of its architecture and enjoy the same benefits:

- Flexible database schema
- Server-to-server, server-to-client, and filtered replication

Folder and Application Accessibility

Public folders in Outlook are arranged in a hierarchical tree view, as shown in Figure 3-12. As you can see, this arrangement makes it easy for users to scroll and find information. This hierarchy is actually a virtual view of public folder replicas in the Exchange Server system; users don't have to know on which server the public folders actually exist.

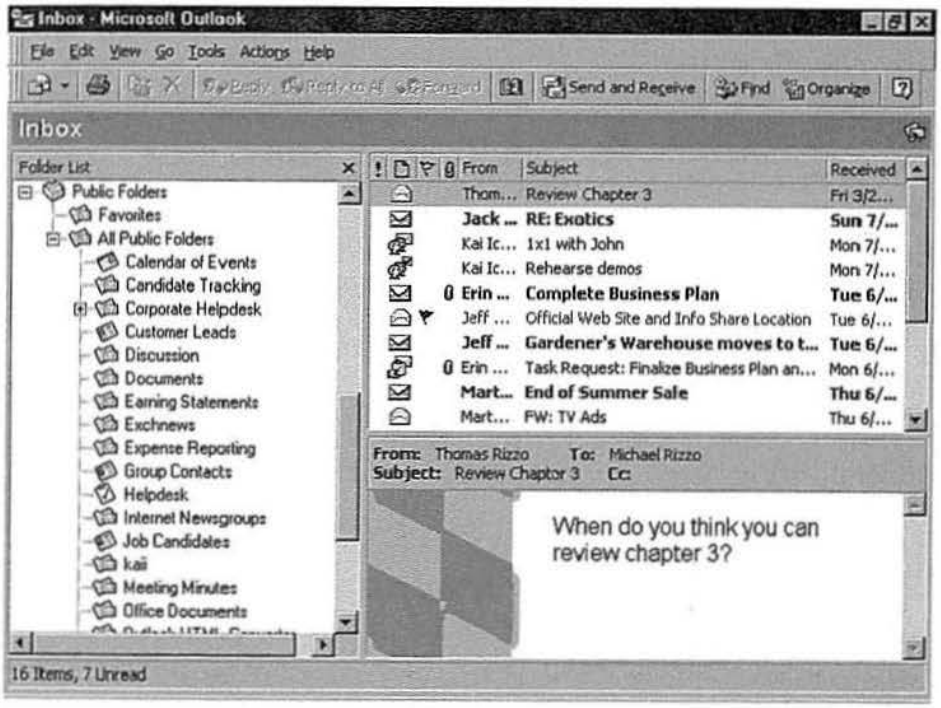


Figure 3-12
The hierarchical tree view of the Public Folder allows users to quickly find information.

Exchange Server can assign costs to different sites so that users connecting to a remote site with a public folder follow the least costly route to that site. This assignment of costs to remote connections for public folders is called public folder affinity. Figure 3-13 shows the administration interface for public folder affinity.

Figure 3-13
By setting the Public Folder Affinity option, users of your application will access one replica of the Public Folder database over another depending on their location on the network.

Public folders are not limited to holding only the data for an application; they also can hold any custom forms associated with the application. The availability of these forms makes your application easier to use by allowing users to go directly to a public folder to select the associated form rather than search for the form in a global forms list.

Security and Content Control

Inherent in public folders is security control. Public folder permissions can be set on three different scales:

- *Global.* Default permissions for everyone in the organization; default permissions for anonymous users

- *Group.* Permissions for a specific list of users
- *Per user.* Individual permissions for a particular folder

All of these permission levels can be combined for a particular folder or set of users. Assigning permissions is easy, as shown in Figure 3-14. Notice that the permissions tab supports predefined roles for users, which you can use to quickly set permissions for a folder.

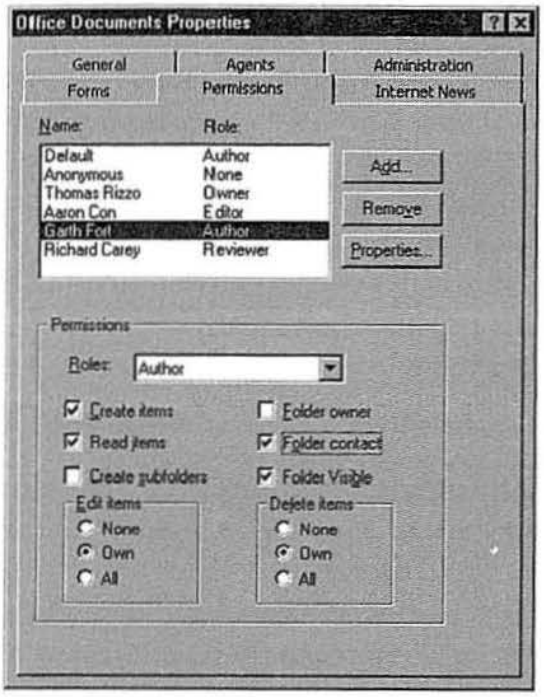


Figure 3-14
Setting permissions for a public folder is easy.

In addition to having roles-based permissions, public folders have built-in moderation capabilities. A moderated public folder allows you to control what content is posted to a folder and who has permission to approve this content. Before any item is posted to a folder, the item is mailed to the selected moderators, who approve the content. You can quickly set up a moderated public folder in the folder properties.

Public folders support the e-mailing of items into a public folder, which makes information available to many users, cuts down on e-mail traffic, and saves disk space. Mailing-list server applications and distribution list applications can really take advantage of public folders. By default, the e-mail address of the public folder is hidden from the address book, but the public folder can be exposed in the address book so that users can browse for its e-mail address.

Internet Standards Support

As you've seen, significant economies can be achieved when information is stored in a central location rather than in individual mailboxes. By using Exchange Server public folders as the central location, organizations can expose information to any standard Internet client that supports the Network News Transfer Protocol (NNTP), the Internet Mail Access Protocol version 4 (IMAP4), or the Hypertext Transfer Protocol (HTTP). These Internet clients can post and read information securely from a public folder. More importantly, these protocols allow users who do not have Outlook on their machines to take advantage of the functionality of an Exchange Server public folder. For example, an organization can set up a customer service public folder that enables internal users to employ Outlook to view folder information and allows external users to choose from several clients, including Outlook, an NNTP newsreader such as Outlook Express, a standard web browser, or even an IMAP4 client such as Netscape Communicator. Exchange Server exposes its collaborative functionality to all of these clients. Let's take a look at some specific information on the Internet protocols supported in Exchange Server.

NNTP

NNTP is an internet standard that defines server-to-server replication of data in the form of articles. These articles exist in a hierarchy of newsgroups, which are similar to discussion folders in Exchange Server. Users can replicate the articles offline, plus the articles are presented in a threaded view so that users can view their history. Exchange Server supports both NNTP server-to-server replication and the ability of any standard NNTP client to read information in Exchange Server public folders. This allows any public folder in Exchange Server to be replicated to another NNTP server or read by an NNTP client. Organizations can use this feature to expose public folders and their information to their customers. Figure 3-15 shows an example of a newsreader using NNTP to access an Exchange Server public folder.

IMAP4

IMAP4 is an Internet standard that defines a way for clients to access messaging information on a server. Exchange Server is an IMAP4-compliant messaging server, so any standard IMAP4 client can access the messaging services of Exchange Server. Some of these services include sending and receiving e-mail, synchronizing e-mail to offline storage, and accessing public folders. Accessing public folders with IMAP4 extends the power of public folders to any standard IMAP4 client.

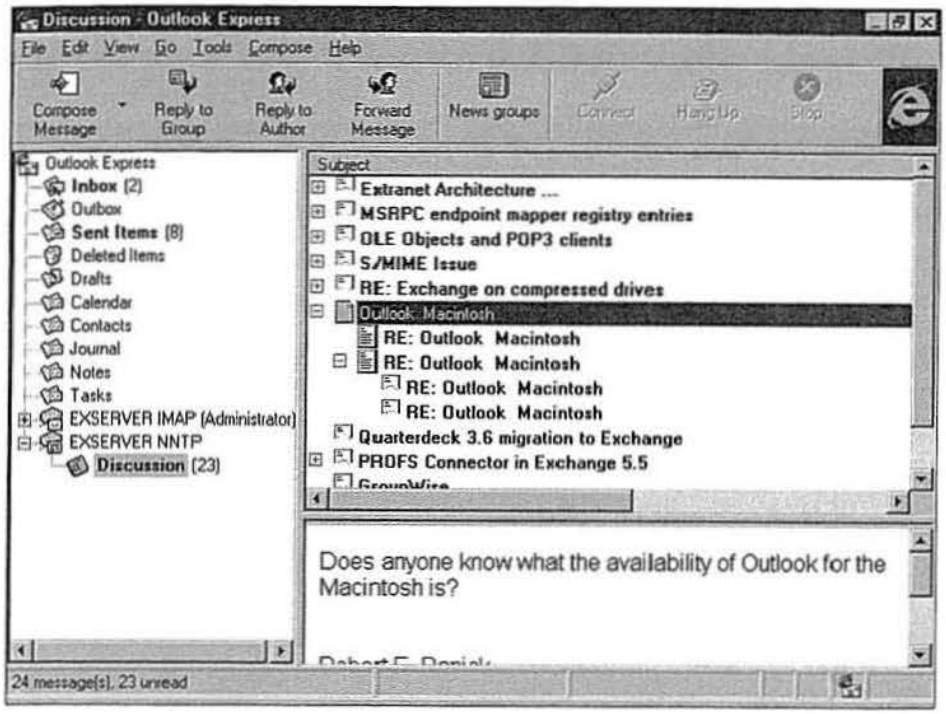


Figure 3-15
A public folder being viewed by Outlook Express, an NNTP newsreader. Exchange Server folders have built-in support for NNTP.

HTTP

HTTP is the primary protocol used to distribute information on the World Wide Web, that is, to transmit graphics and documents from a web server to a web browser. HTTP is a client-to-server protocol, meaning that a client running on the user's machine sends to a server a request for data, and the server receives the request and sends the relevant information back to the client. HTTP servers can do more than just send back simple data—scripts that access other back-end services on the network can run on the HTTP server. These services can be databases, collaboration servers such as Exchange Server, or custom-built applications.

One example of an application that uses the HTTP protocol to access Exchange Server information is Microsoft Outlook Web Access. Outlook Web Access is an application, which we discuss in Chapter 8, that allows any standard web browser to access the information stored inside an Exchange Server. Outlook Web Access is built using Microsoft Collaboration Data Objects (CDO). CDO, which we look at in detail in Chapter 11, is a set of COM objects that exposes the services of Exchange Server to any COM-based development tool.

CDO is the object library for Outlook Web Access, and Internet Information Server (IIS)—especially Microsoft Active Server Pages (ASP), VBScript, and JScript—are the development tools. Figure 3-16 shows the architecture for Outlook Web Access.

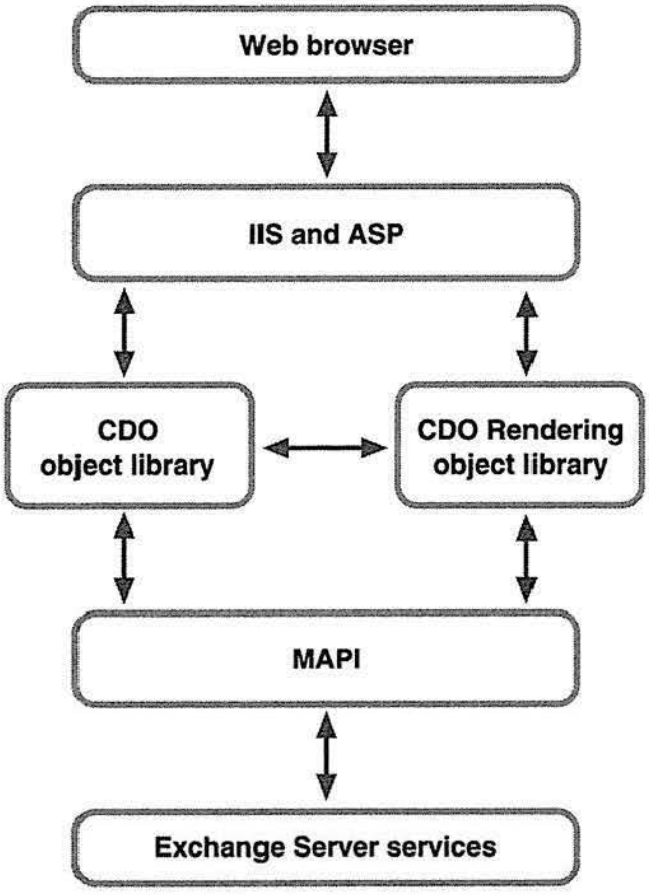


Figure 3-16
The Microsoft Outlook Web Access architecture.

Active Server Pages allows you to write scripts using any standard ActiveX scripting language. With these scripts, which execute on IIS and return HTML to the web browser, you can build dynamic web applications that take advantage of objects on the web server. The following code listing shows a simple Active Server Pages application:

```
<HTML>
<HEAD>
<TITLE>Hello World!</TITLE>
</HEAD>
<BODY>
<H1>
Hello, I was created on <%= Now() %>.<P>
```

```
<% Set BrowserControl = Server.CreateObject("MSWC.BrowserType") %>
You're using <I>
<%= BrowserControl.browser & " " & BrowserControl.Version & " " %>
</I>
as your web browser.
<% set BrowserControl = Nothing %>
</H1>
</BODY>
</HTML>
```

If you browsed this web page using Microsoft Internet Explorer version 4, you would see the image in Figure 3-17. Notice that none of the script is sent back to the client, only the text, and that the date in the text is generated dynamically from the system date on the web server. The browser is also detected by using a component on the web server. Chapter 8 discusses the ASP object model and programming environment in more detail.

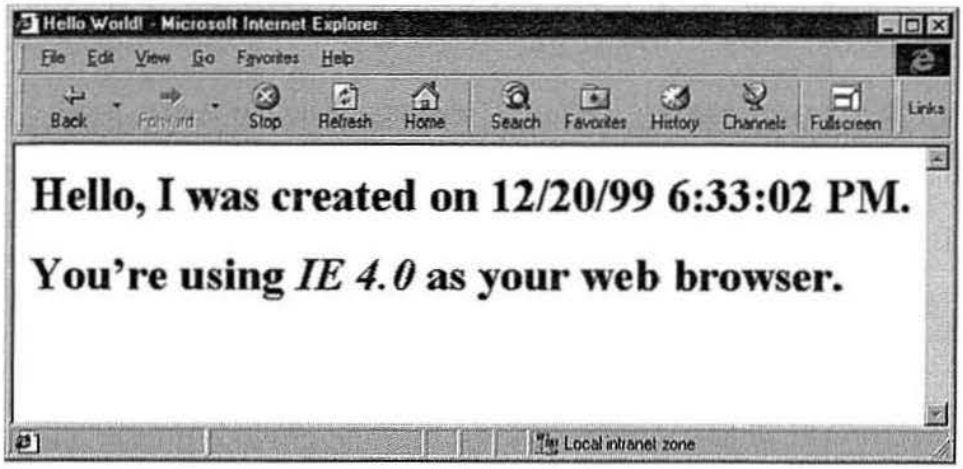


Figure 3-17
Results of browsing the ASP page using Internet Explorer.

By using Active Server Pages, Outlook Web Access can dynamically and securely create web pages based on the information and services in Exchange Server, such as a user's mailbox, calendar, and contacts; and messaging and calendaring services. You can access the same features in your application since the enabling technology for Outlook Web Access is the CDO library.

Integrated, Internet Standards-Based Security

With so many corporations connecting their systems to the Internet and exposing their networks to millions of Internet users, security has become a large concern. While most users on the Internet are not lurking and waiting to break into

corporate networks, some "bad apples" on the Internet are. Exchange Server prevents these users from accessing privileged information by implementing Internet standards-based security in an integrated way.

Windows NT Security

Exchange Server integrates with Windows NT security in two ways. First, users have to be authenticated using a Windows NT account before gaining access to any Exchange Server resource that requires authenticated access. Administrators can set up a Windows NT security infrastructure, and Exchange Server will use that infrastructure for its own security and access permissions. This enables users to log on only once to access both the network and Exchange Server services.

Second, Exchange Server uses the built-in auditing capabilities of Microsoft Windows NT. This integration allows an administrator to detect security breaches by tracking events, across Windows NT and Exchange Server, which occur within a system. All the events can be viewed in one window using the Windows NT event log.

Secure Messaging

Many corporations today use the Internet as a backbone for their corporate communications system. While this is cheaper than leasing lines between servers, it opens a world of security concerns. Exchange Server alleviates these concerns by implementing some key features that allow corporations to securely use the Internet as a communications network backbone. For securely sending messages between servers, Exchange Server supports Secure Socket Layers (SSL) in combination with the Simple Mail Transfer Protocol (SMTP). SMTP is the primary way that different mail systems talk over the Internet. SSL allows systems to encrypt data sent from one system to the other. By implementing SSL with SMTP, an organization can encrypt its data from one Exchange Server to another when sending the data over the Internet.

Secure Applications

SSL is not only supported with use of SMTP, but it is also used with other Internet protocols that Exchange Server supports. By using SSL, Outlook Web Access can encrypt any traffic between a user's web browser and web server. This secures any HTML documents that Outlook Web Access is sending to the user. You can take advantage of SSL when using custom forms in the web forms library of Outlook Web Access.

S/MIME Support

Exchange Server supports encryption and digital signatures by using Secure Multipurpose Internet Mail Extensions, or S/MIME for short. An Internet standard, S/MIME is a method of digitally signing and encrypting messages between users on the same vendor's system or users on different vendors' systems.

S/MIME is built on X.509 version 3 certificates. These certificates are generated by a certificate authority such as VeriSign or Certificate Server included with the Microsoft Windows NT version 4 Option Pack. Since Exchange Server supports X.509 version 3 certificates, it can accept the certificates from other certificate authorities. Similarly, clients can trust certificates from other authorities through the use of Certificate Trust Lists.

Exchange Server also supports the revocation of security certificates. Revoking certificates is useful when a user feels that her security has been compromised and someone else is signing messages on her behalf. Likewise, when a user leaves an organization, you might want to revoke the user's certificate to make sure that all messages sent by this user are marked invalid. When an administrator revokes the certificates for a user, any encrypted messages previously sent by that user will notify other users, upon opening of the messages, that the certificate is invalid. After revoking a certificate, the administrator can issue a new certificate to the user.

As a developer, you can take advantage of the advanced security features of Exchange Server. By building your applications based on the standard Outlook e-mail message, you automatically inherit the advanced security functionality in Outlook. This allows you to digitally sign and encrypt your custom forms before the user sends or posts forms.

Multitiered, Replicated, Secure Forms Library

Locating new applications in an organization can be a hard thing to do because they exist in so many places. For example, an application can exist on one of many possible file servers. Although the emergence of intranets has enhanced the ability to find applications, you still have to find the site with explicit links to the information you want. And if you do find the web server that has the application, you might have to connect to a server halfway around the globe, making connection speeds to that application very slow.

Exchange Server's multitiered, replicated, secure forms library makes it easier to locate applications. The Exchange Server forms library is divided into four main components: an Organizational Forms Library, a folder forms library, a Personal Forms Library, and a web forms library. Some of these libraries can

be synchronized offline, so users can work with the applications, even when the users are disconnected from the network. You can choose which of these is best for your application.

> NOTE: The web forms library has not yet been given an official name. However, web forms library is the name that I will be using throughout the book when referring to it.

Organizational Forms Library

The Organizational Forms Library contains, most often, forms that everyone in an organization needs access to, such as vacation requests, business cards, and travel expense reports. The Organizational Forms Library is contained on the Exchange Server and can be replicated to servers throughout your network, so access to these forms is fast. It lists all the available applications throughout an organization. Figure 3-18 shows an Organizational Forms Library in Microsoft Outlook 98.

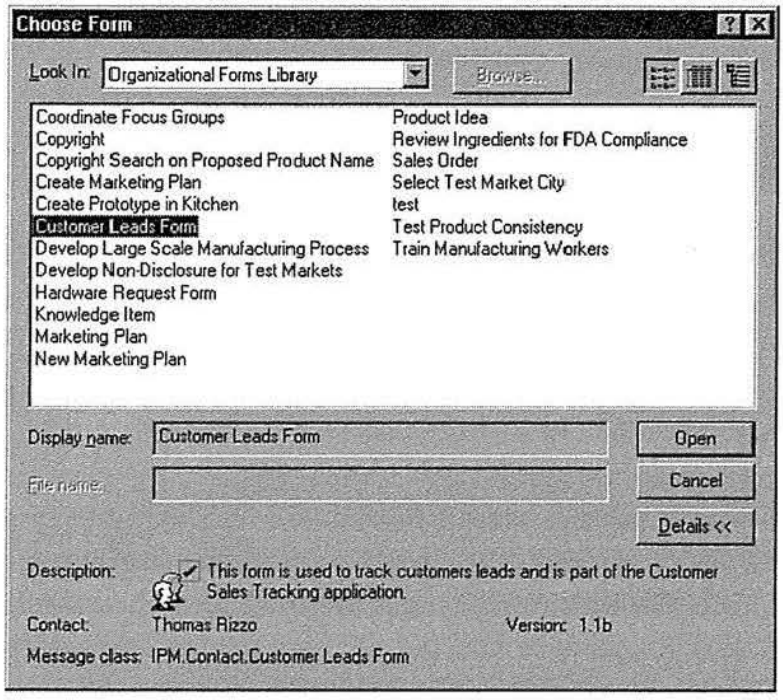


Figure 3-18
You can publish your application in the Organizational Forms Library, and it will automatically be available to your users.

The Organizational Forms Library is secure, so administrators can set which users have permissions to publish or edit information in the forms library. It is

also multilingual; the Exchange Server presents the server-based forms library that corresponds to the language of the client program accessing the forms library. For example, when a Japanese client requests a list of forms in the Organizational Forms Library, the Exchange Server displays all the corresponding Japanese forms. This multilingual capability allows you to customize and deploy your applications to the correct client without writing any code.

Folder Forms Library

The folder forms library is for folder-specific forms. The folder forms library is more secure than the Organizational Forms Library. You would post forms you do not want to share globally in the folder forms library. The forms stored in a *personal* folder forms library are shared only with the users to whom you give access. The forms stored in a *public* folder forms library can be shared with any user who has the correct permission on that public folder. Using the synchronization capabilities of Exchange Server, users can replicate public folders (including their data and forms) offline.

Personal Forms Library

The Personal Forms Library is the most restrictive in terms of sharing its forms with other users; this library "belongs" to a particular user and cannot be shared with any other user in the organization. All forms in the Personal Forms Library can be used both on and off the network. Users can test forms in the Personal Forms Library before publishing them to the Organizational Forms Library or folder forms library.

Web Forms Library

The web forms library is a hierarchy of folders stored in the Windows NT file system where your web server, IIS, runs Outlook Web Access. Exchange Server supports HTML forms as a development environment, so Outlook Web Access has an easy and automatic way for web developers to publish custom forms in the web forms library. To create an HTML-based application, you need only to create a subdirectory in the file system where Outlook Web Access is stored and copy your HTML files to it. The new form will appear in the Launch Custom Forms window of Outlook Web Access. Users can then start working with the application from the web forms library. Figure 3-19 on the following page shows forms in the web forms library.

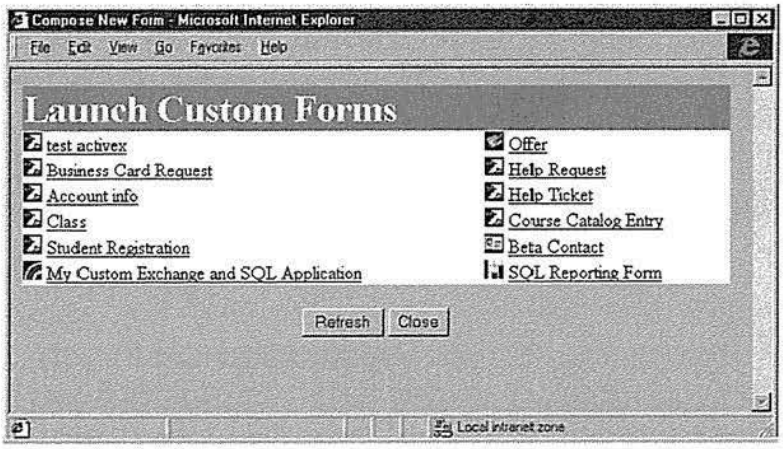


Figure 3-19
The web forms library holds HTML applications that you develop for your organization.

Built-In Information Management Tools

Managing information when building applications can be one of the most tedious tasks for a developer. But public folders, with their built-in and configurable services, handle these tasks automatically for you. They allow you to set the expiration time for information, which prevents public folders from becoming inundated with megabytes of outdated and useless information. Their conflict management features prevent two users from unintentionally saving two versions of the same document. If two users edit the same document stored in a public folder and then try to save their changes, Exchange Server sends a conflict message to both users and to any folder contacts defined on the public folder. The users then have the choice to keep one of the two items or both. Figure 3-20 shows the Conflict Message dialog box.

Rules

To manage the massive amount of information received by an organization, Exchange Server supports rules. Although many other collaborative systems also have this functionality in some form, in most cases a user must be logged on to the system before the rules can be processed. Also, other systems don't allow rules in folders other than a user's personal folders. With Exchange Server, rules are supported for both personal folders such as an Inbox and for public folders.

By setting rules in your public folder application, you can to some extent control the flow of information into and out of it. Public folder rules are configurable by the owner of the public folder and are server-based, which means no client has to be logged on for the rules to fire. Instead, the server fires the rules.

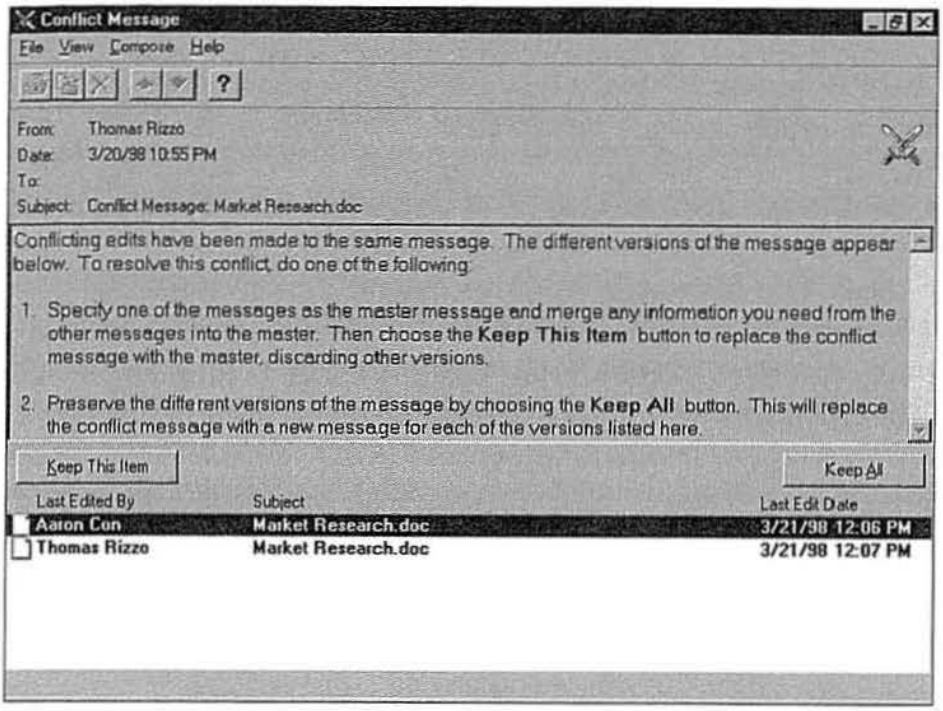


Figure 3-20
Exchange Server automatically detects when conflicts of information occur in your applications. The server will then send a notification to the folder owner and to the users who generated the conflict.

The types of rules you can create range from simple rules, such as "send a thank you e-mail to anyone who sends a message to the public folder," to very complex rules. Complex rules can entail checking multiple fields on an item and taking a specific action based on those fields.

Event Scripting Agent

Sometimes rules are not the best strategy for controlling the flow of information in your application—for example, they might be too constrictive. In these situations, a feature in Microsoft Exchange Server version 5.5, called the Microsoft Exchange Event Scripting Agent, will help you greatly. The Event Scripting Agent (discussed in more detail in Chapter 12) allows you to write custom event handlers for the most common Exchange Server folder events by using standard development tools that support COM. For example, you can write a script event handler that calls COM objects that you create. Plus, as its name implies, the Event Scripting Agent ships with a scripting engine that understands any ActiveX scripting language, so you can write your custom agents using a scripting language such as VBScript or JScript. You choose which development tool you use to write these agents.

Once you write your custom agent, you can place it in an Exchange Server folder, such as your server-based Inbox or a public folder. Your agent can handle four distinct events:

- *OnMessageCreated.* This event fires when any type of new item is posted to the folder, such as an e-mail message, a calendar appointment, or a Microsoft Word document. This event can be generated from any type of client, such as Outlook or a web browser client. An expense report agent might use this event to look up the manager of a person who submits an expense report and then route the report to the manager for approval.
- *OnChanged.* This event fires when any type of item is edited and saved back into the folder. An example agent for this type of event is a resource scheduling agent, which monitors a public folder calendar for conference rooms. When a meeting time is changed, the agent notifies all the attendees and any catering services.
- *OnMessageDeleted.* This event fires whenever an item is deleted from the folder. It's useful when you want to synchronize the contents of a folder with another data source. By writing a custom agent for this event, you could delete from other folders or databases items that are related to the deleted item.
- *OnTimer.* This event fires based on a time limit you specify, which can be weekly, daily, hourly, or on a more granular time. For example, you can customize an event so that it fires every 15 minutes starting at 6:00 P.M. and ending at 3:00 A.M., or set the event to fire only on certain days. An example of an agent using this event is another expense report agent that works in conjunction with the sample expense report agent we just discussed. Suppose the manager does not approve the expense report forwarded by the first expense agent in the specified amount of time—an hour, let's say. A scheduled event, created to check pending expense reports every hour, determines that the expense report needs to be escalated to the manager's manager. The agent could look up that individual using the Exchange Server directory and route the expense report to her.

As you can see, by creating custom agents, you can implement custom functionality that would otherwise be unavailable in public folder rules.

Connectivity and Migration Tools

Information in a corporation is stored in various places. For employees, business partners, and customers to collaborate effectively, these "islands" of information must become connected, and information must be accessible. To enable this, Microsoft Exchange Server has a number of built-in migration and coexistence tools.

A series of connectors enable an Exchange Server to coexist with other types of collaborative systems. These connectors ensure the reliable delivery of messages between Exchange Server and these other systems, but the connector's capabilities do not stop there. The connector can also provide directory synchronization between the two systems. Directory synchronization gives clients on both systems the ability to seamlessly query the directory for users on another system. This global, unified directory in the Exchange Server system makes building collaborative applications easier because it centralizes information. The systems that Microsoft Exchange Server can connect to, send messages from, and synchronize directories with include Microsoft Mail, Lotus cc:Mail, and Lotus Notes. Exchange Server can transfer messages with host-based systems, such as OfficeVision VM (PROFS), and System Network Architecture Distribution Services (SNADS) systems, such as IBM OfficeVision/MVS and Fisher TAO.

Sometimes, corporations find it more cost-effective to have only one collaborative system rather than several. To help organizations move to Exchange Server, migration tools for a large number of systems are included in the product. These tools make it easier for organizations to transfer their users and information into the Exchange Server system. The products supported by these migration tools are Microsoft Mail, Lotus cc:Mail, Novell Groupwise, Collabra Share, and Lotus Notes.

Client Options

The Microsoft Outlook family of clients provides users with a choice of clients to use with Exchange Server. These clients support multiple platforms and provide varying levels of functionality, depending on the needs of the user. In addition, all the clients in the family support a consistent user interface, so moving from one client to another is easy. The following sections describe these clients.

Pocket Outlook

Microsoft Pocket Outlook runs on any handheld device that supports the Microsoft Windows CE version 2 operating system. Pocket Outlook enables communication and enhances collaborative work by supporting e-mail, contacts,

tasks, and scheduling. These services can be synchronized with Outlook 98 by using the built-in ActiveSync technology in Microsoft Windows CE.

Outlook Express

Microsoft Outlook Express is a POP3, IMAP4, and NNTP client that ships for free with Microsoft Internet Explorer version 4. Outlook Express provides basic e-mail and newsgroup functionality that can be customized to meet the needs of the user. When used in conjunction with Exchange Server, Outlook Express has simple calendaring functionality.

Outlook Web Access

Outlook Web Access is a browser-based view of information stored in Exchange Server and is covered in Chapter 8. Outlook Web Access supports e-mail, calendars, public folders, custom views, and directory functions, all from a standard web browser. The technology behind it comprises ASP and CDO.

Outlook for Microsoft Windows Versions 3.x and the Macintosh

For companies supporting employees who use 16-bit Windows and the Macintosh, Microsoft Outlook offers consistent versions for both platforms. Both provide the same user interface as the other members of the Outlook family and include e-mail, personal calendaring, task lists, group scheduling, HTML-based custom forms, and an easy migration path from current Microsoft e-mail clients.

Microsoft Outlook

Outlook is Microsoft's premier e-mail and collaboration client. With Outlook, users can manage many types of information including their e-mail, personal calendar, contacts, and tasks. Group scheduling, task management, and journal capabilities improve user productivity through better information management. Tight integration with Office and Internet Explorer provide major benefits to users of these applications because Microsoft Outlook extends them with enhanced functionality.

Choosing a Client

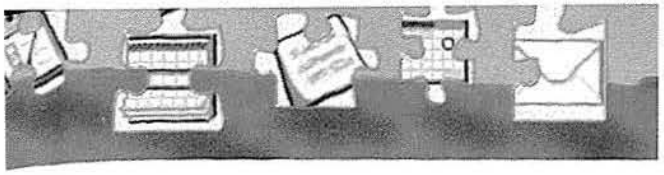
As a developer preparing to build a messaging, tracking, workflow, real-time, or knowledge management application for Exchange Server, your first question is: "Which client interface should I use—Microsoft Outlook or a web browser?" The answer depends on a number of factors.

You need to ask yourself a few questions. For example, does the application need offline support? If the answer is yes, consider using Outlook, which has built-in support for offline forms, and which automatically synchronizes any offline changes to Exchange Server. Compare this functionality to the web browser client, which has limited support for offline forms. Web browsers can cache web pages for offline viewing, but they cannot process server-side scripts such as Active Server Pages without a web server on the local machine. The typical user does not have a web server on a local machine.

Non-Win32 client support, such as Windows 3.1, Microsoft Windows for Workgroups, Macintosh, and UNIX, is another factor to look at when designing applications. The ubiquity of web browsers for multiple operating systems has enabled web-based applications to provide cross-platform client support. Although Microsoft Outlook runs only on Windows 95, Windows 98, and Windows NT, new technology enables Outlook forms to be converted to web-based applications. This technology, called the Outlook HTML Forms Converter, is discussed in detail in Chapter 8.

Your skill as a developer is another factor to consider. Microsoft Outlook offers a very approachable development environment, even for the novice developer. It also provides built-in capabilities that allow power users or developers to customize an application without writing any code—very appealing to those who want to meet a specific need quickly and avoid creating an application from scratch.

If you are more familiar with other development tools, consider that Exchange Server exposes its services through a rich API that can be called from any development environment that supports COM. Some examples of these development environments include Microsoft Visual Basic, Microsoft Office (through Microsoft Visual Basic for Applications), Microsoft Visual C++, and Microsoft Visual J++. This environment flexibility allows you to leverage the tools and skills you currently have.



CHAPTER EIGHT

Outlook and the Web

This chapter looks at how Microsoft Outlook ties into the World Wide Web and is broken into four main sections: Outlook Today, Active Server Pages, Outlook Web Access, and The Outlook HTML Form Converter. We start by examining Outlook Today, which is an HTML feature introduced in Outlook 98. Then we'll cover the basics of Microsoft Active Server Pages (ASP)—the foundation for Outlook and the Web. We'll move on to overview installing and configuring Outlook Web Access. Outlook Web Access technology is based on Active Server Pages and Microsoft Collaborative Data Objects, and it's used to access information on Microsoft Exchange Server and Microsoft Internet Information Server (IIS). We'll finish with a discussion of the Outlook HTML Form Converter. The Form Converter is a tool to help you convert your Outlook forms into a web-based format that integrates Outlook Web Access.

Outlook Today

As you learned at the end of Chapter 7, Outlook 98 includes a feature that takes advantage of the HTML support in Outlook—Outlook Today. Outlook Today allows users to view information in an HTML window, as shown in Figure 8-1 on the next page, rather than as separate modules. You can also customize Outlook Today's HTML code. For example, a customized Outlook Today page might include hyperlinks to Public Folder favorites, intranet sites, or Internet sites. You can even link to other applications, such as a web-based SQL Server application. You can also create multiple Outlook Today pages for the different types of users of your application. Let's take a look at how the Outlook Today page functions and how you modify the Outlook Today page for your specific application needs.

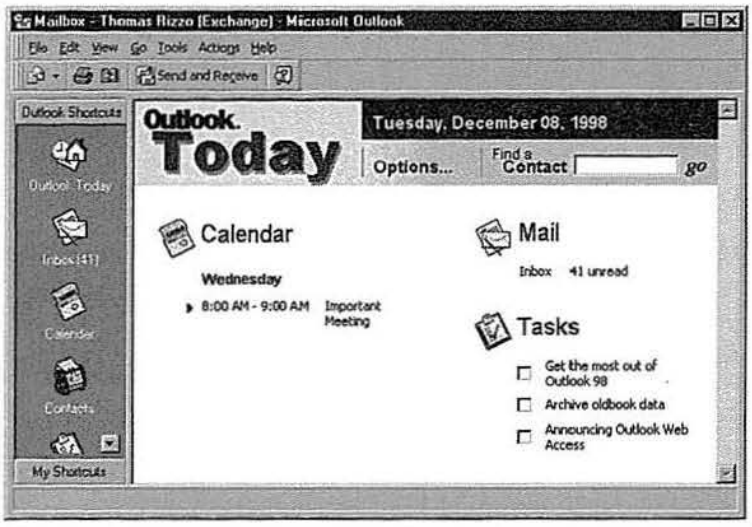


Figure 8-1
Outlook Today is an HTML feature added to Outlook 98. Users can see different types of information in a single window.

Outlook Today Technologies

The standard Outlook Today page takes advantage of a feature in Microsoft Internet Explorer called data binding. Data binding allows Outlook to quickly display the user interface for the Outlook Today page while asynchronously downloading data from the Exchange Server to the Outlook client. Once the data is downloaded, Outlook can modify the data without making additional trips to the server. On the default Outlook Today page, there are three separate data binding tables for your Calendar, Tasks, and Mail. To modify the location of these tables or to add new functionality to the Outlook Today page, you need to know how to modify HTML. Since Outlook Today leverages Internet Explorer, you can even use Dynamic HTML (DHTML) in your customization. However, remember these limitations when customizing the Outlook Today page:

- Modifying the page might slow performance because Outlook has to retrieve information from other data sources. Try not to build complex applications in the Outlook Today window; instead, build either an Outlook form or a standard HTML application, and then add the HTML application to Outlook Web Access.
- Although you can add external links to Internet sites on your Outlook Today page, Outlook Today will not verify the security of the site. To use the security capabilities you have in your browser, you could add a link in Outlook Today that launches a separate browser,

such as Internet Explorer, to render the page. If you are sure of the content that you are linking to, you do not have to browse the link in a separate browser.

- You cannot add the Outlook Today page as an Active Desktop item. Currently, the Outlook Today functionality works only when hosted in the Outlook window.

NOTE: The standard Outlook Today pages are hosted in a resource dynamic link library (DLL), which improves the performance of the Outlook Today application. When modifying your Outlook Today pages, you have the option to place your custom HTML pages and images in a resource DLL. However, this book covers only customizing Outlook Today and saving these customizations as HTML files. To learn how to compile your files into a resource DLL for Outlook Today, refer to the Outlook 98 Deployment Kit. The Deployment Kit is presently not available for retail purchase or download from the Web, but you can access the Deployment Kit documentation from *http://www.microsoft.com/office/98/outlook/documents/O98DKdoc.htm*.

Customizing Outlook Today

Customizing the Outlook Today page and saving the result as an HTML file involves a few steps, which are outlined in the following sections.

Retrieving the Outlook Today Source

To view the source of an HTML page in Internet Explorer, you can right-click on the page and select View Source from the context menu. If you right-click on the Outlook Today page to view the source of the HTML page, however, you will notice that the View Source option is not available—Outlook disables this option. To retrieve the HTML source for the Outlook Today page, you can copy the Outlook.htm file from the Outlook 98 Deployment Kit, or you can use Internet Explorer to retrieve the source by following these steps:

1. Start Internet Explorer. In the address box, type the following URL, adjusting the path as necessary:

   res://c:\Program Files\Microsoft Office\Office\Outlwww.dll/Outlook.htm

2. After you press Enter, you will get a script error. Select No to decline continuing running scripts and debugging the current page.

3. From the View menu, select Source to display the source in Notepad.
4. From the File menu in Notepad, select Save As and save the file to your hard disk as Outlook.htm.
5. Perform a search in Notepad for the three instances of *display:none*, and replace them with *display:*.

Modifying the Registry

You need to modify your Registry settings so that Outlook knows you want Outlook Today to point to a new file for its functionality. You could write a program that performs this step for your customers:

1. Start the Registry Editor (Regedit.exe).
2. Find the following Registry key:

   HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Outlook\Today

   If the Today key does not exist, create it. Right-click on the Outlook key, point to New, and select Key. Type *Today* for the name of the key.
3. Add a new string value to the Today key by right-clicking on the Today key, pointing to New, and selecting String Value. Type *Url* for the value name.
4. Double-click on the URL string icon to edit it. For its value data, type the path to the Outlook.htm file that you saved previously. For example, file://C:\Outlook.htm. When finished, click OK.

Customizing the HTML File

The final step is to customize the HTML file with your functionality. The easiest way to do this is by using a text editor, because the HTML code in the page contains special formatting that will make the file appear incorrectly in Microsoft FrontPage. Let's review some of the ways you can customize the Outlook Today HTML page.

Changing fonts Since Outlook Today uses cascading style sheets, you can easily change fonts and styles by modifying the style sheet. For example, you could change the font for important items by changing the *.itemImportant {color:red}* line in the style sheet to the desired font and color.

Adding text, images, and hyperlinks Using HTML, you can add new text, images, or hyperlinks to your Outlook Today page. Remember that if you link to an external web site, Outlook will not implement the security that you set

in your standard web browser. So use the following code when placing external links in the Outlook Today page:

```
<a style="cursor:hand" class="itemNormal" onclick=
"window.open('http://www.microsoft.com/exchange/','_blank');">
Exchange web site</a>
```

Adding components Since Outlook Today leverages Internet Explorer, you can place any components on your page as long as Internet Explorer supports them. These components can include ActiveX controls as well as Java applets. However, make sure you trust the source of the component because Outlook does not check the component's security credentials.

Adding script Outlook Today supports both JScript as well as VBScript. From script, you can access the Outlook object library and use its functions in your Outlook Today page. You can see an example of a customized Outlook Today page for the Account Tracking application in Chapter 7.

Active Server Pages

In this section, we'll explore Microsoft Active Server Pages (ASP) technology. You should know about ASP for several reasons. First, the Outlook HTML Form Converter, a conversion tool that migrates Outlook forms to HTML forms, utilizes this technology. We'll examine the converter later in this chapter. Second, ASP is used in other areas, including Collaborative Data Objects (CDO) and Active Directory Services Interfaces (ADSI). We look more closely at CDO in Chapter 11 and ADSI in Chapter 14.

ASP Fundamentals

Active Server Pages are standard text files that contain HTML and script. The script can be written using any ActiveX scripting language, such as VBScript or JScript. The HTML files that most web developers write differ from ASP files in two significant ways. First, instead of having an .htm or .html file extension, ASP files have an .asp file extension. When you install IIS, an Internet Server Application Programming Interface (ISAPI) component is installed that processes all files with an .asp extension. This ISAPI component parses the ASP file and executes the appropriate script. Second, the actual script is processed on the web server. The processed results can include client-side scripting code but for the most part is just simple HMTL. Returning only HTML has two benefits: any modern web browser can view the results of an ASP application and the additional capabilities of the browser is less of an issue.

Since Active Server Pages supports VBScript, you can easily move from developing Outlook forms to developing Active Server Pages. The only difference in the development process is that you should use the CDO library to write your Active Server Pages application rather than the Outlook object library, because CDO was designed to be multiuser and server-based.

The following code is an example of an Active Server Pages application. This example uses the VBScript function *Now* to print out the date and time that the Active Server Pages application ran on the web server.

```
<%@ LANGUAGE="VBSCRIPT"%>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">

<HTML>
<HEAD><TITLE>ASP Example</TITLE></HEAD>
<BODY>
<H1>I was created on <%=Now()%></H1>
</BODY>
</HTML>
```

As you can see, the syntax of the ASP script is a little bit different from the syntax for Outlook code. To tell the web server that you want to run a script on the server, you must enclose it in special characters: <% and %>. Active Server Pages supports placing your script directly in your HTML code—the script does not have to be in a separate section of the HTML file.

Take a look at the first line of the code:

```
<%@ LANGUAGE="VBSCRIPT"%>
```

ASP assumes that the default language for server-side script is VBScript. If you replace VBSCRIPT with JSCRIPT, you can write server-side JScript code.

> NOTE: You can specify the default ASP language for an application in the Management Console for IIS. Open the Properties window for an application, and in the Applications Settings area, click the Configuration button. On the App Options tab, type the desired default language in the Default ASP Language text box.

You might be wondering what the <%=*Now()*%> code does in this example. The equal sign (=) indicates that the code should evaluate the expression, which in this case returns the current date and time. As you will see, the equal sign in ASP is a shortcut for calling the *Write* method of the Response object.

Global.asa

If you've viewed the actual directories that contain .asp files, you might have noticed a certain file with the .asa extension: Global.asa. This is a special file in ASP applications that allows you to include global code that executes when an application starts and ends and also when a session starts and ends. One thing to remember is that the Global.asa is an optional file for your web applications. A skeleton Global.asa file is shown here:

```
<SCRIPT LANGUAGE="VBSCRIPT" RUNAT="Server">
    Sub Session_OnStart
        'Put your session startup code here
    End Sub
    Sub Session_OnEnd
        'Put your session termination code here
    End Sub
    Sub Application_OnStart
        'Put your application startup code here
    End Sub
    Sub Application_OnEnd
        'Put your application termination code here
    End Sub
</SCRIPT>
```

The Global.asa file contain stubs for your session and application start and end subroutines. To understand when these subroutines are called, you must understand what exactly constitutes a session and an application inside ASP.

Normally when you browse web pages, the web server does not remember who you are, where you have been, or any variables associated with you. One of the great things about ASP is that it transforms the applications you can build on the HTTP protocol from being stateless to being able to track the state of users. This ultimately lets you create global variables that are maintained for users throughout an application.

An ASP application consists of a virtual directory and associated files. But to understand when an ASP application starts and ends, you'll need a little bit more explanation of how ASP works. For your *Application_OnStart* subroutine to be called, the first user must request an .asp file from the virtual directory of your ASP application. The user can request an HTML file or other types of files from that directory. However, these requests will not cause the *Application_OnStart* subroutine to be called. The user must explicitly request an ASP file. This is the only time this subroutine will be called, unless you restart the application. Restarting the application usually consists of restarting the Web service.

You should use the *Application_OnStart* subroutine to initialize global variables across the lifetime of the web application. For example, a good example of a variable to initialize or set in your *Application_OnStart* subroutine is one that counts the number of users who have used your application. To improve performance, for every user in your ASP application, you should initialize in the *Application_OnStart* subroutine any server components that you will use. Figure 8-2 illustrates a web browser sending a request to an ASP application for the first time.

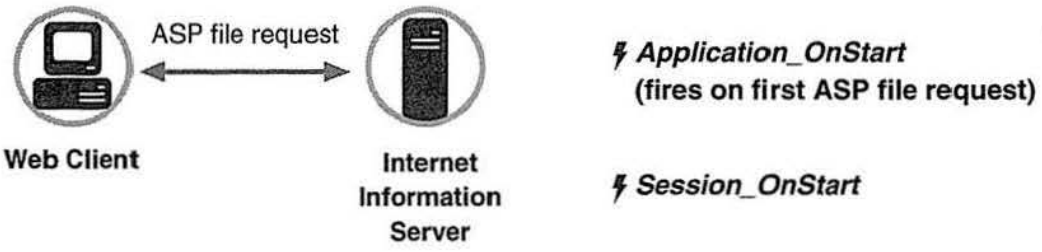


Figure 8-2
When the first user of an application requests an .asp file, the Application_OnStart *event is fired and then the* Session_OnStart *event fires.*

When the user who first requested the ASP page also browses an .asp file in your application, the *Session_OnStart* subroutine is called. Unlike the *Application_OnStart* subroutine, the *Session_OnStart* subroutine is called for any user who makes an application file request. With ASP, each user of your application is considered to have a distinct session with the web server. As a user browses web pages in your ASP application, ASP implements and maintains state in a session by using cookies—whenever a user connects to your application, a file containing information (a cookie) is saved on the user's machine. When his session ends and he closes his web browser, the cookie is removed and the session is invalidated. If he reconnected to your application, his machine would receive a new cookie and a new session would be started. For this reason, the users of your application must support and accept cookies; otherwise, your ASP applications will not fully function. You can still use the server-side script of ASP, but you cannot maintain state information for any of your users.

The *Session_OnStart* subroutine is best used to initialize session variables for individual users. Session scope variables include a connection to Exchange Server for an individual user and personalized information that a user has set in your application—for example, a user could specify a background color for web pages that is stored in a session variable. Then, each page the user accesses

from your site during a session could be displayed in his personalized background color. Figure 8-3 shows each web browser starting a new session when accessing an ASP application.

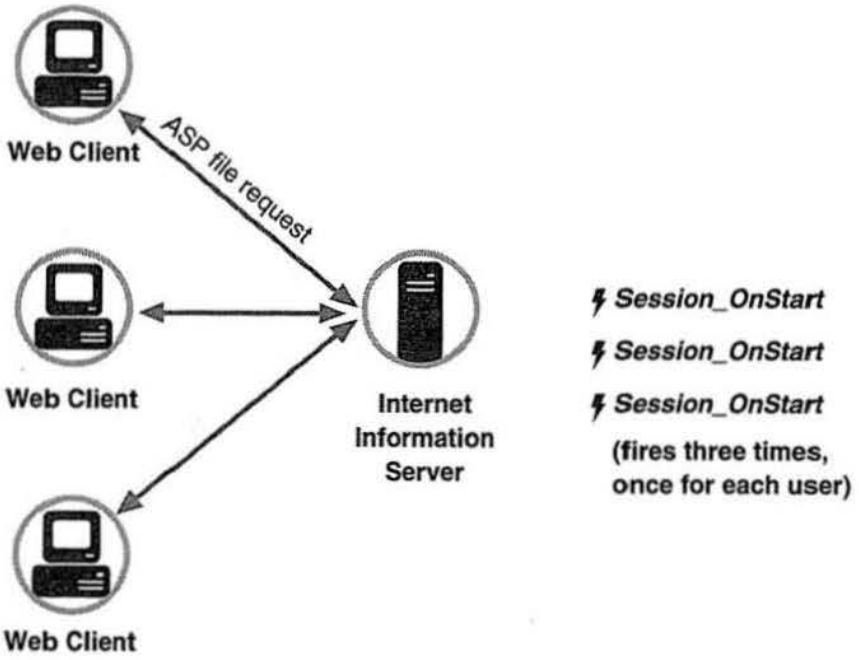


Figure 8-3
Whenever a new user accesses your ASP application, the Session_OnStart *event fires.* Application_OnStart *fires only when the first user accesses your application.*

The *Session_OnEnd* subroutine is called when the session with the web server ends. This end state can be reached in two ways:

- When the user has not requested or refreshed a web page in the application for a specified amount of time
- By explicitly calling the *Abandon* method on the Session object

By default, IIS sets the timeout interval at 20 minutes. You can change this interval either through the administration program for IIS or by setting the TimeOut property on the intrinsic Session object in ASP. For example, to set a particular script timeout to 10 minutes, you would write the following code in your ASP application:

```
<% Session.TimeOut = 10 %>
```

The second way to reach the end state—by explicitly calling the *Abandon* method on the Session object—immediately ends the session and calls the *Session_OnEnd* subroutine.

> NOTE: Applications discussed in later chapters (CDO Help Desk, Event Scripting Expense Report, Routing Objects Expense Report) provide a logout menu option. This option calls another ASP file, which calls the *Abandon* method on the Session object to end the session.

One final note about sessions: you have to be careful when you redirect people to other virtual directories in your application. Developers, including me, commonly make the mistake of redirecting users to another virtual root and forget that this is considered by ASP to be an application. When you do this, the session variables you establish in one application will not transfer to the other application. If you want to share session variables between the two applications, you should place the second application under the same virtual directory in IIS as the first application.

When a web application ends, the *Application_OnEnd* subroutine is called. You end a web application in one of two ways: by shutting down the web server, or by stopping your application by using the Unload button in the IIS administrator. To use the Unload button, you must be running your web application in a separate memory space. So make sure you save any application scope variables to a persistent medium, such as to your Exchange Server or to a database, so that when your application restarts, the *Application_OnStart* subroutine can reload the values. For example, you don't want a user-counter variable to restart at zero every time your application restarts. You should also destroy any server objects that you have created with an application scope. This will eliminate potential memory leaks on your server.

Built-In ASP Objects

The real power of ASP applications is that you can write server-side scripts and use their intrinsic objects. ASP and its built-in objects enable you to generate custom responses and maintain state information. The following section describes, in detail, five built-in objects in ASP: Application, Session, Request, Response, and Server.

> NOTE: The only object not covered here is the ObjectContext object, available in IIS version 4.0. This object can be used for creating ASP applications with transaction capabilities. For more information on the ObjectContext object and transactions, consult the IIS product documentation.

Application Object

The Application object is used to store global data related to an application that can be shared among all users. By using the methods and properties of this object in your application, you can create and set variables that have an application scope. To make sure that you do not run into concurrency issues when setting your application-level variables, since multiple users can be using the same application simultaneously, the Application object provides two methods named *Lock* and *Unlock*. These methods serialize the access to application-level variables so that only one client at a time can read or modify the values. The following example shows how to use the *Lock* and *Unlock* methods to increment a user-counter variable whenever a user accesses the application. The example also shows you how to set and retrieve application-level variables by using the *Application("VariableName")* syntax:

```
<HTML>
<HEAD>
<TITLE>Example: Application Object</TITLE>
</HEAD>
<BODY>
<%
    Application.Lock
    Application("NumVisitors") = Application("NumVisitors") + 1
    Application.UnLock
%>
Welcome! You are visitor #<%=Application("NumVisitors")%>.
</BODY>
</HTML>
```

The Application object also contains two other collections beyond the variables collection—Contents and StaticObjects—which allow you to browse through the application-level objects and variables you have created. You probably won't use either of these collections in your final applications, but both of them provide great debugging functionality. For example, the Contents collection enables you to list all the items that have been added to your application through a script command, and the StaticObjects collection enables you to list all the items with an application scope that have been added using the <OBJECT> tag. By adding debug code to your application at design time, when you run into application object problems, you can make ASP list all the objects you have created with an application scope. The following code illustrates creating debug code for both the Contents and StaticObjects collections. You can see the code output in Figure 8-4 on the following page.

```
<HTML>
<HEAD>
<TITLE>Debugging Application Objects</TITLE>
<%
    'Create some application variables
    Application.Lock
    Set Application("oCDOSession") = _
        Server.CreateObject("MAPI.Session")
    Application("counter") = 10
    Application.UnLock
%>
<P>Objects from the Contents Collection<BR>
<%
    for each tempObj in Application.Contents
        response.write tempObj & "<BR>"
    next
%>
<P>Objects from the StaticObjects Collection<BR>
<%
    for each tempObj in Application.StaticObjects
        response.write tempObj & "<BR>"
    next
%>
</BODY>
</HTML>
```

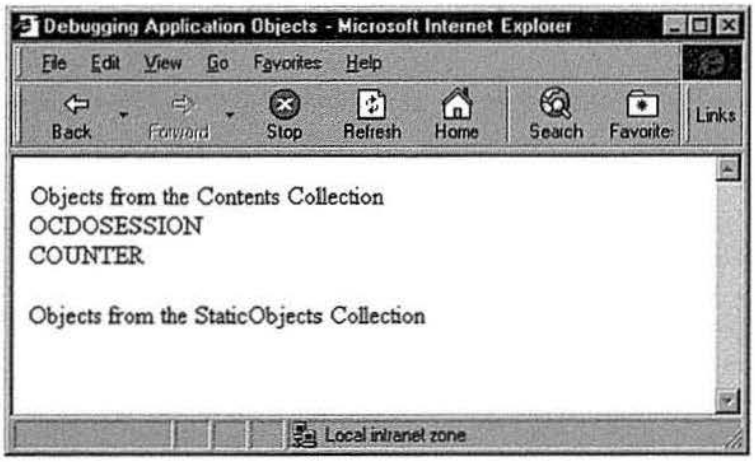


Figure 8-4

The debug output for the Contents and StaticObjects collections. As you can see, objects and variables both can have an application scope.

Session Object

The Session object is one you'll use a lot in your web applications. It holds the variables for individual users across the web pages in your application. When you place a variable in the Session object, that variable is valid only for the current user and cannot be shared across users in the same way that an Application variable can.

Like the Application object, the Session object contains the Contents and StaticObjects collections. You can also create session variables in the same way you create *Application* variables, by using the syntax *Session("VariableName")*.

The properties for the Session object include CodePage, LCID, SessionID, and TimeOut. The CodePage property represents the language code page that will be used to display the content for the HTML page. The Outlook HTML Form Converter, which you'll learn about later in this chapter, uses this property in its converted forms, as shown here:

```
<% @LANGUAGE=VBSCRIPT CODEPAGE = 1252 %>
```

You can use the LCID, or locale identifier, property in conjunction with the CodePage property. The LCID property stores a standard international abbreviation that uniquely identifies a system-defined locale.

The SessionID property returns to you the unique session identifier for the current user. You should remember, however, that this ID is unique only during the lifetime of the ASP application. If you restart your web server and therefore restart your web applications, the web server might generate the same IDs it already generated for the users before the web application was restarted. For this reason, you should avoid storing these IDs and attempting to use them to uniquely identify a user of your application. If you always need to uniquely identify your users whenever they access your application, you should use globally unique identifiers (GUIDs) in cookies, which are saved on the users' computers.

The fourth property of the Session object is the Timeout property. This property enables you to change the timeout period associated with a particular ASP session. Remember that by default, the timeout is set to 20 minutes. If you know that your application will be used for less than 20 minutes, you might want to decrease the duration of the timeout so that sessions end more quickly and resources are returned to the web server at a faster rate.

The only method of the Session object is the *Abandon* method. As mentioned earlier, by calling this method, the user's session with the web server as well as any associated objects and variables for that session are destroyed. If the user attempts to reconnect to the web application, a new session starts on the server.

Request Object

The Request object allows you to access the information that was passed from the web browser to your web application. The Request object is crucial in ASP applications since it enables you to access user input for your server-side scripts. For example, suppose a user fills out an HTML form that you created. Once the user clicks the Submit button on the form, the Request object contains the form information that was passed to the server. By using the collections of the Request object, you can retrieve that information and design your application to respond based on the user's input.

Request object collections The Request object collections are created when the user submits a request to the web server either by requesting an ASP file or by submitting an HTML form via clicking the Submit button. The three collections of the Request object that you'll primarily work with in your ASP applications are the Form, QueryString, and ServerVariables collections.

> NOTE: For information on the other two collections, ClientCertificate and Cookies, refer to the IIS documentation.

To understand when to use these collections, you first need to know about the different ways information can be passed from the web browser to the web server. Normally in your web applications, you use HTML forms to gather input from the user so that you can use it in your calculations or store it in a data source. There are two main ways input can get passed to the web server from the client browser: via the *GET* method and via the *POST* method. The following example shows an HTML page that contains both methods on the same page:

```
<html>
<head>
<title>Forms Galore</title>
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
</head>
<body>
<form method="GET" action="getinfo.asp" name="GetForm">
    <p>What is your email address?</p>
    <p><input type="text" name="email" size="20"></p>
    <p><input type="submit" value="Submit" name="GetSubmit"> </p>
</form>

<form method="POST" action="getinfo.asp" name="PostForm">
    <p>What is your First Name?</p>
    <p><input type="text" name="firstname" size="20"></p>
    <p><input type="submit" value="Submit" name="PostSubmit"> </p>
</form>
</body>
</html>
```

The Action attribute for each of the HTML forms specifies the same ASP file, getinfo.asp. The getinfo.asp file is shown here:

```
<HTML>
<HEAD>
<TITLE>Post and Get Methods Example</TITLE>
</HEAD>
<%txtRequestMethod = Request.ServerVariables("REQUEST_METHOD")%>
You selected to use the <B><%=txtRequestMethod%></B> Method.
<P><% if txtRequestMethod="GET" then %>
You entered your e-mail address as:
<B><%=Request.QueryString("email")%></B>
<% else %>
You entered your first name as: 
<B><%=Request.Form("firstname")%></B>
<% end if %>
</BODY>
</HTML>
```

This ASP code uses the ServerVariables collection of the Request object to check whether the form's *Request* method was a *POST* or a *GET* method. Once the file determines which method was used, it displays the correct information for that particular type of form. Figure 8-5 shows a sample of the *GET* method.

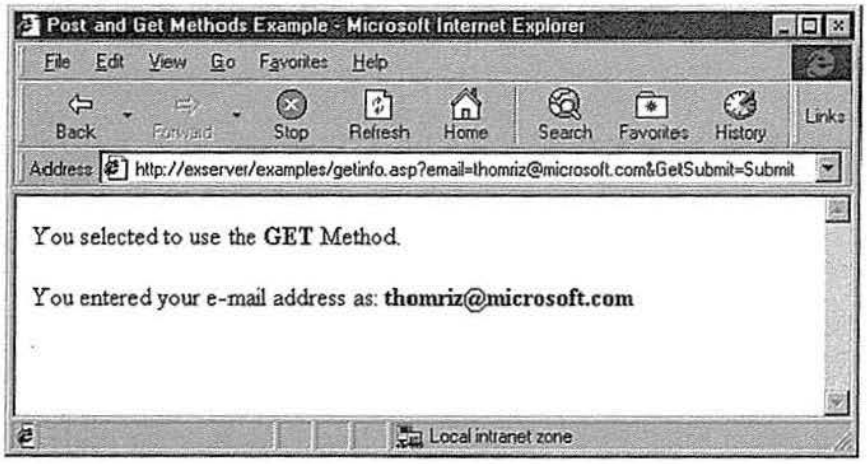


Figure 8-5
When a user types an e-mail address and submits the form, the GET *method is used to pass the information to the* Request *object.*

NOTE: You can also retrieve other server variables such as HTTP_USER_AGENT, which returns information about which browser the client is using; and LOGON_USER, which represents the Microsoft Windows NT account the user is currently logged on to. For a complete list of server variables, please see the IIS documentation.

As you can see in Figure 8-5 with the *GET* method, the information from the form is actually appended to the URL—for example: *http://exserver/examples/getinfo.asp?email=thomriz@microsoft.com&GetSubmit=Submit*. When data is appended to the URL using the *GET* method of a form, you use the QueryString collection of the Request object to retrieve the data. When using the QueryString collection, follow this format to retrieve the information:

```
Request.QueryString("VariableName")
```

Because the information that is passed to your application appears in the address of the user's browser, the user can see it, so you might want to limit when you use the *GET* method. Instead, consider using the *POST* method.

The *POST* method places the form information inside the HTTP header, hiding the information from the client. However, when the *POST* method is used to submit form variables, you cannot use the QueryString collection. Instead, you need to use the Forms collection of the Request object. In the preceding example, the line

```
Request.Form("firstname")
```

retrieves the information the user typed into the First Name text box on the form. You can use this same syntax in your applications to retrieve information from an HTML form.

Response Object

The Response object is used to control the content that is returned to the client. For example, when you calculate a value on the server, you need a way to tell the ASP engine that you want to send the information back to the client. You do this by using the *Write* method of the Response object.

The *Write* method of the Response object will be the most commonly used method in your ASP applications. Even though you have not seen any explicit Response.Write statements in the examples, they are there. The syntax *<%=Variant%>* is equivalent to *<% Response.Write Variant %>*. The shorthand version makes it easier for you to put these statements in your code quickly.

The Response object has a number of other collections, properties, and methods that you can use, such as the Expires property, which tells the web browser how long to cache a particular page before it expires. If you do not want your clients to cache your web pages, you would add the following line to your ASP files to cause your web page to expire immediately on the user's local machine:

```
<% Response.Expires = 0 %>
```

The Response object allows you to buffer the output of your ASP page. This is useful if you want to hold back the output of your ASP code until the script completes its processing. The best example for using buffering is to capture errors in your code. For example, by turning buffering on using the command *Response.Buffer = True*, you can check throughout your ASP code whether an error has occurred. If one has, you can clear the buffer without sending it by using the *Response.Clear* method, and then you can replace the output with new output such as *Response.Write "An error has occurred. Please contact the administrator."* Finally, you can call the *Response.End* method, which sends the new buffer to the client and stops processing any further scripts in the ASP.

Server Object

The Server object provides you with utility methods and properties to modify the information on your web server. This object is used extensively in ASP applications because it contains both the *CreateObject* method and the ScriptTimeout property.

The *CreateObject* method allows you to create an object on the web server by passing in the ProgID for the object. Let's look at an example. To create a CDO object, you would type this in your ASP file:

```
Set oSession = Server.CreateObject("MAPI.Session")
```

ASP creates an object and passes that object to you in the *oSession* variable. By default, when you do this on an ASP page, the object has page-level scope. This means that when ASP is done processing the current page, the object is destroyed. Therefore, you might want to create objects on a page and then store them by assigning them to either session variables or application variables, as shown in this code snippet:

```
<%
    Set oSession = Server.CreateObject("MAPI.Session")
    Set Session("oSession") = oSession
%>
```

As you learned earlier, an object that is assigned either a session or an application scope will be destroyed when either the session or the application ends, respectively. The one issue to watch out for with the *CreateObject* method and some objects is potential performance loss. You can instantiate almost every object on your web server as an ASP object, but some objects are specifically designed to run in a server-based, multiuser environment such as CDO. When you instantiate an object that was not designed for an ASP environment, the application performance might suffer if many people hit the page containing that object at the same time.

The ScriptTimeout property of the Server object allows you to specify how long a script should run before it is terminated. By default, an ASP script can run for 90 seconds before it is terminated, but this might not be enough time to retrieve data from a data source. By using the following syntax for this property, you can increase or decrease the amount of time the script will run before termination:

```
Server.ScriptTimeout = numseconds
```

Avoid increasing this number much beyond 90 seconds, because users who are waiting for long periods of time might assume the page did not load correctly, and they might hit their Stop and then Refresh buttons continuously, flooding your web server with requests.

Server-Side Include Files

One other powerful feature beyond the intrinsic objects of ASP is the ability to use Server-Side Include files in your ASP files. Include files are just text files containing script or HTML that you want to add to your ASP page. Microsoft Outlook Web Access, which you will learn about later in this chapter, relies heavily on Server-Side Includes for common code libraries in its ASP files. The following lines are examples of Server-Side Includes:

```
<!-- #include file="library/vbsfunctions.inc" -->

<!-- #include virtual="/library/vbsfunctions.inc" -->
```

Server Components

ASP can take advantage of built-in objects and also use server components to add functionality to ASP. An example of two such components are Microsoft ActiveX Data Objects (ADO) and CDO. ADO allows you to connect to many types of databases; CDO allows you to connect to Exchange Server and other messaging servers. You can also write your own components using any COM-based development tool.

> NOTE: There are a number of other components packaged with ASP that you can use in your applications, including Ad Rotator, Browser Capability, Content Linking, Content Rotator, File Access, Page Counter, and Permission Checker. If you want to learn more about these components, you should refer to the documentation that ships with IIS version 4.0.

Outlook Web Access

Microsoft Outlook Web Access is an ASP application that Microsoft ships with Exchange Server version 5.5. This ASP application allows you to access your mailbox, calendar, and contacts as well as directory information using any standard web client. The Outlook Web Access application is built on CDO and is one of the best tools for learning CDO.

In this section, you'll learn how to install Outlook Web Access on your web server, which also installs the CDO library. You'll also learn about security when using Outlook Web Access. This security architecture is important since it also applies to any custom CDO applications you develop using ASP.

Installing Outlook Web Access

Before installing Outlook Web Access, you must have installed either IIS version 3.0 or IIS version 4.0 with Active Server Pages. IIS 4.0 and Exchange Server 5.5 both require Windows NT 4.0 Service Pack 3, but it is recommended that you install Service Pack 4. You can download the Window NT 4.0 Service Pack 4 from *http://www.microsoft.com/windows/downloads/*. If you don't install Service Pack 4, you will need to install the Windows NT related fixes required for Outlook Web Access. You can download these hot fixes from *ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes/usa/nt40/hotfixes-postsp3/roll-up/*.

You can install Outlook Web Access on the same server as your Exchange Server or on a separate server. Be aware that if you do install Outlook Web Access on a separate server, you cannot use Windows NT Challenge/Response as your authentication method. You'll learn more about security implications later in the chapter.

The architecture for your web servers and Exchange Servers can vary depending on the topology of your network environment and requirements of your applications. For example, if few users will be accessing Outlook Web Access but you have a number of Exchange Servers and you do not want to set up multiple Outlook Web Access servers for each Exchange Server, you can set up just one Outlook Web Access server to talk to multiple Exchange Servers. The opposite is true as well. You can have multiple Outlook Web Access servers talking to just one Exchange Server. Think of it as a web farm of Outlook Web Access servers. This will work as long as you make sure that when a user starts a session with an Outlook Web Access server in a web farm, that user stays with the same Outlook Web Access server until her session expires or she logs out. Remember that ASP sessions do not span separate ASP applications. If you use DNS round-robin techniques to farm a user out to multiple Outlook Web Access servers, that user's session will be lost when she changes to a different server.

To install Outlook Web Access, follow these steps:

1. Insert the Exchange Server version 5.5 CD in your CD-ROM Drive.
2. If the Exchange Server welcome screen does not start automatically, launch it by double-clicking on Launch.exe.
3. Click on Server Setup And Components.
4. Click on Microsoft Exchange Server 5.5.
5. When setup starts, click Complete/Custom installation. (If you already have Exchange Server installed, click Add/Remove.)
6. In the Options list, check Outlook Web Access and click Continue. If you don't have the Windows NT Service Pack 4 (or the Windows NT related fixes) and IIS installed, a message box will be displayed and you won't be able to continue the installation of Outlook Web Access.
7. The setup program will prompt you for the name of an Exchange Server that Outlook Web Access should connect to. Type a name of a server that contains an entire replica of the Exchange Server directory. This sets up Outlook Web Access so that it automatically redirects itself to the Exchange Server where the mailbox of the user resides. It also allows you to set up one Outlook Web Access web server that talks to multiple Exchange Servers.

After completing the Outlook Web Access installation, you need to update it by installing Service Pack 1 for Exchange Server 5.5. You can download or order this service pack from *http://backoffice.microsoft.com/downtrial/moreinfo/ex55sp1.asp*. This service pack includes a number of enhancements for the Outlook Web Access client, such as the ability to access Outlook contacts from the Web.

After running the update, you need to set the proper permissions in the User Manager For Domains. Ensure that the Exchange users who will use Outlook Web Access have the following rights: "Log On Locally" and "Access This Computer From Network".

Access your new Outlook Web Access server by typing the following URL in your browser: *http://*OWAServer*/exchange*, replacing *OWAServer* with your Outlook Web Access server name. From the displayed page, you can log in as an Exchange user or log in with anonymous access. (Anonymous access is discussed in more detail in Chapter 11. Figure 8-6 shows how Outlook Web Access looks when you log on as an Exchange user.)

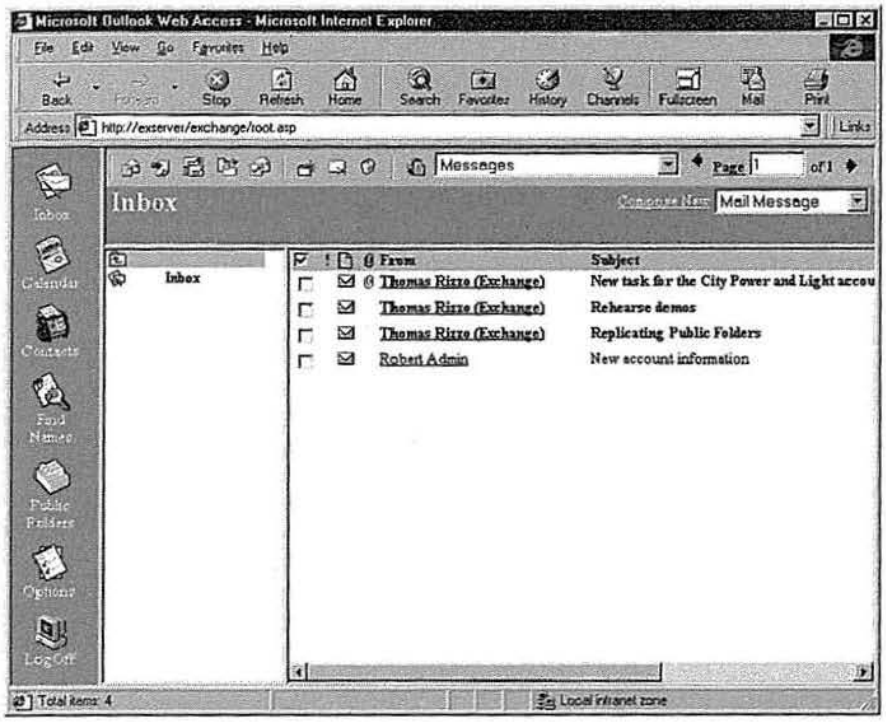


Figure 8-6
Outlook Web Access.

Outlook Web Access and ASP Security

Since the Outlook Web Access application utilizes Active Server Pages, you need to understand the ASP security architecture and how best to configure it for your environment. If you configure the environment incorrectly, you will run into problems when attempting to use authenticated access to the Outlook Web Access application or to any of your CDO web applications requiring authenticated access. This section describes how ASP security works and how you should set up Windows NT to support the type of security you want for your web applications.

ASP Security

When IIS is first installed, it creates a Windows NT user account called IUSR_*computername*, where *computername* corresponds to the current computer name. This account is assigned to the Guests account group, is given a random password, and is granted the right to Log On Locally. Whenever a user browses a web page, this account attempts to access the page on behalf of the user. If the IUSR_*computername* account does not have the proper permissions

to access the page, the request is rejected with this error message: 401 Access Denied. The web server then informs the web browser which authentication methods the web server will support—either Basic authentication or Windows NT Challenge/Response authentication—depending on your settings for your IIS server.

Basic authentication Basic authentication is supported across all web browsers. When the web server informs the client that it supports Basic authentication, the web browser displays a message box asking the user for a user name and password. Once the user types this information in, the web server tries to invoke the request using the identity of the supplied user rather than the IIS anonymous account. It is a good idea to pass in your domain name as well as the user name in the authentication dialog box in the web browser using the syntax *domain\username*.

Basic authentication, if used over Internet connections, can present some security concerns because the user name and password typed into the authentication dialog box is transmitted to the server as clear text. If you do use Basic authentication over Internet connections, use it in conjunction with Secure Sockets Layer (SSL). SSL will encrypt the connection between the web browser and the web server so that any information passed between the two cannot be viewed by unwanted individuals.

For the web server to impersonate the user whose name is typed into the authentication dialog box, the web server must log on as that user. By default, Windows NT does not give regular users the Log On Locally right on the server computer. For this reason, you must give all the users you expect will use your web application with Basic authentication enabled the Log On Locally right on your Windows NT server, which runs your web server. The easiest way to do this is to grant all your domain users the Log On Locally right in the User Manager for Domains.

Windows NT Challenge/Response authentication Windows NT Challenge/Response, or NTLM, authentication is the most secure form of authentication because the user name and password are not sent from the web browser to the web server. Instead an encrypted challenge/response handshake mechanism is used. Unlike Basic authentication, NTLM typically does not prompt the user for a name and password. The Windows NT security credentials of the web user currently logged on are sent to IIS and are used to access the requested resource. IIS then changes to the context of the specified user and attempts to access the resource. If this fails, the user will be prompted for a user name and password.

NOTE: For Windows NT Challenge/Response to work correctly, the users you are trying to authenticate must have "Access This Computer From Network" rights in User Manager for Domains. This is normally enabled for users by default.

A one-way encryption method is used, meaning the mechanism validates the user without sending the password to IIS. IIS doesn't know the user information and cannot use it to access other resources on other machines. Essentially, this is a problem of delegation. When IIS attempts to access a resource on another machine, the other machine will prompt IIS for user credentials. Since IIS does not have the password for the user, it cannot return the correct information to the other machine. For this reason, you cannot use the Windows NT Challenge/Response authentication method with Outlook Web Access when your Outlook Web Access server is on a server different from your Exchange Server. IIS cannot remotely send the authentication to the Exchange Server when the Windows NT Challenge/Response method is used. This problem is being fixed for Windows 2000, but it's a gotcha for now.

A second gotcha of the Windows NT Challenge/Response method is that you cannot use it over proxy connections for the same reasons just discussed. So when setting up your web server, consider NTLM's security advantages as well as its limitations.

A third gotcha for NTLM is that at the time of this book's publication, NTLM is supported only by Microsoft Internet Explorer. This means that if you have a mixture of web browser clients accessing your application, you might want to enable both Windows NT Challenge/Response as well as Basic authentication. If you enable only Windows NT Challenge/Response, when your Netscape Navigator users attempt to access a secure resource or page, they'll receive a message denying them permission. With both security methods set up, if Windows NT Challenge/Response fails, Basic authentication will be used.

ACLs Another way to restrict access to your web pages is by setting NTFS file permissions, or access control lists (ACLs), on your actual ASP files and directories. Doing so controls who can and cannot read the files. IIS respects the ACLs on the files, and if you have authentication enabled on your IIS server, IIS will use it to attempt to verify users and their individual permissions on the files. Be careful when setting permissions on files, however, because if the permissions you set are too restrictive, users will not be able to use your application.

Special Considerations for Setting Up Outlook Web Access

ASP security and Outlook Web Access security work in the same way when a user is being authenticated, but when you set up Outlook Web Access on your web server, you need to keep some access issues in mind. The following section describes file permission issues when users try to access Outlook Web Access on a web server, problems that could ultimately cause trouble in your CDO applications.

File Permissions for the Outlook Web Access Files

Outlook Web Access is installed by default in a subfolder under Exchsrvr, named Webdata. If you change the NTFS file permissions for this folder, at the minimum you should enable Read access on it and its subfolders. For temporary work, Outlook Web Access uses another subfolder under Exchsrvr, named WebTemp. Make sure that the permission for this folder is set to Change because Outlook Web Access needs to create and delete items in it.

Exchange Server Search Permissions

If you are using the new Search permissions in Exchange Server version 5.5 to restrict access to information in the Exchange Server directory for your users, you need to make sure that the Everyone and Directory Anonymous accounts have Search permissions at the Exchange Site or Configuration container level. If you do not grant these permissions, a user might get an error message stating that the Exchange Server is down or that HTTP access has been disabled. Your CDO applications could fail as well. For more information on this error, be sure to check out the following Knowledge Base articles in MSDN: Q173455 "OWA Returns Exchange Server Down Error Message"; Q175892 "Permissions Required for Outlook Web Access"; Q180417 "Error Msg: Sorry! The Microsoft Exchange Server Is Down."

Installing Outlook 8.03 on Your Outlook Web Access Server

If you install Outlook 8.03 on your web server after installing Outlook Web Access, Outlook will register an older version of the CDO library. Most commonly, users won't be able to access or render calendar information in Outlook Web Access or CDO applications because the older version of the library did not support this. To fix this problem, type the following at the Run command, which is accessed from the Start menu on your Outlook Web Access server: *regsvr32 cdo.dll.*

The Outlook HTML Form Converter

In previous chapters, you learned how to develop Outlook solutions using the features of Outlook, such as forms. These forms, however, work only with Outlook on machines running Microsoft Windows 95 or later version, or Microsoft Windows NT 4.0 or later versions. There are still many 16-bit, UNIX, and Macintosh clients for whom developers need to design collaborative solutions. To provide cross-platform support for forms, Microsoft offers the Outlook HTML Form Converter. This converter allows you to take your Outlook solutions and turn them into HTML, ASP, and CDO-based applications that can be viewed by any standard browser such as Internet Explorer and Netscape Navigator. Once you convert your application to HTML, you can use any standard web development tool—for example, Microsoft FrontPage—to edit the HTML output of the converter. Once you convert the form, your users can work with either the Outlook version of the application or the HTML version of the application.

While this technology is a great step forward for cross-platform collaborative solutions, the HTML environment has some limitations and does not provide the same level of functionality as Outlook. This section describes what the Outlook HTML Form Converter is, how the converter works, and what the web forms library for Outlook Web Access is. I also provide tips for developing Outlook solutions that can be more easily converted to web solutions.

Software Requirements of the Converter

Before you attempt to convert your forms, you must meet a few software requirements. First, you must have Outlook Web Access installed on one of your Internet Information Servers. Installing Outlook Web Access was discussed in the previous section. Second, you must have either Outlook 97 (version 8.03 or later), Outlook 98, or Outlook 2000 installed on the machine on which you are going to convert the forms. Make sure that you install the converter *after* you install one of these versions of Outlook. Third, you need to have Exchange Server 5.5 with Service Pack 1. The service pack includes the Outlook HTML Form Converter as well as some improvements to Outlook Web Access that allow you to view Outlook contacts from any standard web browser. To install the Outlook HTML Form Converter, run Fcsetup.exe in the Formscnv folder of the Service Pack 1 CD for Exchange 5.5. Finally, on the client, users of converted forms can use any version of Outlook or no version of Outlook— that is, they don't even need to have Outlook. They need only a web browser to use the forms.

Components of the Converter

The Outlook HTML Form Converter's architecture consists of a number of components:

- *Conversion wizard.* This is the user interface that walks you through converting the form.
- *OFT-HTML COM object.* This reads the layout and data-binding information from the Outlook form and writes the corresponding HTML code to one or more files on the web server.
- *Form Converter templates.* These templates are used as base templates for the converted file.
- *Template processor object.* This object customizes the base templates to create the converted form.

Features of the Converter

Before stepping through the actual Outlook HTML Form Converter, you should know about its features. The Form Converter does provide a large feature set that you can take advantage of, but it also has some limitations.

Form Locations

The Form Converter allows you to convert forms from multiple forms libraries as well as forms from the file system that are saved as .oft files. The types of forms libraries that the Form Converter supports are the Personal Forms Library, the Organizational Forms Library, and the Folder Forms Library. Using the Form Converter wizard, you can specify either the forms library or the specific .oft files you want to use. The Form Converter also supports selecting multiple forms for simultaneous conversion.

Form Types

The Form Converter currently supports forms based on the following types:

- *IPM.Note.* Mail message
- *IPM.Post.* Post form
- *IPM.Contact.* Contact form

IPM.Task, IPM.Appointment, and IPM.Activity (journal entry) are not supported by the Form Converter, but if you need to convert the user interface for any of them, you can copy their controls to a supported form type. You can

then convert the supported form type with the copied controls and customize the converted form using an HTML development tool. If you try to convert an unsupported form type, the Form Converter will display an error message.

Convertible Features

The Form Converter can convert many of the Outlook controls with their corresponding layouts. Following is a list of features that the Form Converter can convert to HTML. Limitations are described.

- *Label control.*
- *TextBox control.*
- *ComboBox control.* If the Outlook form contains an editable ComboBox control, the Form Converter changes it to a noneditable ComboBox.
- *ListBox control.*
- *CheckBox control.*
- *OptionButton control.*
- *Frame control.*
- *CommandButton control.* Any images placed on the CommandButton control are lost since HTML does not support images on buttons.
- *MultiPage control.*
- *Image control.* Images that are bitmaps are converted to GIF files automatically by the Form Converter. In addition, any images that are clipped in Outlook by the Image control are shrunk automatically by the Form Converter for the HTML version of the form.
- *Background images on a form.*
- *ActiveX controls.* The Form Converter adds a commented out Object tag to the HTML form for the ActiveX control. However, the Form Converter does not package the ActiveX control as a CAB file nor does it add a CodeBase statement to the Object tag to point to the control's CAB file. To make the control appear on the form, you can package the control, add the CodeBase statement, and then remove the Form Converter–generated comments.
- *Initial values.*
- *Required fields.* If a user attempts to change a tab in the HTML version of the form, the form will display a warning message that one

of the fields is required on the form and also display the identifier text of the field. This text might appear in a format like SUBJECT_41_0_G, which might not be meaningful to your users. You can modify the error code to make it display the friendly name of the control rather than the identifier text.

- *Type checking and formatting.*
- *Read and compose layout.*
- *Hidden controls.* In most cases, the initial values of hidden controls are not maintained.
- *Built-in and custom actions.* If your actions call a custom form, be sure to also convert these forms, or the HTML version will return an error when the user invokes the custom action. The HTML version of the form can use only two rows of buttons to invoke custom actions. Since the width of the button is based on the amount of text on the button, and you're allowed only two rows of buttons, keep the length of the names of custom actions to a minimum.
- *Limited support for non-English forms.* The Form Converter provides limited support for non-English forms. It generates the ASP files and places them on the client machine in the correct subfolder for the language. For example, the output of a German form will be placed under the GER folder in Outlook Web Access, not under the USA folder. The Form Converter also places in the Form.ini file the appropriate code page for the language in which the form must be rendered. You must have installed on Outlook Web Access the language pack for the character set of the form you want to convert before running the Form Converter wizard. If you do not, you will not get the international options in the wizard.

Unconvertible Features

Following is a list of features that are not supported by the Form Converter. Details are provided for a few of these.

- *ScrollBar control.*
- *SpinButton control.*
- *TabStrip control.*
- *ToggleButton control.*

- *Formulas.* Even though the Form Converter does not convert formulas, it places the code for the formulas in commented out text in the HTML file. You can then uncomment and modify the formulas according to the needs of your application.
- *Script code.* The VBScript behind an Outlook form is not converted. Instead, it is placed in a text file, named Script.txt, which is in the folder that contains the ASP files for the converted form. The reason VBScript is not included in the HTML form is to accommodate cross-browser support. Since Netscape browsers do not support VBScript, you can either change the script in your form to server-side VBScript or rewrite the script as client-side JavaScript.
- *Overlapped controls.* Since HTML does a poor job of supporting overlapped controls and layouts, the Form Converter does not convert overlapped controls. Instead, it places the controls as close as possible to one another on the form.
- *Calculated fields.* In a Contact form in Outlook, there are calculated fields such as FullName whose values are derived from other fields, such as FirstName and LastName. The Form Converter will convert forms that contain calculated fields, but these fields will become static fields. For example, the FullName field will not automatically change in an HTML form when either the FirstName or LastName field is changed.

Stepping Through a Conversion

Before you attempt to convert a form, you must first share the Webdata folder on your Outlook Web Access web server. For a default installation of Outlook Web Access, this folder is located at C:\exchsrvr\Webdata. You must give yourself and other developers in your organization who will use the Form Converter at least Read and Write access to the share. Also, be sure to name the share *Webdata*. If you do not share this folder, the Form Converter will not allow you to finish the wizard.

To start the Form Converter, click the Start button, point to Programs, and select Microsoft Outlook HTML Form Converter. This will display a starting screen for the Form Converter. Click Next to begin the conversion process.

Selecting a Form Location

On the second screen of the Form Converter, shown in Figure 8-7, you can select the type of form you want to convert. As mentioned earlier, you can select forms from the Personal Forms Library, the Organizational Forms Library, and the

Folder Forms Library or Outlook templates from the file system. On the second screen, you also specify the name of the Outlook Web Access Server where the ASP file will be placed after the conversion.

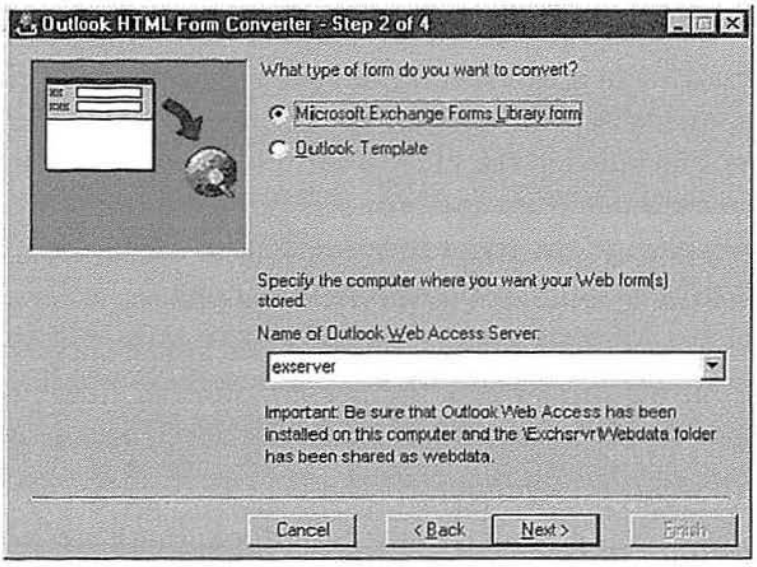


Figure 8-7
Selecting the type of form and specifying the Outlook Web Access server in the Form Converter wizard.

Selecting Specific Forms

Clicking Next might display a Choose Profile dialog box. If so, choose a profile or create a new profile, and click OK. Depending on the type of form you selected in the second screen, the Form Converter wizard will present you with a forms library view, shown in Figure 8-8, or the Open Outlook Template dialog box, shown in Figure 8-9. You can select multiple forms from either of these interfaces. The forms library view enables you to display form categories rather than form names in the Outlook Forms list box. For forms in which these category properties were specified, this can make finding forms easier.

Choosing Conversion Options

In the final step of the conversion process, shown in Figure 8-10 on page 244, you can choose how you want the forms converted. If there are international language packs installed on the Outlook Web Access server, this screen also provides a drop-down list from which you can select the language for the converted form. This final step of the wizard also gives you the option of always overwriting your existing form. If you do not check this option, the Form Converter will prompt you during the conversion about overwriting the existing form. Checking the

Layout Debug Mode check box enables debug mode on the converted form. If debug mode is enabled, the table borders for the HTML version of the form will be made visible so that you can easily see where and how the tables are laid out. You can use the borders to adjust the size and placement of controls on the converted form.

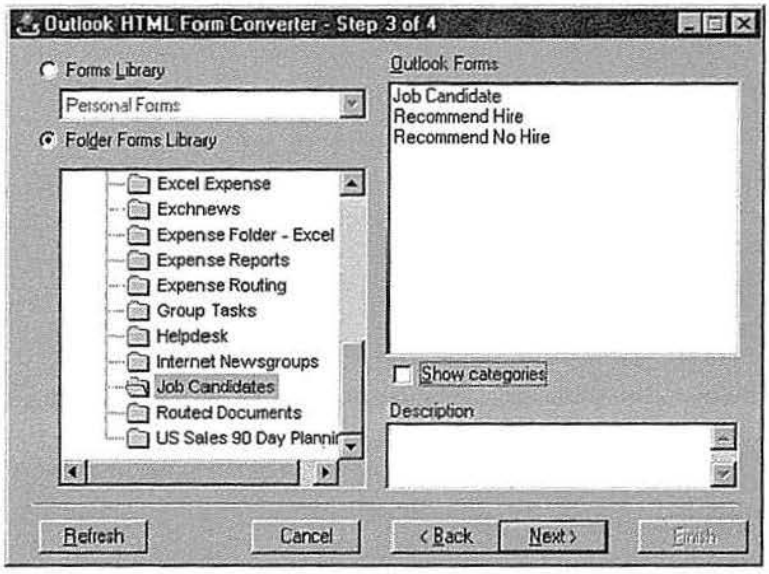


Figure 8-8

The Forms Library view of the Form Converter wizard. You can view your forms by category or by display name.

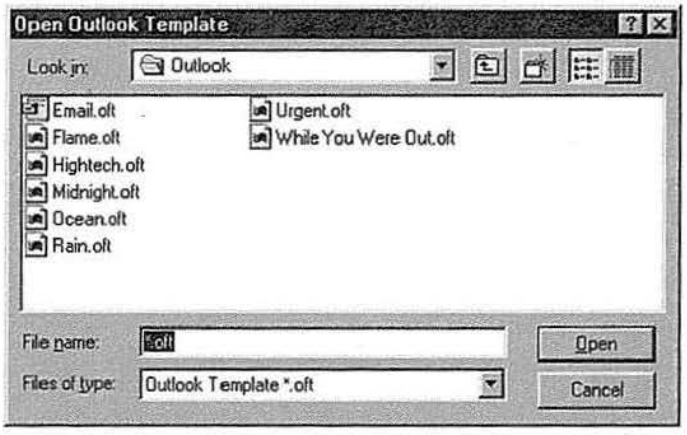


Figure 8-9

The Open Outlook Template dialog box presented by the Form Converter wizard. You can select multiple forms to convert from the dialog box by using the Ctrl key.

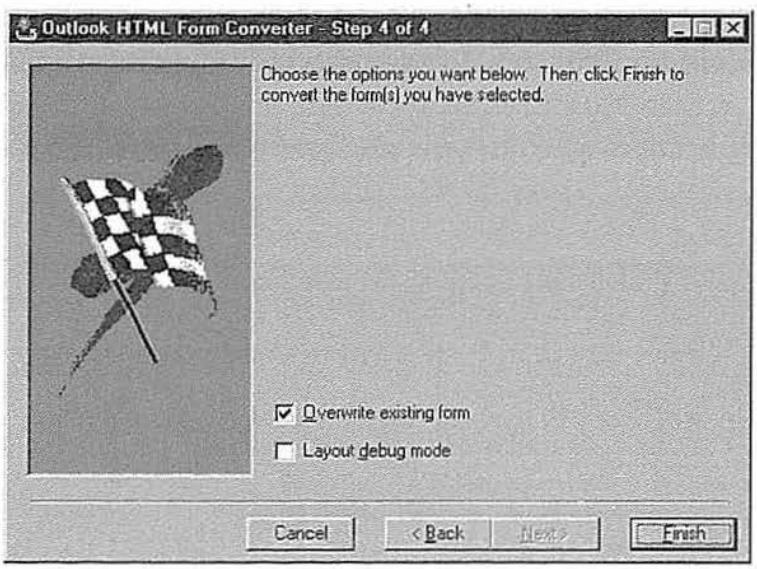


Figure 8-10
On this page, you choose options for conversion: you can overwrite your existing forms or enable the debug mode for your converted forms.

Results of the Conversion Process

When you click the Finish button, the wizard converts your form. At the end of the conversion, the wizard displays its results. There can be three results:

- *Successful conversion.* This result means that the form was converted and the Form Converter has no suggestions for improving the layout and functionality of the form through post-conversion edits. A successful conversion is shown in Figure 8-11.

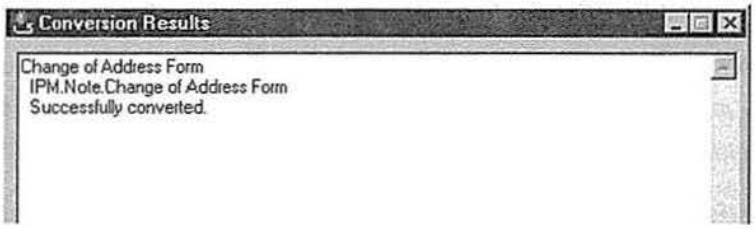


Figure 8-11
The Form Converter lets you know when you've converted a form successfully.

- *Successful conversion with To-Do list.* This result means that although the form was converted successfully overall, the Form Converter describes some post-conversion enhancements that you can make or

some functionality on the form that did not convert. The Form Converter creates a text version of the To-Do list it displays, named ToDo.txt, and copies it into the same folder it places the ASP files for the converted form. Figure 8-12 shows a successful conversion with a To-Do list.

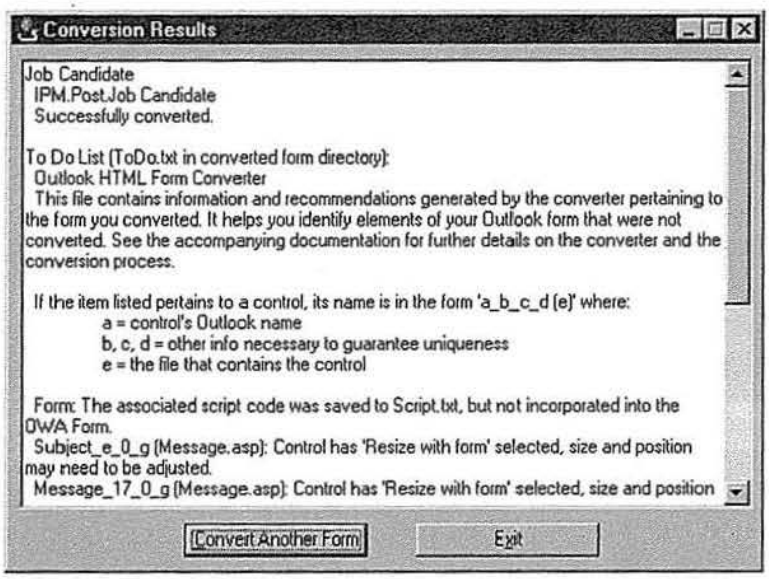


Figure 8-12
A successful conversion with a To-Do list offers suggestions.

- *No Conversion.* This result means that the Form Converter could not convert the specified form. This unsuccessful conversion could be due to a number of issues, the most common being that the form you are trying to convert does not fall into one of the three supported form types. Figure 8-13 shows an example of an unsuccessful attempt to convert a Task form from an .oft file.

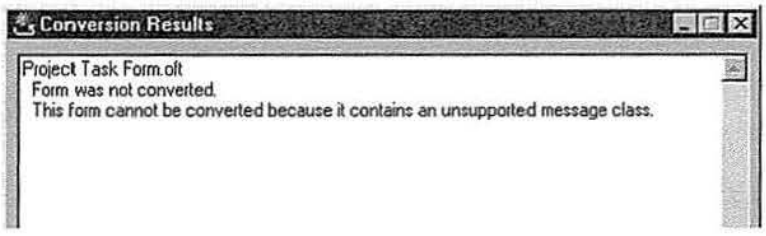


Figure 8-13
Attempting to convert a task form results in the Form Converter returning a No Conversion result.

Viewing the Results

The most common way to view the results of your conversion is to launch your web browser and type *http://OWAServer/exchange/* for the URL, replacing *OWAServer* with your Outlook Web Access server name. When the Outlook Web Access Log On page is displayed, log on. Select Custom Form from the Compose New drop-down list as shown in Figure 8-14, and click the Compose New link.

Figure 8-14
Selecting Custom Form from the Compose New drop-down list in Outlook Web Access to view forms converted with the HTML Form Converter.

After you select Custom Form and click the Compose New link, the Launch Custom Forms window is displayed, as shown in Figure 8-15. This window lists the custom forms in your web forms library; the forms you converted using the Form Converter will be listed here. Click the link of the custom form that you want to test.

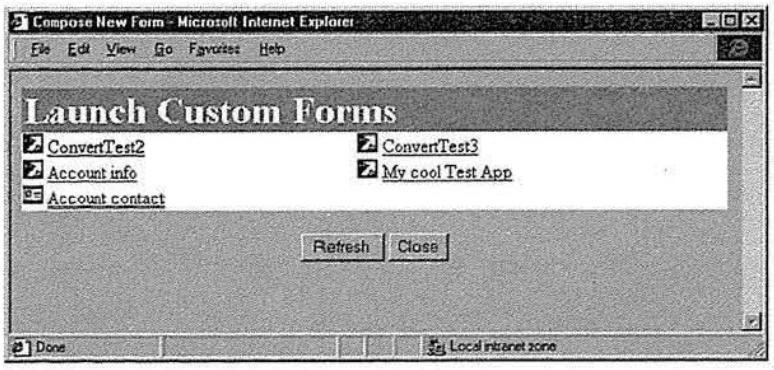


Figure 8-15
The Launch Custom Forms window lists the forms in your web forms library.

Examples of Conversions

Now let's take a look at a few figures showing Outlook forms before and after being run through the Form Converter. Some of the figures contain the limitations mentioned earlier.

Figure 8-16 shows the converted Compose form for the Account Tracking application discussed in Chapter 7. The Account Tracking application was not designed with conversion to the web in mind; therefore, a large amount of dynamic user-interface script was generated for the form. Typically, when forms use a lot of script to change user interface elements (for example, when you dynamically disable controls on your form based on the values a user types in another control), you will need to manually code many of the changes on the converted HTML form.

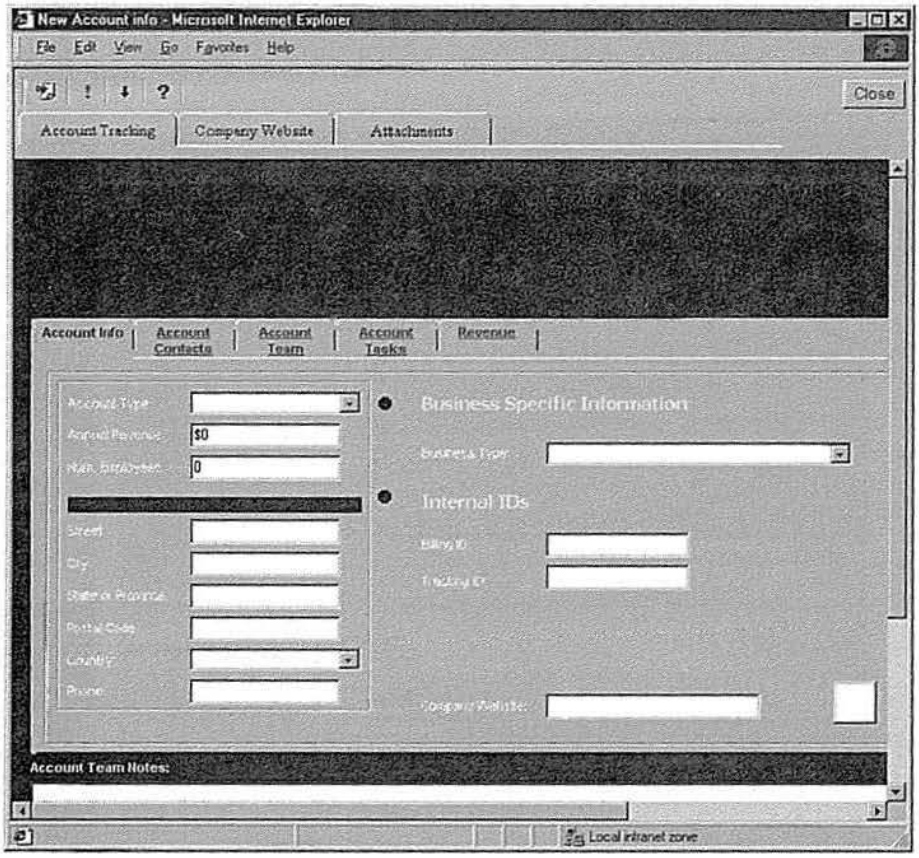


Figure 8-16
The Compose form for the Account Tracking application. Notice how the picture does not come across on the CommandButton control.

Figure 8-17 on the next page shows the Read form for the Account Tracking application. The form's custom actions display as buttons in the HTML version. Also notice that the Form Converter automatically converts the Read form

for your Outlook application. The CDO rendering library, which you will learn about in Chapter 11, also automatically recognizes that the user is reading information for an application and thus launches the Read form when that user clicks on data in your application. All the binding of data in the form is generated by the Form Converter. You should be aware, however, that the HTML version of this form does not have the full functionality of the Outlook version. For example, the HTML version cannot display the embedded Tasks or Contacts for the account because these are populated by using VBScript in the Outlook form.

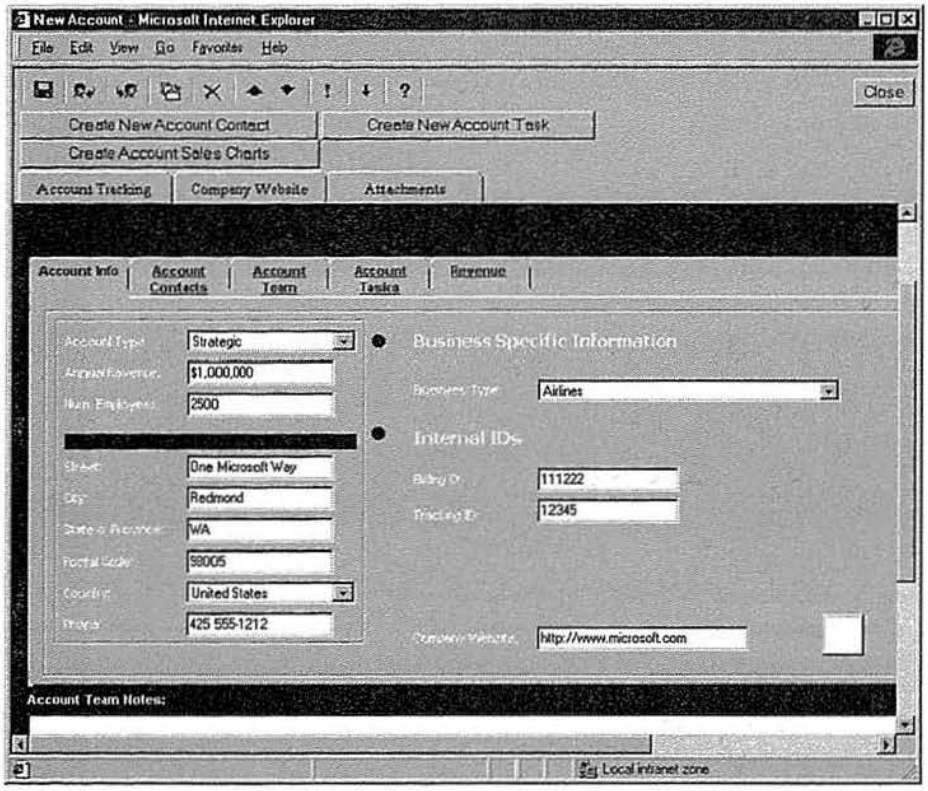


Figure 8-17
The Read form for the Account Tracking application. Notice that the custom actions come across as buttons on the converted form.

Figure 8-18 shows a helpdesk application before conversion to HTML, and Figure 8-19 shows the application after conversion. Notice the Opened By text box is automatically filled for the Outlook version but not filled for the web version. However, the web version does have text boxes where the default values are specified.

Figure 8-20 and Figure 8-21 on page 250 show the before and after versions, respectively, of another helpdesk application. The Form Converter automatically converts the rocks bitmap to a GIF.

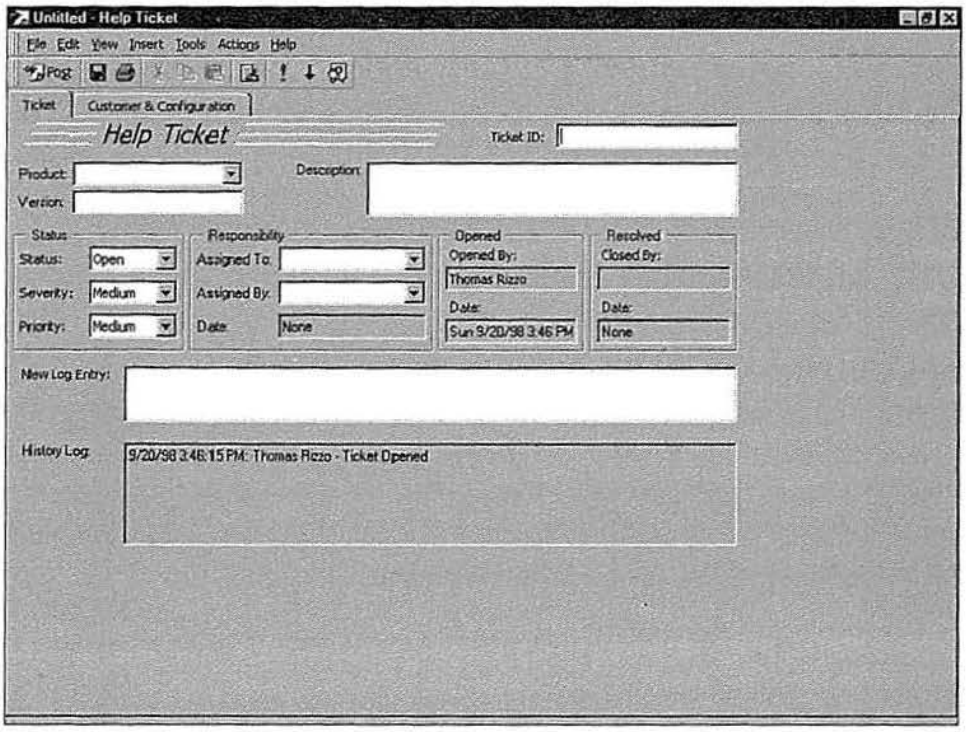


Figure 8-18
The Outlook version of a helpdesk application.

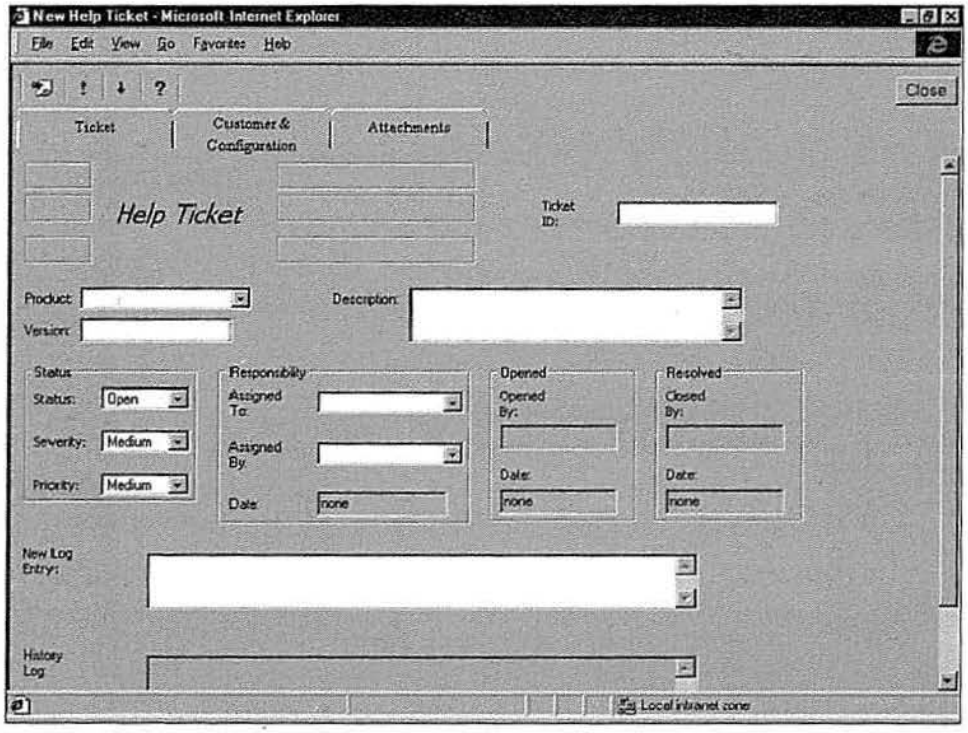


Figure 8-19
The web version of a helpdesk application.

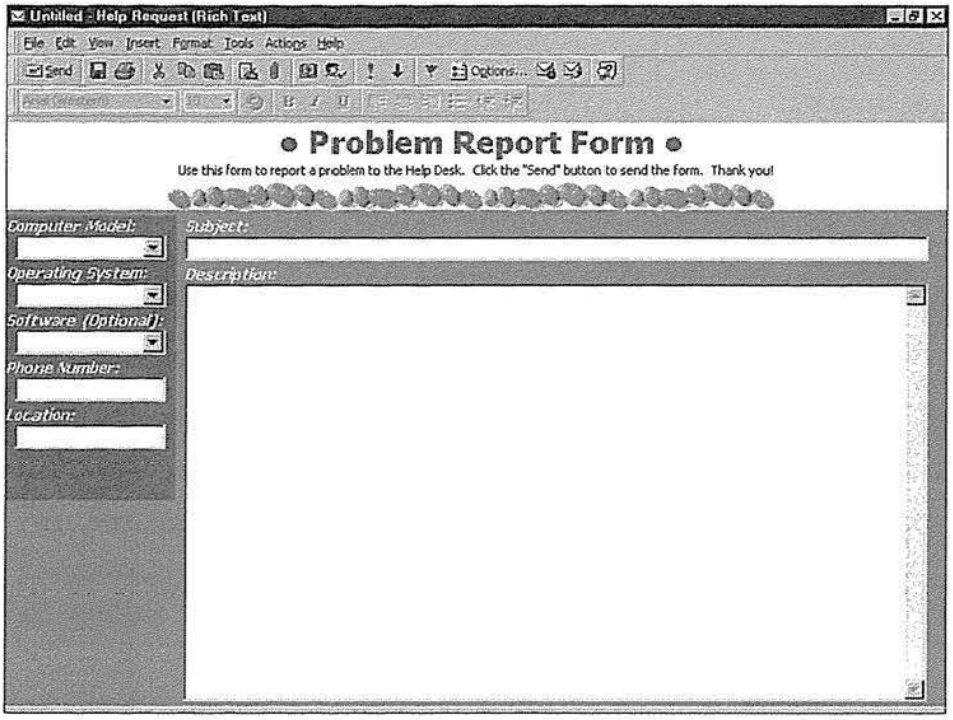


Figure 8-20
The Outlook version of another helpdesk application.

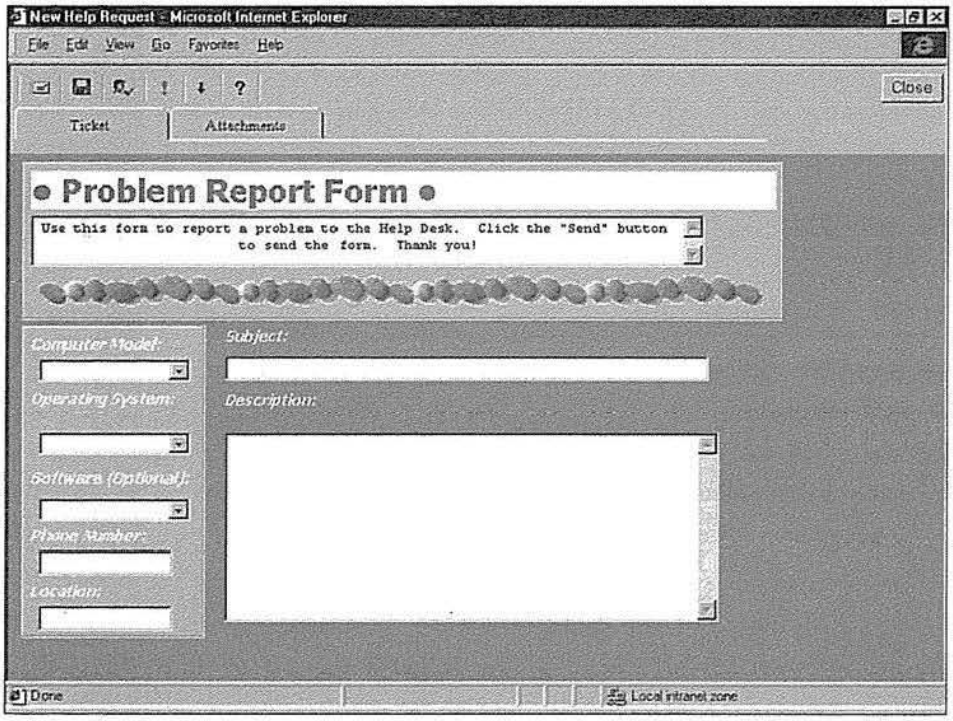


Figure 8-21
The web version of the helpdesk application in Figure 8-20.

Files Created for Converted Forms

Forms converted by the Outlook HTML Form Converter consist of a number of files that you can customize. These files are placed in the web forms library, which we will discuss in the next section. Let's look at the set of HTML and ASP files that comprise the core architecture of the Form Converter.

FrmRoot.asp

This file is the entry point of the converted or custom form. For the Form Converter, this file includes script and an HTML frameset that displays the other components of the form. This file is discussed more in the next section on the web forms library.

Posttitl.asp

This file is the frame of the form, located near the top of the screen. This frame contains the toolbar and the tab strip. The main purpose of this file is to handle form commands generated by clicking the toolbar buttons and the tab strip.

Page_N.asp and Page_N-Read.asp

Every page in a converted form is represented by a separate .asp file. Certain pages of a form have predefined names—for example, the Options tab is named Options.asp. Custom pages of a form are assigned system-generated names, such as Page_3.asp. These custom form pages are generated by the Form Converter. If you created a separate compose and read layout for a particular form page, its file will have two different versions: Page_N.asp for composing the custom form, and Page_N-Read.asp for reading the custom form.

Commands.asp

Commands.asp is used to implement utility functions and event handlers for the converted form. Commands.asp is never seen by the user but rather is a hidden .asp file that is called to handle functions such as standard actions (*On_Send*, *On_Reply*, and so on) and also custom action event handlers.

Form.ini

The Form Converter automatically creates and publishes a Form.ini file for your application. This file contains the form's display name and code page, and indicates whether the form should be hidden in the Launch Custom Forms window. This file is discussed more in the next section on the web forms library.