

17 Extending the Platform

After completing this chapter, you will be able to:

- ✓ List the major features of the Web Storage System
- ✓ Describe Exchange 2000 as a Web service provider
- ✓ Name the requisite services of any repository-based enterprise application
- ✓ Describe the process flow of an Outlook Web Access (OWA) service request
- ✓ List the steps involved in displaying an object through OWA
- ✓ Discuss how browser versions affect the OWA interface
- ✓ Describe the process flow in accessing an Outlook Inbox using Web Distributed Authoring and Versioning (WebDAV)
- ✓ List the advantages of front-end/back-end (FE/BE) configurations
- ✓ Differentiate between generalized and specific markup languages
- ✓ Distinguish features of Hypertext Markup Language (HTML) and Extensible Markup Language (XML)
- ✓ Name the specific components that enhance Internet Explorer 5 (IE5)
- ✓ Discuss the benefits of using Web Forms
- ✓ Discuss the Universal Data Access architecture
- ✓ Outline differences between ActiveX Data Objects (ADOs) and Object Linking and Embedding for Databases (OLE DB)
- ✓ Describe the Microsoft.NET framework architecture

This chapter concludes a long discussion of Exchange 2000. We began this book by describing a paradigm shift in the field of information technology (IT) from data processing in the late 1960s through information sharing in the 1970s and 1980s to knowledge management (KM). Microsoft has used several catch phrases to describe this theme. The one most relevant to this chapter is information “anytime, any place, from any device.” This chapter shows how Exchange 2000 provides the foundation for that KM. We begin our chapter by discussing the Web Storage System. We show throughout the chapter how a .NET Server like Exchange 2000 with inherent multitier features—such as front-end and back-end server topology, partitioned services, and extensible Web Store—has been designed to easily provide Web-based service. We continue with a discussion of Outlook Web Access (OWA), the Web Distributed Authoring and Versioning (WebDAV) and Hypertext Transport Protocol (HTTP) protocols, and front-end/back-end (FE/BE) servers. The combination of standard browser client, OWA, and Web Store meets the Microsoft objectives of “information from anywhere, at any time, using any device.” Exchange 2000 is positioned as the information broker in this scenario, much like how Lotus positioned its premier product, Notes/Domino—a field-tested application platform that provides communication, collaboration, and programmatic control—in the 1990s.

Next, we discuss markup languages, including Hypertext Markup Language (HTML) and Extensible Markup Language (XML). Then, we cover Internet Explorer 5 (IE5), which is becoming an alternative client for Exchange Server and the Web. It has undergone substantial enhancements that support Internet protocols, including its XML engine. This browser client will play an increasing role in future Web-based server management, in addition to its data-entry role through application interfaces like Web Forms. We then discuss various Web solutions. Next, we discuss various application programming interfaces (APIs) such as Object Linking and Embedding for Databases (OLE DB) and its more accessible subset, ActiveX Data Objects (ADOs), from the perspective of the Microsoft .NET framework. These programming interfaces have provided access to many objects stored within the messaging and collaborative structure of Exchange. We continue our discussion with these specific Collaborative Data Objects (CDOs) and how Exchange 2000 has enhanced their functionality. We conclude with a discussion of the possible role of Exchange 2000 in a distributed services environment and vision of its role as a .NET framework.

Web Storage System

The Web Storage System is the cornerstone on which Exchange 2000, a messaging and collaboration system, and Windows 2000, a provider of directory services and network operating system (NOS) core services like security and auditing, have come to rest. The Web Storage System schema defines the formats of the objects that are stored in the Web Store and integrates a wide, extensible range of knowledge

sources into a central repository. Exchange 2000 provides an interface to its contents through intranet clients like Outlook and Internet clients like a standard browser and OWA. To simplify our discussion, we use the terms Web Store and Web Storage System interchangeably.

As a consumer of information services, you request services from either a broker like Exchange or directly from the Web Store. These service providers are extensible, so you can define a set of properties for a specific item and then control how it is used. The property that defines intention or purpose is called the *content class*. What makes the Web Store significant is a design that provides access to every item within its namespace. The ability to locate those items as resources in a systematic and human-readable way is provided by HTTP through the now common Uniform Resource Locator (URL). The concept of the Web Store is to provide a namespace of all collaborative objects using an HTTP URL and a standard Web browser. Thus, consumers/end users can directly access folders or collaborative objects like calendars by appending a named directory or object to the end of the URL used to access their mailbox. Named URLs also allow users to perform operations on these messaging and collaborative objects through explicit URL addressing.

Microsoft suggests that the Web Store is a scalable infrastructure within which we can consolidate all kinds of information, including messaging, collaborative information, file systems, and Web-related data. The Web Store is one of several supporting legs on which Microsoft is building a platform for its vision of “knowledge workers without limits.” The integration of knowledge sources is found in one repository that provides the following key services to any enterprise-based application:

- *File system services*—To integrate structured data (such as databases with unstructured data like streaming data from the Web and information stored in proprietary Microsoft Office documents), a data model must accommodate both hierarchical and heterogeneous collections of data.
- *Database services*—Services must accommodate an extensible assortment of data types. A data query language must be capable of more complex searches than those typically performed within a file system on a Windows platform. Support issues here include the updating process of viewed data, where applications maintain consistent and accurate views of all items.
- *Collaboration services*—These services now extend beyond simple messaging, the collection of contacts, and calendaring objects to realtime collaboration that involves audio and video information.

The file system and database services require an extensible way to identify and manage content. The manners in which these services are used also require tremendous flexibility. Creation, manipulation, and deletion of isolated data, or larger knowledge objects like group calendars and task lists, will occur spontaneously and

without previous preparation or planning. Although you do not have to define a property as a file system or database object before you use the Web Store, you can ensure rapid execution of search queries to retrieve it from the data store once it is placed there by creating some definition of its nature or content. All three systems must be able to support such spontaneous activity. This storage system supposedly removes barriers that obstruct the sharing of heterogeneous data by combining the features of these three service areas as they relate to Exchange 2000 and other Microsoft server products, as well as Microsoft Office 2000 products. Microsoft views the Web Store as an enterprise-based application service that provides ubiquitous client access. The benchmarks for success are improved productivity and a lower total cost of ownership (TCO).

Extending Data Access

The Web Store is the focus for many of the technologies discussed in this chapter because it is actually the container for both structured and unstructured data. The Store now has the potential of providing a central repository for messages, their attachments, collaborative objects like calendars and task lists, and the heaps of data stored in the various Microsoft Office application formats like Excel's .xls, Word's .doc, PowerPoint's PPT, and Access's MDB files. To accommodate the greater number of categories, we need to extend the transport protocol to move these objects between the service provider and the consumer requesting the service. This extension must provide support for a variety of document formats, such as rich HTML and other Web-based scripting languages, Win32 file structures, streaming file formats, offline folder objects, and native collaborative objects. Database management plays a major role in the extension of these services.

Rich HTML Support

Especially with the introduction of IE5, but beginning with IE4, Dynamic HTML (DHTML) support of scripting and robust Document Object Models (DOMs) has added richness to the standards-based HTML. A DOM is a standardized API that provides programmatic control over a document's content, structure, formats, and processes. The Web Store leverages these native features in the more current versions of the browser client. Legacy browser versions 3.x are also supported with functionality provided by increased server-side support. HTML, IE5, and DHTML are discussed in more detail later in this chapter.

Scripting Support

Web-based applications are collections of distributed components and services. Like their standalone counterpart, component-based software, distributed applications require rich methods and properties to manage the many transactions and service requests passed to other services providers across the stateless environment of the Internet. Solution providers must compensate as they transition from the client-side

richness of DHTML to deficiencies inherent in the simple scripting of HTML/XML pages. Solution providers do not necessarily encounter this change in feature sets when they move Win32 component applications from one hardware platform to another.

The advantages of porting component-based application software, however, do not outweigh the expense involved in scaling services or extending functionality. Applications using distributed services are alternatively easier to scale and deploy than their component-based counterparts. The Web Store supports the development of Web-based applications by hosting technologies like Active Server Pages (ASP), discussed later in this chapter. Scripting support of this technology, however, is not sufficient to deliver the quality of productivity tools available when you use Win32 component architecture. Thus, the Web Store supports the same database interfaces that Microsoft ADO scripts use to deliver the common database-intensive software application. The Web Store's OLE DB 2.5 provider is a complex foundation upon which ADO, a subset of OLE DB scripting, runs. Thus, database services—including data query and data manipulation of language components, as well as transaction support—are available. The Web Store also includes full-text content indexing. We discuss ADO and OLE DB in more detail later in this chapter.

Microsoft Win32 Support

The Web Store must support extensible Web-based data types as well as legacy (Win16) file types and document models typically found in the network-based Win32 interface. It leverages resources in existing file systems by supporting an interface that virtualizes the NetBIOS (Win32) share name. You can utilize folder-based resources in the Web Store environment, providing access to unstructured data that heretofore was inaccessible to Web-based manipulations. It is significant especially in terms of TCO that end users accustomed to sharing folders will not notice any differences in handling the data through their browser client across the Internet as compared with Windows Explorer in a LAN environment. There is no need for end-user support or software configuration when you access these resources.

Another important characteristic of the Web Store is its ability to provide property promotion, especially when dealing with Office file types stored in the Win32 file system. The Web Store provides consistent views of the information independent of the client used to access that information, which again minimizes the need for end-user training. Properties from documents are automatically promoted and populate all record fields used in the viewing interface. Thus, for example, the Outlook Inbox displays all file types as common entities in a unified view of information consistent across multiple clients. This universal Inbox integrates information, eliminates concurrent access to multiple applications, and enhances productivity by providing a customizable view of data that can be replicated on demand.

Streaming Store

Just as Exchange 2000 has accommodated new types of multimedia messaging with audio and video components, the Web Store can natively accommodate very large message units with attachments. The important streaming store feature is the file streaming interfaces that provide the retrieval access. When so many storage types are involved, it is especially important to be able to ensure that the intended meaning of data is retained. To maintain the quality of information, the Web Store minimizes the number of file type conversions; the data is natively stored in the Web Store.

Offline Folders

The Web Store allows you to integrate online/offline data stores so that you can support the needs of a user population that is more often remote than on site. Beginning with the Briefcase object in Windows 95, synchronizing online folders and documents with offline stores has become a necessity among users who tend to find themselves accessing network resources from portable platforms. The shift toward Web-based access has freed users from the relatively synchronous connections that a corporate workstation provides. Although the need to explicitly synchronize folders and their contents is critical to version control systems, end users often overlook this. With offline folders available in Windows 2000, the Web Store synchronizes the folder contents regardless of how the data is accessed. This replication model provides support without an increase in TCO because it is transparent to the end user and therefore requires no additional training.

Native Collaboration Objects

The Web Store allows you to automatically integrate what might be more appropriately termed *knowledge objects* with collaborative functions like messaging, contacts, calendaring, and workflow. Thus, a financial report in the form of an Excel workbook can be scheduled for review on a specific view. This integrative process is achieved through an object model that leverages on-demand and programmatic access to data objects. A knowledge object is not the same as a data object; a knowledge object is truly an entity that the consumer or end user manipulates independent of the Web Store or Exchange. The ability your users have to shape their working environment according to individual needs with personalized knowledge objects like calendars or task lists is an example of the difference between data processing and KM. Data access is standardized using well-known APIs such as ADO and OLE DB, so development time is minimized. In addition, popular authoring systems like Microsoft FrontPage further expand access to the collaborative resources that the collaborative platform provides.

Database Management

To deliver database services, the Web Store must perform very fast searches and lookups. The native database manages index documents for common key fields as

well as attachments. The Web Store offers transaction logging at the database level. It uses write-ahead transaction logs like Exchange 2000 to ensure data integrity through redundancy. Data replication is supported at the folder level so that replicas of data can exist on multiple servers in the Exchange organization. Folder replica management also balances application loads, especially when you are dealing with data that is frequently accessed. Using multiple databases is also another way to improve reliability. Finally, at the system level, the Web Store supports Windows 2000 active/active clustering. Web Store services on one machine can assume control over data managed by the Web Store services on another machine if hardware malfunctions. Active/active clustering minimizes downtime and, when actually implemented, is totally transparent to the consumer/end user.

Web Store Events

Exchange 2000 allows you to programmatically respond to store events and build the rules to make decisions. It can respond to events that involve data objects such as adds, moves, and changes, as well as the starting and stopping of services. Developers can create applications or scripts that program the responses to these events. When a specific event occurs in the Web Store, the Exchange system fires an event. You can program the System Services layer to notify other applications or services in response to that event. These applications that respond to a specific event are called *sinks*. In addition to allowing you to define your own event sink, Exchange uses predefined sinks for numerous tasks. Thus, a program that monitors a workflow can initiate specific processes in response to the system's conditions or states and programmatically redirect that workflow without supervision or outside control. Alternatively, event sinks can validate items when they are saved to the Web Store. Other event sinks can be embedded or launched from scripts written for a specific application.

Web Storage System events are the responses the Web Store provides to service requests from other service providers between the Data Services layer and the Application Logic layer; the consumer/end user does not directly interface with this architecture.

With improvements in hardware capacity and response times, enhancements in scripting language, a broader use of APIs (discussed later in this chapter), and the ability to offload more processing to the client side of the transaction, the event architecture in Exchange 2000 has changed from a legacy client-host, component-oriented architecture to multitier, distributed systems. There are three application types:

- *Desktop*—This application model is monolithic; Presentation, Application Logic, and Data Services layers are all located on the computer.
- *Two-tier*—This model is the classic client-server model where a workstation hosts the Presentation layer and data services are located on the host/server. Business or application logic can be located on the client, the host, or both machines.

- *Three-tier (or multitier)*—Traditionally, the multitier approach has been conceptualized as three distinct service layers. In a three-tier architecture, presentation, business logic, and data elements are conceptually separated. Presentation elements are what the user manipulates or uses to make service requests. These elements thus handle requests for services developed by the application logic, which out of necessity is located on the server side. The business logic (or rules) provides whatever services are requested, either directly or through the contribution of another service provider. The data layer provides the repository structure.

If, for the purposes of this discussion, we divide multitier architecture into Presentation, Business (Application) Logic, and Data Services layers, we see a design that is Web and service oriented. The fundamental layer in a multitier architecture is the Data Services layer, where resources are stored and retrieved. The technologies now available through Exchange 2000 and other application servers have shifted the IT paradigm from data processing through information services to the management of knowledge objects in highly collaborative environments. Thus, the sophistication of the Presentation layer in terms of specifying extensible types of objects must grow in tandem with the capacities of the Data Services layer, where these objects are stored, catalogued, and retrieved.

Web Services

Microsoft has defined a Web service in technical literature as an application delivered as a service that is integrated with other Web-based services using Internet standards. This means that you can access resources using standard URL addressing. A Web service, running transparently and leveraging other services to minimize its own overhead, should provide some value that theoretically is not available from any other source. In Chapter 1, one model we described—Generic Architecture for Information Availability (GAIA)—is of particular interest because it provides a terminology well suited for a distributed services environment. GAIA describes roles and services involved in brokering information, which changes the administrative role in maintaining Exchange 2000 Server from simply “fixing pipes” to custom tailoring collaborative solutions.

We have slightly reworked the GAIA model so that we refer to consumers, brokers, and service providers to accommodate the distributed services environment found in a multitier, e-commerce architecture. We use these roles at many functional levels of exchanging information and managing knowledge. If you approach Exchange Server as a conceptual “knowledge broker,” you will be better prepared to manage the many services it provides, from messaging to realtime collaboration. Our emphasis on providing solutions rather than fixing “broken pipes” echoes Microsoft’s test objectives; in the Windows 2000 environment and especially dealing with Exchange 2000, your job description is more of “architect” than

“plumber.” Given the functional role models of broker and service provider, you can, as administrator, successfully leverage the accessibility that OWA provides with the rich feature set of commands available from extensions to the HTTP transport protocol. These extensions have been added through WebDAV. From a structural viewpoint, the careful planning of topology through the use of front-end and back-end servers (FE/BE) adds reliability through redundancy, and optimized performance through service load balancing. The ability to provide solutions of even greater specificity through, for example, Web-based Forms using Web-based scripting languages (discussed later in this chapter), significantly changes the Exchange administrator’s potential scope of activities and job description.

OWA Architecture

OWA has shipped with Exchange for quite some time. It uses ASP technology to render the contents of a user’s mailbox in HTML. Users log in from anywhere using a standard browser with a well-known interface to read their mail. OWA provides an access method that accommodates Web-based, roving users. The standardized browser client and simplified configuration are critical issues in providing a kind of universal access to messaging and collaborative services, in addition to the Web stores. For many years, Microsoft has considered its Exchange client as a universal Inbox. Now with changes in the API and greater exposure of stored objects through OWA, digital dashboard (providing a ready-made framework within which knowledge workers can customize their selection of displayed data and services) has been substituted for the Inbox moniker. Web services offered through OWA—provided especially by IE5 with its HTML and XML enhancements—can expose more than just mailboxes.

In general, OWA provides many functional benefits, including support for:

- *Light messaging*—OWA provides an alternative to the full Outlook 2000 client through the well-known and readily accessible standard browser client. Especially over public networks and where control over configuration is not practical, this form of access to messaging services provides a practical solution to communication needs.
- *Roving users*—With OWA, you minimize the need to configure a client, and you don’t need Messaging API (MAPI) profiles to configure the appropriate workspace for a workstation. Therefore, you can administer remote or roving users using more general system policies and standard profiles than you can with network users. MAPI is discussed in more detail later in this chapter.
- *Kiosk and Information Sources*—This new service outlet provides access to mailboxes and public information with little maintenance or overhead.
- *Migrations and rapid deployment*—OWA provides continuity of services during interim periods of migration, or loss of services during maintenance periods or deployment.

Requesting and Delivering Services

OWA in Exchange 2000 does not use MAPI to access the mailbox store, nor does it use ASP technology for client access. You access OWA via HTTP. We can conceive of the OWA interface as a brokering service or proxy for all messaging traffic directed toward messaging and CDOs. When the OWA client requests services, Internet Information Services (IIS) acts as a proxy for the messaging traffic between the Web client and Exchange Server. IIS acts as a broker by accepting the client requests that use the WebDAV extensions to HTTP (discussed later in this chapter). IIS passes the requests to the OWA Internet Information Services API (ISAPI) layer, which either directly accesses a Database layer or proxies the request to a remote server that provides this service. ISAPI, the API for IIS, then locates the requested object's URL and redirects the client to a script that provides the logic to respond to the service request associated with a target object.

The request and delivery of services are described in the following process flow:

1. A user enters a URL such as **http://servername/exchange/mailbox** in a browser Address box.
2. The Web server authenticates the user and determines the user's Windows 2000 account by displaying a pop-up Secure Attention Sequence (SAS) dialog box that requests the username, password, and domain, just like when a user attempts to access network resources on a local workstation.
3. The location of the mailbox is determined from a query to Active Directory (AD).
4. OWA services return an HTML page composed of navigational bar and mailbox contents similar in appearance to the multidocument interface (MDI) used by Outlook to display a user mailbox and its contents.

When an end user accessed OWA from a legacy Exchange server, OWA used a separate logon page to authenticate users rather than an SAS pop-up dialog box (as described in Step 2). Other distinctions between legacy OWA and Exchange 2000 OWA are that legacy Exchange used cookies to monitor the client session and provided a logout button that terminated the session. Exchange 2000 OWA does not provide a logout button; instead, when ending a session, the user must close all browser windows and thus all running OWA instances.

Displaying Objects

Exchange 2000 OWA displays objects using the following process flow:

1. A consumer/end user, through a standard browser, requests an email message or another collaborative object.
2. The Exchange ISAPI layer in IIS 5 receives and brokers the request.

3. The Web Storage Service determines the type of object, the specific item, and whether the user is authorized to access it. If the authenticated consumer has authorization to access the object, it is returned to the ISAPI application.
4. Exchange ISAPI determines, based on the object's attributes, the appropriate form for this object type from form definitions in the Forms Registry. If a form definition is not found, a default form stored in `wmtemplates.dll` provides the services. If the browser language is not English, other template libraries are accessed in `\Exchsrvr\Res\Directory`.
5. Once the form is selected, the ISAPI Application layer parses the form and makes service calls to the Web Storage System to complete the form with the appropriate data objects.
6. Exchange ISAPI renders the form in the appropriate HTML or XML scripting code and returns the form to the browser.
7. The browser renders the scripting code and displays the information in an HTML document.

It is significant that in this process flow, OWA renders a markup language appropriate for the client browser. Depending, for example, on the version of the browser, HTML or a combination of HTML and XML scripting is returned to the browser client for display. Non-Microsoft browsers receive HTML code that conforms to the older HTML 3.2 standard, whereas IE5 and more recent versions receive DHTML code that performs more complex operations within the browser DOM. Various markup languages are discussed later in this chapter.

Server Access

Outlook clients interact directly with Exchange Server services. OWA clients, using any standard browser, interact with IIS Web services. The browser client communicates typically from outside the enterprise namespace by using the HTTP protocol with WebDAV extensions. When IIS receives a request to access an object in the Web Storage System, it passes the request to the Exchange ISAPI application, which, in turn, communicates with the Data Store layer in the Web Storage System. The Web Storage System responds to the service request, the ISAPI layer renders it in HTML, and the browser on the client side presents it to the consumer/end user.

HTTP and WebDAV

Basic HTTP has been extended to accommodate a broader range of data types and service operations. WebDAV—Request for Comments (RFC) 2291, 3023—is a new technique that transforms the entire Web Store into a collaborative writable medium. Since 1999, WebDAV has provided HTTP with an increased feature set for providing services. WebDAV, written by the Internet Engineering Task Force (IETF) with help from Microsoft, extends the versatility of the Web Storage System as well as many other applications, especially because of its integration with XML

(discussed later in this chapter). WebDAV enables collaborative publishing to Microsoft Web servers across the Internet, supports Web folders in Microsoft Office 2000, and provides an interface that allows access to the hierarchical database found in the Web Store. WebDAV allows Web-based applications with APIs using HTTP requests to manipulate data objects like their networked application counterparts.

Key features of the WebDAV extensions include:

- *Locking features*—The use of long-duration concurrency or locking controls and shared write locks allows collaborative writing to the same resource without fear of corruption in the integrity of the object. The functioning of this locking mechanism is independent of the specific network connection in use. Thus, this feature provides for wide-scale collaboration without disruption (because public network connections are unreliable) and improved scalability (because open connections consume local resources).
- *Properties*—XML scripting provides properties for metadata that can further refine the definition of an element. WebDAV can read and manipulate the XML syntax. XML is discussed later in this chapter.
- *Namespace manipulation*—WebDAV provides an extensible feature set as the containers that hold data objects change location in Internet space.

In general, browser clients access, for example, their Universal Inbox using WebDAV according to the following process flow, shown in Figure 17.1:

1. The browser client issues an HTTP **get** request.
2. IIS receives this service request and passes it to the Exchange ISAPI layer, `davex.dll`, for processing.

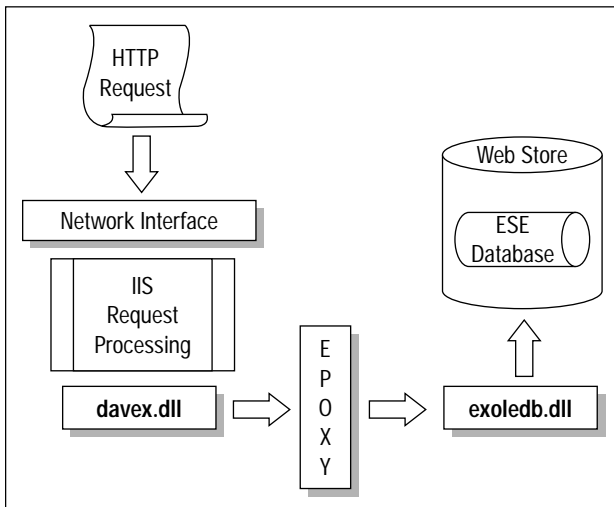


Figure 17.1 Using WebDAV to access an Exchange Inbox.

3. WebDAV forwards this request to the Web Storage System in Exchange Server, `exoledb.dll`, across the specialized EXchange InterProcess Communication channel (EXIPC)—more commonly called EPOXY, as you can see in Figure 17.1. `exoledb.dll` renders and processes this service request by accessing the Web Store or Extensible Storage Engine (ESE) database to retrieve the specific object's properties.
4. Once the Inbox properties are retrieved and parsed, Exchange Server routes information back to the client along the same path in reverse order.

Thus, a browser client, using HTTP, can retrieve a document (**get**) or submit or store a document (**put** or **post**) through this process flow into the Web Store. In fact, WebDAV provides commands to search, move, copy, delete, lock, and unlock data objects, as well as make new collections of resources in folders.

FE/BE Architecture

The division of labor in an FE /BE architecture—introduced in Chapter 1—is very similar to that of a multitier architecture, which these days is typically becoming the foundation for Web application development and e-commerce projects. In a distributed services or multitier architectural environment, one or more FE servers broker OWA service requests by redirecting them to BE servers that physically host the mail stores. BE servers form a more efficient database service layer than a single monolithic server because their physical separation from the front end provides less work for the local system and greater opportunity to scale the data stores more easily. Such a distributed environment is also better because it provides simpler options for scalability and the introduction of fault-tolerant architectures. In addition, distributed services help workflow through easier load balancing.

Advantages of FE/BE architecture include:

- *Single namespace*—Best practices recommend referring to all servers that provide a common service like mailbox storage with a single name. As you add local storage resources to accommodate increased user demand, the separation of FE and BE equipment makes scaling the Data Services layer a simpler task than if data were stored in a single hardware storage unit. You can add other BE servers without changing the front end because the FE has been freed from the more resource-intensive activities through the partitioning of services. The FE servers provide a single namespace from both the client/consumer viewpoint and from the perspective of the BE data layer to which service requests are directed. Thus, an OWA client uses a nonspecific URL such as **http://owa.coriolis.tch/exchange/username**. Once you redirect server access to the appropriate BE machine, fulfilling the service request for data is the same. You can also apply other features like Network Load Balancing to the FE servers to further expand capacity. Finally, the single namespace provides a subtle security

advantage over a monolithic server because the mission-critical data stores are never exposed to public access.

- *Offload Secure Sockets Layer (SSL)*—The argument for separating the more resource-intensive operations of storage and retrieval from brokering service requests also applies to cryptographic services. Encrypting and decrypting messages are processor-intensive operations. Providing these services impacts tremendously on a machine's local resources. If you provide such services on an FE machine rather than on the same server, partitioned service layers greatly enhance the performance of the entire messaging platform.
- *Public folder referrals*—One of the limitations of Internet Message Access Protocol 4 (IMAP4) is its inability to support public folder referrals. FE/BE architecture can accommodate this weakness in the protocol by actually retrieving the public folder information.
- *Server location*—Separation of services simplifies the deployment of, for example, firewall technologies. You can physically separate BE servers from FE servers by placing the BE servers behind firewalls. You can configure the firewalls to pass traffic from only specific FE servers.
- *Security*—Security is enhanced because FE servers exposed more directly to public access can only broker or redirect a service request; they do not provide their own data resources.

Another benefit is a division of labor in the use of FE servers that broker service requests and BE servers that fulfill the requests. If the local Exchange server has a mailbox data store, OWA accesses the mail store directly. We define an FE server in the context of our discussions as an Exchange 2000 server that does not host a public folder or mailbox data store, but instead functions as a proxy or broker. We can call the FE a proxy or broker that forwards service requests to a remote server able to process the request. A BE server, on the other hand, is an Exchange 2000 server that maintains the data store. Thus, if the local Exchange server is an FE server, OWA uses HTTP to proxy the service request to the appropriate BE server. The FE server uses Lightweight Directory Access Protocol (LDAP) to query a directory services provider (in this case, AD) for the path to the BE data store that hosts the user's mailbox. HTTP/WebDAV, Post Office Protocol 3 (POP3), and IMAP4 browser clients support FE/BE architecture.

Markup Languages

Scripting languages specify the operational flow of events and the manipulation of data structures that contain many different types of Web-based resources. Markup languages, a part of this interpreted rather than compiled family of source codes, use their tags to describe the appearance or meaning of the data structures defined

and manipulated by other scripting languages like ECMAScript (formerly Javascript), Personal Home Page (PHP), Active Server Pages (ASP) technology, and Perl. The many changes scripting languages are undergoing, from a static page description language (HTML) through dynamic interactive scripts (DHTML) to extensible metalanguages (XML), are radically altering the scope of control that both developers and end users can programmatically exert on Web-based data objects. The origin of markup languages, which provide portability across different software applications and environments to the underlying content they modify, is rooted in the printing industry, where the phrase *to mark up text* referred to editing a printed document. The term *markup* specifically refers to tagging elements in digitally rendered documents to either modify the appearance of the text or establish a document's structure and meaning for output to a medium like a printer or computer console. Many word processing applications use this technology. Many office workers who used earlier versions of the WordPerfect product could reveal the code or tags by simply toggling screens.

The process of markup consists of inserting tags (or *tokens* or *wickets*) in front of and (usually) behind a string of text to change the appearance or meaning of the enclosed information. Tagged text is often referred to as *source code* or *a document*; the Web browser software renders it. Rich-text format (RTF), a common file type among word processors, is a markup language that is rendered by word processing software. It can produce italicized words, bulleted lists, and block quotes. To view the actual markup in an unrendered RTF document file, use a 32-bit text editor like WordPad. Both Microsoft Word and Corel WordPerfect process RTF file format so that the content is rendered as a fully formatted document on the display monitor. In our example, it does not matter what machine or word processor package we use to view the document; it is still fully formatted.

Specific vs. Generalized Markup Languages

Two types of markup languages are used: specific and generalized. Specific markup languages generate code specific to an application or device. Generalized markup languages describe the structure and meaning of the text, but not how the text should be used. Thus, RTF uses a specific collection of tags to mark up (format) text using popular word processing packages; it does not work with text editors. The problems with specific markup languages are that the set of tags are limited, the document is not universally portable, and the markup language is not standards based (that is, the source code is not easily translated to other markup languages used in other situations or on other media running on other systems).

Generalized markup languages are based on a concept of describing text independent of the Presentation layer—the platform device and/or software application. Thus, the language describes structure rather than format or style. To develop cross-compatibility and extensibility, you need to strictly control the syntax of the language

to minimize hybrid and proprietary dialects of the parent code. Beginning in the early 1970s, IBM proposed the Document Composition Facility Generalized Markup Language (DCF GML). This working language led to the development of Standard Generalized Markup Language (SGML), which the International Organization for Standardization (ISO) adopted in 1986 (ISO 8879). The original objective of SGML was to format information for efficient distribution, search, and retrieval. It is actually a *metalanguage*—a language that defines other languages. Because of its intended scope and metalinguistic tendencies, SGML is very complex. For example, it identifies the characters used in the document, uses document type definitions (DTDs) to define structure, provides for identification of logical objects called *entities*, and allows for the incorporation of external data. It also supports a minimization technique that allows for relaxation in some of the syntactical patterns. An important omission is the absence of tags that describe how the data should appear in a chosen medium.

Web Markup Languages

Whereas specific markup languages originated in the printing industry, Web markup languages have roots in the nonlinear medium of the online world. Both HTML and XML are derived from SGML and are thus standards based. Both languages provide a grammar for text-based scripts interpreted by now commonplace browser client software. Whereas the older HTML describes the appearance of elements within the layout of some Web page, XML provides a metalanguage extending this tagged grammar to encompass and define the content of any data structure that can be rendered by some browser client, such as Internet Explorer.

HTML

HTML, a text-based scripting language that describes how data should appear on the Web browser screen, was developed so that document specifications could be separate from appearance. SGML defines HTML at all levels and documents these specifications in a series of DTDs that you can reference at **www.w3.org**. From the beginning, however, HTML suffered from conflicting de facto standards promulgated by both Netscape and Microsoft through the use of proprietary HTML extensions or dialects of the parent code. In addition, there was not a move toward international language support until version 4.01, when a markup tag that specified language was added to the vocabulary. Prior to this version, the HTML Working Group of the IETF (**www.ietf.org**) had not defined language in specifications. Furthermore, URLs are written into the script of the document (commonly referred to as *hard coding*). This limitation in the language forces you to continually maintain any script because of the dynamic nature of referenced Internet resources. HTML does not allow you to associate hyperlinks to specific elements other than specially targeted hypertext references (commonly referred to as *internal links*) or multiple locations using one reference. HTML also follows the minimization technique of SGML in that this coding syntax is not extremely rigid.

Validation of HTML code implies only technical or syntactical integrity, not a strict compliance with the DTD. The browser can typically interpret poorly written code and successfully render it. This presents difficulties to application developers, who depend on a consistent interpretation of how programming is written. Another problem when you use HTML is that you can't maintain data in the form of variable names, values, or data structures across separate sessions. HTML was designed to format a table, not store the data in it. With HTML, you cannot define both document format and data structure.

The components of HTML (that is, the elements and attributes) are categorized as physical and logical. Physical elements and attributes define exactly how the content will be displayed. Thus, we can display a horizontal rule as 50 percent of the viewable browser window or 200 *pixels* (picture elements that are the distinguishable clusters of luminescent phosphorous) from the left side of the viewable window. Logical HTML elements or attributes, similar to XML elements (discussed shortly), describe the format of content enclosed within the tags. With logical elements, the browser uses the markup elements and attributes to identify content and then displays it accordingly.

The rebirth of HTML came in its marriage, in January 2000, to XML. The markup language spawned from this union is called Extensible Hypertext Markup Language (XHTML). This more rigid scripting language, in which source code is syntactically well formed and valid, provides a way to define content as well as describe appearance.

XML

In 1996, an XML Working Group began developing a new markup standard that was structurally more capable than HTML but considerably less complex than SGML. The objectives of this new language, XML, included:

- Universal usability across software clients and applications
- Compatibility with SGML so that legacy applications that comply with the SGML standard can easily interface with this new language
- Simplified development, which includes a minimal number of development tools, minimal syntactical options and language dialects, rapid application development through interpretation rather than compilation, and source code that can be easily sight-read

A theme of the working group has been to keep the standard concise rather than adding many options that facilitate authoring but complicate the XML markup language itself. Another important theme is that XML is the perfect medium for creating Web applications rather than designing page layouts. Other characteristics of XML include:

- It creates and passes messages to call methods to provide a programming interface.

- It separates content from presentation using XML documents and Extensible Style Language (XSL) pages (we will discuss XSL shortly).
- It uses several other languages (in addition to HTML)—such as Simple Object Access Protocol (SOAP)—to package requests for services provided by another server and thus supports a wide variety of applications, especially those using SGML-compatible message formats.

The basic components of an XML document are elements, attributes, and comments. Development of XML applications requires elements and attributes that are logical in origin and under the user's control. XML documents require a consistent syntax within which authors can define their own tag sets so that these defined elements are reusable according to a stated set of rules. Documents are thus self-describing scripts that contain all the rules for that particular class of document. You typically use Microsoft XML Notepad to work in a document. Elements mark up sections of an XML document, and content is usually contained within the XML tags. You also have elements, called *empty tags* or *singletons*, which do not contain content. Attributes modify the behavior of the markup tag. Comments can be embedded in the code to help orient the reader. You must follow very specific rules when using and placing tags so that all XML documents are well formed and valid. A well-formed XML document:

- *Contains a DTD*—The XML DTD is important because XML is extensible; you define or extend the meaning and structure of the XML tags. Unlike with HTML, which has predefined tagged keywords, you create the language that describes your text. The DTD provides the syntactical rules for the descriptive tags you create. The server-side or client-side processor can verify the integrity of the document according to the author's stated rules. A valid XML document is a subset of valid SGML code, ensuring both portability and extensibility of the content to other platforms, both today and in the future.
- *Has the proper hierarchical structure*—This is where a single container tag or root surrounds all other entities and tags declared and written in a syntactically correct and balanced manner.
- *Has properly declared tags*—A major difference between HTML and XML is in tag symmetry; in a proper hierarchical structure, empty tags do not exist.

When formatting your XML document, you use either a Cascading Style Sheets (CSS) text file or an XSL style sheet. The use of style sheets was popularized with the introduction of DHTML—a marketing term both Netscape and Microsoft use to differentiate their version 4 Web browsers from earlier products. CSS was a key component in this cross-browser technology and an important tool in purifying the HTML code for a rebirth and refocus in functional purpose. The latest version of CSS—level 2 (released in March 1998)—includes commands for object positioning on the display screen without major changes in scripting syntax.

You create a simple XML style sheet using XSL by loading the XML data into the client-side DOM, formatting the contents according to the style sheet specifications, and displaying the information. In fact, XSL is actually two languages in one; it consists of a transformation language that can transform XML into a well-formed HTML document and a language that applies formatting objects to XML code. These two features can work independently of each other so that you can transform XML without applying formatting rules, and vice versa. XSL improves upon CSS because it enables you to format and display both XML elements and attributes; CSS works only with the element. Also, XSL enables you to dynamically manipulate data; CSS, an older technology, displays only static XML data.

XML derives its flexibility from the use of an extensible metalanguage that is basically a grammar describing another grammar. XML allows you to build your own tags that define your own content or data. You write a script that executes a SQL-based query via ADO and obtain a recordset or equivalent data structure. Your XML tags describe various aspects of your data dictionary or field elements of that data structure. If your Presentation layer needs to display the content of the recordset in a particular way, XSL modifies the view of the data. For example, the following code snippet, written at the top of an XML script, transforms XML content into an HTML table based on the view defined by another text file, `simple.xsl`:

```
<?xml-stylesheet type="text/xsl" href="simple.xsl"?>
```

XSL as a language provides two major benefits. It is a transformation language for XML documents that enables you to transfer the content into other document formats that conform to SGML. It also provides an XML vocabulary for formatting semantics that provides the foundation for more complex formatting rules applied to more complex XML content. No software available supports all aspects of these specifications. The presentation or view of the data is independent of the programmed output through the combination of XSL and XML. Instead of integrating styling instructions with a Visual Basic-based application, current Web browser versions like IE5 can take full advantage of a markup language like XSL and direct browsing capabilities. XSL also provides for document transformation. XSL transforms an XML page into an HTML page or document rendered in another compatible medium, as shown in Figure 17.2.

Internet Explorer 5 (IE5)

Microsoft IE5, the latest version of the browser client, allows users to view and navigate XML documents just as they do HTML documents. IE5 can receive both HTML 4 and XML 1, which provide greater breadth in object selection and manipulation on the client side of the message exchange than earlier browser versions that rendered legacy HTML. The outward appearance of the document in CSS or XSL is identical to how HTML looks. The XSL software processor in IE5

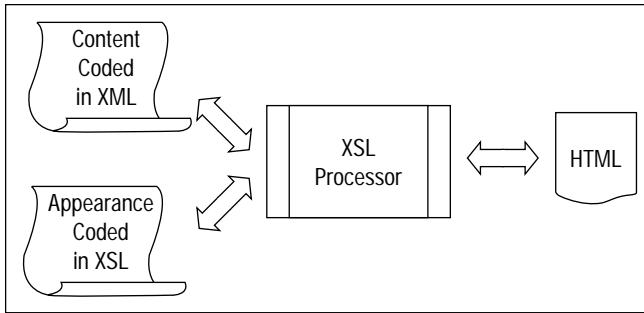


Figure 17.2 XML information transformed by XSL for any display device.

applies style sheets to XML content in a way that uncouples content from appearance. Thus, large amounts of client-side scripting overhead are removed from the content, often referred to as an XML *tree*. This separation allows developers to provide the same content to, for example, devices like a software browser, palm-top device, or wireless phone by simply applying different style sheets. The XSL processor can function on both the server and client sides of the Internet exchange. If it's run on the server side, all XML content is transformed to HTML or another display format before being sent to the client. Alternatively, if the XSL processor is executed in the client browser, both the XML document and XSL style sheet are sent to the browser, where the final display of the material is rendered.

The DHTML enhancements in IE5 provide greater client-side functionality than the features available in earlier scripting languages. DHTML also improves client response times, which is especially important when you are accessing resources across a public network. In addition, these enhancements improve server workload because processing is done within the client-side browser. Older client software reverts back to server-side processing to provide these OWA services. You can access DHTML behaviors more easily using simple CSS syntax than through scripting because CSS can use one external text file that contains format specifications affecting an entire collection of HTML documents; scripting typically affects only the document it is written in. The DHTML behaviors differ from ActiveX controls (discussed later in this chapter) in the way they are built; they use HTML components (HTC files) rather than C++ or Java compiled code. Thus, a more predictable security model is applied with the DHTML behaviors than with regular HTML because the security is a part of the native HTML scripting rather than from the operating system (OS) in which the document is displayed.

IE5 also provides greater programmatic control over other aspects of the document than legacy browser clients. For example, developers have greater control over the printing of documents than they did with earlier versions of the browser. You can access the Print dialog box by using the **window.print()** function, as well as adjust page layout before or after printing is completed.

Web Forms

Legacy Exchange 5.5 used ASP technology and OWA to create a Web-based application that looks and acts, for the most part, like its component-based counterpart, Outlook. The Outlook Forms Converter in Exchange 5.5 created custom HTML forms from the same ASP technology. Exchange 2000 has improved on these scripting approaches by using the Web Store.

Web Forms is a customized Web-based application that uses the Web Store and works within OWA as well as customized applications. The product is fundamentally based on HTML, so it is both browser and platform independent, running on both the newer IE4 and IE5 client through DHTML behaviors and older browsers running HTML 3.x or earlier versions. Thus, in the IE5 browser, Web Forms leverages both HTML and XML elements so that a developer can associate the fields with other objects or information in the Web Store to more rapidly create customized applications. FrontPage 2000, and especially the FrontPage Web Form Authoring add-in, simplify this process.

Exchange 2000 provides a more functional OWA Web Store than the legacy Exchange ASP scripting. It includes a broader range of Exchange objects like Outbox, calendar, and scheduling information to which the fields in Web Forms can link. Thus, the HTML scripting in the form prompts for a value, and, when the Web form is submitted, delivers it directly to a newly created object in the Web Store. All messaging and collaborative information is displayed in either the browser window or frames. You can access forms from a variety of libraries available in a drop-down list within the Choose Form dialog box, as shown in Figure 17.3, accessed from Choose Form in Tools | Forms.

Web Solutions

Collaborative application development has been a major theme in Exchange Server over the years. In earlier versions, Visual Basic Forms Designer was the only tool that developers had. It was robust and useful, but it too was intimidating and not well integrated with Exchange objects. MAPI, though available, was too complex for rapid application development, and coding a quick form required access to one or two data objects. This situation changed, however, with the introduction of technologies like ASP that have simplified the complexity of the coding process through interpretative scripts rather than component coding compiled into some kind of machine-executable code.

With the development of interpreted scripting, a variety of gateway methods have been developed to interface various scripting languages with different Web services. Rather than accessing database information directly, common gateway interface (CGI) scripts translated service requests into coded instructions that Web servers

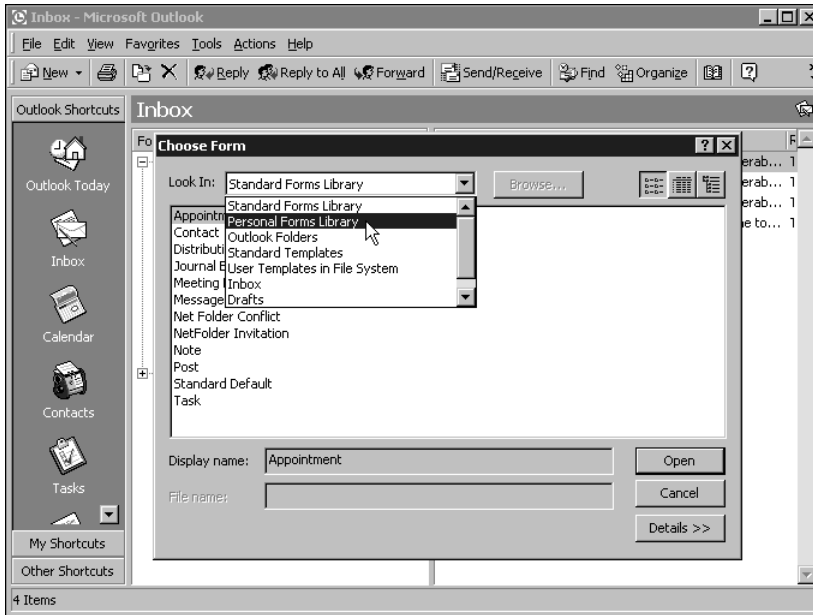


Figure 17.3 Accessing Web Forms through Outlook.

could process. Several methods access system and Registry information to process programmed instruction that provides server-side operations and resources. They include Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), ColdFusion, and Active Platform.

ODBC

This Microsoft interface provides a simple way to access database information without specialized knowledge or even complicated code. The ODBC driver is a layer of software that communicates directly with a database that is assigned a data name called a data source name (DSN). The ODBC driver manager functions as an interface for a script or application that has generated a service request using, for example, SQL. The driver manager directly references the specific database provider using appropriate security credentials and other parameters. The script or application can then communicate through the established connectivity to access the data resources. The ODBC standard, though originally designed as a proprietary interface, is now considered a de facto standard that relational database products like IBM's DB2 and Oracle, as well as desktop applications like Microsoft Access and Excel, use. In NT 4 and Windows 2000, you must register the ODBC-compliant database as a system data source. In addition, for the ODBC-compliant database to be accessible to a user's interface with the resource from a Web-based environment, the system Registry must know that the database is there. This is necessary for service requests to be properly directed in the system.

JDBC

This Sun Microsystems interface provides an alternative interface to ODBC for a server to work with SQL-compliant databases. This, along with ODBC, has become a de facto standard in the server-side manipulation of database information. JDBC, like ODBC, uses system Registry entries to provide database services to Java through the Java API. SQL statements are processed within Java programs as Java objects. Although both ODBC and JDBC require specific drivers to operate, Java-related drivers tend to be compatible with more programming languages and OSs than ODBC-related drivers.

ColdFusion

Originally a CGI script, ColdFusion is a compiled program that is packaged as both a Web authoring system and a separate Cold Fusion server that is necessary to facilitate database access using the Cold Fusion Markup Language (CFML). This solution uses a proprietary markup language, ColdFusion Markup Language, which provides an application-development environment with high-level query and retrieval functions but limited complexity in programming. It interfaces with the Web server through the appropriate API (such as ISAPI or Netscape API) as an add-on or plug-in.

Active Platform

Microsoft Active Platform is the foundation for designing and developing Internet and intranet business applications. It is a three-tier client/server model that has an extensible component-based architecture. Active Platform provides a simple-to-learn and easy-to-implement development environment because you use the same set of tools for both the client-side and server-side components. The platform is divided into three parts: ActiveX, Active Server, and Active Client.

ActiveX

This term connotes not a programming language but a set of technologies for delivering services to the Internet and intranets. ActiveX includes ActiveX controls and ActiveX scripts. ActiveX controls are versions of OLE software controls that have been reduced in size and overhead to optimize their use on the Internet. Both ActiveX and OLE are based on the Component Object Model (COM), which Microsoft uses to build software applications. COM provides specifications that are designed to ensure that two objects can interact and communicate with each other regardless of programming language or OS platform. *ActiveX scripting* is the collective term for JScript, Microsoft's open implementation of JavaScript, and VBScript. JScript 5.5 is the first scripting language that fully conforms to European Computer Manufacturers Association ECMAScript specifications.

Active Server and Active Client

The term Active Server describes IIS or any other Web server that supports server-side scripting (more commonly known as ASP technology). An ASP script leverages both server-side processing power and HTML by using reusable software objects built into a DOM. The ASP application is made up of various elements that together form a usable interface that provides a service. Each application is composed of a collection of various text-based files, server objects, and components joined together by ASP scripting coded directly in the HTML document. ASP is easier to use than, for example, CGI scripting, because it is native to the Windows NT and 2000 platforms, often working directly through the ISAPI layer. It thus works more intimately with other software layers in the Microsoft OSs than an interpreted CGI script located “out of process”..

Active Client is any browser that can support ActiveX components. The Internet Explorer family has naturally been able to complement Active Server because Microsoft developed both of these software components. Active Client is the primary presentation layer of any Web-based application. It must support the client-side scripting of Dynamic HTML, which mixes static HTML descriptive elements with interactive pieces of script written in interpreted codes like ECMAScript (formerly JavaScript), VBScript, and more recently, XML. Thus, the Active Client provides a robust user interface that leverages server-side functionality.

With Active Server Pages technology, you can create scripts with the same tools as Active Client and embed the specialized code within HTML. In addition, ASP can use Active Server components to keep an HTTP connection with the database server open and functioning in a persistent manner over time through a feature called *keep alive*, which optimizes the scripting performance. A CGI script will, alternatively, create or spawn a new process on its server platform for every new service request.

Active Server comes with the following predefined Active Server components that help build applications within an application:

- *Server-side scripting*—VBScript, ECMAScript, Perl, CGI, REXX, Java, ISAPI
- *Client-side scripting*—VBScript, ECMAScript, ActiveX controls, Java
- *Server-side objects*—Built-ins that control the ASP engine
- *File system objects*—Virtual, physical, and root directories, Access Control Lists (ACLs)
- *Server-side components*—COM-based objects that are scalable, manageable, and distributable
- *Text*—ASCII text that forms both HTML and XML tags

As you can see, there are many components of ASP technology. They are actually part of a larger model called Universal Data Access architecture, shown in Figure 17.4.

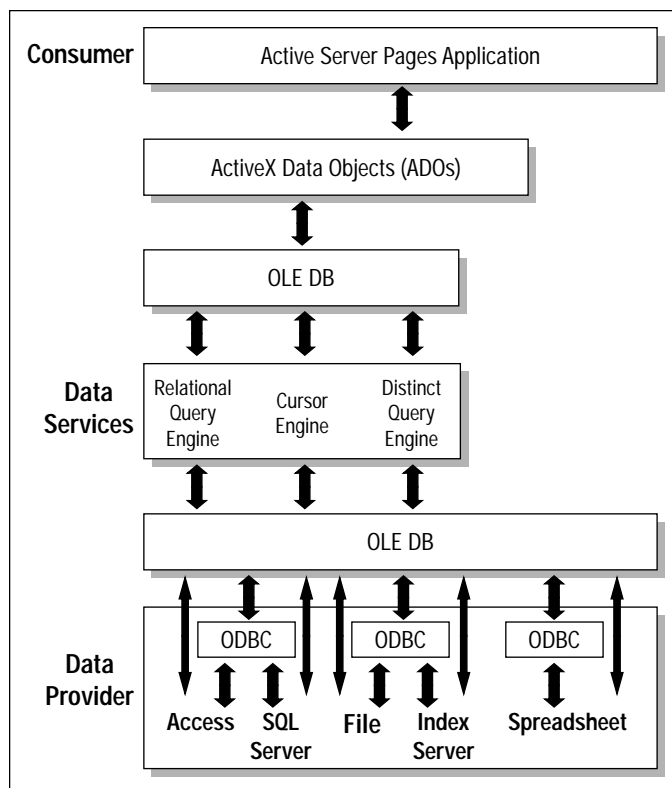


Figure 17.4 Universal Data Access architecture.

Universal Data Access is yet another Microsoft strategy to provide access to data that is based in a client/server environment or intranet. It is relational or non-relational. Diverse data sources are made accessible through the Microsoft Data Access Components (MDAC). This consists of new versions of ADO, OLE DB, and ODBC, which are supported as one release.

Exchange APIs

Exchange offers a large range of technologies that you can use to rapidly develop applications as well as leverage predefined and tested methods to access data objects from the Web Store. Exchange 2000, like Windows 2000, is built as an extensible platform. It provides this feature by using standards-based and well-known APIs. These component interfaces are actually libraries of reusable functions that collectively form dynamic link library (DLL) files. A developer can write source code that requests these functions to perform a service. Aside from the fact that using these functions is simpler than writing new code, the function specifications are well known, the code is reusable, and the performance is guaranteed to work.

OLE DB and ADO

One component that builds data-driven dynamic Web applications is ADO, a subset of a larger, more complex API, the OLE DB model. OLE DB provides a scriptable interface that permits servers to share both structured and unstructured data types, including multimedia. It is a specification for a set of data-access interfaces that enable a multitude of data stores to work as a single unit. Exchange 2000's OLE DB provider allows direct access to the Web Store. OLE DB, unlike ODBC, accesses both structured and unstructured information; databases; and, for example, Word and other Office file types. In highly technical terms and as shown in Figure 17.5, ADO provides an interface, whereas OLE DB, like ODBC, actually accesses the data source. This is why OLE DB, which deals with a broad range of data, is so much more complex than ADO as a programming API.

ASP works with ADO to provide database connectivity. ADO is part of a bigger picture, the COM that Microsoft formulated to support the development of applications through the conceptualization of separate software objects, written in different languages yet working in a complementary and supporting capacity. As mentioned earlier in this chapter, you use the ADO interface to access objects in the Web Store. ADO provides support for all Microsoft database APIs and contains built-in features that provide developers with access to client-side dynamic objects. Thus, for example, you can use the ADO API to query specific records in a relatively easy sequence of programmatic steps.

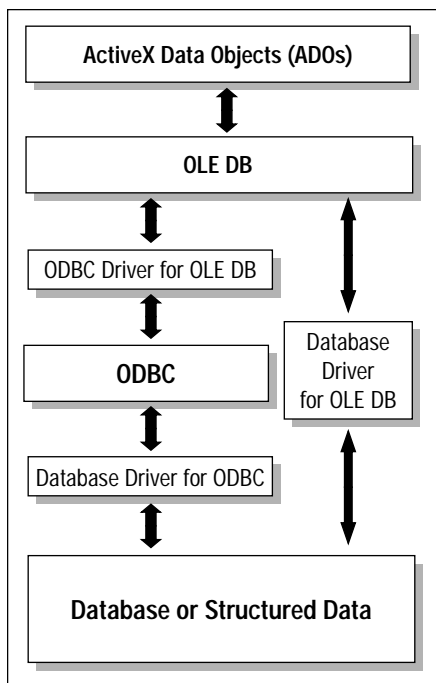


Figure 17.5 Accessing a database using ADO.

Active Directory Services Interface (ADSI)

ADSI is a standardized Windows-based interface for multiple or metadirectory applications. A *metadirectory* is a network-level directory service that organizes and manages resource and permission information from separate network OSs. You use ADSI to access the entire range of objects found in AD directory services—from machines and groups to user attributes. This interface also provides access to user information from a Web-based application. ADSI can leverage capabilities and features from other network OSs and directory services that use well-known or standards-based protocols such as LDAP, Novell Directory Services (NDS), and NT Directory Services (NTDS).

CDO

The CDO library provides an API that simplifies complex operations that specifically involve messaging objects (such as calendars, tasks, and scheduled events) so that they are easily accessible and manageable in the Web Store. CDO is an API specification that defines the objects, interfaces, functions, and properties of each messaging and collaborative object. CDO, along with OWA, was introduced with Exchange 5.5. CDO in Windows 2000 is a Simple Mail Transfer Protocol (SMTP)-based library. Exchange 2000 upgrades Windows CDO during its installation and adds new features and a larger range of functions to the CDO libraries. Exchange 2000 CDO is compatible with all existing libraries and works with ADO and OLE DB.

MAPI

You use this industry-wide standard for writing messages and workflow applications to create and access many of the oldest mail applications and messaging systems. It provides a uniform environment for development and standard use. You can program a simple request from any program to access a messaging service through an entry point in the DLL and pass a record to an externally defined messaging structure. Full MAPI conforms to the Microsoft COM. CDO was designed to simplify much of the coding needed to use the many MAPI features.

.NET Framework

Exchange has been described as the first .NET server. Microsoft's .NET software initiative proposes to simplify the development and deployment of Web applications and services. Even if Microsoft's initiative never achieves full fruition, Exchange 2000 remains an extensible platform for messaging and collaborative services fundamental to any e-commerce undertaking. Exchange 2000 blends transparently into the infrastructure of an enterprise while delivering levels of service appropriate to the 24x7 world of e-commerce on the Internet. This reliability and extensibility have positioned Exchange Server as a fundamental Web service layer in the proposed .NET landscape.

The .NET framework, shown schematically in Figure 17.6, gives you a unified approach when you are developing and deploying Web applications and services. It is a layered infrastructure that rests on core network OS services. This initiative is a necessary step in providing loosely connected distributed services in both a private namespace (like a corporate enterprise) and a public network (like the Internet). The framework provides a common language runtime engine, though “engine” probably does not carry the correct connotation. .NET is a framework within which solutions are assembled or interconnected rather than a platform on which these component parts are run.

It is important to remember that unlike legacy Exchange 5.5 operating on NT 4, Exchange 2000 requires that the network OS it runs on have IIS 5 installed as a properly configured component. This is another example of partitioning services that specifically characterizes .NET servers and Windows 2000 in general. We can discuss the .NET framework or any generic multitier architecture built on top of OS services and still use Figure 17.6.

As we have discussed throughout this book, the theme of partitioning system services so that you can distribute them throughout an enterprise in Exchange has been a significant architectural trend. Although the details are outside the scope of this book, the .NET architecture is significant in that Exchange is considered the first in a family of distributed service providers. Although the concepts in this chapter are more often associated with Web application development, our thesis here is that all these features are built into the Exchange 2000 design and were intended to create a central broker of information services useful in any Internet-based endeavor. Component-based software applications have powered enterprise and commercial interests for many years. As their popularity has grown, constraints concerned with distribution and licensing have increased as well. Web services are evolving into a virtual ocean filled with floating objects and functionality into which you can plug your standard browser software. These Web-based objects and

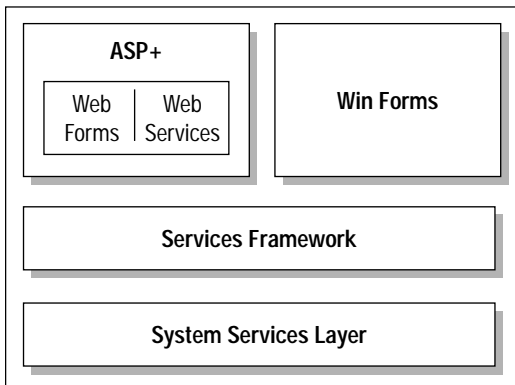


Figure 17.6 The .NET framework architecture.

services, though still problematic especially in the area of security, attract much attention because they clearly provide more value than component-based software in the areas of scalable services.

In the first chapter of this book, Figure 1.1 shows a pyramid composed of the various new service providers Microsoft has released since the beginning of 2000. Commerce Server 2000, providing scalable business-to-consumer (B2C) and business-to-business (B2B) sites with analytic tools, sits at the apex of this family of .NET servers; we have put Exchange 2000 at the base. The development of online services is based on messaging rather than process. A component-based software application doesn't require a messaging platform, but it also doesn't scale well across an enterprise, much less the Internet. The World Wide Web has entangled our earth and our information technologies in a digital mesh. In less than 50 years we have evolved from blinking lights, banks of toggle switches, and 128 kilobytes memory to digital dashboards, knowledge objects, and memory measured in gigabytes. As inhabitants of a global village, we have come to expect information wherever we are, at any time, and from any available device. Exchange 2000 provides the messaging and collaborative platform on which document and content management are built. It provides an infrastructure much like Active Directory provides a namespace because communication and collaboration are fundamental to activities, culture, and the social cohesion of humankind. No matter how great the interest in online services, the need for reliable messaging will always overshadow process features because there is no workflow without communication and no progress without collaboration. If an extensible platform combines both messaging and collaborative services, it will, in fact, provide the perfect foundation on which to manage knowledge or engage in forms of public or private activity. Exchange 2000 is positioned as that Web-based application platform; as its administrator, you are positioned to coordinate and direct its services.

Chapter Summary

This chapter reviews the issues that are involved in extending the Exchange platform:

- ▶ The Web Storage System is based on three key service areas: file system (Win32) support services, database services, and collaboration services. The Exchange 2000 structure is partitioned so that the Web Storage System handles all database issues.
- ▶ Web Store support of both the Web-based data types and older unstructured data like Office file types in the Win32 file system extends the range of data objects that can be accessed and managed through programmatic control. It's significant that these data objects are accessible from the same interface and displayed without discernable differences through an interface like a digital dashboard. Features like property promotion, which automatically provides a

consistent view of data, support the concept of a universal Inbox as well as the platform and application independence of displaying the data.

- Rich HTML support, specifically in the forms of DHTML and XML, leverages the native features of the Web Store and provides client-side processing performance benefits.
- You can monitor Web Storage events by using event sinks, which are applications to which notifications are sent if a defined system condition is met or occurs. Event sinks form the basis of the programmatic control of dynamic events that occur with Exchange 2000 or another associated service layer.
- Web service clients differ in terms of their DOM. These DOMs render information encoding in scripting languages. Thus, differences in the display of content arise from how the DOM interprets the syntax of the scripting language.
- You can think of server components like OWA as brokering service layers for all messaging traffic. For example, in the case of proxy requests, ISAPI receives service calls to IIS 5 and relays them to the Web Store. This partitioning of services mirrors the Presentation layer, Business Logic layer, and Database layer multitier architecture model.
- OWA in Exchange 2000 is accessed by HTTP and interacts with IIS rather than the legacy approach of using MAPI and ASP technology.
- Significant support for Web Storage Services comes from the HTTP and WebDAV extensions, which provide a way, using URL addresses, to locate and manipulate CDOs. WebDAV provides an extensible way to identify and manipulate content and extends the capability of HTTP with a greater set of commands and features like concurrency controls, namespace manipulations, and enhanced properties.
- FE/BE architecture provides various benefits, including a single namespace and the offload of CPU-intensive processing like encryption, public folder referrals, and isolated server location to simplify security like firewall technology.
- Markup languages use specialized tags to indicate special treatment of text in a text-based script. Specific markup languages generate code that is specific to an application or platform; generalized markup languages describe a structure and meaning of the text but not how the text should be used. Thus, generalized markup languages are based on describing text independent of the Presentation layer.
- HTML and XML are standards-based markup languages derived from SGML. HTML is a page description language, whereas XML primarily defines content. XML is an attempt to create universal usability; compatibility with other SGML-compliant scripting languages; and simplified, rapid development of Web-based applications. XML is well defined and valid. XSL can both format

and translate the content defined by the XML script. The transformation of XML to, for example, HTML allows you to easily alter the view of data to accommodate different methods of presentation, such as desktop display and cellular phone display.

- ▶ Script support helps to extend the Exchange messaging platform because processing can be offloaded to the client side when IE5 and DHTML are used as coding instructions. In addition, the Web Store supports the same ADO interfaces as the older component-based programs use to manipulate data. These scripts can access the same OLE DB provider compiled programming language used to handle data.
- ▶ Server access is enhanced in Exchange 2000 because Outlook clients make service requests directly to Exchange Server. Browser clients access the message stores through OWA, which interfaces with IIS Web Services. This partitioning of service responsibility on the server side improves response performance. IE5—with native capacity to support HTML, DHTML, and XML—provides a broader range of accessible objects than legacy browsers and offloads services to the client.
- ▶ Web Forms and the use of OWA provide advantages such as easily deployed light messaging, low-overhead management of roving users, easily deployed kiosk and information source outlets, and backup support during migration.
- ▶ Two cultural approaches have become de facto standards in the server-side manipulation of database information: Microsoft's ODBC and Sun's JDBC. Other implementations include ColdFusion.
- ▶ Microsoft's Active Platform is composed of ActiveX, Active Server, and Active Client (IE). ASP technology is a key solution to server-side scripting.
- ▶ Exchange 2000 provides several different APIs for programmatic control of data objects, including ADO, ADSI, CDO, MAPI, and OLE DB.
- ▶ Exchange has been described as the first .NET server because of its design and positioning in relation to the Windows NOSs and other servers and service layers that rely on messaging and collaborative support.

Review Questions

1. You can transform XML to HTML using what language?
 - a. HTML
 - b. XML
 - c. XSL
 - d. XHTML

2. If you needed to write scripting code that modifies the appearance rather than the content of information, which language(s) would you use?
 - a. HTML
 - b. XSL
 - c. XML
 - d. Answers a and b
3. In a three-tier application model, what are the three tiers? [Check all correct answers]
 - a. Presentation layer
 - b. Data layer
 - c. Network layer
 - d. Business Logic layer
4. In OWA architecture, what layer or service authenticates the user?
 - a. AD
 - b. IIS
 - c. ISAPI
 - d. SAS
5. What service or layer in OWA determines the proper response form to a service request?
 - a. IIS
 - b. ISAPI
 - c. EXIFS
 - d. Forms Registry
6. The browser client communicates with what service or layer when requesting data services?
 - a. Web Store
 - b. EXIFS
 - c. IIS
 - d. EPOXY
7. When OWA returns a request to an IE5 browser, what scripting languages can it use to render its response? [Check all correct answers]
 - a. DHTML
 - b. HTML
 - c. XML
 - d. XSL

8. What protocol(s) does OWA use to proxy service requests to BE servers?
 - a. HTTP
 - b. LDAP
 - c. SMTP
 - d. Answers a and b
9. What benefits does an FE/BE architecture provide? [Check all correct answers]
 - a. A single namespace
 - b. A single signon
 - c. More efficient SSL encryption/decryption services
 - d. Easier deployment of security measures
10. What are the key specific management functional areas (SMFAs) built into the Web Store? [Check all correct answers]
 - a. Database services
 - b. File system services
 - c. Security services
 - d. Protocol services
11. Which is not a key feature of the Web Store?
 - a. Support for rich HTML
 - b. Support of Win16 applications
 - c. Support for ADOs
 - d. Folder replication
12. What benefit does OWA provide with significantly less overhead than the full-featured Outlook application?
 - a. Policy-based management
 - b. Messaging and collaborative services
 - c. Remote user support
 - d. Web Forms
13. In what scripting languages are forms rendered for IE5 browser clients? [Check all correct answers]
 - a. HTML
 - b. The same markup language as IE3
 - c. XML
 - d. The same markup language as IE4

14. What features are provided with WebDAV extensions that are not found in HTML alone? [Check all correct answers]
 - a. Concurrency control
 - b. Log tracking
 - c. Protocol control
 - d. Extensible code
15. What is not an objective of the XML language design?
 - a. Universal usability
 - b. SGML compatibility
 - c. Rapid development
 - d. Backward compatibility with HTML
16. Which statement about DTD is false?
 - a. DTD defines how elements are displayed.
 - b. DTD defines rules that validate a document's syntax.
 - c. DTD means that a document is a self-describing script and more portable than a script without DTD.
 - d. HTML uses DTD.
17. What are the most significant XML characteristics? [Check all correct answers]
 - a. Documents are well formed.
 - b. Documents are valid.
 - c. Documents are cross compatible.
 - d. Documents are standardized.
18. Where is the XSL processor located?
 - a. The processor is on the server side.
 - b. The processor is on the client side.
 - c. It is a separate processor running complementary to a server.
 - d. Answers a and b.
19. Which statement about IE5 is false?
 - a. It allows you to view both XML and HTML code.
 - b. It provides a greater opportunity to perform client-side tasks than legacy browsers like IE3 and IE4.
 - c. It provides greater programmatic support for scripting than legacy browsers.
 - d. It uses C++ and Java compiled code rather than .htc files.

20. Do you have to change your standard browser configuration to read XML code?
 - a. Yes.
 - b. No.
 - c. You can configure the browser to read XML if you know the browser version.
 - d. You can configure IE4 or later browsers using an XML processor plug-in; IE5 reads XML code directly.

Real-World Projects

Harry has noticed that when he opens email using a standard browser and the URL **http://servername/exchange/harrys**, a hyperlink on the right side of his screen says, View As A Web Page. If he clicks on the link, he can see the message just like a Web page. He wants to learn more about how this is done. He knows this message has been rewritten in HTML.

Harry wants to see how the HTML source code is written. He is using IE5 to view the HTML document. In the IE5 menu bar, Harry selects View | Source. A Notepad window showing the HTML source code opens. Harry examines this source code carefully. He notices that the script contains the markup tags that begin with “< “ and end with “>”. He recognizes the lengthy DTD and notices several <META> key words. Finally, enclosed within several tags, he sees his message.

Harry wants to try writing his own simple script.

Project 17.1

To create a simple HTML message:

1. From the desktop, right-click on the display area and select New | Text Document.
2. While the label is highlighted, type “testdoc.htm” and press the Enter key. The three-letter extension is important. You should see an error message that says, “Rename—If you change a filename extension, the file may become unusable. Are you sure you want to change it?” Click on Yes. Your object should reappear with the IE icon.

***Note:** Hiding an object’s file extension as a desktop preference is the default. To avoid problems when writing scripting language code, best practices recommend showing file extensions. To do so, access the View option from any Explorer window and disable this feature. In Windows 2000, it is located under Tools | Folder Options.*

3. Open the newly created document in your IE browser window. It is a blank screen.
4. Select View | Source from the menu bar. You have opened a second window in Notepad.

5. To create a simple HTML script, enter this code exactly as follows:

```
<html>
<head><title> Test Script</title></head>
<body bgcolor="#FF0000">
This is a test message.
</body>
</html>
```

6. Select File | Exit and then click on Yes to save your changes.
7. Either press F5 on your keyboard or select View | Refresh. You will see a red screen with your message in the browser window.

Project 17.2

To create a simple XML script from the HTML script you just created:

1. Reopen the text editor view of testdoc.htm by selecting View | Source.
2. Add the following line of code to the top of the HTML source code above the opening **<HTML>** tag:

```
<?xml version="1.0">.
```

3. Go to the **<body>** tag and change **#ff0000** to **#ffff00** which replaces the hexadecimal code for the color red with the code for the color yellow.
4. Select File | Exit and then click on Yes to save your changes.
5. Press the F5 key on your keyboard to refresh your browser. The screen will turn yellow.

Harry knows that HTML and XML code are very similar. In fact, XML uses HTML code words except that it applies more rigorous grammatical rules to the code. It is said to be well defined and must be valid. Harry also realizes that one item missing from the XML script is a DTD. Nevertheless, he knows that adding the opening **<?xml version="1.0">** is sufficient to have the code parsed by the XML processor in the IE5 browser. Harry expects the code to work. He realizes that he actually didn't use XML to define any content. He is going to try doing that as soon as he can find a book about XML syntax.