This section includes articles and whitepapers from a variety of sources, both internal and external to Microsoft.

® Technical Articles

® Technical Whitepapers

For information about the publishers of this material, see Publishers in the Resources section.

This section includes a list of books available from Microsoft Press, and includes a sample chapter from the MS Press book, *Inside COM*.

See Microsoft Press in the Resources section, for more information about the publisher.

Welcome to Mastering Microsoft Visual Basic 5

This CD-ROM-based training application is designed to teach you how to build sophisticated solutions with ActiveX controls, create database solutions, and enable your Visual Basic applications to take advantage of the Internet.

{ewc MVIMG, MVIMAGE,!welcome.shg}

For an introduction to this course, click this icon.
{ewc mvimg, mvimage,!exppov.bmp}

Welcome to Mastering Microsoft Visual Basic 5

This CD-ROM-based training application is designed to teach you how to build sophisticated solutions with ActiveX controls, create database solutions, and enable your Visual Basic applications to take advantage of the Internet.

This course assumes that you are familiar with basic programming concepts.

The course is organized into the following 12 chapters.

1 Visual Basic Review

Chapter 1 covers user interface design, writing and debugging code, and creating an executable file.

2 Using the Data Control

Chapter 2 discusses how to use the Data control to view, edit, and update information in a database. You will learn about data control events, and how to use the data-bound grid and data-bound combo box to view and update data.

3 Using Data Access Objects

Chapter 3 introduces data access objects (DAO) and how to use them to open a database and to work with recordsets. You will learn to write applications that access data by using DAO and that use stored queries.

4 Advanced Database Development

Chapter 4 teaches you how to create an application that accesses external data and can handle multi-user locking issues, referential integrity, rule-violation errors, and ODBCDirect.

5 Using Dynamic-Link Libraries

Chapter 5 identifies the benefits of using DLLs, and provides general information about how to use them. You will learn to declare and invoke a routine in a DLL, and to create an application that uses various functions from the Win32 API.

6 Creating ActiveX Clients

Chapter 6 introduces Automation as a way to integrate applications. You will learn to use Visual Basic to create client applications and receive notifications from a server. You will also learn how to embed and link a Microsoft Excel object into a Visual Basic form.

7 Creating ActiveX Code Components

Chapter 7 introduces you to creating ActiveX code components, formerly known as OLE servers. You will learn how to create servers that provide objects by using Visual Basic, and how to implement an interface and extend code component functionality.

8 Creating ActiveX Controls

Chapter 8 describes how to create ActiveX components that expose objects, properties, and methods.

9 Using ActiveX Components in Internet Explorer

Chapter 9 covers the basics of using an ActiveX control on a Web page, including how to add a control to an HTML page, download the control and related files, and add Visual Basic, Scripting Edition to interact with the control.

10 Creating and Using ActiveX Documents

Chapter 10 describes how to create ActiveX documents and use them in Internet Explorer and Office Binder.

11 Creating Internet-Aware Applications

Chapter 11 describes how to use the new Internet control to build Internet-aware Visual Basic applications.

12 Application Setup and Optimization

Chapter 12 teaches you how to create a Setup program, optimize your application, and use resource files when creating an application.

Return to Welcome Screen

action query

A query that copies or changes data. Action queries include append, delete, make-table, and update queries. Delete and update queries change existing data; append and make-table queries move existing data. In contrast, select queries return data records. An SQL pass-through query may also be an action query.

action QueryDef

A data access object that contains the definition of an action query. An action QueryDef object is used only with the Execute method, and it defines, moves or changes data instead of returning rows.

ActiveX
The Microsoft brand name for the technologies that enable interoperability by using the Component Object Model (COM).

ActiveX component

Physical file (for example, .exe, .dll, .ocx) that contains classes, which are definitions of objects. You can use these objects in your Visual Basic application.

ActiveX control

An object that you place on a form to enable or enhance a user's interaction with an application. ActiveX controls have events and can be incorporated into other controls. These controls have an .ocx file name extension.

ActiveX designer
Visual designer that third-party developers can create for the Visual Basic's development environment. Examples include Forms^3 and the Connection Designer.

ActiveX document

A Visual Basic application that can be viewed in another container provided by an application, such as Internet Explorer version 3.0 or later or Microsoft Office Binder. Forms created with Visual Basic can be easily converted into ActiveX documents by using the Visual Basic ActiveX Document Migration Wizard.

ActiveX scripting
Microsoft technology for hosting scripts in Internet Explorer and other browsers.

ActiveX terms

ActiveX has brought several new terms to Visual Basic programming, some of which replace terms you may have used in connection with OLE. Each of the new terms is defined in this glossary. The following table lists the ActiveX terms, and connects them with the OLE terms they may be replacing.

ActiveX term	OLE term
active document	No corresponding OLE term
ActiveX	No corresponding OLE term
ActiveX code component	OLE server, OLE Automation server
ActiveX component (umbrella term for ActiveX document, ActiveX control, and ActiveX code component)	OLE Automation server, OLE Component, OLE control, custom control, OLE server
ActiveX control	OLE control, custom control
ActiveX designer	No corresponding OLE term
ActiveX document	No corresponding OLE term
ActiveX scripting	No corresponding OLE term
Automation	OLE Automation

In general, OLE terms related to drag-and-drop functionality remain the same. The following OLE terms do not change.

® compound document

® object linking and embedding

® OLE container control

® OLE drag and drop

® OLE object (for an object that can be linked and/or embedded)

® Remote Automation

add-in
A customized tool that adds capabilities to the Visual Basic development environment. You select available add-ins by using the Add-In Manager dialog box, which is accessible from the Add-Ins menu.

aggregate function

A function, such as Sum, Count, Avg, and Var, that you can use to calculate totals. In writing expressions and in programming, you can use SQL aggregate functions (including the four listed here) and domain aggregate functions to determine various statistics.

aggregation
A composition technique for implementing component objects. This technique allows you to build a new object with one or more existing objects to support some or all of the new object's required interfaces.

append
Add objects, characters, or records to the end of a collection, recordset, or file.

append query
An action query that adds new records to the end of an existing table or query. Append queries do not return records (rows).

application object
The top-level object in an application's object hierarchy. The application object identifies the application to the system, and typically becomes active when the application starts.

application programming interface (API)
The set of commands that an application uses to request and carry out lower-level services performed by a computer's operating system.

application project
Visual Basic project that will be made into an .exe file.

asynchronous call
A function call whereby the caller does not wait for the reply.

asynchronous processing

A type of I/O in which some file I/O functions return immediately, even though an I/O request is still pending. This enables an application to continue with other processing and wait for the I/O to finish at a later time.

In asynchronous mode, the client issues a request but continues processing until a response is returned. The client may issue multiple requests and can field them in whatever order they return. Asynchronous communications are network independent, and clients can issue requests even if the network or remote system is down.

asynchronous query
A type of query in which SQL queries return immediately, even though the results are still pending. This enables an application to continue with other processing while the query is pending completion.

attached table

A table in one database is linked to another database. Data for attached tables remains in the external database where it may be manipulated by other applications.

Authentication
The level of data integrity guaranteed for communication between two computers across the network.

Automation object
An object that is exposed to other applications or programming tools through Automation interfaces.

Automation server

An application, type library, or other source that makes Automation objects available for programming by other applications, programming tools, or scripting languages.

bind

To associate two pieces of information with one another, most often used in terms of binding a symbol (such as the name of a variable) with some descriptive information (such as a memory address, a data type, or an actual value).

binding

The process of putting an object into a running state so that operations (such as edit or play) supplied by the object's application can be invoked. The type of binding determines the speed by which an object’s methods are accessed by using the object variable.

bound control

A data-aware control that can provide access to a specific column or columns in a data source through a RemoteData or Data control. A data-aware control can be bound to a RemoteData or Data control through its DataSource and DataField properties. When a RemoteData or Data control moves from one row to the next, all bound controls connected to the RemoteData or Data control change to display data from columns in the current row. When users change data in a bound control and then move to a different row, the changes are automatically saved in the data source.

browse
To walk through data a record or row at a time.

browse back
Button; looks like <<.

browse forward
Button; looks like >>.

browse mode
In VBSQL only, this mode supports the ability to perform updates while viewing data.

browse sequence
Order in which topics are displayed when user presses the >> button (forward) or the << button (backward).

browser
Software that interprets the markup of HTML, formats it into Web pages, and displays it to the user. Some browsers can also contain ActiveX components, and make it possible to play sound or video files.

business object

Representations of the nature and behavior of real-world things or concepts in terms that are meaningful to the business. For example, in an application, a customer, order, product, or invoice can be represented as a business object encapsulated for manipulation by users.

business rule

The combination of validation edits, login verifications, database lookups, and algorithmic transformations which constitute an enterprise's way of doing business. Also known as business logic.

business service

The logical layer between user and data services, and a collection of business rules and functions that generate and operate upon information. They accomplish this through business rules, which can change frequently, and are thus encapsulated into components that are physically separate from the application logic itself.

by reference

A way of passing the address of an argument to a procedure instead of passing the value. This allows the procedure to access the actual variable. As a result, the variable's actual value can be changed by the procedure to which it is passed. Unless otherwise specified, arguments are passed by reference.

Including bitmaps by reference refers to bitmaps not located in the directories specified in the ROOT or BMROOT option.

by value

A way of passing the value, rather than the address, of an argument to a procedure. This allows the procedure to access a copy of the variable. As a result, the variable's actual value can't be changed by the procedure to which it is passed.

calling convention
The coding convention used to make a function call.

certificate authority
A third-party that issues digital certificates, which provide your application with a digital signature, and handles legal and liability issues for broken security.

class

The formal definition of an object. The class acts as the template from which an instance of an object is created at run time. The class defines the properties of the object and the methods used to control the object's behavior.

class factory
An object that implements the IClassFactory interface, which allows it to create other objects of a specific class.

class factory table
A list of registered class factory objects for specific class identifiers (CLSIDs), where a server for each CLSID is currently running.

class identifier (CLSID)
A unique identifier (UUID) that identifies an object. An object registers its CLSID in the system registration database so that it can be loaded and programmed by other applications.

class name

Defines the type of an object. Applications that support Automation fully qualify class names using either of the following syntaxes: application.objecttype.version or objecttype.version, where application is the name of the application that supplies the object, objecttype is the object's name as defined in the object library, and version is the version number of the object or application that supplies the object, e.g., Excel.Sheet.5.

class of an object
The class or type of an Automation object (for example, Application, WorkSheet, Toolbar).

client
Any application or component that accesses or otherwise makes use of services provided by components.

client batch cursor library

A library that provides client-side cursor support for ODBCDirect database applications. This library supports all four types of cursors (keyset, static, dynamic, and forward-only) and provides a number of other features including the ability to dissociate connections and perform optimistic batch updates.

client/server

A term generally applied to a software architecture in which processing functions are segmented into independent collections of services and requesters on a single machine or segmented among several machines. One or more processing servers provide a set of services to other clients on the same or across multiple platforms. A server completely encapsulates its processing and presents a well-defined interface for clients.

code component
An .exe or .dll file that provides objects created from one of the classes that the component provides. Formerly server and Automation server.

code module

A module containing public code that can be shared among all modules in a project. A code module is referred to as a standard module in later versions of Visual Basic.

code signing
A means to certify that code downloaded over the Internet has not been tampered with. Also known as digital signing.

collection

An object that contains a set of related objects. For example, a collection named Tax Preparation Objects might contain the names of objects such as EndOfYear, RoyaltyCalc, and ExemptionCalc. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection may vary.

collection list
A list of named groups of related collections. For example, Tested Components might be a list of all components that have been tested.

collection object

A grouping of exposed objects. You create collection objects when you want to address multiple occurrences of an object as a unit, such as when you want to draw a set of points.

column

The visual representation of a field in a grid. A column defines the data type, size, and other attributes of one field of a row (record) of data. All columns taken as a set define a row (record) in the database. An individual column contains data related in type and purpose throughout the table; that is, a column's definition doesn’t change from row to row.

COM

component

Any software that supports Automation, meaning it can be used programmatically in a custom solution. This includes ActiveX controls (.ocx files), ActiveX documents, and ActiveX code components.

component catalog

A sharable database of information that describes and manages components—generally ActiveX components (formerly called servers). A component catalog does not contain the objects themselves, but contains references to where the objects reside on a computer or network.

Component Object Model (COM)

An industry-standard architecture for object-oriented development. The Component Object Model defines interfaces on which ActiveX components are built.

compound query

A query that is composed of at least one action query (a query that copies or changes data) and at least one select query (a query that returns a **Recordset** without changing data). In DAO, a compound query is created by putting two or more SQL statements (separated by semicolons) in the **SQL** property of a **QueryDef** object.

connection string

A string used to define the source of data for an external database. The connection string is usually assigned to the **Connect** property of a **QueryDef**, **TableDef**, **Connection**, or **Database** object or as an argument to the **OpenDatabase** method.

consistent

The state of a multiple-table Recordset object that allows you to only perform updates that result in a consistent view of the data. For example, in a Recordset that is a join of two or more tables (a one-to-many relationship), a consistent query would not allow you to set the many-side key to a value that isn't in the one-side table.

control
A file in a Visual Basic project with an .OCX filename extension that is associated with a visible interface. The Grid and CommonDialog controls are examples of controls.

control array

A group of controls that share a common name, type, and event procedures. Each control in an array has a unique index number that can be used to determine which control recognizes an event.

create an instance

To create an instance of a class (instantiate); that is, to allocate and initialize an object's data structures in memory.

cross-platform development
Visual SourceSafe supports transparent file-compatibility support across multiple processors and operating systems.

current database

The **Database** object returned by the **CurrentDB**( ) function. A reference of `DBEngine.Workspaces(0).Databases(0)` returns the first database opened. This concept applies only to Microsoft Access.

current index

For an indexed table-type **Recordset** object, the index most recently set with the **Index** property. This index is the basis for ordering records in a table-type **Recordset**, and is used by the **Seek** method to locate records. A **Recordset** object can have more than one index but can use only one index at a time (although a **TableDef** object may have several indexes defined on it). The Microsoft Jet database engine may use more than one index to evaluate a query.

current record

The record in a Recordset object that you can use to modify or examine data. Use the Move methods to reposition the current record in a recordset. Use the Find methods (with a dynaset- or snapshot-type Recordset object) or the Seek method (with a table-type Recordset object) to change the current record position according to specific criteria. Only one record in a Recordset can be the current record; however, a Recordset may have no current record. For example, after a dynaset-type Recordset record has been deleted, or when a Recordset has no records, the current record is undefined. In this case, operations that refer to the current record result in a trappable error.

current transaction

All changes made to a Recordset object after you use the last BeginTrans method and before you use the Rollback or CommitTrans method.

cursor

Keeps track of the driver's position in the result set. The cursor is so named because it indicates the current position in the result set, just as the cursor on a CRT screen indicates current position. Cursors let the user scroll through and update a result set with fewer restrictions than browse mode.

data access object (DAO)

An object that is defined by the Microsoft Jet database engine. You use data access objects, such as the Database, TableDef, Recordset and QueryDef objects, to represent objects that are used to organize and manipulate data in code.

Data control

A built-in Visual Basic control used to connect a Visual Basic application with a selected data source. Bound controls require use of the Data control as a source of data.

Data Definition Language (DDL)
The language used to describe attributes of a database, especially tables, fields, indexes and storage strategy.

data services

Support the lowest visible level of abstraction used for the manipulation of data within an application. This support implies the ability to define, maintain, access, and update data. Data services manage and satisfy requests for data generated by business services.

data source

A named Open DataBase Connectivity (ODBC) resource that specifies the location, driver type and other parameters needed by an ODBC driver to access an ODBC database. A data source can be any source of database information.

data source name (DSN)
Name of a registered data source.

data-aware
Describes an application or control that is able to connect to a database.

data-definition query
An SQL-specific query that can create, alter, or delete a table, or create or delete an index in a database.

database

A set of data related to a particular topic or purpose. A database contains tables and can also contain queries and indexes as well as table relationships, table and field validation criteria and linkages to external data sources.

Database object

A Database object is a logical representation of a physical database. A database is a set of data related to a specific topic or purpose. A database contains tables and can also contain queries and indexes as well as table relationships, table and field validation criteria and linkages to external data sources.

database objects

The database objects are Recordset, Field, Index, QueryDef, TableDef, Relation, User, Group, Property and Document.

DB-Library ™

Enables the database to become an integral part of an application. Transact-SQL statements can be incorporated into the application, allowing the application to retrieve and update values from a database. Through DB-Library, values from the database can be placed in program variables for manipulation by the application. Conversely, values in program variables can be inserted into the database.

deadlock

Occurs when one user has locked a data page and tries to lock another page that is locked by a second user who, in turn, is trying to lock the page that is locked by the first user. While such occurrences are rare, the longer that a record (or file) is locked the greater the chance of a deadly embrace.

dependent object

Dependent objects can only be accessed by using a method of a higher-level object. For example, the Cells method of the Microsoft Excel Worksheet object returns a Range object.

DispTest

A tool, similar to Visual Basic, whose programming language manipulates the exposed objects of other applications.

dynamic cursor

A cursor where committed changes made by anyone and uncommitted changes made by the cursor owner become visible the next time the user scrolls. Changes include inserts and deletes as well as changes in order and membership.

Dynaset

A recordset variable that identifies a Dynaset created using the CreateDynaset method. The Dynaset object is outdated. It is recommended that you use the dynaset-type Recordset object instead which supports all of the outdated Dynaset properties and methods.

dynaset

A type of Recordset object that returns a dynamic set of pointers to live database data. Like a table- or a snapshot-type Recordset, a dynaset returns data in records (rows) and fields (columns). Unlike a table-type Recordset, a dynaset-type Recordset can be the result of a query that joins two or more tables. The records in dynaset-type Recordset objects are updatable if their Updatable property is True, the Field being changed is updatable, and the data page containing the current record isn't locked.

early bound

A form of binding where object variables are declared as variables of a specific class. Object references that use early-bound variables usually run faster than those that use late-bound variables.

Error statement

A keyword used in Error Function, Error Statement, On Error Statement. Error is also a Variant subtype indicating a variable is an error value.

error trapping

An action recognized by an object, such as clicking the mouse or pressing a key, and for which you can write code to respond. Events can occur as a result of a user action or program code, or they can be triggered by the system.

error-handling routine
User-written code that deals with some kinds of errors at run time.

event

An action recognized by an object, such as clicking the mouse or pressing a key, and for which you can write code to respond. Events can occur as a result of a user action or program code, or they can be triggered by the system.

event procedure
A procedure automatically invoked in response to an event initiated by the user, program code, or system. Event procedures are private by default.

event-driven
Describes an application that responds to actions initiated by the user or program code, or that are triggered by the system.

exception handling
Where a service is able to inform their client in some uniform way that an exception was raised or encountered.

exclusive

Indicates whether a database or a table can be shared by other users in a multiuser environment. If the database or table is opened for exclusive use, it can't be shared by other users.

executable code

Code that Visual Basic translates into a specific action at run time, such as carrying out a command or returning a value. In contrast, nonexecutable code defines variables and constants.

executable file

A Windows-based application that can run outside the development environment. An executable file has an .EXE filename extension.

executable statement

A statement that Visual Basic translates into a specific action at run time. Most Visual Basic statements are executable. The main exceptions are declarations, constant definitions, comments, and user-defined type definitions.

explicit declaration
A declaration in which a variable is explicitly declared using DIM, STATIC, PUBLIC, or PRIVATE statements.

expose
To make available to other applications via Automation. An exposed object can be a document, a paragraph, a sentence, a graph, and so on.

extender object

An object, containing methods and properties, that is attached to another object, thereby 'extending' the latter's capabilities.

external database

Either an ODBC database such as SQL Server that resides on a remote server, or one of the external databases such as Btrieve, Paradox, dBase, FoxPro, Microsoft Excel or Microsoft Access.

foreign table

A database table used to contain foreign keys. Generally, you use a foreign table to establish or enforce referential integrity. The foreign table is usually on the many side of a one-to-many relationship. An example of a foreign table is a table of state codes or customer orders.

form code
All the procedures and declarations saved in the same file as a form and its controls.

form module

A file in a Visual Basic project with an .FRM filename extension that can contain graphical descriptions of a form; its controls and their property settings; form-level declarations of constants, variables, and external procedures; and event and general procedures.

form-level variable
A variable recognized by all procedures attached to a form.

forward-only scrolling snapshot

A snapshot-type Recordset object in which records can be searched only from beginning to end; the current record position can't be moved back toward the first record. Forward-scrolling snapshots are useful for quickly scanning data, such as when you're searching for a particular record.

function pointer
A stored memory location of a function's address.

Function procedure

A procedure that performs a specific task within a Visual Basic program and returns a value. A Function procedure begins with a Function statement and ends with an End Function statement.

general procedure
A procedure that must be explicitly called by another procedure. In contrast, an event procedure is invoked automatically in response to a user or system action.

GUID
Globally unique identifier used to identify objects and interfaces precisely.

handle

A unique integer value defined by the operating environment and used by a program to identify and access an object, such as a form or control.

IID

implicit declaration
Declaration that occurs when a variable is used in a procedure without previously declaring its name and type.

inner join

A join in which records from two tables are combined and added to a Recordset only if the values of the joined fields meet a specified condition. For instance, an equi-join is an inner join in which the values of the joined fields must be equal.

installable ISAM

A driver you can specify that allows access to external database formats such as Btrieve, dBase, Microsoft Excel, and Paradox. ISAM is an acronym for Indexed Sequential Access Method. The Microsoft Jet database engine installs (loads) these ISAM drivers when referenced by your application. The location of these drivers is maintained in the Microsoft Windows registration database.

instance

Any one of a set of objects sharing the same class. For example, multiple instances of a Form class share the same code and are loaded with the same controls with which the Form class was designed. During run time, the individual properties of controls on each instance can be set to different values.

instantiate

interface
A set of semantically related functions (methods) used to manipulate data.

interface identifier (IID)
Unique identifier tag associated with each interface; applications use the IID to reference the interface in function calls.

interface negotiation
The process by which an object or container can query another object about a specified interface and have the object return a pointer to that interface if it is supported.

intrinsic constant

A constant provided by an application. Visual Basic constants are listed in the Visual Basic object library and can be viewed using the Object Browser.

ISAM
Indexed sequential access method.

IUnknown
A base interface that describes the group of functions all objects must support.

join

A database operation that combines some or all records from two or more tables, such as an equi-join, outer join, or self-join. Generally, a join refers to an association between a field in one table and a field of the same data type in another table. You create a join with an SQL statement.

When you define a relationship between two tables, you create a join by specifying the primary and foreign table fields. When you add a table to a query, you need to create a join between appropriate fields in the SQL statement that defines the query.

key

The Windows Registry stores data in a hierarchically structured tree. Each node in the tree is called a key. Each key can contain both subkeys and data entries called values.

In a database, a key is a column used as a component of an index.

keyset
The set of key values that are buffered in the client by DB-Library.

late bound

Object references are late-bound if they use object variables declared as variables of the generic Object class. Late-bound binding is the slowest form of binding, because Visual Basic must determine at run time whether or not that object will actually have the properties and methods you used in your code.

locked
The condition of a data page, Recordset object, or Database object that makes it read-only to all users except the one who is currently entering data in it.

locking
A system of ensuring that two processes do not try to affect the same record in a database at the same time.

marshaling
The processing of packaging and sending interface parameters across process boundaries.

MDI

MDI child

A form contained within an MDI form in a multiple-document interface (MDI) application. To create a child form, set its MDIChild property to True.

MDI form

A window that makes up the background of a multiple-document interface (MDI) application. The MDI form is the container for any MDI child forms in the application.

member
A constituent element of a collection, object, or user-defined type.

member function
One of a group of related functions that make up an interface.

method
A member function of an exposed object that performs some action on the object.

modal

Describes a window or dialog box that requires the user to take some action before the focus can switch to another form or dialog box.

modeless

Describes a window or dialog box that does not require user action before the focus can be switched to another form or dialog box.

module
A set of declarations and procedures.

module level

Describes code in the declarations section of a module. Any code outside a procedure is referred to as module-level code. Declarations must be listed first, followed by procedures. For example:

```
Dim X As Integer     'This is a module-level variable declaration
Const RO = "Readonly"    'This is a module-level constant declaration
Type MyType 'This is a module-level user-defined type declaration
    MyString As String
    MyAge As Integer
End Type
```

module variable

A variable declared outside of Function, Sub, or Property procedure code. Module variables must be declared outside any procedures in the module. They exist while the module is loaded, and are visible in all procedures in the module.

multiple-document interface (MDI)
Interface that allows you to create an application that maintains multiple forms within a single container form. Microsoft Excel and Word for Windows are examples of applications that have MDIs.

multiple-document interface (MDI) application

An application that can support multiple documents from one application instance. MDI object applications can simultaneously service a user and one or more embedding containers.

multiple-object application
An application that is capable of supporting more than one class of object; for example, a spreadsheet program might support charts, spreadsheets, and macros.

named argument

An argument that has a name that is predefined in the object library. Instead of providing values for arguments in the order expected by the syntax, you can use named arguments to assign values in any order. For example, suppose a method accepts three arguments:

```
DoSomeThing namedarg1, namedarg2, namedarg3
```

By assigning values to named arguments, you can use the following statement:

```
DoSomething namedarg3:=4,namedarg2:=5,namedarg1:=20
```

Note that the arguments need not be in their normal positional order.

Null

A value indicating that a variable contains no valid data. Null is the result of:

® An explicit assignment of Null to a variable.

® Any operation between expressions that contain Null.

null

A value that indicates missing or unknown data. Null values can be entered in fields for which information is unknown and in expressions and queries. In Visual Basic, the Null keyword indicates a Null value. Some fields, such as those defined as containing the primary key, can't contain null values.

null field

A field containing no characters or values. A null field isn't the same as a zero-length string (" ") or a field with a value of 0. A field is set to null when the content of the field is unknown. For example, a Date Completed field in a task table would be left null until a task is completed.

object

A combination of code and data that can be treated as a unit, for example a control, form, or application. Each object is defined by a class.

An object is an instance of a class that combines data with procedures.

Automation: A unit of information that resides in a compound document and whose behavior is constant no matter where it is located or used.

Object Browser
A dialog box that lets you examine the contents of an object library to get information about the objects provided.

object class
A type of object that is registered in the registration database and serviced by a particular server.

object data type
A collective term for objects defined by Visual Basic or by the Microsoft Jet database engine.

Object Description Language (ODL)
The language used to describe an application's interface. ODL scripts are compiled into type libraries using the MkTypLib tool.

object expression

An expression that specifies a particular object. This expression may include any of the object's containers. For example, if your application has an Application object that contains a Document object that contains a Text object, the following are valid object expressions:

® Application.Document.Text

® Application.Text

® Documention.Text

® Text

object library

Data stored in a .olb file or within an executable (.exe, .dll, or .ocx) that provides information used by Automation controllers (such as Visual Basic) about available Automation objects. You can use the Object Browser to examine the contents of an object library to get information about the objects provided.

object model

object type

A type of object exposed by an application through Automation; for example, Application, File, Range, and Sheet. Refer to an application's documentation (Microsoft Excel, Microsoft Project, Microsoft Word, and so on) for a complete listing of available objects.

object variable
A variable that contains a reference to an object.

ODBC data source

A term used to refer to a database or database server used as a source of data. ODBC data sources are referred to by their Data Source Name. Data sources can be created using the Windows control panel or the RegisterDatabase method.

ODL

Open Database Connectivity (ODBC)

A standard protocol that permits applications to connect to a variety of external database servers or files. ODBC drivers used by the Microsoft Jet database engine permit access to Microsoft SQL Server and several other external databases.

The ODBC applications programming interface (API) may also be used to access ODBC drivers and the databases they connect to without using the Jet engine.

optimistic concurrency control

Concurrency: An attempt to maximize the number of simultaneous transactions, or users.

Optimistic Concurrency control: You can use Browse Mode, or DB-Library® cursors, to implement optimistic concurrency control. In this mode, no locks are held, under the assumption that no other transactions will change the data being accessed. When the transaction is committed, if another transaction has indeed changed the data, the original transaction is required to retrieve the new data and resubmit the query.

out of scope

When a variable loses focus or is out of scope. Scope is defined as the visibility of a variable, procedure, or object. For example, a variable declared as Public is visible to all procedures in all modules in a directly referencing project (unless Option Private Module is in effect). Variables declared in procedures are visible only within the procedure and lose their value between calls unless they are declared Static.

parameter query

A query that requires you to provide one or more criteria values, such as Redmond for City, before the query is run. A parameter query isn't, strictly speaking, a separate kind of query; rather, it extends the flexibility of other queries.

parent project
A project that contains one or more subprojects. A project can be both a parent project and a subproject at once, if it is in the middle of the project hierarchy.

pass-through query

An SQL-specific query you use to send commands directly to an SQL database server (such as Microsoft SQL Server). With pass-through queries, you work with the tables on the server instead of attaching them. Pass-through queries are used to execute SQL queries and system specific commands written using SQL dialects known only to the server.

A pass-through query may or may not return records. If it does, they are always returned in a snapshot.

pessimistic

A type of locking in which the page containing one or more records, including the record being edited, is unavailable to other users when you use the Edit method and remains unavailable until you use the Update method. Pessimistic locking is enabled when the Lockedits property of the Recordset object is set to True.

pointer
In programming, a variable that contains the memory location of data rather than the data itself.

primary key
One or more fields whose value or values uniquely identify each record in a table. Each table can have only one primary key. An Employees table, for example, could use the social security number for the primary key.

primary table

A database table used to contain primary keys. Generally, a primary key table is used to establish or enforce referential integrity. The primary table is usually on the one side of a one-to-many relationship with a foreign table.

Private
Private variables are available only to the module in which they are declared.

procedure
A named sequence of statements executed as a unit. For example, Function, Property, and Sub are types of procedures.

Procedure box

A list box at the upper-right of the Code and Debug windows that displays the procedures recognized for the object displayed in the Object box.

procedure call
A statement in code that tells Visual Basic to execute a procedure.

procedure level

Describes statements located within a Function, Property, or Sub procedure. Declarations are usually listed first, followed by assignments and other executable code. For example:

```
Sub MySub() ' This statement declares a sub procedure block.
    Dim A   ' This statement starts the procedure block.
    A = "My variable"    ' Procedure-level code.
    Debug.Print A    ' Procedure-level code.
End Sub ' This statement ends a sub procedure block.
```

Note In contrast, module-level code resides outside any procedure blocks.

---

procedure stepping
A debugging technique that allows you to trace code execution one statement at a time. Unlike single stepping, procedure stepping does not step into procedure calls; instead, the called procedure is executed as a unit.

procedure template

The beginning and ending statements that are automatically inserted in the Code window when you specify a Sub, Function, or Property procedure in the Insert Procedure dialog box.

programmability
The ability of a server to define a set of properties and methods and make them accessible to Automation controllers.

programmable
Capable of accepting instructions for performing a task or operation. A programmable object (Automation object) can be manipulated programmatically with its methods and properties.

project

A group of related files, typically all the files required to develop a software component. Files can be grouped within a project to create subprojects. Projects can be defined in any way meaningful to the user(s)—as one project per version, or one project per language, for example. In general use, projects tend to be organized in the same way file directories are.

project file
A file with a .VBP filename extension that keeps track of the files, objects, project options, environment options, EXE options, and references associated with a project.

Project window

A window that displays a list of the form, class, and standard modules; the resource file; and references in your project. Files with .OCX and .VBX filename extensions don't appear in this window.

Properties window

A window used to display or change properties of a selected form or control at design time. Some custom controls have customized Properties windows.

property

A named attribute of an object. Properties define object characteristics such as size, color, and screen location, or the state of an object, such as enabled or disabled.

A property is a data member of an exposed object. Properties are set or returned by means of get and let accessor functions.

Property list
A two-column list in the Properties window that shows all the properties and their current settings for the selected object.

Property procedure

A procedure that creates and manipulates properties for a class module. A Property procedure begins with a Property Let, Property Get, or Property Set statement and ends with an End Property statement.

property setting
The value of a property.

proxy
An interface-specific object that packages parameters for that interface in preparation for a remote method call. A proxy runs in the address space of the sender and communicates with a corresponding stub in the receiver's address space.

Public

Variables declared using the Public statement are available to all procedures in all modules in all applications unless Option Private Module is in effect; in which case, the variables are public only within the project in which they reside.

query
A formalized instruction to a database to either return a set of records or perform a specified action on a set of records as specified in the query. For example, the following SQL query statement returns records:

® SELECT [Company Name] FROM Publishers WHERE State = 'NY'

You can create and run select, action, crosstab, parameter, and SQL-specific queries.

query parameter data types

The set of data types for a field in a parameter query. The Microsoft Jet database engine has 13 query parameter data types.

Category	Data type
Table fields	Currency, Date/Time, Memo, Automation Object, Text, and Yes/No correspond to the same data types in table fields.
Number	Byte, Single, Double, Integer, and Long Integer correspond to the five Field Size options of the Number data type in table fields.
Generic	Value is a Generic data type that doesn't accept any type of data.
Binary	You can use the Binary data type in parameter queries directed to attached tables that recognize it.

QueryDef

A data access object that contains the SQL statement that describes a query and its properties, such as DateCreated and ODBCTimeout. Querydef objects with an ODBC Connect property may contain an SQL statement that can only be executed on an external database like Microsoft SQL Server. This is called an SQL pass-through query.

Recordset
A logical set of records. The three types of Recordset objects are dynaset, snapshot, and table.

reference count

The number of instances of an object loaded. This number is incremented each time an instance is loaded and decremented each time an instance is unloaded. Ensures an object is not destroyed before all references to it are released.

referential integrity

Rules that you set to establish and preserve relationships between tables when you add, change, or delete records. Enforcing referential integrity prohibits users from adding records to a related table for which there is no primary key, changing values in a primary table that would result in orphaned records in a related table, and deleting records from a primary table when there are matching related records.

If you select the Cascade Update Related Fields or Cascade Delete Related Records option for a relationship, the Microsoft Jet database engine allows changes and deletions but changes or deletes related records to ensure that the rules are still enforced.

registration
The process of adding a class, container, or object to the registration database.

registration database
A database that provides a system-wide repository of information for containers and servers that support Automation.

remote procedure call (RPC)

A mechanism through which applications can invoke procedures and object methods remotely across a network. Using RPC, an application on one machine can call a routine or invoke a method belonging to an application running on another machine.

reusable code
Software code written so that it can be used in more than one place.

row

A set of related columns or fields used to hold data. A row is synonymous with a record in the Microsoft Jet database engine. A table is composed of zero or more rows of data.

run time
The time when code is running. During run time, you interact with the code as a user would.

run-time error

An error that occurs when code is running. A run-time error results when a statement attempts an invalid operation.

scope

Defines the visibility of a variable, procedure, or object. For example, a variable declared as Public is visible to all procedures in all modules in a directly referencing project (unless Option Private Module is in effect). Variables declared in procedures are visible only within the procedure and lose their value between calls unless they are declared Static.

select query

A query that asks a question about the data stored in your tables and returns a Recordset object without changing the data. Once the Recordset data is retrieved, you can examine and make changes to the data in the underlying tables. In contrast, action queries can make changes to your data, but they don't return data records.

sequential access
A type of file access that allows you to access records in text files and variable-length record files sequentially; that is, one after another.

server

An application or DLL that provides its objects to other applications. You can use any of these objects in your Visual Basic application.

services model

A way of viewing applications as a set of features or services that are used to fulfill consumer requests. By encouraging the developer to model an application as a collection of discrete services, features and functionality can be packaged for reuse, sharing, and distribution across functional boundaries.

services-based architecture

An application model in which the feature set of the business application is expressed conceptually as a collection of services. These services can be grouped based on common characteristics, for example, sematics, behavior, analysis and design techniques. MSF identifies three such groupings of services: user services, business services, and data services.

session

A session delineates a sequence of operations performed by the Microsoft Jet database engine. A session begins when a user logs on and ends when a user logs off. All operations performed during a session form one transaction scope and are subject to permissions determined by the logon user name and password. Sessions are implemented as Workspace objects in DAO.

set
To assign a value to a property.

single-object application
An server that exposes only one class of object.

snapshot

A static copy of a set of records retrieved from the database and copied into memory. Snapshot-type Recordset objects can be created from a base table, a query, or another recordset. Snapshots are not updatable. All SQL pass-through queries return snapshots.

snapshot-type Recordset

A snapshot-type Recordset object is a static set of records that you can use to examine data in an underlying table or tables. A snapshot-type Recordset can contain fields from one or more tables in a database but can't be updated. One of the three types of Recordset objects.

SQL database
A database that can be accessed through use of Open Database Connectivity (ODBC) data sources.

SQL pass-through query

SQL PassThrough permits your application to effectively bypass the Microsoft Jet database engine query processor and route foreign-dialect SQL statements to an external database server like SQL Server.

This technique is used to pass single or multiple SQL commands, stored procedures, or other SQL statements to an external database server to be executed. Whenever you specify a SQL PassThrough query, the Jet database query processor is bypassed, but its recordset processor is used to create and manage the result sets that may be generated by your query.

SQL Server

A relational database engine running on a network-accessible server. SQL Servers are responsible for comprehensive management of one or more relational databases residing on the server. They are controlled by and information is passed to and from these servers by way of Structured Query Language (SQL) statements. There are two types of SQL Server: Microsoft SQL Server and Sybase SQL Server.

SQL statement/string

1. An expression that defines a Structured Query Language (SQL) command, such as SELECT, UPDATE, or DELETE, and includes clauses such as WHERE and ORDER BY. SQL strings and statements are typically used in queries, Recordset objects, and aggregate functions but can also be used to create or modify a database structure.
2. A set of commands written using a dialect of Structured Query Language used to retrieve or pass information to a relational database. SQL statement syntax is determined by the SQL Server or other relational database engine it is intended to execute on.

stack
A fixed amount of memory used by Visual Basic to preserve local variables and arguments during procedure calls.

Static
A Visual Basic keyword you can use to preserve the value of a local variable.

static cursor

Neither the cursor owner nor any other user can change the results set while the cursor is open. Values, membership, and order remain fixed until the cursor is closed. You can either take a 'snapshot' (temporary table) of the results set, or you can lock the entire results set to prevent updates. When you take a snapshot of the results set, the results set diverges increasingly from the snapshot as updates are made.

Structured Query Language (SQL)

A language used in querying, updating, and managing relational databases. SQL can be used to retrieve, sort and filter specific data to be extracted from the database.

You can use SQL SELECT statements anywhere a table name, query name, or field name is accepted. For example, you can use an SQL statement in place of a table name in the OpenRecordset method.

Sub procedure

A procedure that performs a specific task within a program, but returns no explicit value. A Sub procedure begins with a Sub statement and ends with an End Sub statement.

subroutine
A section of code that can be invoked (executed) within a program.

synchronous call

A function call in which the caller waits for the reply before continuing. Most interface methods are synchronous calls. An operation that completes synchronously performs all of its processing in the function call made by the application. The function returns different values depending on its success or failure.

synchronous processing
When the data interface blocks until an operation is complete or at least until the first row of the results is ready. Opposite of asynchronous processing.

table
The basic unit of data storage in a relational database. A table stores data in records (rows) and fields (columns) and is usually about a particular category of things, such as employees or parts. Also called a base table.

Table object

A type of recordset that is a logical representation of a physical object within a database that contains data about a particular subject. Like all recordsets, a Table object has records (rows) and fields (columns). The Table object is outdated. It is recommended that you use the table-type Recordset instead.

TableDef object

An object in a database that defines the structure of a table. In contrast, a table is the physical container of the data in a database.

table-type Recordset
A logical representation of a base table in a database.

three-tiered architecture

An application model in which the feature set of the business application is expressed conceptually as three layers, each of which supplies services to the adjacent layers: user presentation, core business, and data management. Note that this is a conceptual architecture; while all business applications can be expressed conceptually in terms of the these three layers, the implementation architecture may not be three layers.

transaction

A series of changes made to a database's data and schema. Mark the beginning of a transaction with the BeginTrans statement, commit the transaction using the CommitTrans statement, and undo all your changes since BeginTrans using the Rollback statement.

There is an implicit transaction while action queries are running. If a query doesn't complete for any reason, it is automatically rolled back.

Transactions are optional and can be nested up to five levels. Transactions increase the speed of data changing operations and enable changes to be reversed easily.

Transactions are global to the referenced database object's Workspace.

trigger

Record-level event code that runs after an insert, update, or delete. Different actions can be attached to the different events. Triggers run last, after rules, and don't run during buffered updates. They are most often used for cross-table integrity.

type description

The information used to build the type information for one or more aspects of an application's interface. Type descriptions are written in Object Description Language (ODL) and include both programmable and nonprogrammable interfaces. The component of an Automation controller used to write programming tools that create type libraries. The type description interfaces provide a way to read and bind to the descriptions of objects in a type library. The descriptions are used by Automation controllers when they browse, create, and manipulate Automation objects.

type information

1. Information that describes the interfaces of an application. Type information is created from type descriptions by using Automation tools, such as the **MkTypLib** or **CreateDispTypeInfo** function. Type information can be accessed through the **ITypeInfo** interface.
2. Type information is the Automation standard for describing exposed objects, properties, and methods to an application or programming tool that accesses an exposed object. You provide type information in one of two ways: As a type library written in Microsoft Object Description Language (ODL) and compiled by **MkTypLib**, or as a data structure exported at run time.

type library

A compound document file that includes information about types and objects exposed by an server. May contain any of the following:

® Information about data types

® Descriptions of one or more objects (each of these descriptions is commonly referred to as a typeinfo)

® References to type descriptions from other type libraries

You can ship a type library in any of the following forms: as a resource in a DLL; as a resource in an .EXE file; as a subfile within a compound document file (sometimes called a 'docfile'); or as a stand-alone binary file.

union query

A SQL-specific select query that creates a snapshot-type Recordset object containing data from all specified records in two or more tables with any duplicate records removed. To include the duplicates, add the keyword ALL.

For instance, a union query of the Customers table and the Suppliers table results in a snapshot-type Recordset that contains all suppliers that are also customers.

unmarshaling

The processing of unpackaging parameters that have been sent across process boundaries. In a given call, the method arguments are marshalled and unmarshalled in one direction, while the return values are marshalled and unmarshalled in the other direction.

update query
An action query that changes a set of records according to criteria you specify. An update query doesn't return any records.

validation
The process of checking whether entered data meets certain conditions or limitations.

validation properties

Properties used to set conditions on table fields and records. Validation properties include ValidationRule, Required and AllowZeroLength.

validation rule

A rule that sets limits or conditions on what can be entered in one or more fields. Validation rules can be set for a Field or a TableDef object. Validation rules are checked when you update a record containing fields requiring validation. If the rule is violated, a trappable error results.

VBSQL
Visual Basic library for SQL Server.

Windows API

The Windows API (Application Programming Interface) consists of the functions, messages, data structures, data types, and statements you can use in creating applications that run under Microsoft Windows. The parts of the API you use most are code elements included for calling API functions from Windows. These include procedure declarations (for the Windows functions), user-defined type definitions (for data structures passed to those functions), and constant declarations (for values passed to and returned from those functions).

Click here to connect to the Knowledge Base page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Knowledge Base contains detailed technical articles and samples that are created and maintained by Microsoft Product Support Services (PSS). This comprehensive database is updated daily and contains more than 50,000 detailed articles with technical information about Microsoft products, fix lists, documentation errors, and answers to commonly asked technical support questions. The Knowledge Base is a primary Microsoft product information source used by Microsoft support engineers every day.

This course includes lab exercises that give you experience using the skills presented in the chapters. Most of the labs contain code to get you started and code for the lab solution. You can access the labs from within each chapter, or by clicking **Labs** on the **Contents** menu.

If you selected the Complete installation option during Setup, the lab files are copied to the Labs folder in the location where you installed the application. If you selected the Typical installation option, the lab files will not be copied to your local computer. To install the lab files on your local computer, re-install the application, and select the Complete installation option during Setup.

{ewc MVIMG, MVIMAGE,!library.shg}

This course includes a number of audio and video demonstrations and animations. They illustrate the concepts and techniques discussed during the course.

When you see this icon {ewc MVIMG, MVIMAGE,!democlip.bmp} in the course contents, click it to launch a demonstration of the topic being discussed.

When you see this icon {ewc MVIMG, MVIMAGE,!anim.bmp} in the course contents, click it to launch an animation of a chapter introduction.

Note To see the text of a demonstration or animation, click **Closed Caption** on the **View** menu.

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **45 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 1.

Exercise 1: Creating Forms

In this exercise, you will create two simple forms for your application.

Exercise 2: Adding Code

In this exercise, you will add code to make the application functional.

Exercise 3: Creating an Error-Handling Routine

In this exercise, you will trap errors in an event procedure by using an error-handling routine.

Exercise 4: Restricting Input

In this exercise, you will restrict user input on a form.

Exercise 5: Creating an Executable File

In this exercise, you will compile your project into an executable application, and then test the application.

By the end of this lab, you will be able to:

® Create forms and controls.
® Write code in event procedures.
® Trap and handle a run-time error.
® Restrict input in a **TextBox** control.
® Compile an executable application.

Before working on this lab, you should be familiar with the following concepts:

® The contents of this chapter.

To complete this lab, you need the following setup:

® Microsoft Visual Basic 5.0 or later.

In this exercise, you will create two simple forms for an application.

Create the main form

1. Start a new Visual Basic project.
2. Add controls to the form, as shown in the following illustration.

   {ewc MVIMG, MVIMAGE,!v01g045.bmp}

   For more information about adding controls to a form, see Using Controls.
3. Using the Visual Basic Menu Editor, add menus to the form, as shown in the following illustration.

   {ewc MVIMG, MVIMAGE,!v01g050.bmp}
3. Save the project in the folder \<*Install Folder*>\Labs\Lab01.

Create the About form

1. Add a new form to the project that will be used as the About box for the application.
2. Add controls to the form, as shown in the following illustration.

   {ewc MVIMG, MVIMAGE,!v01g055.bmp}

   The picture in the upper-left corner is an **Image** control. You can select any icon you want from the Visual Basic \Icons folder.
3. Save the project.

In this exercise, you will add code to make the application functional.

Add code for the temperature conversion

1. In the KeyUp event for the Celsius text box, convert the number entered to Fahrenheit and place the converted number in the Fahrenheit text box.

   The conversion formula is: Fahrenheit = (Celsius * 9/5) + 32
2. In the KeyUp event for the Fahrenheit text box, convert the entered number to Celsius, and place the converted number in the Celsius text box.

   The conversion formula is: Celsius = (Fahrenheit - 32) * 5/9.
3. Test the application. What happens if you enter text characters in one of the text boxes?

To see an example of how your code should look, click this icon.
{ewc MVIMG, MVIMAGE,!code.bmp}

Add code for menus and an OK button

1. In the Click event for the **Exit** menu command, **Unload** the form and **End** the application.
2. In the Click event for the **About** menu command, **Show** the **About** box as a modal form.

   For information about modal forms, see the **Show** method in the Help for Visual Basic.
3. In the Click event for the **OK** button on the About form, **Unload** the About form.
4. Save the project.
5. Test the application.

To see an example of how your code should look, click this icon.
{ewc MVIMG, MVIMAGE,!code.bmp}

In this exercise, you will compile your project into an executable application, and then test the application.

u **Create an executable file**

1. On the **File** menu, click **Make <Project>.exe**.
2. Name the application Convert.exe, and save it in the folder \<*Install Folder*>\Labs\Lab01.
3. To compile the application, click OK.

u **Test the executable file**

1. Using the Windows Explorer, navigate to the folder \<*Install Folder*>\Labs\Lab01.
2. Test the application.

```
Private Sub txtCelsius_KeyUp(KeyCode As Integer, Shift As Integer)
    txtFahrenheit.Text = (txtCelsius.Text * 9 / 5) + 32
End Sub

Private Sub txtFahrenheit_KeyUp(KeyCode As Integer, Shift As Integer)
    txtCelsius.Text = (txtFahrenheit.Text - 32) * 5 / 9
End Sub
```

Main Form Code

```
Private Sub mnuFileExit_Click()
    Unload Me
    End
End Sub

Private Sub mnuHelpAbout_Click()
   frmAbout.Show vbModal
End Sub
```

About Form Code

```
Private Sub cmdOK_Click()
    Unload Me
End Sub
```

In this exercise, you will restrict user input for the **Username** text box to a limited length and to use only uppercase characters.

Restrict user input

1. Use the **MaxLength** property of the **Username** text box to limit the number of characters that can be entered to 15.
2. In the KeyPress event of **Username** text box, convert all characters entered in the text box to uppercase characters.

   For information about validating text in text boxes, see Validating Field Information.
3. Save the project.
4. Test the application.

   Is text box input limited to 15 characters? Are all characters converted to uppercase characters?

In this exercise, you will add error handling to trap for the Type Mismatch run-time error that occurs if users type a non-numeric value into one of the text boxes.

Create an error handler

1. In the KeyUp event procedure for the Celsius **TextBox** control, add an **On Error Goto** statement to enable error handling.
2. Create the error-handling routine specified in the **On Error Goto** statement.
3. In the error-handling routine, add code to test for run-time error 13: Type Mismatch.

   For information about error handling, see Handling Run-Time Errors.
4. If error 13 is found, set the **Text** property of the txtFarhenheit text box to **Can't Convert**.
5. Test the application.

   To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **60 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

In this lab, you will be working with the Nwind.mdb database. To see the schema for this database, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Note The Nwind.mdb database for this lab is located in the <*Install Folder*>\Labs directory and is accessed by the lab through a relative path. To ensure that the lab solution project finds the database (that is, that the solution directory is the current directory), open the solution project in Visual Basic 5 by double-clicking the .vbp file in Explorer. If you start Visual Basic and then open the project, the Visual Basic installation directory will be the current directory and the database file will not be found.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 2.

Exercise 1: Creating a Data Entry Form

In this exercise, you will use the Data Form Wizard to create a Master/Detail data entry form using the Orders and Order Details tables in the Nwind.mdb database.

Exercise 2: Using the DBCombo Control

In this exercise, you will enhance the Orders form by replacing the Employee ID text box with a data-bound combo (**DBCombo**) control.

Exercise 3: Using the Validate Event

In this exercise, you will use the **Data** control's Validate event to confirm changes made to the Orders table.

In this lab, you will use the **Data** control to build a data-entry form.

After completing this lab, you will be able to:

® Use the Data Form Wizard to build a Master/Detail form.
® Manipulate a database by using the **Data** control and data-bound controls.
® Use the **DBGrid** control to view and edit data.
® Use the **DBCombo** control for data entry.

Before working on this lab, you should be familiar with the following concepts:

® Basic understanding of databases

® The contents of this chapter

To complete this lab, you need the following setup:

® Visual Basic 5.0 or later

In this exercise, you will use the Data Form Wizard to create a Master/Detail data-entry form by using the Orders and Order Details tables in the Nwind.mdb database.

u Create the Orders form

1. Create a new Standard EXE project.
2. Using the Add-In Manager, load the Data Form Wizard.

   The Data Form Wizard command should now appear on the **Add-Ins** menu.
3. On the **Add-Ins** menu, click **Data Form Wizard**.
4. Select the Data Form Wizard options, as shown in the following table.

   For more information about the Data Form Wizard, see Using the Data Form Wizard.

Option	Value
Database format	Access
Database Name	<*VB Install Folder*>\Nwind.mdb
Form Layout	Master/Detail
Master Record Source	Orders table
Master Fields	OrderID, CustomerID, EmployeeID, OrderDate, ShipName
Detail Record Source	Order Details table
Detail Fields	All fields in the Order Details table
Field to link Master and Detail	OrderID
Available Control	All
Form Name	frmOrders

5. In the **Project Properties** dialog box, set **Startup Object** to **frmOrders**.
6. Run the application.

   Your form should resemble the following illustration.

   {ewc mvimg, mvimage,!v02g015.bmp}

Note The Data Form Wizard creates a control array for all text boxes on a form. For more information about control arrays, search on “control arrays” in the Visual Basic Help index.

---

u Examine the form created by the Data Form Wizard

1. Quit the application, and note the following properties and events of the form.
   a. The **DatabaseName** and **RecordSource** properties of the **Data** control.
   b. The **Data** control event procedures, in particular the Reposition event, which ensures that the Master/Detail records are synchronized.
   c. The **DataSource** and **DataField** properties of each of the text boxes.
   d. The Click event procedure for each of the command buttons.
   e. The **DataSource** property of the **DBGrid** control.
2. Save the form as Orders.frm, and save the project as Orders.vbp in the folder <*Install Folder*>\Labs\Lab02.

In this exercise, you will enhance the Orders form by replacing the Employee ID text box with a data-bound combo (**DBCombo**) control. This will enable users to select employees by name without having to know their ID.

u Add a data control

1. Open the Orders form.
2. Add a new **Data** control to the form.
3. Set the **Name** property to datEmployees.
4. Set the **DatabaseName** property to <*VB Install Folder*>\Nwind.mdb.
5. Set the **RecordSource** property to Employees.
6. Set the **Visible** property to **False**.

   The **Data** control is hidden because it is used only to supply information to the data-bound combo box. The user does not interact with this control directly.

u Add a DBCombo control

1. On the Orders form, replace the Employee ID text box with a data-bound combo box.

   You will need to add the control to the Toolbox before you use it on the form.
2. Set the **RowSource** and **ListField** properties of the data-bound combo box so it contains the Employee Last Name. Set the **RowSource** property to datEmployees, and the **ListField** property to LastName.
3. Set the **DataSource**, **DataField**, and **BoundColumn** properties of the data-bound combo box so that the correct data is returned to the Orders table.

   Set the **DataSource** property to datPrimaryRS (the **Data** control created by the Data Form Wizard), and set the **DataField** and **BoundColumn** properties to EmployeeID.

   {ewc mvimg, mvimage,!tip.bmp}
4. Test the application.

   Your form should resemble the following illustration.

   {ewc mvimg, mvimage, !v02g015.bmp}

   As you move between records, does the data bound combo box display the correct information?

   If you select a new employee, is that information saved correctly in the Orders table?

   For more information about using the **DBCombo** control, see Using the DBCombo Control.

In this exercise, you will use the **Data** control's Validate event to confirm changes made to the Orders table.

1. Edit the Validate event for the **Data** control in the Orders table.
2. If any edits have been made to the current record, use a message box to prompt the user to the save changes.
3. If the user answers No to saving changes, delete the changes by using the **Save** argument.
4. Run the application.
5. Edit the ShipName field, and move to the next record.

   You should be prompted to save the changes. Choose No, and check to see if the changes are canceled.

   If your code is not working properly, click this icon to see the solution for the lab.
   {ewc mvimg, mvimage,!code.bmp}

```
Dim iResponse As Integer
If Save = True Then 'data has changed
    'get rid of changes if user does not want to save
    iResponse = MsgBox("Save changes?", vbYesNo)
    If iResponse = vbNo Then
        Save = False
    End If
End If
```

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **60 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

In this lab, you will be working with the Nwind.mdb database. To see the schema for this database, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Note The Nwind.mdb database for this lab is located in the <*Install Folder*>\Labs directory and is accessed by the lab through a relative path. To ensure that the lab solution project finds the database (that is, that the solution directory is the current directory), open the solution project in Visual Basic 5 by double-clicking the .vbp file in Explorer. If you start Visual Basic and then open the project, the Visual Basic installation directory will be the current directory and the database file will not be found.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 3.

Exercise 1: Creating and Navigating a Recordset

In this exercise, you will use data access objects (DAO) to open a database and create a recordset.

Exercise 2: Adding and Editing Records

In this exercise, you will add new records and edit existing records using various DAO methods.

Exercise 3: Finding Records

In this exercise, you will let the user search for specific records in the Categories table, based on text within the Description field.

Exercise 4: Using Queries

In this exercise, you will use a saved query to return information from the Nwind.mdb database.

Exercise 5 (Optional): Disabling Buttons During Edit

In this exercise, you will enable and disable the appropriate command buttons while the user is navigating or editing data on the Categories form.

Exercise 6 (Optional): Using a Parameter Query

In this exercise, you will use a parameter query to return sales information from the Nwind.mdb database.

In this lab, you will build and test forms that use data access objects (DAO).

After completing this lab, you will be able to:

® Open a database by using DAO.
® Navigate and manipulate a recordset by using DAO.
® Find information in a table.
® Create a recordset based on a query.
® Use a parameter query.

Before working on this lab, you should be familiar with the following concepts:

® The data access object model.

® The contents of this chapter.

To complete this lab, you need the following setup:

® Visual Basic version 5.0 or later.

In this exercise, you will use data access objects (DAO) to open a database and create a recordset.

Create Categories form

1. Create a new Visual Basic Standard EXE project.
2. Name the default form frmCategories.
3. Add controls to the form, as shown in the following illustration.

   {ewc mvimg, mvimage,!v03g050.bmp}

Note Because the ID data will not be edited by the user, the Category ID field is a **Label** control.

Open a database and create a recordset

1. On the Categories form, declare two module-level variables, as shown in the following code:

```
Dim dbCurrent As Database
Dim recCategories As Recordset
```

2. Set a reference to the Microsoft DAO 3.5 Object Library.
3. In **Load** event procedure of the Categories form, follow these steps:
   a. Open \Program Files\DevStudio\VB\Nwind.mdb and save the resulting database object in the variable dbCurrent.
   b. Create a recordset by using the Categories table in the Nwind.mdb database, and capture the resulting **Recordset** object in the variable recCategories.
   c. Move to the first record in the recordset.
   d. Read the Category ID, Category Name, and Description fields into the appropriate controls. To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}
4. In the form **Unload** event procedure, **Close** the database.
5. Test the application. Is the first record in the Categories table displayed correctly?

Write code for the Move Previous and Move Next buttons

1. Add code to make the **Move Previous** and **Move Next** command buttons functional.
   - After a move, fill in the fields on the form with data.
   - Test for the beginning and the end of the recordset. To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

Test the project

1. Run the form.
2. Click the **Move Previous** button.
3. Click the **Move Next** button.

   You should see different Category records.

In this exercise, you will add new records and edit existing records using various DAO methods.

Modify the form

1. Modify the Categories form by adding the command buttons shown in the following illustration.

   {ewc mvimg, mvimage,!v03g055.bmp}

Write code for the Add, Edit, Save, Cancel, and Delete buttons

1. To clear the text fields and enter Add mode, write code for the **Add** button.

   Note Because the Category ID field is a counter field, it will automatically contain a new value as soon as you issue the **AddNew** method. You can then place the value for this field in the Category ID label. For example, execute the following code immediately after the **AddNew** method.

   ```
   lblCategoryID.Caption = recCategories("CategoryID")
   ```

   ---

2. To enter Edit mode, write code for the **Edit** button. For more information about editing records, see Updating Records.
3. To write changes from the text boxes back to the recordset and update it, write code for the **Save** button.
4. To cancel the pending update and update the fields with the current record in the recordset, write code for the **Cancel** button.
5. To delete the current record, write code for the **Delete** button. Move to the next record and retrieve the fields.

   To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

Test the application

1. Add a new record.
2. Cancel the Add. Is the previous information restored correctly?
3. Add and save a new record.
4. To ensure that you are on the correct record after the Add, test the **Move Next** and **Move Previous** buttons.
5. Delete the new record.
6. Try to delete the first category.

   The referential integrity constraints on this table should cause a run-time error. In a later lab, you will trap for that error.

In this exercise, you will let the user search for specific records in the Categories table, based on text within the Description field.

Write code for the Find button

1. Use the **InputBox** function to prompt users for a string that represents a portion of the Description field.
2. Using the string returned from the Input box, create an SQL string to select the matching records from the Categories table, as shown in this example:

   ```
   "select * from categories where [Description] like '*pasta*'"
   ```

3. Create a recordset based on the SQL query, and assign the results to the **recCategories** variable. This enables the other buttons to operate in the new recordset.
4. If no records are found (recCategories.RecordCount = 0), display a message saying that no records were found. Otherwise, display the first record in the new recordset.

   To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

Test the application

1. Run the form.
2. Click the **Find** button.
3. Enter **sweet** in the Input box.

   You should see two records that contain the text **sweet** in the description.

In this exercise, you will use a saved query to return information from the Nwind.mdb database.

Query for the ten most expensive products

1. Create a new form.
2. The saved query **Ten Most Expensive Products** in the Nwind.mdb database returns the ten highest priced products. In the **Form_Load** event procedure on the new form, create a recordset based on this query, and display the results in a **DBGrid** control, as shown in the following illustration.

   {ewc mvimg, mvimage,!v03g060.bmp}

   To see examples of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

   {ewc mvimg, mvimage,!tip.bmp}

Test the application

1. On the **Project** menu, click **Project Properties**.
2. Change the Startup Object to the new form, and then click OK.
3. Run the form.

   You should see the ten most expensive products in the grid.

In this exercise, you will enable and disable the appropriate command buttons while the user is navigating or editing data on the Categories form.

Enable and disable controls

1. When a user navigates in the Categories form, disable the text boxes, and the **Save** and **Cancel** button, so changes cannot be made to the data unless the user selects the **Edit** or **Add** button, as shown in the following illustration.

   {ewc mvimg, mvimage,!v03g065.bmp}

2. When a user selects the **Edit** or **Add** button, enable the text boxes, and the **Save** and **Cancel** buttons. Disable all other buttons on the form.

   After you have finished the form, it should resemble the following illustration.

   {ewc mvimg, mvimage,!v03g070.bmp}

   To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

3. In the **Save** and **Cancel** buttons, add code that:

   - Enables the command buttons **Add**, **Edit**, **Delete**, **Find**, **Move Previous**, and **Move Next**.
   - Disables the text boxes.
   - Disables the **Save** and **Cancel** buttons.

Test the application

1. Add a new record and save your changes.
2. Edit the new record and save your changes.
3. Edit the new record again and cancel your changes.

In this exercise, you will use a parameter query to return sales information from the Nwind.mdb database.

Use a parameter query

1. Create a new form that uses the **Employee Sales by Country** parameter query to display sales records in a date range, as shown in the following illustration.

   {ewc mvimg, mvimage,!v03g075.bmp}

   The parameter names for the query are **Beginning Date** and **Ending Date**. To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}
2. Test the application.

```
Private Sub Form_Load()
    Set dbCurrent = OpenDatabase("C:\Program Files\DevStudio\VB\Nwind.mdb")
    Set recCategories = dbCurrent.OpenRecordset("Categories")
    recCategories.MoveFirst
    FillFields
End Sub

Sub FillFields()
    ' this procedure will populate the fields on the form.
    ' it will be called from multiple procedures.
    lblCategoryID.Caption = recCategories.Fields("CategoryID")
    txtCategoryName.Text = recCategories.Fields("CategoryName")
    txtDescription.Text = recCategories.Fields("Description")
End Sub
```

```
Private Sub cmdPrevious_Click()
    recCategories.MovePrevious
    If recCategories.BOF Then
        Beep
        recCategories.MoveFirst
    Else
        FillFields
    End If
End Sub

Private Sub cmdNext_Click()
    recCategories.MoveNext
    If recCategories.EOF Then
        Beep
        recCategories.MoveLast
    Else
        FillFields
    End If
End Sub
```

```
Private Sub cmdEdit_Click()
    recCategories.Edit
End Sub

Private Sub cmdSave_Click()
    recCategories.Fields("Category Name") = txtCategoryName.Text
    recCategories.Fields("Description") = txtDescription.Text
    recCategories.Update
    recCategories.Bookmark = recCategories.LastModified
End Sub

Private Sub cmdCancel_Click()
    recCategories.CancelUpdate
    FillFields
End Sub

Private Sub cmdDelete_Click()
    recCategories.Delete
    recCategories.MoveNext
    If recCategories.EOF Then
        recCategories.MoveLast
    End If
    FillFields
End Sub
```

```
Private Sub cmdFind_Click()
    Dim strSQL As String
    Dim strAnswer As String
    strAnswer = InputBox("Enter any portion of the Description", "Find Records")
    strSQL = "select * from categories where [Description] like " & _
        "'*" & strAnswer & "*'"
    Set recCategories = dbCurrent.OpenRecordset(strSQL)
    If recCategories.RecordCount = 0 Then 'no records found
        MsgBox "No matching records found. Displaying all records."
        Set recCategories = dbCurrent.OpenRecordset("Categories")
    Else 'at least one record was found
        recCategories.MoveFirst
        FillFields
    End If
End Sub
```

```
Private Sub Form_Load()

    Dim dbCurrent As Database

    Set dbCurrent = DBEngine.Workspaces(0).OpenDatabase("C:\Program Files\
DevStudio\VB\Nwind.mdb")

    Set datProducts.Recordset = OpenRecordset("Ten Most Expensive Products")

End Sub
```

```
Private Sub cmdEdit_Click()
    ButtonEditAddMode
    rs.Edit
End Sub

Sub ButtonEditAddMode()
    cmdSave.Enabled = True
    cmdCancel.Enabled = True
    cmdAdd.Enabled = False
    cmdEdit.Enabled = False
    cmdDelete.Enabled = False
    cmdFind.Enabled = False
    cmdClose.Enabled = False
    cmdPrevious.Enabled = False
    cmdNext.Enabled = False
    txtCategoryName.Enabled = True
    txtDescription.Enabled = True
End Sub
```

```
Private Sub cmdExecute_Click()

    Dim strSQL As String
    Dim dbCurrent As Database
    Dim recSales As Recordset
    Dim qrySales As QueryDef


    Set dbCurrent = DBEngine.Workspaces(0).OpenDatabase _
                    ("C:\Program Files\DevStudio\VB\Nwind.mdb")
    Set qrySales = dbCurrent.QueryDefs("Employee Sales by Country")
    qrySales.Parameters("Beginning Date") = CDate(txtBegin)
    qrySales.Parameters("Ending Date") = CDate(txtEnd)
    Set recSales = qrySales.OpenRecordset()
    Set datSales.Recordset = recSales

End Sub
```

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **60 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

In this lab, you will be working with the Nwind.mdb database. To see the schema for this database, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Note The Nwind.mdb database for this lab is located in the <*Install Folder*>\Labs directory and is accessed by the lab through a relative path. To ensure that the lab solution project finds the database (that is, that the solution directory is the current directory), open the solution project in Visual Basic 5 by double-clicking the .vbp file in Explorer. If you start Visual Basic and then open the project, the Visual Basic installation directory will be the current directory and the database file will not be found.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 4.

Exercise 1: Handling Referential Integrity Violations

In this exercise, you will trap a run-time error produced by violating referential integrity rules.

Exercise 2: Multi-User Issues

In this exercise, you will trap run-time errors that occur when multiple users try to edit the same database information.

Exercise 3 (Optional): Using ODBCDirect

In this exercise, you will access external data stored in a Microsoft SQL Server database with ODBCDirect.

In this lab, you will add functionality to a project that uses advanced data access concepts.

After completing this lab, you will be able to:

® Handle referential integrity.
® Understand and work with a multi-user environment.
® Access external data with ODBCDirect.

Before working on this lab, you should be familiar with the following concepts:

® The data access objects (DAO) object model.

® The contents of this chapter.

To complete this lab, you need the following setup:

® Visual Basic version 5.0 or later.

In this exercise, you will trap a run-time error produced by violating referential integrity rules.

Check the referential integrity error

1. In the folder \Labs\Lab04, open NWindAdv.vbp.
2. Run the application.
3. Try to delete the first record. What happens?
4. End the application.

Trap the referential integrity violation

1. Open the Categories form.
2. In the Click event of the **Delete** button, add an error-handling routine to trap and handle the error produced above. In the handler, display a message box saying that the item was not deleted.
   {ewc mvimg, mvimage,!code.bmp}

Test the application

1. Run the application.
2. Try to delete the first category. What happens?

In this exercise, you will trap run-time errors that occur when multiple users try to edit the same database information.

u Add locking options

1. Add option buttons to the Categories form, as shown in the following illustration.

   {ewc mvimg, mvimage,!v04g050.bmp}

2. Set the **Value** property of the **Pessimistic** option button to **True**.
3. To set the **LockEdits** property, add code to each of the option buttons.
4. Add the following code to the **Save** button after the **Update** method, and to the **Cancel** button after the **CancelUpdate** method.

   ```
   DBEngine.Idle dbFreeLocks
   ```

   This will free any database locks that were set during the add or edit process.

5. Save the project.

u Test for multi-user errors

1. Create an executable for the application.
3. Run two instances of the application.
4. Using the **Pessimistic** locking option for both instances, try to edit the same record in both instances of Visual Basic, and note the error that occurs.
   {ewc mvimg, mvimage,!tip.bmp}
5. Using the **Optimistic** locking option for both instances, edit the same record in both instances of Visual Basic.

   In the first instance, change the Category Name and save the record.

   In the second instance, change the Category Name and try to save the record.

   Note the error that occurs.
   {ewc mvimg, mvimage,!tip.bmp}
6. In the first instance of Visual Basic, set locking to **Pessimistic**, and edit the first record.

   In the second instance of Visual Basic, set locking to **Optimistic**, edit the first record, change the Category Name, and save the record.

   Note the error that occurs.
   {ewc mvimg, mvimage,!tip.bmp}
7. End both instances of the application, and close the second instance of Visual Basic.

u Trap multi-user run-time errors

1. Add code to the **Edit** command to check for the error encountered in Step 4 of the previous procedure, and notify the user if an error occurs.
   {ewc mvimg, mvimage,!code.bmp}
2. Add an error-handling routine to the **Save** command Click event.

   In the error-handling code, check for the error encountered in Step 5 of the previous procedure. If the error is found, notify the user and cancel the update.

   In the error-handling routine, check for the error encountered in Step 6 of the previous procedure. If the error is found, notify the user.
   {ewc mvimg, mvimage,!code.bmp}
3. Save the application.
4. Repeat the steps in this procedure to test the application.

In this exercise, you will access external data stored in a Microsoft SQL Server database by using ODBCDirect.

u Query the external database

For this exercise, you will need a server running Microsoft SQL Server and its sample Pubs database.

1. Create a new Visual Basic standard EXE project.
2. Create a new form with a command button and a list box.

   Your form should resemble the following illustration.

   {ewc MVIMG, MVIMAGE,!V04G055.bmp}
3. In the Click event for the command button, do the following:
   a. Create an ODBCDirect workspace.
   b. Create a new connection to the workspace.
   c. Create a recordset based on the Authors table.

      To see a code sample of the SQL statement, click this icon.
      {ewc mvimg, mvimage,!code.bmp}
   d. Populate the list box with the last names (field au_lname) from the Authors table.

```
HandleError:
    If Err.Number = 3200 Then
        MsgBox "Can not delete a Category which has " & _
            "related products in the Products Table."
    Else
        MsgBox "Error " & Err.Number & ": " & Err.Description
    End If
```

```
Private Sub cmdEdit_Click()

On Error GoTo EditErrorHandler

    recCategories.Edit
    ButtonEditAddMode

Exit Sub

EditErrorHandler:

    Select Case Err.Number
    Case 3260 'page currently locked
        MsgBox "Record is currently locked. Try again later."

    Case 3197 'data has changed
        MsgBox "Data has been changed and will be refreshed."
        'get updated data
        recCategories.Bookmark = recCategories.Bookmark
        FillFields
        recCategories.Edit

    Case Else
        MsgBox Err.Number & ": " & Err.Description
    End Select

End Sub
```

```
Private Sub cmdSave_Click()

On Error GoTo SaveErrorHandler

    ' save the record
    recCategories.Fields("CategoryName") = txtCategoryName.Text
    recCategories.Fields("Description") = txtDescription.Text
    recCategories.Update
    recCategories.Bookmark = recCategories.LastModified

    ButtonNonEditAddMode

    DBEngine.Idle dbFreeLocks

Exit Sub

SaveErrorHandler:

    Select Case Err.Number
    Case 3260
        MsgBox "Record is currently locked. Try Save again later or cancel
changes."

    Case 3197
        MsgBox "Data has been changed by another user. Your changes have been
canceled."
        cmdCancel_Click

    Case Else
        MsgBox Err.Number & ": " & Err.Description

    End Select

End Sub
```

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **60 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 5.

Exercise 1: Locating the Windows Folder

In this exercise, you will use the Windows API function **GetWindowsDirectory** to find the folder that contains Microsoft Windows.

Exercise 2: Creating a Topmost Window

In this exercise, you will use the Windows API function **SetWindowPos** to create a topmost window (or form) that remains on top of other windows, regardless of the current focus.

Exercise 3: Using Callbacks

In this exercise, you will use the **AddressOf** operator to establish a callback function to receive timer notifications.

In this lab, you will call two Windows DLLs to find the location of the Windows folder and create a topmost window. In addition, you will call a Windows timer by using a callback function.

After completing this lab, you will be able to:

® Declare a DLL.

® Call a function of a DLL.

® Manipulate strings returned from a DLL.

® Locate the Windows folder by using the Windows API.

® Create a topmost window by using the Windows API.

® Use Windows timers without a form on which to place a timer control.

Before working on this lab, you should be familiar with the following concepts:

® The API Text Viewer

® Standard modules

® The contents of this chapter

To complete this lab, you need the following setup:

® Visual Basic version 5.0 or later

In this exercise, you will use the Windows API function **GetWindowsDirectory** to find the folder that contains Microsoft Windows.

u Create a form

1. Create a new Standard EXE project.
2. Save the project in the <*Install Folder*>\Labs\Lab05.
3. Add controls to the form, as shown below.
   {ewc mvimg, mvimage,!v05g010.bmp}

u Declare a procedure

1. Add a standard module to the project.
2. From the API Viewer, copy the **GetWindowsDirectory Declare** statement.
   For more information about copying declarations, see Using the API Viewer.
3. Paste the **Declare** statement into the new module.

u Create an event procedure

1. Create a Click event procedure for the Windows Directory button.
2. In the event procedure, add code that will display the path to the Windows folder in a message box.
   Be sure to follow the rules for using strings, as discussed in Passing Strings.
   {ewc mvimg, mvimage,!code.bmp}
3. Save and test the project.

In this exercise, you will use the Windows API function **SetWindowPos** to create a window or form that remains on top of other windows, regardless of the current focus.

1. Create a new project.
2. Save the project in the <*Install Folder*>\Labs\Lab05.
3. Add controls to the form, as shown below:

   {ewc mvimg, mvimage,!v05g015.bmp}
4. Create event procedures for the **Topmost window** and **Non-topmost window** option buttons.
   {ewc mvimg, mvimage,!code.bmp}
5. Save and test the application.

```
'DLL Declarations
Declare Function GetWindowsDirectory Lib "kernel32" _
    Alias "GetWindowsDirectoryA" (ByVal lpBuffer As String, _
    ByVal nSize As Long) As Long

Private Sub cmdWinDir_Click()
    Dim WinDir As String
    Dim ReturnSize As Long
    WinDir = String(255, 0) 'fill string w/nulls
    'call API to get win directory and capture the
    'size of the return string
    ReturnSize = GetWindowsDirectory(lpBuffer:=WinDir, _
        nSize:=Len(WinDir))
    'trim the string down to the correct size
    WinDir = Left(WinDir, ReturnSize)
    MsgBox "The Windows directory is: " & WinDir
End Sub
```

For background information on this lab, click each of these topics.

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **60 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 6.

Exercise 1: Controlling Microsoft Excel

In this exercise, you will control Microsoft Excel using Automation.

Exercise 2: Creating a Client Application

In this exercise, you will create a client application that uses the functionality of a version of the Credit Card component, which is created in later chapters.

In this lab, you will use Automation to control and exchange information with Microsoft Excel and Microsoft Access.

After completing this lab, you will be able to use Automation to:

® Open a Microsoft Excel workbook.

® Change information on a worksheet.

® Create a chart from worksheet information.

® Open a Microsoft Access database.

® Use Microsoft Access to open a report based on parameters of your Visual Basic application.

Before working on this lab, you should be familiar with the following concepts:

® The major constructs of COM

® The major constructs of Automation.

® Fundamentals of the Automation objects in Microsoft Excel.

To complete this lab, you should have the following setup:

® Visual Basic, version 5.0, or later.

In this exercise, you will control Microsoft Excel using Automation. A Microsoft Excel workbook named Earn.xls has been created for you and is located in the <*Install Folder*>\Labs\Lab06 folder.

A portion of the workbook is shown in the following illustration. Note that the cells C3 and C4 have been named "Growth" and "Inflation," respectively. Also, cells C16:E16 have been named "Net_Profit."

{ewc mvimg, mvimage,!v06g045.bmp}

Create a project

1. Start a new Visual Basic project.
2. Modify Form1 to resemble the following:

   {ewc mvimg, mvimage,!v06g050.bmp}

   If you do not want to build the form, you can get a copy of it in the <*Install Folder*> \Labs\Lab06 folder.
3. Save the project as XLAuto.vbp in the \Lab06 folder.

Create an instance of the Microsoft Excel application object

1. Add a reference to the Microsoft Excel 5.0 Object Library.
2. Dimension module-level variables named **xl** and **xlChart**.
3. In the Click event procedure for the **Create XL Object** button, use the **CreateObject** function to create an instance of the Microsoft Excel **Application** object, and assign that object to the variable created in the previous step. For more information see Creating an Instance of Microsoft Excel.
4. Set the **Visible** property of Microsoft Excel to **True**.
5. Use the Microsoft Excel **Open** method to open Earn.xls.

   To see sample answer code, click this icon.
   {ewc mvimg, mvimage,!code.bmp}
6. Test the application.
7. Exit.

Send information to Microsoft Excel

1. In the Click event procedure for the **Estimate Earnings** button, activate the Earnings worksheet and set the values of cells Growth and Inflation to the values of the Growth and Inflation text boxes, respectively.

   To see sample answer code, click this icon.
   {ewc mvimg, mvimage,!code.bmp}
2. Test the application.
   - Run the application.
   - Click the **Create XL Object** button to start Microsoft Excel and open the workbook.
   - Switch to Visual Basic, enter percentages in the text boxes, and test the functionality of the **Estimate Earnings** button.
3. Exit Microsoft Excel.

Create a chart

1. In the Click event procedure for the **Chart Earnings** button, if the **xlChart** variable added above is **Nothing**, use the **Select** method to select the named **Range** Net_Profit of the Earnings worksheet.

   Note In these steps, you should use the **xl** module-level variable so it can be used from within all procedures in this exercise.

   ---
2. Use the **Add** method of the **Chart** object to create a new chart in the workbook and initialize **xlChart**.
3. Set the **Type** property of the **Chart** object to xl3DColumn. This is a constant in the Microsoft Excel object library.
4. If the **xlChart** variable is not **Nothing**, use the **Activate** method to make it the active sheet.

To see sample answer code, click this icon.
{ewc mvimg, mvimage,!code.bmp}

5. Test the application.
   - Run the application.
   - Click the **Create XL Object** button to start Microsoft Excel and open the workbook.
   - Switch to Visual Basic and click the **Chart Earnings** button.

Exit Microsoft Excel

1. In the Click event procedure for the **Exit** button, use the **Close** method to close the **ActiveWorkbook** object.
2. Use the **Quit** method to close Microsoft Excel.
3. Unload the form and end the application.
4. Test the application.
   - Run the application.
   - Click the **Create XL Object** button to start Microsoft Excel and open the workbook.
   - Switch to Visual Basic and click the **Exit** button.
5. Save the project.

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **90 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder *<Install Folder>*\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Note The solution to this lab will not execute properly on the current build of Windows NT 4.0.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 7.

Exercise 1: Creating a Code Component

In this exercise, you will create a code component that exposes one object.

Exercise 2: Debugging and Error Handling

In this exercise, you will raise a custom error from the **CreditCard** object, and trap that error in your client application.

Exercise 3: Adding Component Information and Help

In this exercise, you will define procedure attributes that provide descriptive information and context-sensitive Help for the properties and methods of the **CreditCard** object.

Exercise 4: Defining and Using Events

In this exercise, you will use events to provide status information to the client application while the **Approve** method is processing.

Exercise 5: Compiling and Registering the Component

In this exercise, you will compile the code component and client application, and test the compiled versions.

Exercise 6: (Optional) Creating an Asynchronous Method

In this exercise, you will create a method that executes asynchronously.

After completing this lab, you will be able to:

® Create an ActiveX code component.
® Declare methods and properties within a code component.
® Test the component from a second instance of Visual Basic.
® Compile and register the component.
® Create an object model.
® Build and test an ActiveX code component.

Before working on this lab, you should be familiar with the following concepts:

® Automation.

® The contents of this chapter.

To complete this lab, you need the following setup:

® Visual Basic version 5.0 or later

In this exercise, you will create a code component that exposes one object. The **CreditCard** object will be used to simulate the validation of a credit card purchase.

This table shows the interfaces you will implement.

Name	Type
CardNumber	Property
ExpireDate	Property
PurchaseAmount	Property
Approve	Method

u Create a code component

1. Create a new project.
2. When prompted for a project type, click **ActiveX DLL**, and then click **Open**.

   The class module and project properties are set automatically when you choose an ActiveX DLL project.
3. Set the properties for the project, as shown in the following table.

Property	Setting
Project Name	Lab
Project Description	CreditCard Object
Unattended Execution	Selected

4. Set the **Name** property of the class module to CreditCard.
5. Save the class module as Cc.cls, and the project as Cc.vbp in the folder <*Install Folder*>\Labs\Lab07.

u Create properties and methods

1. In the class module, use public variables to create two properties with the information in the following table.

Property name	Data type
CardNumber	Integer
ExpireDate	Date

   For information about creating properties and methods, see Creating Properties.
2. Using property procedures, create a property named **PurchaseAmount**.
   a. Create a private module-level variable to store the value of the purchase amount.
   b. In the **Property Get** procedure, return the value of the private variable.
   c. In the **Property Let** procedure, test to see whether the return value of the private variable is greater than zero. If it is greater than zero, set the private variable to the purchase amount. If it is less than zero, set the private variable to zero.

      For more information about property procedures, see Using Property Procedures to Create Properties.
3. Using a **Public** function, create a method named **Approve** that returns a **Boolean** value.

   If the credit card is approved, the method should return a value of **True**, and if it is declined, a value of **False**.

   For this exercise, use the following logic to validate the credit card.

If...	Then...
PurchaseAmount < 1000 And ExpireDate > Now()	Approve = **True**
PurchaseAmount >= 1000 And ExpireDate <= Now()	Approve = **False**

4. Make **CardNumber** the default property for the CreditCard class.
5. Save the project.
6. Compile the project as Cc.dll.

   This .dll file will enable **Project Compatibility** automatically in the **Project Properties** dialog box. This will ensure that as you make changes to the project, the GUID will remain the same, and references from your client application will not be lost.

   For information about version control, see Version Compatibility.

u Create a client application

1. To add a new project to the CreditCard project, click **Add Project** on the **File** menu.
2. When prompted for a project type, click Standard EXE, and then click **Open**.

   This project will be used as the client that tests the component.
3. In the References dialog box, add a reference to the CreditCard project.
4. Dimension a module-level variable named cc as type Lab.CreditCard.
5. Add a **CommandButton** control to the form.
6. In the Click event procedure for the **CommandButton** control, add the following code:

```
set cc = New Lab.CreditCard
cc.ExpireDate = "1/1/99"
cc.PurchaseAmount = 50
cc.CardNumber = 1234
MsgBox cc.Approve
```

7. In the **Project Group** window, right-click Project1, and then click **Set as Start Up**.

   This command sets Form1 as the startup form for the project.
8. Press F8 to run the client application in Step mode, and step through the code.

   When you set the **PurchaseAmount** property and call the **Approve** method, you are actually stepping into the CreditCard project.
9. Try changing the value of **PurchaseAmount** to 2000. Does the **Approve** method return **False**?

   To see the code for the CreditCard class module if your code is not working properly, click this icon {ewc mvimg, mvimage,!code.bmp}

In this exercise, you will raise a custom error from the **CreditCard** object and trap that error in your client application.

u Add error-handling code

1. Open the CreditCard class module.
2. Select the **PurchaseAmount Property Let** procedure.
3. Rewrite the **Property Let** procedure, so that if the value of the amount the user has entered is negative, a run-time error will be raised with the following arguments.

Argument	Value
Number	vbObjectError + 1000
Source	Lab.CreditCard
Description	Purchase amount must be greater than zero.

4. In the **General** tab of the **Options** dialog box, set the **Error Trapping** option to **Break in Class Module**.
5. Change the client code to pass a purchase amount of –100.
6. Run the client application.

   Note that the run-time error occurs in the class module.
7. In the **Options** dialog box, set the **Error Trapping** option to **Break on Unhandled Errors**.
8. Run the client application.

   Note that this time, the run-time error occurs in the client application.
9. Save the CreditCard project.

u Error-handling in the client

1. In the client application, add an error handler to the Click event for the **CommandButton** procedure.
2. In the error handler, test for the error generated in the class module.
3. If the error is found, display a message box telling the user that the purchase amount must be a positive value.

   For more information about error handling, see Raising Run-Time Errors.

   To see the error-handling code to use if your code is not working properly, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

In this exercise, you will define procedure attributes that provide descriptive information and context-sensitive Help for the properties and methods of the **CreditCard** object.

u Set procedure attributes

1. In the **Project Properties** dialog box, set **Help File Name** to Lab.hlp.

   The file Lab.hlp is located in the folder <*Install Folder*>\Labs\Lab07. It is a Help file with three sample topics.
2. Open the CreditCard class module.
3. In the **Procedure Attributes** dialog box, add a description for the **Approve** method and **PurchaseAmount** properties.
4. In the **Procedure Attributes** dialog box, set **Help Context ID** for the **Approve** method to 1 and the **Help Context ID** for the **PurchaseAmount** property to 2.
5. Save the project.

u Test procedure attributes

1. Open the client project.
2. Display the **Object Browser** dialog box, and select the **Lab Library**.
3. Check to see that the descriptions for the **Approve** method and **PurchaseAmount** property are displayed correctly.
4. Test the Help information for the **Approve** method and **PurchaseAmount** property as shown in the following illustration.

   {ewc mvimg, mvimage,!v07g035.bmp}

In this exercise, you will use events to provide status information to the client application while the **Approve** method is processing.

u Define and use the Status event

1. In the CreditCard class module, add the following code to the beginning of the **Approve** procedure:

```
Dim sngEndTime As Single

RaiseEvent Status("Dialing bank...")
'Simulate delay dialing bank.
sngEndTime = Timer + 2
Do While Timer < sngEndTime
    DoEvents    ' Yield to other processes.
Loop

RaiseEvent Status("Processing card...")
'Simulate delay processing card.
sngEndTime = Timer + 2
Do While Timer < sngEndTime
    DoEvents    ' Yield to other processes.
Loop
```

   This code adds two delays of two seconds each to simulate the time necessary to dial the bank and process the credit card information. Before each delay, an event is generated to inform the client application of the current status.

2. In the CreditCard class module, declare the Status event with one string argument.
3. In the client application, add a label to hold status information, as shown in the following illustration.

   {ewc mvimg, mvimage,!v07g040.bmp}

4. In the client application, add the **WithEvents** keyword to the cc variable declaration.

   This adds a new object with one new procedure named Status to the code window, as shown in the following illustration.

   {ewc mvimg, mvimage,!v07g045.bmp}

5. In the Status event, display the status text in the **Label** control.
6. Save both projects, and test the application.

   Each of the status messages should be displayed for two seconds in the label.

   To see the code, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

In this exercise, you will compile the code component and client application, and then test the compiled versions.

Compile the component and test the application

1. Compile the credit card project as a .dll file.
2. Compile the client project as an .exe file.
3. Close Visual Basic.
4. In the folder <*Install Folder*>/Labs/Lab07, run the file Client.exe.
5. Test the client application.

In this exercise, you will create a method that executes asynchronously.

Create the initial project

1. Create a new ActiveX EXE project.
2. Set the project name to AsyncSrv.
3. Set the class module name to Async.
4. Add a form to the project and name it frmTimer.

Implement a Timer control

1. Add a **Timer** control to the frmTimer form.
2. At the form level, add a public variable called **CallBack** of type **Async**.
3. Add code to the Timer event procedure in the **Timer** control to disable the **Timer** control and call the **DoWork** method in the **Callback** object. For example:

```
Private Sub Timer1_Timer()
    Timer1.Enabled = False
    'Start the real processing
    CallBack.DoWork
End Sub
```

Implement the Async class

1. Add a class level variable called **CurrForm** of type **Form**.
2. Add a method called **AsyncMethod** that does the following:
   a. Initializes the CurrForm variable with a new instance of the frmTimer form.
   b. Sets the Callback variable of the form to point to the **Async** class.
   c. Enables the timer on the form.

```
Public Sub AsyncMethod()
'Create instance of form so it is unique to this class.
    Set CurrForm = New Form1
'Pass current class to form instance.
    Set CurrForm.CallBack = Me
'Enable timer to start async processing
'and return immediate control to client
    CurrForm.Timer1.Enabled = True
End Sub
```

3. Add a **Friend** method called **DoWork**.
4. Add a **Single** variable to **DoWork** called **sgnEndTime**.
5. Destroy the instance of CurrForm.
6. Add the following code to simulate a long, asynchronous task:

```
sngEndTime = Timer + 5
    Do While Timer < sngEndTime
        DoEvents    ' Yield to other processes.
    Loop
```

7. Add an event called Complete that passes the string "Hello, World" back to the client once the task is complete.

   The following example shows the completed code for the DoWork procedure:

```
Public Event Complete(Result As String)

Friend Sub DoWork()
    Dim sngEndTime As Single
```

```
    'Get rid of form instance.
    Set CurrForm = Nothing

    'Delay to simulate a long procedure.
    sngEndTime = Timer + 5
    Do While Timer < sngEndTime
        DoEvents    ' Yield to other processes.
    Loop

    'Raise event to client to signify end of processing.
    RaiseEvent Complete("Hello, world")
End Sub
```

Build the project

1. Save all of the project files as Asyncsrv in the folder <*Install Folder*>\Labs\Lab07.
2. Compile the project as an ActiveX EXE project.

   For more information about asynchronous methods, see Making Asynchronous Calls with Events.

Create an asynchronous client

1. Start a second instance of Visual Basic.
2. In the second instance, create a new Standard EXE project.
3. In the new project, add controls to the form as shown in the following illustration.

   {ewc mvimg, mvimage,!v07g050.bmp}
4. Set the **Enabled** property of the text box to **False**.
5. Create a reference to the AsyncSrv project.
6. Add a form level variable called **mAsync** to hold the **Async** object created above.

   Be sure the client can receive events from the object.
7. In the Load event for the form, initialize the **mAsync** variable with a new instance of the **Async** object.
8. In the Click event of the **Start Async Method** button, set the text in the text box to indicate that the component is processing, and then call the **AsyncMethod** method of the component.
9. In the Complete event of the **Async** object, update the text in the text box to indicate that the asynchronous task is complete and display the resulting text. The following example shows the completed code for the client:

```
Dim WithEvents mAsync As AsyncSrv.Async

Private Sub cmdStartAsyncMethod_Click()
    Text1 = "Processing..."
    mAsync.AsyncMethod
End Sub

Private Sub mAsync_Complete(Result As String)
    Text1 = "Complete. Result = " & Result
End Sub

Private Sub Form_Load()
    Set mAsync = New AsyncSrv.Async
End Sub
```

10. Test the asynchronous method.

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **45 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 12.

Exercise 1: Using Resource Files

In this exercise, you will use a resource file within your application.

Exercise 2: Using the Registry

In this exercise, you will store application-specific information in the Windows registry.

Exercise 3: Using the Setup Wizard

In this exercise, you will use the Setup Wizard to build a Setup program that will install the application you created in this lab.

Exercise 4: (Optional) Using Visual Basic Code Profiler

In this exercise, you will use the Visual Basic Code Profiler to examine the performance of a Visual Basic-based application.

Note Make sure that you copy the folder \Tools\Unsupprt\Vbcp from your Visual Basic 5.0 CD-ROM to the Visual Basic folder on your hard drive.

By the end of this lab, you will be able to:

® Create a resource file for your application.

® Save application settings in the registry.

® Use the Setup Wizard to create a Setup program for your application.

® Use the Visual Basic Code Profiler.

Before starting this lab, you should be familiar with the following concepts:

® The purpose and structure of a resource file.

® The purpose and structure of the Windows registry

® The purpose and structure of a Setup program

To complete this lab, you need the following:

® Visual Basic 5.0 or later

In this exercise, you will use a resource file within your application. For more information about resource files, see Using Resource Files.

Create a form

1. Create a new project.
2. Add controls to the form, as shown in the following illustration.

   {ewc mvimg, mvimage,!v12g010.bmp}

   The control in the upper-left corner is a picture control. Place any icon you want in the **Picture** property. Icons are provided in the Visual Basic \Icons folder.
3. Save the application.

Use resources

1. In the folder <*Install Folder*>\Labs\Lab12, add the resource file Lab.res to the project.
2. In the Form_Load event, load resource picture 100, and assign it to the **Picture** property of the **Image** control.
3. Load resource string 100, and assign it to the **Caption** property of the **Label** control.
   {ewc mvimg, mvimage,!code.bmp}
4. Test the application. If the resources are loaded correctly, your form should resemble the following illustration.

   {ewc mvimg, mvimage,!v12g015.bmp}
5. To load the picture and the string resources numbered 200, change the code in the Form_Load event appropriately.
6. Test the application. How has it changed?

   {ewc mvimg, mvimage,!answer.bmp}

In this exercise, you will store application-specific information in the Windows registry.

Save information in the Windows registry

1. In the form **Unload** event procedure, use the **SaveSetting** statement to save the information in the first text box to the Windows registry. Use the arguments listed in the following table.

Argument	Value
AppName	Polishing
Section	General
Key	Text1
Setting	Text1.Text

   To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

2. In the form **Load** event procedure, use the **GetSetting** function to read the information from the Windows registry into the first text box. Use the arguments listed in the following table.

Argument	Value
AppName	Polishing
Section	General
Key	Text1
Setting	Default

   To see an example of how your code should look, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

3. Test the application. Is the information saved and loaded correctly?

   You can also examine the Windows registry by running Regedit.exe.

   If you run Regedit after completing this exercise, you can find the application-specific information in the registry, as shown in the following illustration.

   {ewc mvimg, mvimage,!v12g020.bmp}

In this exercise, you will use the Setup Wizard to build a Setup program that installs the application you created in this lab.

Build a Setup program

1. Run the Setup Wizard.
2. In the **Select Project and Options** dialog box, choose to create a Setup program without a dependency file, and to have the executable rebuilt.
3. For the distribution method, select **Single Directory**.
4. For the destination folder, choose <*Install Folder*>\Labs\Lab12.
5. Do not add any ActiveX servers to the Setup.
6. In the **File Summary** dialog box, add the Readme.txt file located in the folder <*Install Folder*>\Labs\Lab12, and review the file summary information.
7. In the **Finished** dialog box, click the **Finish** button to build the Setup program.

   For more information about the Setup Wizard, see Using the Setup Wizard.

Test Setup

1. Run the Setup program you just created, and install the application in the default folder.
2. On the **Start** menu, click **Programs**.

   The application should be listed.
3. Run the application.
4. View the contents of the application folder. Has the file Readme.txt been installed correctly?
5. On the **Start** menu, point to **Settings**, and then click **Control Panel**.
6. Inn the **Control Panel** dialog box, double-click **Add/Remove Programs**.
7. On the **Install/Uninstall** tab, remove the application you installed.

In this exercise, you will use the Visual Basic Code Profiler to examine the performance of a Visual Basic application.

Note Make sure that you copy the folder \Tools\Unsupprt\Vbcp from your Visual Basic CD-ROM to the Visual Basic folder on your hard drive.

---

u **Install the Code Profiler add-in**

1. Register the DLL for the Code Profiler (Vbcp.dll) by using the file Regsvr32.exe.
2. Edit the file VBAddin.ini located in the Windows folder to include the entry **VBcp.VBcpClass=0** under the section **[Add-Ins32]**.
3. In the **Add-In Manager** dialog box, select the **Visual Basic Code Profiler** to add it to the Visual Basic design environment.

u **Use the Code Profiler**

1. Save the project before using the Code Profiler.
2. To profile code, click the **Visual Basic Code Profiler** on the **Add-Ins** menu.
3. To add code to the project, click **Add Profiler Code** in the **Code Profiler** dialog box.
4. Close the **Code Profiler** dialog box, and run the application.
5. Execute various functions, and end the application.
6. To view the results, click **Visual Basic Code Profiler** on the **Add-Ins** menu.
7. On the **File** menu in the **Code Profiler** dialog box, click **View Results**.

   Line-timing analysis is displayed.

```
Private Sub Form_Unload(Cancel As Integer)
    SaveSetting AppName:="Polishing", Section:="General", _
        Key:="Text1", Setting:=Text1.Text
End Sub
```

```
Private Sub Form_Load()
    Label1.Caption = LoadResString(100)
    Image1.Picture = LoadResPicture(100, 1)
    ' get registry settings
    Text1.Text = GetSetting(AppName:="Polishing", Section:="General", _
        Key:="Text1", Default:="Default")
End Sub
```

```
Private Sub Form_Load()
    Label1.Caption = LoadResString(100)
    Image1.Picture = LoadResPicture(100, 1)
End Sub
```

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

To complete this course successfully, you should have a basic understanding of how to work with Visual Basic. This chapter provides a brief review of the skills you should have to create Visual Basic applications.

Objectives

By the end of this chapter, you will be able to:

® Create a simple application by using Visual Basic, and create an executable file for users.

® Set properties and add code to controls on forms.

® Describe the differences in the scope of data and code.

® List the files that comprise a Visual Basic application.

® Validate data.

® Debug and trap errors in an application.

The first step in creating an application in Visual Basic is to design the forms that will be used in the application.

Users want applications that are easy to learn and that help them become productive quickly. Design your forms carefully, and work with users to determine the best design.

This section reviews how to create a simple form.

This section includes the following topics:

® Using Controls

® Setting Properties

® Adding a Menu

For more information about user interface conventions, refer to *The Windows Interface Guidelines for Software Design* available from Microsoft Press.

To connect to the Microsoft Press Web site, click this icon.
{ewc mvimg, mvimage,!intjump.bmp}

When you build an application in Visual Basic, you begin by creating the user interface.

Using the Toolbox, you draw or place controls on a form to create the visual elements of your application. The Toolbox is available at design time only.

Using the Toolbox

The Toolbox contains built-in Visual Basic controls and any ActiveX controls or any insertable objects you have added to the project. If the Toolbox is closed, you can open it by clicking **Toolbox** on the **View** menu.

To place a control on a form, you can either double-click the control, or single-click it and draw the control on the form.

Adding Controls to the Toolbox

You can extend the Toolbox by adding ActiveX controls. The Professional Edition of Visual Basic provides additional ActiveX controls. You can also purchase ActiveX controls from third-party vendors.

u To add an ActiveX control to the Toolbox

1. On the **Project** menu, click **Components**.

   Visual Basic displays the **Components** dialog box.
2. On the **Controls** tab, click the control you want to include, and then click OK.

   Visual Basic adds the control to the Toolbox.

Naming Controls

When naming controls, follow standard user interface and control naming conventions. This will make your code easier to read and debug.

For information about control naming conventions, search on **Coding Conventions** in Visual Basic Books Online, and then see **Control Naming Conventions**.

Working with Objects and Classes

An object is a combination of code and data that is treated as a unit. An object can be an element of an application, such as a control or a form. It can also be an entire application.

A class is a template for an object, just as a blueprint is a template for a house. All objects are created as identical copies or instances of their class. Once they exist as individual objects, their properties can be changed.

In Visual Basic, each object is defined by a class. The controls in the Toolbox represent classes. When you place a control on a form, you create an object of the control’s class.

Using an Object's Properties and Methods

All objects have properties and methods. Properties are values you set to determine the object’s appearance and behavior. Methods are procedures provided by the object.

For example, a form provides a **Show** method that causes the form to be displayed.

The benefit of working with objects is that objects provide code that you do not have to write. You simply set the object’s properties and call the object’s methods, and the object performs various functions.

In this course, you will learn how to use Visual Basic objects, as well as objects from other applications, such as Microsoft Excel and Word.

When building the user interface of a Visual Basic application, you must set the properties for the objects you create.

To see a demonstration of how to add controls to a form and set properties, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Setting Properties at Design Time

Some properties can be set at design time. To set these properties, you can use the Properties window or property pages.

For information about creating property pages, see Creating Property Pages in Chapter 8: Creating ActiveX Controls.

To access the Properties window, right-click an object, and then click **Properties**. The Properties window displays only the properties that are common to all of the selected controls. Any change you make to a property applies to all of the controls.

To see an illustration of the Properties window, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

You can also set properties for multiple controls at the same time. To select multiple controls, click and drag the mouse.

Setting Properties at Run Time

At run time, you can write code to set or retrieve properties. The following code sets a font to bold for a text box named txtData.

```
txtData.Font.Bold = True     'Set text to bold.
```

This code sets the **Text** property of the text box txtData.

```
txtData.Text = "Hello World" 'Set value of text.
```

If you omit the property name, you set the default property of the control. The default property of a text box is the **Text** property. The default property of a label is the **Caption** property.

The following code sets the default text and caption properties for a text box and a label control.

```
txtData = "Set the Text property or text box"
lblData = "Set the Caption property of label"
```

Getting Properties at Run Time

You can use the following code to get properties at run time.

```
Dim sName As String
sName = txtName.Text
```

You can enhance your application with menus. Menus provide users with a convenient way to execute commands.

Using the Menu Editor

The Menu Editor is an interactive tool that enables you to create and modify menus with a minimum of coding. With the Menu Editor, you can add new commands to existing menus, replace existing menu commands with your own, create new menus and menu bars, and change and delete existing menus and menu bars.

To see an illustration of the Menu Editor, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

u **To add a menu to a form**

1. Click the **Menu Editor** button on the Standard toolbar to display the **Menu Editor** dialog box.
2. In the **Caption** field, enter the menu or command name that you want to appear on the menu bar in your menu.
3. Use the arrow command buttons to move menu items up or down in the command list.

At run time, you can modify the appearance of a menu and add new menu items. You can also use the **PopupMenu** method to display a pop-up menu.

Once you have designed forms and set properties, you are ready to add code to your application.

Visual Basic is a complete programming language that supports the programming constructs found in most other programming languages.

This section describes how to create events and general procedures, the scope of procedures, and how to use the Object Browser to list the procedures. It also discusses the different data types available in Visual Basic, how to declare variables, and the scope and life of variables.

This section includes the following topics:

® Creating Procedures

® Scope of Code

® Using the Object Browser to View Procedures

® Using Variables, Constants, and Data Types

® Scope of Variables

® Life of Variables

There are two types of procedures you will work with in Visual Basic: event procedures and general procedures.

Event Procedures

Visual Basic automatically calls event procedures in response to keyboard, mouse, or system actions. For example, command buttons have a Click event procedure. The code you place in a Click event procedure is executed when the user clicks a command button.

To open a code window, double-click the control or form, or click **Code** on the **View** menu.

Each control has a fixed set of event procedures. The event procedures for each control are listed in a drop-down list box in the code window.

The following code shows the Click event procedure for a command button named cmdOK:

```
Private Sub cmdOK_Click()
    MsgBox "Hello"
End Sub
```

General Procedures

General procedures are **Sub** or **Function** procedures that you create to perform specific tasks. To execute a general procedure, you must explicitly invoke it.

To create a general procedure, you open a code window and click **Add Procedure** on the **Tools** menu. You can also create a new procedure by typing the procedure heading **Sub**, followed by your procedure name, on a blank line in a code window.

If you have duplicate code in several event procedures, you can place the code in a general procedure, and then call the general procedure from the event procedures.

Sub Procedures

Sub procedures do not return values. For example:

```
Private Sub UpdateName(S as String)
```

You invoke a **Sub** procedure by specifying only the procedure name, or by using the **Call** statement with the procedure name. For example:

```
Call sortList("myname")
```

If you use the **Call** statement, you must enclose the argument list in parentheses. If you omit **Call**, you must also omit the parentheses around the argument list.

Function Procedures

Function procedures return values. In the following code, the **Function** procedure receives a number and returns the number squared.

```
Function Square(I As Integer) As Integer
    Square = I * I
End Function
```

If you want to save the return value, you must use parentheses when invoking a function, as shown in this code:

```
j = Square(5)
```

If you omit the parentheses, you can ignore the return value or not save it in a variable. This can be useful if you want to execute a function and you do not want the return value. For example:

```
Square 5
```

Note Both **Sub** and **Function** procedures can accept and modify arguments. You can use the **Optional** keyword to define arguments as optional.

For information about optional arguments, search on **Optional Arguments** in Visual Basic Books Online, and then see **Passing Arguments to Procedures**.

In addition to declaring code in a module associated with a form, you can also declare procedures in a standard code module.

Standard code modules contain only Visual Basic code, and they are a good place to store code that is not specific to a single form. Procedures can be declared as **Private** or **Public**.

Private procedures can be called only by other procedures located in that form, module, or class.

Public procedures on a form become methods of the form. The procedure can be called from anywhere in the application by specifying the form and procedure names.

Public procedures in a module are available throughout the application, and can be called by specifying the procedure name.

The following code declares a public procedure.

```
Public Sub myproc()

End Sub
```

If you declared the procedure in a form module, you call it with this code.

```
Form1.myproc
```

If you declared the procedure in a standard module, you call it with this code.

```
myproc
```

If you declared a procedure with the same name in two standard modules, you must qualify the procedure name when you call it, as shown in the following code.

```
Module1.myproc
```

Once you have added code to your application, you can use the Object Browser to list all of the procedures you have created, jump quickly to a specific procedure, or paste a procedure call into a code window.

The Object Browser displays the classes that are available from any object libraries you have referenced.

To display the Object Browser, press F2 in a code window.

The following illustration shows how to use the Object Browser to view the procedures in a module.

{ewc MVIMG, MVIMAGE,!v01g035.bmp}

This topic explains how to declare variables, constants, and data types.

Declaring Variables

To declare a variable, you use the **Dim** statement. The following code declares the variable **iCounter** as an **Integer** variable.

```
Dim iCounter As Integer
```

Although Visual Basic lets you use variables that you have not declared, your code will be easier to debug if you declare all variables. To require variable declaration, include the **Option Explicit** statement to the General Declarations section of a form or module. If you try to use a variable that you have not declared when **Option Explicit** is set, you will receive a run-time error.

To automatically include the Option Explicit statement in all new projects

1. On the **Tools** menu, click **Options**.
2. On the **Editor** tab, select **Require Variable Declaration**, and then click OK.

Declaring Constants

You use the **Const** statement to declare a constant. You cannot change the value of a constant.

The following code declares the constant **PI** as a variable of type **Single**.

```
Const PI As Single = 3.14
```

Visual Basic provides many built-in constants that are useful with various statements.

The following code uses the predefined constants **vbYesNo** and **vbYes** with the **MsgBox** statement.

```
iResult = MsgBox("Do you want to continue?", vbYesNo)
If iResult = vbYes Then
    'Do something.
End If
```

Using Data Types

The data type of a variable determines the type of information that a variable can store and the range of possible values. For example, if you want a variable to act as a counter for a loop, you assign it the data type **Integer**.

For information about data types, see the **Dim statement** in Visual Basic Help.

If you do not provide a data type when you declare a variable, the variable is given the **Variant** data type. The **Variant** data type can store null, numeric, date/time, or string data. However, the **Variant** data type requires more memory than other data types. If you want to create concise and fast code, you should use explicit data types where appropriate.

Declare each variable on a separate line to avoid confusion, as shown in the following code.

```
'sFname is Variant and sLname is String.
Dim sFname, sLname As String

'Instead, use
Dim sFname As String
Dim sLname As String
```

Visual Basic coerces data types if necessary. For example, if you use a string variable in an arithmetic expression, Visual Basic will convert the string variable to a numeric value, if possible.

The following code shows how Visual Basic adds 1 to the value in the text box.

```
Dim iNewRate As Integer
txtRate.Text = "5"
iNewRate = txtRate.Text + 1
```

Generally, you should use data type conversion statements to explicitly convert data types, as shown in the following code.

```
iNewRate = CInt(txtRate.Text) + 1
```

For information about the data type conversion statements, search on **Data Type** in Visual Basic Help, and see the topic **Type Conversion Functions**.

Visual Basic provides interactive tools for finding run-time errors and errors in program logic. You can access all of the debugging tools by using the **Debug** menu or the **Debug** toolbar.

To see a demonstration of how to use the debugging tools in Visual Basic, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Visual Basic debugging support includes:

® Breakpoints and break expressions

Set a breakpoint to stop a program while it is running. You can set a breakpoint at design time, or at run time while in break mode.

® Watch expressions

Use watch expressions to monitor a particular variable or expression. The value of each watch expression is updated at breakpoints.

® Step options

Use the step options to run portions of your code either one statement or one procedure at a time.

® Call Stack

Use the Call Stack to view all active procedure calls and trace the execution of a series of nested procedures.

® Immediate window

In Break mode, you can test an executable statement by typing it in the Immediate window. Visual Basic runs the statement immediately so that you can evaluate your code.

® Locals window

This window automatically displays all of the declared variables in the current procedure, along with their values.

Creating an executable file in Visual Basic is a simple process.

u To create an executable file

1. On the **File** menu, click **Make <Project. Name>.exe**.
2. Enter the name for the executable file.
3. To add version-specific information:

   In the **Make EXE File** dialog box, click the **Options** button. On the **Make** tab, type the version numbers and text for the version information, and then click OK.

   The following illustration shows the **Version Number** options on the **Make** tab.

   {ewc MVIMG, MVIMAGE,!v01g025.bmp}

You can also start Visual Basic from an MS-DOS prompt to create an executable file. This enables code generators to create .exe files without user intervention, and enables developers to start large compile jobs programmatically.

The following code shows how to compile an application from the command line:

```
vb32 /make projectname exename
```

In addition to the executable file, you must provide various DLLs and other files to your users. You should create a Setup program that installs your application onto the user's computer.

The Visual Basic Setup Wizard makes it easy to create distribution disks or to create a distribution server folder for your application. Users can then run the resulting Setup program on their computers to install and register the appropriate files.

For information about creating a Setup program, see Chapter 12: Application Setup and Optimization.

Visual Basic is available in three editions, each designed to meet a specific set of development requirements. *Mastering Microsoft Visual Basic 5* is based on Microsoft Visual Basic, Professional Edition.

Learning Edition

The Visual Basic Learning Edition enables you to create powerful applications for Microsoft Windows 95 and Windows NT. It includes all intrinsic Visual Basic controls, along with grid, tab, and data-bound controls.

Professional Edition

The Professional Edition provides a full-featured set of tools for developing solutions for others. It includes all of the features of the Learning Edition, along with additional ActiveX controls, including Internet controls and the Crystal Reports for Visual Basic.

Enterprise Edition

The Enterprise Edition enables you to create robust distributed applications in a team setting. It provides all of the features of the Professional Edition, and also includes features such as the Automation Manager, Component Manager, database management tools, and the Microsoft Visual SourceSafe project-oriented version control system.

Visual Basic provides a variety of resources to help you find the information you need when you are working in the development environment.

For a demonstration of different ways to get assistance in the development environment, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Online Help

Visual Basic provides extensive online Help. The Help file contains many code samples that you can copy directly into an application.

Visual Basic version 5.0 Help is context-sensitive. To use context-sensitive Help in a code window, enter the word for which you want information, and then press F1. For example, if you want information about the **Open** statement, type **Open** and press F1.

If you receive a run-time error, you can press F1 when the error message is displayed to get Help for that error message.

Books Online

In addition to context-sensitive Help, the Visual Basic CD-ROM includes an online version of the print documentation for Visual Basic. To gain access to Books Online, click **Books Online** on the **Help** menu in Visual Basic.

Note If you have worked with a previous version of Visual Basic, see **What's New in Visual Basic 5.0?** in Books Online for a summary of the new features.

Code Editor

The Visual Basic code editor automatically provides you with relevant information as you enter code. For example, if you type the name of a control, followed by the dot (**.**) operator, the properties and methods of that control will be automatically displayed in a list box. You can then choose the appropriate property or method to complete the statement.

When you enter a function name in the code window, Visual Basic automatically provides you with the syntax of the function.

Sample Applications

Visual Basic includes a number of sample applications for you to use. These can be found in the folder \Vb\ Samples.

Microsoft Web Site

To find the most up-to-date information about Visual Basic, and gain access to a library of technical information, visit the Visual Basic area on the Microsoft Web site. To connect to the site, click this icon.
{ewc mvimg, mvimage,!intjump.bmp}

1. **You are creating a Windows 95 application for your accounting department with the project name ACCPROJ. Which command would you enter at the DOS prompt to compile your project to an executable named ACC.EXE?**

{ewc mvimg, mvimage,!answer.bmp} A. vb32 /make ACCPROJ ACC.EXE

{ewc mvimg, mvimage,!answer.bmp} B. vb32 ACCPROJ ACC.EXE

{ewc mvimg, mvimage,!answer.bmp} C. vb32 /compile ACCPROJ ACC.EXE

{ewc mvimg, mvimage,!answer.bmp} D. vb32 /compile ACC.EXE ACCPROJ

2. **You want to display the message "This is a text box" in a text box named txtMyBox by setting the value of the Text property with the following code:**

```
txtMyBox.Text = "This is a text box"
```

By mistake, you enter the code shown below:

```
txtMyBox = "This is a text box"
```

What is the result?

{ew A. "This is a text box" is stored in a string variable named txtMyBox.

{ewc mvimg.mvimage.!answer.bmp}

B. The **Text** property of txtMyBox is set to "This is a text box".

{ewc mvimg.mvimage.!answer.bmp}

C. The **Name** property of txtMyBox is set to "This is a text box".

{ewc mvimg.mvimage.!answer.bmp}

D. The **DataField** property of txtMyBox is set to "This is a text box".

{ewc mvimg.mvimage.!answer.bmp}

3. What is the primary difference between Sub procedures and Function procedures?

{ewc mvimg.mvimage.!answer.bmp}

A. **Sub** procedures are private to the local module, while **Function** procedures can be used in any module.

{ewc mvi

B. You can pass arguments to **Function** procedures, but not to **Sub** procedures.

{mg.mvimage,!answer.bmp}

C. You can pass arguments to **Sub** procedures, but not to **Function** procedures.

{ewc mvimg,mvimage,!answer.bmp}

D. **Function** procedures can return values, but **Sub** procedures do not.

{ewc mvimg,mvimage,!answer.bmp}

4. You have created a procedure in a form with the code shown below:

```
Private Sub myroutine()
     ' some code
End Sub
```

From where in your project can this procedure be called?

{ewc mvimg,mvimage,!answer.bmp}

A. From anywhere in your application by specifying the form name and the procedure name.

{ewc mvimg,mvimage,!answer.bmp}

B. From procedures on the same form or in any .bas module in the application by specifying the procedure name.

{ewc mvimg,mvimage,!answer.bm

p}

{ewc mvimg, mvimage,!answer.bmp}

C. From anywhere in your application by specifying the procedure name.

{ewc mvimg, mvimage,!answer.bmp}

D. From procedures located on the same form by specifying the procedure name.

5. Where can a variable declared in a procedure be used?

{ewc mvimg, mvimage,!answer.bmp}

A. Only in the procedure.

{ewc mvimg, mvimage,!answer.bmp}

B. In any procedure in the form or module where the declaring procedure resides.

{ewc mvimg, mvimage,!answer.bmp}

C. In the declaring procedure and any .bas module.

{ewc mvimg, mvimage,!answer.bmp}

{ewc mvimg.mvimage.!answer.bmp}

D. Anywhere in the application.

6. How should you declare a variable so it will be available the entire time your application is running?

{ewc mvimg.mvimage.!answer.bmp}

A. As a public variable in the first form loaded by your application.

{ewc mvimg.mvimage.!answer.bmp}

B. As a public variable in the last form unloaded by your application.

{ewc mvimg.mvimage.!answer.bmp}

C. In a standard module.

{ewc mvimg.mvimage.!answer.bmp}

D. As a non-static variable in a procedure.

7. Which Visual Basic debugging tool is used to monitor a particular variable or expression?

{ewc mvimg, mvimage,!answer.bmp}
A. Break expressions

{ewc mvimg, mvimage,!answer.bmp}
B. Watch expressions

{ewc mvimg, mvimage,!answer.bmp}
C. Step options

{ewc mvimg, mvimage,!answer.bmp}
D. Call Stack

8. Which Visual Basic debugging tool can you use to trace the execution of a series of nested procedures?

{ewc mvimg, mvimage,!answer.bmp}
A. Breakpoint expressions

{ewc mvimg.mvimage.!answer.bmp}

B. Watch expressions

{ewc mvimg.mvimage.!answer.bmp}

C. Step options

{ewc mvimg.mvimage.!answer.bmp}

D. Call Stack

You can prevent some data entry errors and improve the usability of your application by validating information as it is entered into the fields of your application.

Restricting Choices with Controls

One way to ensure valid input is to restrict the number of options from which a user can choose. For example, you can use a list box to let users select a product name on a form. Because users must choose from a predefined list, they cannot enter an invalid product name.

You can also use radio buttons for a small number of mutually exclusive options, or use check boxes for Boolean choices.

Using the MaxLength Property

The **MaxLength** property determines the maximum length of a string in a text box. The system beeps when the user tries to type a string that exceeds the maximum length. If you want to display an error message, you need to trap the keystroke in the KeyPress event.

Using the Locked Property

The **Locked** property determines if users can modify data in a text box. If **Locked** is **True**, users can only view and copy data in a text box.

Using the KeyPress Event to Validate Data

You can use the KeyPress, KeyDown, and KeyUp events to validate data as the user types. You can prevent the user from entering certain characters (for example, you can limit data entry exclusively to numeric values). You can also modify data as it is entered (for example, you can convert all characters to uppercase).

The KeyPress event occurs whenever the user enters a standard ASCII character. This does not include most of the special keys, such as function keys, the arrow keys, or the DEL key. To respond to these keys, use the KeyDown and KeyUp events.

The following code changes characters to uppercase as the user types.

```
Sub Text1_KeyPress(KeyAscii As Integer)
    KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

To see a code sample that prevents users from entering characters other than numeric values in a text box, click this icon.
{ewc mvimg, mvimage,!code.bmp}

For a list of ASCII values, search for **ASCII** in Visual Basic Help.

In addition to using field-level techniques to validate data as it is entered, you can write code that validates the data in all fields on a form at the same time. This topic describes the form-level validation techniques supported by Visual Basic.

Enabling the OK Button

One approach to validating form information is to ensure that a user has entered data in all fields on a form before the user is allowed to proceed. You can accomplish this by disabling the OK button on a form until the user has filled in all the fields, as shown in the following illustration.

{ewc MVIMG, MVIMAGE,!v01g020.bmp}

To test each keystroke entered on a form, set the form's **KeyPreview** property to **True**. The form receives the keyboard event first, and then the control receives the event.

To see a code sample that enables the OK button, click this icon.
{ewc mvimg, mvimage,!code.bmp}

Validating All Fields on a Form

An easy way to validate all fields on a form at the same time is to put the validation code in the Click event of the **OK** button. In this scenario, the application allows the user to complete the form, and then it validates all fields. The application sets the focus to the first field that contains invalid data.

To see a code sample that validates all of the numeric fields on the form shown above, click this icon.
{ewc mvimg, mvimage,!code.bmp}

Using the QueryUnload Event

The QueryUnload event occurs just before the Unload event when the form is unloaded. The QueryUnload event enables you to determine how an Unload event was initiated, and to cancel the Unload event. This is useful when a user has not completed the data entry on a form, or when you would like to prompt the user to save changes before closing the form.

The QueryUnload event has the following two arguments.

® The **UnloadMode** argument indicates how the Unload event was initiated.

® The **Cancel** argument cancels the Unload event. If Cancel is set to **True**, the application remains as it was before the unload was attempted.

The following code cancels the Unload event and prompts the user before closing the form.

```
Private Sub Form_QueryUnload(Cancel As Integer, _
                             UnloadMode As Integer)
    Dim iResult As Integer
    iResult = MsgBox("Are you sure you want to quit?", _
                         vbYesNo)
    If iResult = vbNo Then
        Cancel = True
    End If
End Sub
```

No matter how well you design your application, run-time errors will occur. Users forget to put disks in disk drives, systems run low on memory, and files are not located where you expect to find them. By adding effective error handling code to your application, you create a more robust application.

Understanding the Error Handling Process

The error handling process involves the three following steps.

1. Enable an error trap that specifies where execution will branch to when an error occurs.
2. Write the error handling code.
3. Exit the error handling code.

The **On Error GoTo** statement enables an error trap and specifies where the execution will jump to when an error occurs. If a run-time error occurs, execution jumps to the label indicated by the **On Error GoTo** statement. The error handler runs error handling code, followed by a **Resume** statement that indicates where processing should continue.

The following code shows how to use the **On Error GoTo** and **Resume** statements.

```
Private Sub Command1_Click()
    Dim AppName As String
    On Error GoTo CheckError
    AppName = InputBox("Enter application name")
    Shell AppName
    Exit Sub

CheckError:
    If Err.Number = 53 Then 'File not found.
        MsgBox "Application not found."
        If MsgBox("Try again?", vbYesNo) = vbYes Then
            AppName = InputBox("Enter application name")
            Resume 'Try again.
        Else
            Resume Next 'Run next statement.
        End If
    Else
        MsgBox "Unknown error"
    End If
End Sub
```

In error handling code, use the properties and methods of the **Err** object to check which error occurred, clear an error value, or raise an error.

Err Object Properties

The **Number** property is an integer that indicates the last error that occurred. To determine which error has occurred, you check the value of **Err.Number**. In some cases, you can correct an error and allow processing to continue without interrupting the user. Otherwise, you must notify the user of an error, and take some action based on the user's response.

The **Description** property is a string that contains a description of the error.

The **Source** property contains the name of the object application that generated the error. This is helpful when using Automation. For example, if you launch Microsoft Excel and it generates an error, Microsoft Excel sets **Err.Number** to the correct error code and sets **Err.Source** to **Excel.Application**.

Err Object Methods

The **Clear** method sets the value of **Err.Number** back to zero. You use the **Clear** method primarily when you handle in-line errors.

The **Raise** method causes an error. You use the **Raise** method to pass an error back to a calling procedure, or to test your own error handling code. For example:

```
Err.Raise 53 'File not found error.
```

Resume Options

To specify where your application will continue processing after handling an error, you use the **Resume** statement. The following table lists the three types of **Resume** statements available in Visual Basic.

Statement	Description
Resume	Return to the statement that caused the error. Use **Resume** to repeat an operation after correcting the error.
Resume Next	Return to the statement immediately following the statement that caused the error.
Resume line or label	If there is no **Resume** statement, the procedure will exit.

It can be difficult to debug code that has error handling enabled. Visual Basic may execute error-handling code when you want to enter Break mode and debug the application.

Visual Basic provides options to disable error handling when debugging.

To change how errors are handled

1. On the **Tools** menu, click **Options**.
2. On the **General** tab, under **Error Trapping**, click the option you want, and then click OK.

This table describes the error handling options that are available under **Error Trapping** on the **General** tab.

Option	Description
Break on All Errors	If you click this option, Visual Basic ignores any **On Error** statements and enters Break mode if any run-time errors occur.
Break in Class Module	Set this option when debugging an ActiveX component. This causes the ActiveX component to enter Break mode rather than passing the error back to the client application.
Break on Unhandled Errors	Visual Basic enters Break mode on any error for which you do not supply specific error-handling code.

For an overview of database development, covered in Chapters 2 through 4, click this icon.
{ewc mvimg, mvimage,!exppov.bmp}

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

In Visual Basic, you can use the **Data** control to create database applications for a variety of database formats. The **Data** control interacts with the Microsoft Jet database engine, and lets you create data-aware applications with a minimal amount of coding.

This chapter will teach you how to use the **Data** control and several advanced data-bound controls to view, edit, and update information in a database.

Objectives

At the end of this chapter, you will be able to:

® Describe how Visual Basic gains access to databases.

® Differentiate between tables, fields, and records in a database.

® Describe the purpose of an index.

® Use Structured Query Language (SQL) to retrieve records from a database.

® Use the **Data** control and standard bound controls to view, add, delete, and update the contents of a Microsoft Access database.

® Describe the situations that cause events of the **Data** control (Validate, Reposition, and Error) to occur.

® Use the data-aware **DBList** control to add data to a table.

® Describe the differences between the properties **DataField**, **ListField**, and **BoundColumn** of the data-aware **DBList** control.

® Display information in the data-aware **DBGrid** control.

This topic lists several options that Visual Basic provides for accessing data.

Using the Jet Database Engine

Data access objects (DAO) and the **Data** control use the Microsoft Jet database engine to access databases. The Jet database engine can access the following three types of databases.

® Jet databases

These databases are created and manipulated directly by the Jet engine. Microsoft Access and Visual Basic use the same Jet database engine.

® Indexed Sequential Access Method (ISAM) databases

The formats of these databases include Btrieve, dBASE, Microsoft Visual FoxPro, and Paradox.

® Open Database Connectivity (ODBC) – compatible databases

These databases include client/server databases that conform to the ODBC standard, such as Microsoft SQL Server. Most databases that support ODBC can be accessed by using Visual Basic.

Other Methods of Data Access

Other methods of data access supported by Visual Basic include:

® Remote Data Source Control

This is a control that uses ODBC to gain access to ODBC databases such as Microsoft SQL Server and Oracle. The remote data source control is available only in the Visual Basic Enterprise edition.

® ODBC Libraries

These libraries enable you to call the ODBC applications programming interface (API) directly, and are available as a separate product.

® Visual Basic SQL Libraries (VBSQL)

These libraries provide a direct link to a Microsoft SQL Server, and are available as a separate product.

Most database systems use a relational database model.

The relational database model presents data as a collection of tables. A table is a logical group of related information. For example, the Northwind database contains a table that lists employees, and another table that lists orders for a fictitious company.

The Nwind.mdb database is a sample database included with Microsoft Access. It is also included on this *Mastering Microsoft Visual Basic 5* CD-ROM.

The following illustration shows you a diagram of the Northwind database.
{ewc mvimg, mvimage,!illust.bmp}

Elements of a Table

The Northwind database contains a number of tables that group information. These tables include Orders, Customers, and Employees.

In a Jet database, table rows are referred to as records, and columns are referred to as fields.

The following illustration provides a diagram of the Employees table.
{ewc mvimg, mvimage,!illust.bmp}

Primary Key

Each table includes a primary key, which is a field (or combination of fields) that is unique for each row in the table. For example, the EmployeeID field is the primary key for the Employees table.

A table can also contain foreign keys, which are fields that reference a row in another table. For example, in the Northwind database, the Orders table contains a CustomerID field. This field is a foreign key because it references the "foreign" Customer table rather than the Orders table. Instead of duplicating all of the customer information for each order, there is only one customer for each order, and one customer can have many orders. In terms of a database, the relationship between the customer information and the Orders table is a one-to-many relationship.

Records

A record contains information about a single entry in a table. Generally, you do not want two records in a table to have the same data. For example, a record in the Employees table contains information about a single employee.

Fields

Each field in a table contains a single piece of information. For example, the Employees table includes fields for Employee ID, Last Name, First Name, and so on.

Indexes

Database table indexes are sorted lists that are faster to search than the tables themselves. To enable faster database access, most databases use one or more indexes. For example, the Employees table might have an index for the EmployeeID column.

The Data control in Visual Basic provides you with the ability to write powerful database applications with very little code.

In this section, you will learn how to build database applications by using the **Data** control and the associated **Recordset** object. You will also learn how the Data Form Wizard can build a database application that includes the **Data** control.

This section includes the following topics:

® Accessing Data with the Data Control

® Using Data Control Properties and Methods

® The Recordset Object

® Using Recordset Properties and Methods

® Using the Data Form Wizard

To specify what data you want to retrieve, you must set the **DatabaseName** and **RecordSource** properties of a **Data** control. In addition, you can set the following properties and method.

® **Connect** property

® **Exclusive** property

® **ReadOnly** property

® **Recordset** property

® **BOFAction** and **EOFAction** properties

® **Refresh** method

This topic describes these additional properties and method.

Connect Property

The **Connect** property specifies the type of database to open. It can include arguments such as user ID and password.

Exclusive Property

The **Exclusive** property determines whether or not you have exclusive use of the database. If you set the **Exclusive** property to **True**, and then successfully open the database, no other application will be able to open the database until you close it.

ReadOnly Property

The **ReadOnly** property determines whether or not you can update the database. If you do not plan to update the database, it is more efficient to set the **ReadOnly** property to **True**.

Recordset Property

The **Recordset** property is an object that contains the set of records returned by the **Data** control. This property contains properties and methods that you can use to work with the returned records.

BOFAction and EOFAction Properties

These properties determine what action the **Data** control will take when the **BOF** or **EOF** properties of the recordset are **True**.

For example, if the **EOFAction** property of the **Data** control is set to vbActionAddNew, and you use the **Data** control to move past the last record in the recordset, the **Data** control will automatically execute the **AddNew** method so you can enter a new record.

The Refresh Method

The **Refresh** method refreshes the **Recordset** object. If you change the **RecordSource** property at run time, you must call the **Refresh** method to refresh the recordset.

The following code shows how to use the **Refresh** method:

```
Data1.RecordSource = "SELECT * FROM Employees " & _
                "WHERE [Employee ID] = " & txtEmpID.text
Data1.Refresh
```

To retrieve information from the recordset, you use properties and methods of the **Recordset** object to move through records, and add, update, or delete records.

BOF and EOF Properties

The **Recordset** object properties **BOF** and **EOF** indicate whether the position of the current record is before the first record or after the last record in the recordset. If there are no records in the recordset, then both **BOF** and **EOF** properties are **True**.

Recordset AddNew Method

To add a new record to a recordset, you use the **AddNew** method. When you execute the **AddNew** method, Visual Basic clears the bound controls and sets the **EditMode** property of the **Data** control to dbEditAdd.

The new record will not be added to the database until an **UpdateRecord** or **Update** method is explicitly executed, or until the user moves to another record.

The following code shows how to add a new record to a recordset:

```
Sub cmdAdd_Click ()
    Data1.Recordset.AddNew
```

Data Control UpdateRecord Method

To save the current record to a database, you use the **UpdateRecord** method.

The following code shows how to save the current record and update the database:

```
Sub cmdUpdate_Click ()
    Data1.UpdateRecord
```

Data Control CancelUpdate Method

To cancel an **AddNew** or **Edit** method and refresh the bound controls with data from the recordset, you use the **CancelUpdate** method.

For example, if a user has modified the fields on a form, but has not yet updated them, the **CancelUpdate** method will refresh the fields with the original data from the recordset.

If a user selects an **Add** button on a form, and then decides not to add the record, the **CancelUpdate** method will cancel the operation and display the current record.

The following code shows how to cancel the adding or editing of a record:

```
Sub cmdCancel_Click ()
    Data1.CancelUpdate
```

The Delete Method

To remove a record from a database, you use the **Delete** method. The deleted record will remain as the current record until the user moves to a different record, as shown in the following code.

```
Sub cmdDelete_Click ()
    Data1.Recordset.Delete
    Data1.Recordset.MoveNext
    If Data1.Recordset.EOF Then
        Data1.Recordset.MoveLast
    End If
End Sub
```

Note The Northwind database has referential integrity rules defined that prohibit you from deleting employees that have orders. For information about how to handle referential integrity violations, see Referential Integrity in Chapter 4: Advanced Database Development.

The Data Form Wizard is a Visual Basic utility that generates simple database forms that use the **Data** control. You can use the Data Form Wizard to quickly build the basic forms for a database application.

To load the Data Form Wizard, you use the **Add-In Manager** command on the **Add-Ins** menu.

To see a demonstration that shows how to use the Data Form Wizard to create a form that displays information from the Northwind database, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

The **Data** control provides three events that you can use to enhance your database application: Validate, Reposition, and Error. These events allow you to override some of the default behavior of the **Data** control.

This section includes the following topics:

® Using the Validate Event

® Using the Reposition Event

® Using the Error Event

To verify data before a record is saved to a database, you can use the Validate event. This event occurs just before Visual Basic writes changes from the bound controls to the database and repositions the current record pointer to another record in the database.

You can use the Validate event to prompt users to confirm changes.

Syntax

The Validate event has the following syntax:

Private Sub Data1_Validate (*index* **As Integer**, *action* **As Integer**, *save* **As Integer**)

The *Action* Argument

The *action* argument indicates the operation that caused the Validate event to occur. The Validate event occurs as a result of any one of the following operations:

® **MoveFirst**, **MovePrevious**, **MoveNext**, **MoveLast**

® **AddNew**

® **Update**

® **Delete**

® **Find**

® Setting the **Bookmark** property

® Closing the database

® Unloading the form

To cancel any of these actions, set the *action* argument to vbDataActionCancel.

The *Save* Argument

The *save* argument indicates whether or not the record will be saved. If *save* is **True**, the bound data has been changed. To cancel the save, you can set *save* to **False**.

The following code prompts the user to confirm changes to the database. If the user responds with No, the changes will be canceled.

```
Private Sub Data1_Validate (Action As Integer, Save As Integer)
    Dim iResponse As Integer
    If Save = True Then
        iResponse = MsgBox ("Save Changes?", vbYesNo)
        If iResponse = vbNo Then
            Save = False
            Data1.UpdateControls ' Refresh fields.
        End If
    End If
End Sub
```

To modify the appearance of a form, or take any action required when moving to a new record, you use the Reposition event.

This event occurs when Visual Basic repositions the current record pointer to another record in the database. It also occurs when the database is first opened.

Modifying the Appearance of a Form

To change the way in which a form displays information based on the currently selected record, you use the Reposition event. For example, you can modify the caption of the **Data** control to display Record *n*.

To display the current record number, you use the **AbsolutePosition** property of the **Recordset** object. The record number is relative to zero, so the first record is 0.

The following code shows how to display the current record number:

```
Private Sub Data1_Reposition()
    Data1.Caption = "Record Number " & _
             Data1.Recordset.AbsolutePosition + 1
End Sub
```

Handling Changes When Moving to a New Record

When a user moves to a new record by using the **Data** control, data on the form may need to be displayed differently for that record.

For example, a form showing employee records may contain different options for permanent employees, contractors, hourly employees, or salaried employees. Each record would display information about a different employee, and those options would not be the same for all records. To enable the correct option to be selected for each record, you can place code in the Reposition event.

The following code uses the Reposition event to modify the appearance of a form.

```
Private Sub Data1_Reposition()
    Data1.Caption=Data1.Recordset.AbsolutePosition
    If Data1.Recordset("Employee ID") > 5 Then
        optSenior.Value = True
    Else
        optJunior.Value = True
    End If
    Data1.Caption = "Record " & _
    Data1.AbsolutePosition + 1
End Sub
```

The following illustration shows the form based on the previous code.

{ewc mvimg, mvimage,!v02g030.bmp}

The Error event occurs when a user interacts with the **Data** control, and a data-access error results. To add custom error handling to the **Data** control, you can use the Error event.

For example, if a user modifies a field and then clicks the **Data** control to move to the next record, the D**a**ta control will update the current record. If a data-access error results during the update, the Error event will occur.

After an error has occurred, the values in the bound fields will not change. The user can correct the values, and then click the **Data** control to try updating the record again.

Displaying a Custom Error Message

If you do not add any error-handling code to the Error event, and an error occurs when a user interacts with the **Data** control, Visual Basic will display the error message and continue running the application.

If you do not want to display the standard error message, you can set the Response argument to zero and display a custom error message, as shown in the following code:

```
Private Sub Data1_Error(DataErr As Integer, Response As Integer)
    If DataErr = 3022 Then 'Duplicate Key error
        MsgBox "Enter unique Emp ID number"
        txtEmpID.SetFocus
        Response = 0
    Else
        Response = 1 'display standard error message
    End If
End Sub
```

You can use SQL to retrieve a subset of rows only, or to retrieve information from two tables at once. SQL enables you to specify exactly which records to retrieve and in what order.

If you have an error in your SQL statement—for example, if you provide an invalid field name—you will receive a run-time error message similar to **Too few parameters. Expected 2**. If you receive this error, check your SQL string and verify that the table and field names you listed are valid.

SQL SELECT Syntax

Use the SQL **SELECT** statement to retrieve records. The syntax for **SELECT** is:

SELECT fieldlist

FROM *tablenames*

WHERE *searchconditions*

ORDER BY *fieldlist*

The following code retrieves only the Last Name and Employee ID fields, where the Employee ID is greater than 5. The records are retrieved in descending Employee ID order.

```
SELECT [LastName], [EmployeeID]
    FROM Employees
    WHERE [EmployeeID] > 5
    ORDER BY [EmployeeID] DESC
```

WHERE

Use the **WHERE** clause to limit the selection. The number sign (**#**) indicates date literal values. Date literal values entered in an SQL statement must be in U.S. date format (for example, May 9, 1996 is written as 5/9/96).

To see a code sample that shows various SQL statements, click this icon.
{ewc mvimage, MVIMAGE,!code.bmp}

ORDER BY

Use the **ORDER BY** clause to create a recordset in a particular order. The **ASC** option indicates ascending order, and **DESC** indicates descending order.

This code selects all fields sorted by last name from the Employees table.

```
SELECT * FROM Employees ORDER BY [LastName] DESC
```

Using SELECT with Multiple Tables

You can use a join operation to combine records from multiple tables. For example, if you want to display category names and product names from different tables, you can join records from the Categories table and the Products table.

The **INNER JOIN** clause specifies that you want the records for which the Category ID from the Categories table matches the Category ID from the Products table.

This code joins information from the Categories table and the Products table.

```
strSQL = "SELECT categories.[CategoryName], " & _
    "Products.[ProductName] " & _
    "FROM  Categories " & _
    "INNER JOIN Products ON " & _
    "Products.[CategoryID] = Categories.[CategoryID]"
```

To see an illustration that shows the result of the join operation, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

In addition to the intrinsic bound controls, Visual Basic provides a number of data-bound ActiveX controls.

This section describes some advanced data-bound ActiveX controls.

This section includes the following topics:

® Using the DBGrid Control
® Using the MSFlexGrid Control
® Using the DBCombo Control

The data-bound grid control (**DBGrid** control) enables users of your database application to work with multiple records at the same time.

Displaying Multiple Records

The **DBGrid** control is an ActiveX control that displays a series of rows and columns that represent records and fields from a **Recordset** object. When you set the **DataSource** property of the **DBGrid** control to a **Data** control, the **DBGrid** control will be automatically filled with data, and its column headers will be automatically set from recordset of the **Data** control.

Unlike many of the other data-bound controls, the **DBGrid** control enables you to view and edit several records at the same time.

The following illustration shows a form that uses a **DBGrid** control to display records from the Northwind database.

{ewc mvimg, mvimage,!v02g035.bmp}

u To add the DBGrid control to your project

1. On the **Project** menu, click **Components**.
2. On the **Controls** tab, select **Microsoft Data Bound Grid Control,** and then click OK.

The **DBGrid** control has several properties that specify how the control behaves. For example, if you set the **AllowUpdate** property to **True**, a user can modify data in the control. You can also set properties for individual columns in the **DBGrid** control.

u To set column properties

1. Right-click the **DBGrid** control.
2. Click **Properties**.
3. Click **Columns**.

You can change the caption of the column, change the DataField to which the column is bound, add default values, and so on.

To set the **DBGrid** control column headings, right-click the control at design time, and then click **Retrieve Fields**.

Getting and Setting Text of the Current Cell

Use the **Columns** collection of the **DBGrid** control to retrieve the text of the currently selected cell at run time. For example:

```
MsgBox DBGrid control1.Columns(DBGrid control1.Col).Text
```

To change information in the **DBGrid** control, change the associated **Recordset** object. For example, if the **DBGrid** control is bound to the recordset Data1, you would execute the following code:

```
Data1.Recordset.Edit
Data1.Recordset.Fields("Product Name") = "Floppy Disk"
Data1.Recordset.Update
```

Using the BeforeUpdate Event

The BeforeUpdate event occurs before data is moved from a **DBGrid** control to the data control's copy buffer. You can validate the data and cancel the update if necessary.

The **MSFlexGrid** control provides advanced features for displaying data in a grid. It is similar to the **DBGrid** control; however, when bound to a **Data** control, the **MSFlexGrid** control displays read-only data.

You can use the **MSFlexGrid** control to merge rows or columns of information as a way of grouping related information.

The following illustration shows records that are grouped by Order ID in the Order Details table.

{ewc mvimg, mvimage,!v02g040.bmp}

To merge cells

1. Set the **MergeCells** property to a value other than zero.
2. Set the **MergeRow()** and **MergeCol()** array properties to **True** for the rows and columns you want to merge.

For example, to merge the cells in the Order Details table, add the following code to the form that contains the **MSFlexGrid** control:

```
Private Sub Form_Load()
    MSFlexGrid1.MergeCells = flexMergeFree
    MSFlexGrid1.MergeCol(0) = True
End Sub
```

To add the MSFlexGrid control to your project

1. On the **Project** menu, click **Components**.
2. On the **Controls** tab, select **Microsoft FlexGrid Control 5.0**, and then click OK.

For more information about the **MSFlexGrid** control, refer to Visual Basic Books Online.

You can use the data-bound combo (**DBCombo**) control or the data-bound list box (**DBList**) control to automatically display a list of values from a recordset. You can use these controls to provide valid values.

For example, if a user must enter a shipper ID for an order, use the **DBList** control to automatically display a list of all shipper IDs from the Shippers table.

Getting Information from a Lookup Table

You can also use these controls in lookup table applications.

For example, you can display a list of valid category names (rather than IDs), and use the corresponding ID when the user adds or modifies data.

The following illustration shows a form that uses the **DBCombo** control to display category names for the Products table.

{ewc mvimg, mvimage,!v02g045.bmp}

Setting DBCombo Control Properties

To determine what value is displayed in the **DBCombo** control, set the **RowSource** property to a **Data** control name, and set the **ListField** property to a field name.

The data bound combo box contains all the values for that field. To determine what database field is updated when a user changes a value, set the **DataSource** and **DataField** properties. To establish the relationship between the table providing the lookup values and the actual table being edited, set the **BoundColumn** property.

To see a demonstration of how to use the **DBCombo** control, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

1. **If a user changes the data in a bound control, what happens to the data when the user moves to another record?**

{ewc mvimg, mvimage,! answer .bmp}
A. The change is automatically discarded.

{ewc mvimg, mvimage,! answer .bmp}
B. The change is automatically written to the database.

{ewc mvimg, mvimage,! answer .bmp}
C. The change is automatically written to a data cache on the client computer.

{ewc mvimg, mvimage,! answer .bmp}
D. The change is automatically written to a data cache on the server.

2. **When connecting to a dBase, Paradox, or Btrieve database, which property of the Data control is used to specify the directory that contains the database files?**

{ewc mvimg, mvimage,! answer
A. **DatabaseName**

{ewc mvimg, mvimage,!answer.bmp}

B. **RecordSource**

{ewc mvimg, mvimage,!answer.bmp}

C. **DataSource**

{ewc mvimg, mvimage,!answer.bmp}

D. **DataField**

{ewc mvimg, mvimage,!answer.bmp}

3. What are the rows in a Jet database table called?

{ewc mvimg, mvimage,!answer.bmp}

A. Primary keys

{ewc mvimg, mvimage,!answer.bmp}

B. Fields

{ewc mvimg.mvimage.!answer.bmp}

C. Indexes

{ewc mvimg.mvimage.!answer.bmp}

D. Records

4. When would each record in a table contain a single piece of information?

{ewc mvimg.mvimage.!answer.bmp}

A. When each row in the table is unique.

{ewc mvimg.mvimage.!answer.bmp}

B. When the table has a single record.

{ewc mvimg.mvimage.!answer.bmp}

C. When the table has a single field specified as the primary key.

{ewc

D. When the table has a single column.

{mvimg,mvimage,!answer.bmp}

5. What is the purpose of an index?

{ewc mvimg,mvimage,!answer.bmp} A. To enable the Jet database engine to sort the table.

{ewc mvimg,mvimage,!answer.bmp} B. To make access to the table faster.

{ewc mvimg,mvimage,!answer.bmp} C. To define the primary key for a table.

{ewc mvimg,mvimage,!answer.bmp} D. To specify the order in which data in a table should be displayed.

6. What is the current record after the code shown below has run?

```
Sub cmdDelete_Click ()
```

```
    Data1.Recordset.Delete
End Sub
```

{ewc mvimg, mvimage,!answer.bmp}

A. The first record in the table.

{ewc mvimg, mvimage,!answer.bmp}

B. The deleted record.

{ewc mvimg, mvimage,!answer.bmp}

C. The record before the deleted record.

{ewc mvimg, mvimage,!answer.bmp}

D. The record after the deleted record.

7. Which Data control event occurs when a database is first opened?

{ewc mvimg, mvimage,!answer.bm

A. Reposition

p}

{ewc mvimg, mvimage,!answer.bmp}

B. Validate

{ewc mvimg, mvimage,!answer.bmp}

C. Connect

{ewc mvimg, mvimage,!answer.bmp}

D. BOF

8. Which property of the DBList control specifies the name of the field containing the data you want to display in the list box?

{ewc mvimg, mvimage,!answer.bmp}

A. **ListField**

{ewc mvimg, mvimage,!answer.bmp}

B. **DataField**

{ewc mvimg.mvimage,!answer.bmp}

C. **RowSource**

{ewc mvimg.mvimage,!answer.bmp}

D. **RecordSource**

```
'Basic Where
strSQL = "SELECT * FROM Employees WHERE [LastName] = " & _
        "'" & txtLName.text & "'"

'Where In
strSQL = "SELECT Employees.[LastName] FROM Employees " & _
    "WHERE Employees.State in ('NY','WA')"

'Where Between
strSQL = "SELECT [OrderID] FROM Orders WHERE " & _
    "([OrderDate] BETWEEN #01/01/93# AND #01/31/93#)"

strSQL = "SELECT [OrderID] FROM Orders WHERE " & _
        "([OrderDate] BETWEEN #" & CDate(txtStartDate.TEXT) & _
        "# AND #" & CDate(txtEndDate.TEXT) & "#)"
```

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

Data access objects (DAO) are a set of objects that enable you to access and manipulate data programmatically in local or remote databases. You use DAO to manage databases, along with their objects and structure.

In this chapter, you will learn how to use DAO to retrieve and use data in a database.

Objectives

By the end of this chapter, you will be able to:

® Write an application that uses DAO to add, delete, update, and find data.

® Describe the differences between a table-, snapshot-, and dynaset-type recordset, and determine which type of recordset to use in a given scenario.

® Write an application that uses queries stored in a database.

To access and manipulate data programmatically, you should have an understanding of the DAO hierarchy.

The order of objects in DAO is referred to as its object model. The DAO object model enables you to write code that can take advantage of database functionality.

This section includes the following topics:

® The DAO Object Model

® The DBEngine and Workspace Objects

The DAO object model defines the hierarchy of the DAO objects. To incorporate DAO functionality in an application, you must be able to expose an object within this hierarchy.

To see an illustration that shows the DAO object model, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

You refer to this hierarchy when you create or access specific objects.

For example, this following code opens a database.

```
Set dbMydb = DBEngine.Workspaces(0) _
    .OpenDatabase("Mydb.mdb")
```

This table lists some of the data access objects and describes their use.

Object	Description
Workspace	Contains open databases.
Database	An open database.
Recordset	The records in a table or the records that result from running a query.
QueryDef	A stored definition of a query.
TableDef	A stored definition of a table.

Note This course focuses on working with the data in a database. It does not explain how to use Visual Basic to create a new database or modify the structure of an existing database. In most cases, you will want to use a tool such as Microsoft Access to define or modify a database structure.

To access data programmatically with DAO, you first use the **DBEngine** object to open a workspace.

DBEngine Object

The **DBEngine** object is the top-level object in the DAO object model. It contains and controls all other objects in the hierarchy of DAO.

To determine the DAO version number, you can query the **Version** property of the **DBEngine** object. You can also use the **RepairDatabase** or **CompactDatabase** methods of the **DBEngine** object to manipulate a database.

The following code shows the use of the **Version** property and the **RepairDatabase** method.

```
MsgBox "DAO version number is " & _
    DBEngine.Version
DBEngine.RepairDatabase "C:\MYDB.MDB"
```

Workspace Object

The **Workspace** object defines a session for the user, and determines how your application interacts with data. If you open a database without specifying a **Workspace** object, the default, **DBEngine.Workspaces(0)**, is used.

This following code uses the **CreateWorkspace** method of the **DBEngine** object to create a new workspace that will use the Microsoft Jet database engine to interact with data.

```
Dim wspNew as Workspace
Set wspNew = DBEngine.CreateWorkspace _
    ("NewJetWorkspace", "Admin", "", dbUseJet)
```

You can open additional **Workspace** objects as needed. Each workspace has a user ID and password associated with it.
{ewc mvimg, mvimage,!tip.bmp}

The first step in creating a database application is to open the database. You declare a variable as a **Database** object, and then use the **OpenDatabase** method to open the database.

Database Declaration

The **Database** object variable refers to your database. Use the **Set** statement to assign it to a database.

The following code opens a database in the default workspace.

```
Dim dbMydb As Database
Set dbMydb = OpenDatabase _
    ("C:\Program Files\DevStudio\VB\Nwind.mdb")
```

OpenDatabase Method

The **OpenDatabase** method accepts the arguments listed in the following table.

Argument	Purpose
dbname	A string that is the name of an existing database.
options	For a Microsoft Jet database workspace, **True** or **False**. If you open a database with *options* set to **True**, no other user will be able to open the database. The default value is **False**.
readonly	**True** or **False**. If **True**, no modifications are allowed. The default value is **False**.
connect	A string that specifies various connection information, including passwords.

{ewc mvimg, mvimage,!tip.bmp}

You can improve performance by opening a database for exclusive use if you do not have multiple users.

Close Method

When you have finished working with a database, you close it by using the **Close** method, as shown in the following code.

```
dbMydb.Close
```

A recordset contains a set of records from a database. You use the **Recordset** object to retrieve information programmatically from a database.

To see an animation that introduces the **Recordset** object, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

This section covers how to create and use recordsets.

This section includes the following topics:

® Opening a Database

® Creating a Recordset

® Types of Recordsets

® Displaying Records

® Moving Through a Recordset

Once you have opened a database, you can manipulate the data stored in the database by creating a recordset. You declare a variable as a **Recordset** object, and then use the **OpenRecordset** method to create the recordset.

Recordset Declaration

The **Recordset** object variable refers to your recordset. You use the **Set** statement to assign it to a recordset.

The following code opens a database and creates a recordset.

```
Dim dbMydb As Database
Dim recEmployees As Recordset
Set dbMydb = OpenDatabase _
    ("C:\Program Files\DevStudio\VB\Nwind.mdb")
Set recEmployees = dbMydb.OpenRecordset ("Employees")
```

OpenRecordset Method

The following table lists and describes the arguments to the **OpenRecordset** method.

Argument	Description
source	Specifies the source for the recordset. The source is a string that indicates a table name, query name, or SQL statement.
type	The type of recordset object to create, such as **dbOpenTable**, **dbOpenDynaset**, or **dbOpenSnapshot**.
options	Determines how the recordset is opened. For example, you can open a recordset and allow only forward scrolling through the recordset with the **dbForwardOnly** option.
lockedits	Determines the locking for the recordset. For example, if you specify **dbPessimistic** for the *lockedits* argument, the recordset will be locked during editing. For more information, see Microsoft Jet Database Engine Locking in Chapter 4: Advanced Database Development.

This code creates a read-only recordset.

```
Dim recOrders As Recordset
Set recOrders = dbMydb.OpenRecordset _
    ("Orders", dbOpenDynaset, dbForwardOnly)
```

To see a code sample that checks the **Type** property of a **Recordset** object to determine the recordset type, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

For a Microsoft Jet database workspace, there are four types of recordsets: table, dynaset, snapshot, and forward-only.

Table

The table-type recordset corresponds to a single table and can be modified. Only the current record is loaded into memory. A predefined index determines the order of the records in the recordset.

You cannot create a table-type recordset on an attached table. An attached table is a table in another database that is linked to a Microsoft Jet database. The data for an attached table remains in the external database.

For information about attached tables, see Chapter 4: Advanced Database Development.

Dynaset

A dynaset-type recordset is a dynamic set of records that can contain fields from one or more tables in a database. A dynaset also enables you to modify data.

A dynaset stores the primary key for each record in the recordset, instead of the actual data. The actual data is retrieved only when you access a particular record.

Because you are using a keyset to refer to the data, a dynaset reflects modifications to existing records by other users, but does not reflect records that are added by other users. If another user deletes a record after a dynaset is fully populated, the dynaset will contain a pointer to the deleted record. If you attempt to access a deleted record, you will receive a run-time error.

Snapshot

A snapshot-type recordset can contain fields from one or more tables in a database. It contains a fixed or static copy of the data as it existed when the snapshot was created.

A snapshot cannot be updated, and does not reflect changes made to data by other users. A snapshot is not fully populated until you move to the end of the recordset.

Forward-Only

A forward-only-type recordset is similar to a snapshot-type recordset, except that you can only scroll forward through its records.

Comparison of Recordset Types

The type of recordset you use depends on what you want to do with the data in the recordset.

If you want to use indexes to order records, you use a table-type recordset. Searching a table is much faster than searching a dynaset or a snapshot. In general, tables are the fastest type of recordsets, and dynasets are the most flexible.

If you use a dynaset, the Microsoft Jet database engine retrieves only the primary key of each record in the recordset. If you use a snapshot-type recordset, the Microsoft Jet database engine retrieves the entire record. In both cases, the result sets are stored in memory (overflowing to disk, if very large), which enables you to scroll in any direction.

When records of data are needed, a snapshot has the data available locally. A dynaset, on the other hand, must use a separate query to retrieve the complete record. The Microsoft Jet database engine requests the server for clusters of rows specified by their bookmarks, rather than one at a time, to reduce the time it takes to process the query.

If you only want to make a single pass through a recordset, you use a forward-only recordset.

Controls are not bound to a recordset as they are with the **Data** control. You must explicitly copy data from the recordset to controls on your form.

The following code illustrates two different ways to copy data from a field to a text box.

```
txtEmpID.Text = recEmployees("EmployeeID")
'or can use alternate syntax.
txtEmpID.Text = recEmployees![EmployeeID]
```

You can use the **Fields** collection to access the fields in the current record. To see a code sample that prints the data from the fields in a recordset, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

Handling Null Data

If you attempt to assign a null value to a text box control, you will receive a run-time error. To see a code sample that uses the **IsNull** function to avoid this error, click this icon.
{ewc mvimg, mvimage,!code.bmp}

Navigation through recordsets is based on a current record. Imagine that the current record is a particular record highlighted by an invisible cursor as you move through the recordset.

Navigation Methods

You use the **Move** methods to navigate through the records in a recordset.

® **MoveFirst** and **MoveLast** methods move to the first and last records in the recordset.

® The **MoveNext** and **MovePrevious** methods move relative to the current record or bookmark.

® The **Move** method moves a specific number of rows forward or back.

The following code shows uses of the **Move** method.

```
recEmployees.Move +5 'Moves forward 5 rows.
recEmployees.Move -6 'Moves backward 6 rows.
```

Navigation Properties

The **Recordset** object provides properties that you can use while navigating through a recordset.

For example, you can use the **BOF** property to avoid run-time errors when moving past the first record in a recordset.

BOF and EOF

The **BOF** and **EOF** properties indicate the beginning or end of the recordset. If both **BOF** and **EOF** are **True**, there are no records in the recordset.

When you use **MoveNext** or **MovePrevious**, you should check the **BOF** and **EOF** properties to make sure you do not move off the recordset. The **BOF** or **EOF** property is **True** when you move one record past the first or last record. If you move past the BOF or EOF indicators, you receive a run-time error.

To see a code sample that shows how to check for the **EOF** property, click this icon.
{ewc mvimg, mvimage,!code.bmp}

To see a demonstration of how to retrieve and navigate through a recordset, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

RecordCount

The **RecordCount** property of a table-type recordset accurately reflects the number of records in the table, and is affected immediately when any user adds or deletes a record.

The **RecordCount** property of a dynaset- or snapshot-type recordset returns the number of records have been accessed in a dynaset- or snapshot-type recordset. The **RecordCount** property does not indicate how many records are contained in the dynaset or snapshot until the recordset is fully populated.

To fully populate a dynaset- or snapshot-type recordset, you use the **MoveLast** method to cause the last record to be retrieved. However, using the **MoveLast** method in this manner can negatively impact performance. Unless you want to have an accurate record count as soon as you open a recordset, it is better to wait until you populate the recordset with other portions of code before checking the **RecordCount** property.

The following code displays the number of records in a recordset with a message box.

```
MsgBox "Recordset contains " & _
    recEmployees.RecordCount & " records."
```

Note You can also use the SQL **Count** function to determine the number of records in a recordset. For example, `Select Count(*) From Employees` returns the number of records in the Employees table.

AbsolutePosition

The **AbsolutePosition** property returns or sets the relative record number of the current record in a recordset. **AbsolutePosition** is zero-based (that is, the first record in a recordset is record 0). If you set **AbsolutePosition** to 3, the fourth record becomes the current record.

This code displays the absolute position of the current record in a label.

```
lblStatus.Caption = "You are on record " & _
     rs.AbsolutePosition + 1
```

Bookmark

The **AbsolutePosition** property of a record may change as records are added and deleted from a recordset. To return to a specific record, you use the **Bookmark** property.

This code uses a bookmark to save a location and return to it.

```
Dim varBookmark As Variant
varBookmark = recEmployees.Bookmark 'Save location
' [Your code here]
recEmployees.Bookmark = varBookmark 'Return to saved location.
```

Once you have created a recordset, you can use methods of the **Recordset** object to update, add, and delete records.

This section discusses how to update, add, and delete data from a Microsoft Jet database.

This section includes the following topics:

® Updating Records
® Adding Records
® Deleting Records
® Using DAO with the Data Control

To update a record, you invoke the **Edit** method, fill the fields of the record with the appropriate data, and then invoke the **Update** method.

The following code shows how to update data.

```
Private Sub cmdUpdate_Click ()
    recEmployees.Edit
    recEmployees("LastName") = txtLName.Text
    recEmployees.Update
End Sub
```

If you invoke the **Update** method without first using the **Edit** method, a run-time error will occur.

Recordset EditMode Property

You can check the **EditMode** property to determine whether the **Edit** or **AddNew** methods have been invoked. This can be useful in a **Save** command that either invokes the **Update** method immediately if the **Add** method has already been executed, or executes the **Edit** method followed by the **Update** method if the user is editing the current record.

The following table describes the constants that indicate the state of editing for the current record.

Constant	Description
dbEditNone	No editing operation is in progress.
dbEditInProgress	The **Edit** method has been invoked, and the current record is in the copy buffer.
dbEditAdd	The **AddNew** method has been invoked, and the current record in the copy buffer is a new record that has not been saved in the database.

To see a code sample that checks the current **EditMode** property, and invokes the **Edit** method if necessary, click this icon.
{ewc mvimg, mvimage,!code.bmp}

To add a new record to a recordset, you use the **AddNew** method. The record will not actually be saved to the database until you invoke the **Update** method.

The following code provides an **Add** command button that clears the form and invokes the **AddNew** method.

```
Sub cmdAdd_Click ()
    ClearFields
    recEmployees.AddNew
End Sub

Sub ClearFields ()
    txtLName.Text = ""
    ...
End Sub

Sub cmdUpdate_Click ()
    recEmployees("LastName") = txtLName.Text
    ...
    recEmployees.Update
End Sub
```

A user can add the new record, and then click the **Update** command button. The code for the **Update** command button copies the values from the form to the recordset, and saves the record to the database.

Canceling an Add

You use the **CancelUpdate** method to cancel the **AddNew** or **Edit** method, so that changes are not made to a database.

This code cancels an update, and refreshes the contents of two text boxes.

```
Sub cmdCancel_Click ()
    recEmployees.CancelUpdate
    txtLastName.Text = recEmployees("LastName")
    txtFirstName.Text = recEmployees("FirstName")
End Sub
```

To see a demonstration of how to add and edit records in a Microsoft Jet database, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

You use the **Delete** method to remove the current record from a recordset. No warning or prompt will occur before the record is deleted. You can add a warning message in your delete routine, if necessary.

Deleting a record does not automatically cause the next record to become the current record. To cause the next record to become the current record, you use the **MoveNext** method.

The following code deletes a record from a recordset and moves to the next record:

```
Private Sub cmdDelete_Click()
    recEmployees.Delete
    recEmployees.MoveNext
    If recEmployees.EOF Then
        recEmployees.MoveLast
    End If
    FillFields   'User written procedure.
End Sub
```

To determine if you have deleted all of the records in a recordset, you can check its **RecordCount** property. If you delete the last record, the value of the **RecordCount** property will be zero.

You can combine both DAO and the **Data** control in a single application. This enables you to use bound controls in an application while providing your application with the flexibility offered by DAO.

The **Recordset** property of the **Data** control returns a **Recordset** object.

You can set the **Recordset** property of the **Data** control to a **Recordset** object that was created by using the **OpenRecordset** method, as shown in the following code.

```
Dim recEmployees As Recordset
Set recEmployees = dbMydb.OpenRecordset ("Employees")
Set datEmployees.Recordset = recEmployees ' Set Data Control
```

You must make sure that the **DataField** property of the bound controls matches the field names in the **Recordset** object.

This section describes how to use one of the **Find** methods or the **Seek** method to search for data in a recordset.

This section includes the following topics:

® Searching a Dynaset or Snapshot

® Searching a Table

This section discusses how to use stored queries and SQL statements to view or modify data in a database.

This section includes the following topics:

® Using a Select Query
® Using an Action Query
® Introduction to SQL
® Using SQL

To use a stored Select query, you can provide the query name in the **OpenRecordset** method of the **Database** object.

The following code creates a recordset based on a stored Select query.

```
Set recProducts = dbMydb.OpenRecordset("Products On Order")
```

You can also use the **OpenRecordset** method of the **QueryDef** object to retrieve the results of the query.

This code creates a recordset with a **QueryDef** object based on a stored Select query.

```
Dim qryProducts As QueryDef
Set qryProducts = dbMydb.QueryDefs("Products On Order")
Set recProducts = qryProducts.OpenRecordset
```

Setting a Parameter

If the stored query requires parameters, use the **Parameters** collection of the **QueryDef** object to set parameters.

This code sets a parameter in a stored query.

```
Set qryProducts.Parameters("Beginning Date") = _
    CDate(txtBeginDate.text)
```

Creating a New Query

To create a new query at run time, and store the query in a database, use the **CreateQueryDef** method of the **Database** object.

This code creates and stores a new query.

```
Set qryNewQuery = dbMydb.CreateQueryDef _
    ("My Query", "Select * from employees")
```

1. What is the value of the BOF and EOF properties if there are no records in a recordset?

{ewc mvimg, mvimage,!answer.bmp}

A. **BOF** is **True** and **EOF** is **False**.

{ewc mvimg, mvimage,!answer.bmp}

B. **BOF** is **False** and **EOF** is **True**.

{ewc mvimg, mvimage,!answer.bmp}

C. Both **BOF** and **EOF** are **False**.

{ewc mvimg, mvimage,!answer.bmp}

D. Both **BOF** and **EOF** are **True**.

2. With which type of recordset would you use the Seek method to locate a record?

{ewc mvimg, mvimage,!answer.bmp}

A. Dynaset

{ewc mvimg.mvimage.!answer.bmp}

B. Snapshot

{ewc mvimg.mvimage.!answer.bmp}

C. Table

3. What can you do with a Select query?

{ewc mvimg.mvimage.!answer.bmp}

A. Return a record.

{ewc mvimg.mvimage.!answer.bmp}

B. Update a record.

{ewc mvimg.mvimage.!answer.bmp}

C. Insert a record.

{ew c mvi mg. mvi ma ge.! ans wer .bm p}

D. Delete a record.

You can use the **FindFirst**, **FindLast**, **FindNext**, and **FindPrevious** methods to locate a record in a dynaset or snapshot.

The search criteria that you specify must be equivalent to an SQL **Where** statement without the **Where** keyword.

The following code finds the first record that has a specific employee ID.

```
recEmployees.FindFirst "[EmployeeeID] = 5"
```

If you invoke a **FindFirst** or **FindLast** method, and no records are found, the current record is positioned to the first or the last record in the recordset, respectively

You can use the **Bookmark** property to save the current record location before invoking a **Find** method, and return to the previous position if no records are found.

To see a code sample that searches for a specific record and returns to the original position if no record is found in the search, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

This code sample uses the **NoMatch** property to determine whether the **FindFirst** method has found a record or not.

The following code uses a wildcard character (*****) to find the first employee with last name starting with “A”.

```
recEmployees.FindFirst "[LastName] Like 'A*'"
```

As an alternative to using the **FindFirst**, **FindLast**, **FindNext**, and **FindPrevious** methods, you can use the SQL Where statement to locate a record in a dynaset or snapshot.

In this code, the literal quotes must surround the string variable.

```
set strSQL = "Select * From Employees " & _
    "Where [Last Name] = " & _
    "'" & txtLastName.text & "'"
set recEmployees = dbMydb.OpenRecordset _
    (strSQL, dbOpenDynaset)
```

In most cases, using the **Find** methods are slower than creating a recordset by using an SQL query with a **Where** clause. However, if the recordset is small, using a **Find** method might be faster than recreating a recordset, because a **Find** method acts on fewer records.

For example, finding a record in an existing recordset that contains only 10 records will be faster than recreating a recordset with a **Where** clause that must search 20,000 records.

To locate a record in a table-type recordset, you use the **Seek** method. The recordset must have an index defined before you can use the **Seek** method.

The following table describes the possible comparisons you can use with the **Seek** method.

Comparison string	Description
“=”	Equal to the specified key values.
">"	Greater than the specified key values.
“>=”	Greater than or equal to the specified key values.
“<=”	Less than or equal to the specified key values.
“<”	Less than the specified key values.

You use multiple keys for indexes that have multiple fields. For example, if you have an index named “Name” that covers the fields LastName and FirstName, you might use the **Seek** method with the following code.

```
rs.Index = "Name"
rs.Seek "=", "Davolio", "Nancy"
```

Using the **Seek** method to find records on a table is faster than using the **Find** methods on a dynaset or a snapshot. However, with the **Seek** method, you can only search by using defined indexes.

An action query is an SQL statement that updates, deletes, or inserts records in a database. An action query does not return any rows.

To execute an action query, you use the **Execute** method of the **Database** object.

The following code runs an action query and displays the number of affected records.

```
dbMydb.Execute "Update Products", dbFailOnError
Msgbox "This query changed " & _
    dbMydb.RecordsAffected & _
    " records."
```

Use the **dbFailOnError** option to cause the **Execute** method to return a run-time error if any of the affected records are locked or cannot be updated or deleted.

The **RecordsAffected** property returns the number of records that have been modified, inserted, or deleted.

You can also use a **QueryDef** object to run an action query. If the query requires parameters, use the **Parameters** collection of the **QueryDef** object to set the parameters, as shown in this code.

```
Dim qryUpdate As QueryDef
Set qryUpdate = db.QueryDefs("Update Prices")
qryUpdate.Parameters ("Enter Increase") = .10
qryUpdate.Execute dbFailOnerror
Msgbox qryUpdate.RecordsAffected
```

You can use SQL statements in code to retrieve or modify information in a database.
{ewc mvimg, mvimage,!tip.bmp}

SQL SELECT Statement

You can dynamically create an SQL string at run time, and use the **OpenRecordset** method to create a recordset based on the SQL statement. For example:

```
Dim stSQL As String
strSQL = "Select * from Employees"
Set recEmployees = dbMydb.OpenRecordset (strSQL)
```

SQL UPDATE Statement

The SQL **UPDATE** statement modifies existing records. The **dbFailOnError** option causes the update to be rolled back if an error occurs during the update. In addition, your error handler will be invoked.

This following code shows the use of an SQL **UPDATE** statement.

```
strSQL = "UPDATE Products SET [Discount] = 1 " & _
            "WHERE [Discount] = 0"
dbMydb.Execute strSQL, dbFailOnError
```

SQL Insert Statement

You can use the SQL **INSERT** statement to insert a new record. To see a code sample that inserts a record into the Employees table, click this icon.
{ewc mvimg, mvimage,!code.bmp}

```
Select Case recWhatType.Type
    Case dbOpenTable
        Msgbox "This is a table-type recordset"
    Case dbOpenDynaset
        Msgbox "This is a dynaset-type recordset"
    Case dbOpenSnapshot
        Msgbox "This is a snapshot-type recordset"
End Select
```

```
'Prints first field in current row in recordset
Print recEmployees.Fields(0)

'Loops through all fields In current record
For Each fldCurrent In recEmployees.Fields
    Print fldCurrent.Value
Next
```

```
If IsNull(recEmployees("Notes")) Then
    txtNotes.Text = "No Notes"
Else
    txtNotes.Text = recEmployees("Notes")
End If
```

```
recEmployees.MoveNext
If recEmployees.EOF Then
    recEmployees.MoveLast
    Beep
End If
```

```
Private Sub cmdUpdate_Click()
If recEmployees.EditMode = dbEditNone Then
    recEmployees.Edit
End If
recEmployees("LastName") = txtLastName.Text
recEmployees("FirstName") = txtFirstName.Text
recEmployees.Update
End Sub
```

```
Private Sub cmdFindFirst_Click()
    Dim varBookmark as Variant
    varBookmark = recEmployees.BookMark 'Save current location
    recEmployees.FindFirst "[EmployeeID] = 5"
    If recEmployees.NoMatch Then
        recEmployees.BookMark = varBookmark 'Return to saved location
    End If
    FillFields 'user written proc- copies data to form fields
End Sub
```

```
strSQL = "INSERT INTO EMPLOYEES " & _
    "([First Name], [Last Name]) " & _
    "VALUES ('Joe', 'Smith')"
dbMydb.Execute strSQL, dbFailOnError
```

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

You can use the Data control or the data access objects to create basic database applications quickly and with minimal code.

However, if you want your applications to handle errors, manage multiple users, and prevent data integrity errors, you will need to add more code.

In this chapter, you will learn how to provide advanced features for your database application.

Objectives

By the end of this chapter, you will be able to create an application that:

® Handles errors caused by rule and referential integrity violations for a Microsoft Jet database.

® Uses transactions.

® Opens a recordset for exclusive use.

® Resolves locking conflicts.

® Accesses a Microsoft SQL Server database by using ODBCDirect.

When you create a database application, you must ensure that only valid data is entered in the database.

With Microsoft Access, you can create rules and establish referential integrity constraints in a database to help ensure data integrity. If you violate these constraints, you will receive a run-time error. You should provide code that traps these errors, and enables the user to correct the data and continue.

This section explains how to ensure that only valid data is entered into a database.

Topics in this section include:

® Defining Validation Rules

® Referential Integrity

® Transactions

When using a Microsoft Jet database, you can define rules that specify which data is valid for a field or a table. These rules are stored with the database, and are enforced no matter how you modify the data.

The Northwind database has several rules defined. For example, a rule for the Quantity field in the Order Details table specifies that the quantity must be greater than 0.

Rule Properties

With Visual Basic, you can create or view the rules of a database. The **Field** and **Recordset** objects have two properties that relate to rules. The following properties are supported only in Microsoft Jet databases.

® **ValidationRule**

Defines the criteria for the rule.

® **ValidationText**

Provides an error message that can be displayed if the rule is violated.

To see a code sample that displays the **ValidationRule** and **ValidationText** properties for the Quantity field, click this icon.
{ewc mvimg, mvimage,!code.bmp}

Note To define a rule programmatically, you can set the **ValidationRule** property of a **Field** object. This chapter does not discuss how to define a rule programmatically. For information about creating rules, see the **ValidationRule** property in Visual Basic Help.

Rule Violation Errors

If you violate a rule when updating a record, a run-time error will occur. The **ValidateOnSet** property determines when the rule violation error occurs.

If **ValidateOnSet** is **True**, a run-time error will occur when the field is set, as shown in the following code.

```
recOrders("Quantity").ValidateOnSet = True
recOrders("Quantity").Value = 0  'ERROR GENERATED HERE
recOrders.Update
```

If **ValidateOnSet** is **False**, a run-time error will occur when the **Update** method is executed, as shown in the following code.

```
recOrders("quantity").ValidateOnSet = False
recOrders("quantity").Value = 0
recOrders.Update  'Error generated here.
```

Handling Rule Violations

Generally, if a rule violation occurs, you'll want to allow the user to correct the data and attempt the update again.

If you are using the Data control and you violate a rule, the Error event occurs. The Data control displays the **ValidationText** property and cancels the update. The user can modify the data and click the Data control to attempt the update again.

You can add code to the Error event to take a different action when the error occurs. For example, you might display text in a caption rather than in a message box, or change the **ForeColor** property of the fields to indicate that an error has occurred.

If you are using data access objects (DAO) and you violate a rule, you will receive a run-time error. You should trap for this error and inform the user.

To see a code sample that displays a message if a rule violation occurs, click this icon.
{ewc mvimg, mvimage,!code.bmp}

The **Description** property of the **Err** object contains the value of the **ValidationText** property.

You can use the **CancelUpdate** method to cancel the current **Edit** or **AddNew** method, and to refresh the current record. You can either leave the user's data on the form, or update the form with the current data from the database.

Referential integrity refers to the constraints that are placed on a database to preserve the defined relationships between tables when records are added, changed, or deleted.

Defining Referential Integrity Constraints

The Microsoft Jet database engine can enforce referential integrity in a Microsoft Jet database.

For example, in the Northwind database, a relationship is defined between the Customers table and Orders table, and referential integrity is enforced. If you try to add an order with an invalid CustomerID or delete a customer record that contains orders, you will receive a run-time error.

Note You can define relationships between tables either programmatically by using Visual Basic, or manually by using Microsoft Access. For information about creating relationships programmatically, see the **CreateRelation** method in Visual Basic Help.

Cascade Updates and Deletes

When you have established referential integrity, you can specify the options **Cascade Update Related Fields** and **Cascade Delete Related Records**. If you set these options, the Microsoft Jet database engine will enable the user to change and delete records, and it will automatically change or delete related records in related tables.

For example, assume that you have created a relationship between the Customers table and the Orders table. If you select the option **Cascade Delete Related Records** for the Customers table, the Microsoft Jet database engine will delete all related records in the Orders table whenever the user deletes a customer record.

Note The Microsoft Jet database engine does not display a warning when deleting related records. To avoid losing data when deleting records, you must understand how relationships between tables are defined in your database.

Handling Referential Integrity Violations

If you try to violate referential integrity constraints, you will receive a run-time error. You should trap this error and inform the user.

To see a demonstration of how to set referential integrity constraints and handle referential integrity violations, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

To see a code sample that displays an error message if the user attempts to delete a customer record that contains an order, click this icon.
{ewc mvimg, mvimage,!code.bmp}

Notice that the record is not deleted, and the current record is not repositioned. The text boxes on the form remain filled with the current data.

Preventing Referential Integrity Violations

To prevent referential integrity violations, you can provide a list box with valid entries. If the valid entries are too long to store in a list box, you can enable the user to enter search criteria, and then display a list of valid entries.

For example, when you add a new product, you can enable the user to enter a value for a product name, and then display a grid with all products that match that name. The user can then select a valid product from the grid.

A transaction is a series of changes made to a database. You can group several database changes within a transaction.

If all operations are successful, you commit the transaction. If any of the operations fail, you roll back the transaction.

Transaction Methods

The transaction methods apply to the **Workspace** object. Use the **BeginTrans** method before updating the first record. If any subsequent update fails, use the **Rollback** method to undo all of the updates. Use the **CommitTrans** method after you have successfully updated the last record.

To see a code sample that uses a transaction to add an entry to the Orders table and the Order Details table, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

Using Transactions to Improve Performance

In addition to using transactions to maintain data integrity, you can also improve performance by enclosing an SQL **Update** statement in a transaction. The operations in the transaction are buffered, and are not written to disk until the transaction is committed.

The following code encloses an SQL **Update** statement in a transaction.

```
DBEngine.Workspaces(0).BeginTrans
stSQL = "UPDATE Products SET [Unit Price] = [Unit Price] * 0.1"
db.Execute strSQL, dbFailOnError
DBEngine.Workspaces(0).CommitTrans
```

Note Transactions can be nested within a **Workspace** object. Before you can commit or roll back a transaction at a higher level, you must commit or roll back the current transaction.

If you want to have transactions that overlap but are independent of each other, you can create a second **Workspace** object.

For information about creating **Workspace** objects, see the **CreateWorkspace method** in Visual Basic Help.

When you allow multiple users to access a database at the same time, the database application becomes more complex. You'll need to handle several errors that can occur when multiple users access data. For example:

® Two users try to edit the same data.

The Microsoft Jet database engine automatically locks data to prevent two users from editing a record at the same time. If users try to update a locked record, they will receive a run-time error.

® A user changes a record that another user is viewing.

If a user attempts to update a record, and the data has been modified since the **Edit** method was executed, the Microsoft Jet database engine will return an error.

® A user deletes a record that another user is viewing.

If a user attempts to access a record that has been deleted, the Microsoft Jet database engine will return an error.

This section describes multiple user issues that occur when accessing data.

Topics in this section include:

® Opening a Table for Exclusive Use

® Microsoft Jet Database Engine Locking

® Handling Locking Errors

To avoid problems that can occur when you allow multiple users to access a database, you can open a table for exclusive use. Opening a table in this way prevents other users from using the table. Although this is a fairly restrictive solution, it may be appropriate for your situation.

The *options* argument to the **OpenRecordset** method determines how a recordset is opened.

The following code opens a table for exclusive use. You are allowed to modify the Orders table, but the user is not allowed to view or modify data in the table.

```
Set recOrders = dbMydb.OpenRecordset _
    ("Orders", dbOpenTable, dbDenyRead + dbDenyWrite)
```

The following code opens a record for exclusive write. You are allowed to view and modify the Orders table, and the user can view data in the table, but cannot modify the data.

```
Set recOrders = dbMydb.OpenRecordset _
    ("Orders", dbOpenTable, dbDenyWrite)
```

The following code opens a recordset in read-only mode. You can read data in the table, but you cannot modify data. Other users are not affected.

```
Set recOrders = dbMydb.OpenRecordset _
    ("Orders", dbOpenTable, dbReadOnly)
```

The following code opens a recordset in append mode. You can only add records. You cannot view or modify any existing records. Other users are not affected.

```
Set recOrders = dbMydb.OpenRecordset _
    ("Orders", dbOpenDynaset,  dbAppendOnly)
```

If you intend only to add records to a table, use the **dbAppendOnly** option for better performance. The **dbDenyRead** option is available only in table-type recordsets.

For a list of valid values for the *options* argument, see the **OpenRecordset method** in Visual Basic Help.

All database management systems provide some sort of locking mechanism to prevent two users from updating data at the same time.

The Microsoft Jet database engine provides page-level locking. Pages are blocks of records that are 2048 bytes (2K) in size. Visual Basic stores as many records as will fit on each page. When Visual Basic locks the page containing a record you're editing, all the other records on that page will also be locked.

Note Information in this section is specific to databases that use the Microsoft Jet database engine. When you access databases through ODBC, the locking mechanism is handled by the database management system.

Pessimistic vs. Optimistic Locking

The value of the **LockEdits** property of the **Recordset** object determines when a lock is placed on the records in a recordset.

Pessimistic Locking

If the **LockEdits** property is **True**, pessimistic locking is in effect. The page containing the current record is locked as soon as you use the **Edit** method. The page is unlocked when you use the **Update** method.

The default locking strategy locks records for a longer period of time, but ensures that once the **Edit** method has been executed, another user cannot change the data.

Optimistic Locking

If the **LockEdits** property is **False**, optimistic locking is in effect. The page containing the record is locked only while the record is being updated.

Locks are in place for a shorter period of time, and multiple users can use the **Edit** method without locking the page. However, when you use optimistic locking, you'll need to handle possible errors when the user executes the **Update** method.

If you do not expect users to attempt to modify the same record very often, consider setting **LockEdits** to **False**.

Unlocking

The Microsoft Jet database engine marks a page to be unlocked as soon as the update completes. However, locks are not removed until other actions are complete.

You can use the **Idle** method of the **DBEngine** object with the **dbFreeLocks** option to suspend processing and allow the Microsoft Jet database engine to release locks and catch up on other background tasks, as shown in the following code.

```
DBEngine.Idle dbFreeLocks
```

When multiple users are updating a database, you'll need to add code that traps for the following errors.

® **3260 — Couldn't update; currently locked**

Occurs on the **Edit** method when the record is locked. Your code should wait for a short interval, and then call **Edit** again, or inform the user of the error.

® **3186 — Couldn't save; currently locked**

Occurs on the **Update** method when the record is locked. Your code should wait for a short interval, and then call **Update** again or inform the user of the error.

® **3197 — Data has changed; operation stopped**

Occurs on the **Edit** or **Update** method if another user has changed data since you last accessed it. Your code should refresh the form with the latest data or inform the user of the error.

For a complete list of errors available in Visual Basic, see **Trappable Errors** in Visual Basic Help.

To see a demonstration of how to handle locking errors, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Handling Errors on Edit

To see a code sample that handles locking errors that can occur on the **Edit** method, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

Notice that if the record is locked when the user executes the **Edit** method, a message will be displayed. If the data has changed, the code refreshes the form with the latest data.

The statement **recOrders.bookmark = recOrders.bookmark** refreshes the current record in the recordset with the latest data from the database.

Handling Errors on Update

To see a code sample that handles errors that can occur on the **Update** statement, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

Handling Errors on Records Deleted by Other Users

Once your dynaset-type recordset is fully populated, if another user deletes a record from the database, the pointer to the entry will not be deleted from the dynaset.

However, the deleted record will no longer be in the database. You cannot view the record, and if you attempt to access data in that record, an error will occur.

The easiest way to handle this error is to delete the entry that points to the deleted record from the dynaset.

To see a code sample that shows how to handle an error that is generated by an attempt to access a deleted record, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

You can use Visual Basic to access several indexed sequential access method (ISAM) databases, such as dBASE, Paradox, and Microsoft Visual FoxPro.

Visual Basic provides drivers for various ISAM databases. These drivers are listed in the registry.

You can also use Visual Basic to access standard data files, such as comma-delimited text files.

Note Databases that use the Microsoft Jet database engine are considered native to Visual Basic.

This section describes how to access external data with Visual Basic, including ISAM databases and standard data files.

This section includes the following topics:

® Data Access Choices

® Accessing ISAM Databases

® Working with Data Files

You can access data in an external database by attaching the external database or by opening it directly.

Attaching a Table to a Microsoft Jet Database

When an external database is attached to a Microsoft Jet database, the table's data remains in the external database. However, the connection information and table definition are stored in the Microsoft Jet database. You use the table as you would any other Microsoft Jet database table, except that you cannot create a table-type recordset in an attached table.

You can use Microsoft Access to easily attach a table to a Microsoft Jet database. You can also attach a table programmatically by using Visual Basic.

For information about attaching tables programmatically, see the **CreateTableDef method** in Visual Basic Help.

Note You can usually access data in ODBC tables that are attached to a Microsoft Jet database more quickly than you can access data in an ODBC database that you have opened directly. When possible, consider attaching external tables rather than opening them directly.

Opening a Table Directly

When you open an external table directly, you specify the connection information by using the arguments of the **OpenDatabase** method or the **Connect** property of the Data control.

To see a code sample that uses the **OpenDatabase** and **OpenRecordset** methods to open a table directly, click this icon.
{ewc mvimg, mvimage,!code.bmp}

To open an ISAM database directly, set the **Database** argument of the **OpenDatabase** method (or the **DatabaseName** property of the Data control) to the folder name that contains the ISAM tables. If your network supports it, you can use a network path for the database name (for example, \\Myserver\Myshare).

Set the **Connect** property of the **Database** object or Data control to the type of database to be opened. For a list of possible values, see the **Connect property** in Visual Basic Help.

The following code opens a Microsoft Visual FoxPro database directly.

```
Public Sub OpenFoxProTable()
    Dim dbFox As Database
    Dim recAccounts As Recordset
    Set dbFox = OpenDatabase _
        ("\\FoxPro\Data\AP", False, False, _
        "FoxPro 5.0;")
    Set recAccounts = dbFox.OpenRecordset("Accounts")
End Sub
```

To attach an ISAM database to an existing .mdb database, you must create a new **TableDef** object and set the **Connect** and **SourceTableName** properties of the **TableDef** object.

The following code attaches a Visual FoxPro table to a Microsoft Jet database. You can create a new Microsoft Jet database or use an existing one.

```
Dim tbdAttach as TableDef
Set dbMydb=OpenDatabase ("Mydb.mdb")
Set tbdAttach = dbMydb.CreateTableDef ("Attached FoxPro Table")
tbdAttach.Connect = "FoxPro 5.0;DATABASE=\\FoxPro\AP"
tbdAttach.SourceTableName = "Accounts"
dbMydb.TableDefs.Append tbdAttach
```

Once you have attached the table to a Microsoft Jet database, you can open it by using the following code.

```
Set dbMydb = OpenDatabase ("Mydb.mdb")
Set recAttach = dbMydb.OpenRecordset ("Attached FoxPro Table")
```

Distributing the Executable

When you distribute an application that accesses an ISAM database, you must provide the appropriate ISAM drivers to the user.

You can create Visual Basic applications that work with information from standard data files rather than with a database.

When you work with information from standard data files, you can use some of the following options.

® File I/O statements

You can use standard file access statements in Visual Basic to read and write files. For information about the **Open** statement, see **Open** in Visual Basic Help.

® Manually import data into a database

You can use Microsoft Access to manually import a text file into a database. Microsoft Access supports various file formats, such as fixed-length, comma-delimited, and so on.

® Programmatically import into a database

You can create a program to read a text file and add the data to a database.

To see a code sample that shows the basic steps for using file I/O to read a file and insert the data into a database, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

An application's design significantly influences its performance. When creating database applications, consider the following suggestions.

® Know what you are asking for.

Some SQL queries are more efficient than others. For example, sorting on a field that contains an index is faster than sorting on a field that does not contain an index.

For more information about query optimization, search for **Optimizing Performance** in Visual Basic Books Online and click **Optimizing Performance**.

® Request only the data you need.

Avoid creating dynaset-type and snapshot-type recordsets on an entire table. Your application design should enable the user to enter criteria and return a limited recordset.

Rather than selecting all columns in a record, select only the necessary columns.

The following code creates a recordset based on a value entered by the user. For example, if the user enters “A”, the code creates a recordset of all employees with a last name starting with “A”.

```
strSQL = "Select [LastName], [FirstName] From Employees " & _
        "Where [LastName] Like " & _
        "'" & txtName.text & "'" & "*"
Set recEmployees = dbMydb.OpenRecordset (strSQL,dbOpenDynaset)
```

® Use only the functionality required.

If you do not need to update the database, open the database as read-only. This improves performance because the Microsoft Jet database engine does not need to keep track of locks.

If you do not need to update a recordset, open it as read-only.

If you are only adding to a recordset, open it with the **dbAppendOnly** option.

® Use transactions.

Using a transaction reduces disk access to about once per transaction, instead of once per record. This can significantly improve an application's performance. You may want to group transactions. If a transaction is too large, it uses memory that other operations may need.

® Use stored queries.

In general, it is better to create stored queries than it is to use SQL in text. A stored query is precompiled, so you save compilation time at run time. In addition, it is often easier to maintain queries that are stored in an .mdb file, rather than created throughout your application.

If possible, use stored queries or SQL for batch operations, such as updating a group of records, rather that updating records one at a time with the **Update** method.

® Attach SQL tables.

Attach ODBC database tables to a local .mdb database. The data definition and connection information is maintained with the .mdb file, and therefore improves performance.

® Write your own performance test.

There are many factors that affect the performance of an application. The best way to find out if a feature will improve performance is to try it with the data in your environment.

1. **You have defined a rule such that the value of the ShipCost field of your database must be equal to or greater than 4 percent of the value of the OrderPrice field. How can you have your application generate a run-time error when this rule is violated and the user attempts to write the incorrect data to the ShipCost field in the database?**

{ewc mvimg, mvimage,!answer.bmp}
A. Establish referential integrity between the ShipCost field and the OrderPrice field.

{ewc mvimg, mvimage,!answer.bmp}
B. Set the **ValidationRule** property to **True**.

{ewc mvimg, mvimage,!answer.bmp}
C. Set the **ValidateOnSet** property to **False**.

{ewc mvimg, mvimage,!answer.bmp}
D. Set the **ValidationText** property to **True**.

2. **Transactions are used primarily to maintain data integrity, but can also result in improved performance. How can the code shown below improve performance?**

```
DBEngine.Workspaces(0).BeginTrans
    sqltext = _
        "UPDATE Products SET [Unit Price] = [Unit Price] * 0.1"
    db.Execute sqltext, dbFailOnError
DBEngine.Workspaces(0).CommitTrans
```

{ewc mvimg, mvimage,!answer.bmp}

A. The **Update** statement bypasses buffering and writes the transaction to disk immediately.

{ewc mvimg, mvimage,!answer.bmp}

B. When the **Update** statement is used, the operation can be canceled without invoking the **Rollback** method when the transaction fails.

{ewc mvimg, mvimage,!answer.bmp}

C. The **Update** statement uses buffering and writes changes to disk only after the transaction is committed.

{ewc mvimg, mvimage,!answer.bmp}

D. When a transaction fails, the **Update** statement buffers error message text when the **Rollback** method is invoked.

3. With which type of recordset can you use the dbDenyRead argument with the OpenRecordset method?

{ewc mvimg, mvimage,!answer.bmp}

A. Any recordset.

{ew

B. Only dynaset-type recordsets.

{ewc mvimg, mvimage,!answer.bmp}

C. Only snapshot-type recordsets.

{ewc mvimg, mvimage,!answer.bmp}

D. Only table-type recordsets.

{ewc mvimg, mvimage,!answer.bmp}

4. When is pessimistic locking in effect?

{ewc mvimg, mvimage,!answer.bmp}

A. When the value of the **LockEdits** property is **True**.

{ewc mvimg, mvimage,!answer.bmp}

B. After the **Edit** method has been executed and before the **Update** method executes.

{ewc mvimg, mvimage,!answer.bmp}

C. When the value of the **LockEdits** property is **False**.

{ewc mvi

mg.
mvi
ma
ge.!
ans
wer
.bm
p}

D. When the **Update** method is executing.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

```
Sub DisplayRule()

    Set rstOrders = dbMydb.OpenRecordset("Order Details")

    ' Prints ">0"
    Debug.Print rstOrders("Quantity").ValidationRule

    ' Prints "Quantity must be greater than 0"
    Debug.Print rstOrders("Quantity").ValidationText

End Sub
```

```
Private Sub cmdUpdate_Click()
    On Error GoTo update_err
    rs.Edit
    rs("order id") = txtOrder.Text
    rs.Update
    Exit Sub
update_err:
    If Err.Number = 3316 Then 'rule violation
        lblError.Caption= Err.Description
        rs.CancelUpdate
        Exit Sub
    End If
End Sub
```

```
Private Sub cmdDelete_Click()
    On Error GoTo del_err
    rs.Delete
    rs.MoveNext
    FillFields 'user written procedure
    Exit Sub
del_err:
    'Ref Integrity Violation
    If Err.Number = 3200 Then
        lblError.Caption = _
            "Cannot delete customer with orders."
        Exit Sub
    Else
        'handle other errors...
    End If
End Sub
```

```
Sub cmdAddOrder_Click ()
    On Error GoTo AddOrder_Err
    DBEngine.WorkSpaces(0).BeginTrans
        db.Execute "Insert Into Orders...  .", dbFailOnError
        db.Execute "Insert Into [Order Details]...." , dbFailOnError
    DBEngine.WorkSpaces(0).CommitTrans
    lblStatus.Caption = "Updates complete"
    Exit Sub

AddOrder_Err:
    lblStatus.Caption = Err.Description
    Msgbox "Not all updates successful"
    DBEngine.Workspaces(0).Rollback
    Exit Sub
End Sub
```

```
Private Sub cmdEdit_Click()
    On Error GoTo HandleError
    rs.Edit
    ButtonEditAddMode
    Exit Sub
HandleError:
    Select Case Err.Number
    Case 3260 'page currently locked
        MsgBox "Record is currently locked. Try again later."
    Case 3197 'data has changed
        MsgBox "Data has been changed and will be refreshed."
        rs.Bookmark = rs.Bookmark 'get updated data
        FillFields
        rs.Edit
    Case Else
        MsgBox Err.Number & ": " & Err.Description
    End Select
End Sub
```

```
Private Sub cmdSave_Click()
    Dim answer As Integer
    On Error GoTo HandleError
    rs.Fields("Category Name") = txtCategoryName.Text
    rs.Fields("Description") = txtDescription.Text
    rs.Update
    rs.Bookmark = rs.LastModified
    FillFields
    ButtonNavigateMode
    Exit Sub
HandleError:
    Select Case Err.Number
    Case 3260
        MsgBox "Record is currently locked. " & _
            "Try Save again later or cancel changes."
    Case 3197
        answer = MsgBox("Data has been changed by another user. " & _
            "Overwrite changes?", vbYesNo)
        If answer = vbYes Then
            Resume
        Else
            rs.Bookmark = rs.Bookmark 'refresh w/other users changes
            cmdCancel_Click
        End If
    Case Else
        MsgBox Err.Number & ": " & Err.Description
    End Select
End Sub
```

```
Private Sub cmdMoveFirst_Click()
    On Error GoTo err_movefirst:
    retry_MoveFirst:
    rs.MoveFirst
    txtLName.Text = rs("Last Name")  'Error may occur
    Exit Sub
err_Movefirst:
    If Err.Number = 3167 Then
        'record is deleted,
        'remove entry from your dynaset
        rs.Delete
        Resume retry_MoveFirst
    Else
        MsgBox Err.Description
        Exit Sub
    End If
End Sub
```

```
Private Sub Command4_Click()
    Dim fhandle As Integer
    fhandle = FreeFile()
    Open "newCust.txt" For Input As fhandle
    Do
        'Read file using some Input statement
        'Build sql string
        'Execute SQL Insert Statement
    Loop Until EOF(fhandle)
End Sub
```

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

One of the greatest advantages of using Visual Basic is that you can create advanced Windows applications without knowing the Windows application programming interface (API).

If you want your application to include functionality that is not provided by Visual Basic, you can use the Windows dynamic-link libraries (DLLs) directly to access the Windows API.

For example, you can use a DLL to create a form that remains on top of other forms, even when it does not have the focus.

This chapter shows you how to use DLLs to extend the functionality of your Visual Basic applications.

Objectives

At the end of this chapter, you will be able to:

- Describe the purpose of DLLs.
- Map C language data types to Visual Basic data types.
- Describe the purpose of the **Any** keyword.
- Pass a null value to a DLL procedure.
- Explain when to use the ANSI version of a function rather than the Unicode version.
- Describe the difference between the **ByVal** and **ByRef** keywords.
- Implement a callback function by using the **AddressOf** keyword.
- Create a Visual Basic application that:
  - Causes the title bar of a window to flash.
  - Causes a form to remain as the topmost form in an application.
  - Uses a Windows timer with a callback function.

It's useful to have an understanding of what a DLL is and of how it is used, even though you may never write one. As a Visual Basic programmer, you will probably use DLLs at some point during the development of your applications.

Definition of a DLL

A dynamic-link library (DLL) is a set of procedures that are external to your application, and can be called from your application. DLLs are not bound to your executable file, so they can be linked to at run time, instead of being loaded at compile time. The libraries can be updated independently of the application, and multiple applications can share a single DLL.

Advantages of Using DLLs

With DLLs, you can:

® Accomplish tasks that are not possible in Visual Basic.

DLLs can perform tasks that are difficult or impossible to code by using Visual Basic. For example, you can call a DLL procedure that causes your application to remain as the topmost window.

® Improve performance.

Code in a DLL often runs faster than code in Visual Basic. For procedures that require fast performance, you can write a DLL, and then call the DLL from Visual Basic.

® Update independently of the application.

You can update a DLL without recompiling the application that calls the DLL. This makes it easier to manage your application.

There are two basic steps to using a DLL procedure.

1. Use the **Declare** statement to tell Visual Basic which procedure you want to use.
2. Call the DLL procedure from the appropriate place in your code.

These steps are discussed later in this chapter.

This topic briefly introduces both declaring and calling a DLL procedure in Visual Basic.

To see a demonstration of how to declare and use a simple DLL procedure, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Declaring a Procedure

DLL procedures reside in files that are external to your Visual Basic application, so you must declare a procedure before you can call it. Declaring a procedure provides Visual Basic with the information it needs to find and run the procedure you want to use.

The following code declares the **FlashWindow** API procedure.

```
'In the general Declarations of a .BAS module.
Declare Function FlashWindow Lib "User32" _
        (ByVal hWnd As Long, ByVal bInvert As Long) _
        As Long
```

Calling a Procedure

Once you have declared a procedure, you call it just as you would a Visual Basic procedure, as shown in the following code statement.

```
lResult = FlashWindow(form1.hWnd, 1)
```

Note If you declare the function in a module, you can use the Object Browser to paste the call for the procedure into your application.

Saving Your Work

Visual Basic cannot verify that you are passing the correct values to a DLL procedure. If you pass incorrect values, your Visual Basic application may fail. For this reason, it is good idea to save your work often when working with DLLs.

You can use the API Viewer provided with Visual Basic to retrieve the appropriate declarations for the Windows API procedures.

Retrieving Declarations with the API Viewer

When you load an API text file, such as the file Win32api.txt, the API Viewer reads the file and displays the constants, types, and **Declare** statements for the procedures in the Windows API. You can use the API Viewer to copy a declaration to the Clipboard, and then paste it into your Visual Basic application.

Loading a Declaration from a Database

To retrieve specific DLL declarations, it is faster to load and search a database (.mdb) file than to search a text (.txt) file. The API Viewer gives you the choice of searching the API text file or converting the text file to a database file. When you are prompted to create a database, and you respond Yes, the API Viewer will create an .mdb database file. You can then load the .mdb file whenever you use the API Viewer.

For information about the purpose of specific DLL procedures, see the Win32 Software Development Kit (SDK) on the Microsoft Developer Network CD-ROM.

To declare a DLL procedure, you place a **Declare** statement in the Declarations section of a form, standard module, or class module.

Determining the Scope of a DLL Procedure

If you declare a DLL in a standard module, the DLL is public by default, and can be called by code from anywhere in your Visual Basic application. To declare a DLL procedure in a form or class module, you must include the **Private** keyword in the declaration.

Syntax of the Declare Statement

The syntax for the **Declare** statement is:

[**Public|Private**] **Declare Sub** *name* **Lib** "*libname*" [Alias "*aliasname*"] [(arglist)]

– or –

[**Public|Private**] **Declare Function** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [(*arglist*)] [**As** *type*]

The **Declare** statement contains the following elements.

Element	Description
Sub or **Function** procedure	**Function** procedures return values, while **Sub** procedures do not.
name	The procedure name used in your code.

Lib "*libname*"	The name of the library the procedure is in. If you have written your own DLL, you specify the name of your DLL. For example:
	`Declare Function AddNum Lib "c:\add.dll" _`
	`(ByVal a as long, ByVal b as long) As long`
	
	If you do not specify a full path to the DLL, the DLL must be in the search path.

arglist	The arguments required by the procedure. The data type of each argument must match the data type specified in the declaration. The argument must be passed either by value or reference.
Alias	If a DLL contains a name that is not a legal identifier in Visual Basic, you can use the **Alias** keyword to identify the called procedure. The **Alias** keyword specifies a name other than the actual name used in the procedure.

Typically, DLLs are written in the C programming language, so the arguments are defined with C data types. When you use the **Declare** statement for a DLL in Visual Basic, you must match the data type of the argument to the C data type.

The following table lists commonly used C language declarations and their Visual Basic equivalents for 32-bit compilers.

C language declaration	**Visual Basic, declaration**	**Call with**
Pointer to string (LPSTR)	**ByVal** <variable> **As String**	A **String** or **Variant** variable.
NULL	**ByVal** <variable> **As String**	A **vbNullString** constant.
char	**ByVal** <variable> **As Byte**	An expression that evaluates to a **Byte** data type.
Integer	**ByVal** <variable> **As Long**	An expression that evaluates to a **Long** data type.
Windows handle (hWnd, hDC, hMenu)	**ByVal** <variable> **As Long**	An expression that evaluates to a **Long** data type.

Note In 16-bit C compilers, an **int** data type maps to a Visual Basic **Integer** data type. In 32-bit C compilers, an **int** data type maps to a Visual Basic **Long** data type.

For a complete list of C language data types and guidelines for using them in Visual Basic, search on **Converting C Declarations to Visual Basic** in Visual Basic Books Online. Select **Accessing DLLs and the Windows API** to locate the jump to the topic.

When a DLL procedure accepts a parameter with a null value, it expects a value of zero (0) rather than an empty string. To pass a null value, declare the argument with **ByVal As String**, and pass the constant vbNullString, as shown in the following code.

```
Declare Function FindWindow Lib "User32" _
    Alias "FindWindowA" _
    (ByVal lpClassName As String, _
    ByVal lpCaption As String) As Integer

rc = FindWindow (vbNullString, "Microsoft Word")
```

Note In Visual Basic version 3.0, the only way to pass a null value was with **ByVal 0&**. If you used a DLL that took a string or a NULL, you either had to declare the argument **As Any**, or declare the DLL twice, once with a **String** argument and once with a **Long** argument.

Beginning with Visual Basic 4.0, you can declare an argument as **String** and pass the Visual Basic defined constant **vbNullString**.

Visual Basic **String** data types are stored in a different format than C language **String** data types. Most DLLs and all procedures in the Windows API recognize the C language string format. When passing string values in Visual Basic, you must adjust for these differences by declaring string arguments by value, and by allowing for different string lengths.

Declaring String Arguments By Value

If a DLL procedure expects a C language string as an argument, you should declare the Visual Basic argument as a **String** preceded by the **ByVal** keyword. The value of the string variable is actually a pointer to a location in memory that contains the string. Therefore, a DLL procedure can modify a Visual Basic string variable that it receives as an argument.

Allowing for Different String Lengths

A DLL cannot increase the length of a Visual Basic string, and writes beyond the end of the string if it is not long enough. To allow enough space for any string length, use one of the following two methods.

® Fill the character string with the maximum number of characters you expect.

```
Dim FilePath As String
FilePath = String (255, 0)
```

® Define the string as a fixed-length string with the maximum possible length.

```
Dim FilePath As String * 255
```

Strings returned by a DLL procedure contain a null character at the end of the string. If you are combining several strings, you need to remove the null character. Since procedures generally return an integer indicating the string length, you can either use the **Left** function to remove the null character, or you can search for the null character by using **Chr(0)**.

Example of Passing a String

You can use the **GetWindowsDirectory** procedure to return the Windows folder. To do this, you provide an empty string for the path of the Windows folder, which should be long enough to accept the longest possible folder name. You also pass an integer argument to the procedure to indicate the maximum string length.

The **GetWindowsDirectory** procedure returns an integer that indicates the actual number of characters in the modified string. This string will contain a NULL as the last character. You can remove the NULL by searching for it, or by using the **Left** function, as shown in the following code.

```
Declare Function GetWindowsDirectory Lib "Kernel32" _
 Alias "GetWindowsDirectoryA" _
 (ByVal f as String, ByVal fLen as Long) As Long

Sub Command1_Click ()
    Dim sWinDir As String
    Dim lLen As Long
    sWinDir = String(255,0)
    lLen = GetWindowsDirectory(sWinDir, Len(sWinDir))
    sWinDir = Left(sWinDir, lLen)
    sWinDir = sWinDir + "\"
    MsgBox "Windows is in " + sWinDir
End Sub
```

To see a demonstration of how to pass a string, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

When you declare a DLL procedure, you must provide the correct function name. Visual Basic translates string arguments passed from Visual Basic to the DLL procedure.

ANSI and Unicode Character Sets

Every Win32 API procedure that takes a string as an argument comes in two versions: the ANSI character set and the Unicode character set. In ANSI, each byte represents each character. In Unicode, two bytes are used to represent each character.

Translation Between ANSI and Unicode

Visual Basic version 5.0 uses the Unicode character set internally to store strings. However, when you call a function in the Win32 API, you should use the ANSI version of the API function. When you call a DLL procedure, Visual Basic automatically converts a string to ANSI. When you return from a DLL procedure, Visual Basic translates the string back to Unicode.

The Windows API functions are generally named with an 'A' or 'W' suffix for ANSI and Wide (Unicode), respectively. If you paste the **Declare** statement from the API Viewer, you will get the correct ANSI function name.

For example, to use the Win32 API function **SetWindowText**, the correct declaration is:

```
Declare Function SetWindowText Lib "user32" Alias "SetWindowTextA" _
    (ByVal hwnd As Long, ByVal lpString As String) as Long
```

Although your Visual Basic application will call **SetWindowText**, the actual function called will be the ANSI version, **SetWindowTextA**.

If your application calls a particular DLL repeatedly, you can create a procedure to call the DLL, rather than calling it directly each time. This procedure, known as a wrapper procedure, is similar in purpose to a centralized error handler.

Using a wrapper procedure to call a DLL simplifies your application by consolidating the calls and isolating the details of using the DLL.

Writing the Wrapper Procedure

In the following code, the function **GetWinDir** calls the **GetWindowsDirectory** API routine.

```
Public Function GetWinDir () As String
  sWindir = String(255,0)
  lLen = GetWindowsDirectory (sWindir, Len(sWinDir))
  sWinDir = Left (sWinDir, lLen)
  GetWinDir = sWinDir
End Function
```

Each time you want to determine the Windows folder, you invoke the function **GetWinDir**.

Calling the DLL Procedure

In the following code, **GetWinDir**, which is the actual call to the API routine, is embedded in the function.

```
Sub cmdGetWin_Click ()
    Msgbox GetWinDir
End Sub
```

This topic lists some Windows API procedures that you may find useful when writing Visual Basic applications. It also includes an example of how to use the Windows API procedure **SetWindowPos** to create a topmost window.

Frequently Used Windows API Procedures

Here are some DLL procedures from the Windows API that are often used to extend Visual Basic applications.

Windows API procedure	Purpose
BitBlt	Moves a bitmap from a source device context to a destination.
ExtractIcon, **DrawIcon**, **LoadIcon**	Manipulates icons.
FindExecutable	Finds and retrieves the name of the executable file associated with a specific file.
GetSystemDirectory	Gets the path of the Windows system directory.
GetSystemMetrics	Gets the width and height of the display elements in Microsoft Windows.
GetTempFileName	Returns a temporary file name and path by using the TEMP environment variable.
GetWindowPlacement, **SetWindowPlacement**	Gets or sets the show state, and the normal (restored), minimized, and maximized positions of a window.
SendMessage	Sends Windows messages to control applications. For example, the LB_SETTABSTOPS message sets tab stops in a list box. LB_FINDSTRING finds the first string in the list box that matches prefix text. Hundreds of other messages are also available.
BringWindowToTop, **SetActiveWindow**	Sets the focus to a particular window.
DragAcceptFiles, **DragFinish**	Supports drag-and-drop file features.
FindWindow, **ShowWindow**	Checks to see whether or not a specified application is currently running.
GetActiveWindow, **IsWindow**	Determines when a shell function has finished loading a program.
GetWindowText	Gets the caption title of a window or the text in a control, given a window handle.

For additional information about Windows API procedures, refer to the following publications.

- ® *Microsoft Windows Programmer's Reference,* published by Microsoft Press.
- ® *PC Magazine's Visual Basic Programmer's Guide to the Windows API* by Daniel Appleman, published by Ziff-Davis Press. This reference describes most of the Windows API procedures that can be used with Visual Basic.

Using the SetWindowsPos Function

The **SetWindowPos** API function creates a window (or form) that remains on top of other windows, even if it does not have the focus.

To use the **SetWindowPos** function, you must declare the appropriate constants. Use the API Viewer to copy the constants HWND_TOPMOST, SWP_NOSIZE, and SWP_NOMOVE from the text file Win32api.txt.

To see a code sample that uses the **SetWindowPos** function to create a topmost window, click this icon.
{ewc mvimg, mvimage,!code.bmp}

1. Which benefit do DLLs provide for the Visual Basic programmer?

{ewc mvimg, mvimage,!answer.bmp}

A. DLLs are linked to your application at compile time, assuring that your application will be recompiled if the DLL is later modified.

{ewc mvimg, mvimage,!answer.bmp}

B. DLLs can provide functionality not available in Visual Basic.

{ewc mvimg, mvimage,!answer.bmp}

C. DLLs execute more slowly than true Visual Basic code, but development time is reduced significantly when DLLs are used.

{ewc mvimg, mvimage,!answer.bmp}

D. Software distribution is simplified because DLLs are bound to a specific application and cannot be separated.

2. Which Visual Basic data type is equivalent to the int data type in a 16-bit C compiler?

{ewc mvimg, mvimage,!answer.bmp}

A. **Integer**

{ewc mvimg.mvimage.!answer.bmp}

B. **Long**

{ewc mvimg.mvimage.!answer.bmp}

C. **Byte**

{ewc mvimg.mvimage.!answer.bmp}

D. **Variant**

3. Why would you use the Any keyword?

{ewc mvimg.mvimage.!answer.bmp}

A. To enable a DLL to accept any data type as an argument.

{ewc mvimg.mvimage.!answer.bmp}

B. To enable any C data type to be equivalent to any Visual Basic data type.

{ewc mvimg.mvimage.!answer.bmp}

C. To enable any Visual Basic data type to be equivalent to any C data type.

{ewc

{mvimg,mvimage,!answer.bmp}

D. To enable you to declare an argument that will accept any data type.

{ewc mvimg,mvimage,!answer.bmp}

4. How should you pass a NULL value to a DLL?

{ewc mvimg,mvimage,!answer.bmp}

A. Declare the argument as **Any** and pass “” to the DLL.

{ewc mvimg,mvimage,!answer.bmp}

B. Declare the argument as **String** and pass ByVal 0& to the DLL.

{ewc mvimg,mvimage,!answer.bmp}

C. Declare the argument with **ByVal** as **String** and pass vbNullString to the DLL.

{ewc mvimg,mvimage,!answer.bmp}

D. Declare the argument as **Integer** and pass 0 to the DLL.

{ewc mvimg,

{mvimage,!answer.bmp}

5. In the code shown below, which function will actually be called?

```
Declare Function SetWindowText Lib "user32" Alias "SetWindowTextA" _
    (ByVal hwnd As Long, ByVal lpString As String) as Long
```

{ewc mvimg,mvimage,!answer.bmp}

A. **SetWindowText**

{ewc mvimg,mvimage,!answer.bmp}

B. The Unicode version of **SetWindowText**.

{ewc mvimg,mvimage,!answer.bmp}

C. The ANSI version of **SetWindowText**.

{ewc mvimg,mvimage,!answer.bmp}

D. The ANSI or Unicode version of **SetWindowText**, depending on the argument passed to the DLL.

6. Three of the languages below can be printed using either the ANSI character set or a multibyte

character set such as Unicode. Which language requires a multibyte character set?

{ewc mvimg, mvimage,! answer .bmp}
A. English

{ewc mvimg, mvimage,! answer .bmp}
B. Japanese

{ewc mvimg, mvimage,! answer .bmp}
C. French

{ewc mvimg, mvimage,! answer .bmp}
D. Spanish

7. What is the difference between arguments passed by value and by reference?

{ewc mvimg, mvimage,! answer .bmp}
A. If a procedure changes an argument passed by value, the original variable will be changed.

{ewc mvimg.mvimage.!answer.bmp}

B. When an argument is passed by value to a procedure , the address of the variable containing the argument is passed to the procedure.

{ewc mvimg.mvimage.!answer.bmp}

C. Strings are passed to procedures by reference, and other data types are passed by value.

{ewc mvimg.mvimage.!answer.bmp}

D. When a procedure changes an argument passed by reference, the value of the original variable will be changed.

8. You have written a callback procedure EnumProc in Visual Basic. What is the purpose of the AddressOf operator used in the line of code shown below?

```
EnumChildWindows form1.hWnd, AddressOf EnumProc, 0
```

{ewc mvimg.mvimage.!answer.bmp}

A. The **AddressOf** operator must be used with the name of the callback procedure to pass the address of the procedure to the DLL.

{ewc mvimg.mvimage.!answer.bmp}

B. The **AddressOf** operator is used to specify the standard module where the callback procedure is located.

{ewc mvimg.mvimage.!answer.bm

p}

{ewc mvimg, mvimage,!answer.bmp}

C. The **AddressOf** operator is used specify the location of the callback procedure when the procedure is not located in a standard module.

{ewc mvimg, mvimage,!answer.bmp}

D. The **AddressOf** operator is used to pass the name of the DLL to the callback procedure.

9. Which Win32 API function would you use to create a form that remains on top of other forms even when it does not have the focus?

{ewc mvimg, mvimage,!answer.bmp}

A. **SetWindowPlacement**

{ewc mvimg, mvimage,!answer.bmp}

B. **BringWindowToTop**

{ewc mvimg, mvimage,!answer.bmp}

C. **SetWindowPos**

{ewc mvimg.mvimage.!answer.bmp}

D. **IsWindow**

{ewc mvimg.mvimage.!answer.bmp}

E. **SetActiveWindow**

10. You are creating a Visual Basic application using a Windows timer with a callback function. You have named the timer procedure MyTimer and want an interval of a half second, so you use the following parameter values when the SetTimer function is called:

Parameter	Value
HWnd	0
idEvent	0
Interval	500
Callback	AddressOf MyTimer

Why have you set HWnd and idEvent to 0?

{ewc mvimg.mvimage.!answer.bmp}

A. A callback routine requires the NULL handle and an idEvent of 0.

{ewc mvimg.mvimage.!answer.bmp}

B. Visual Basic will automatically insert the correct values for HWnd and idEvent each time the timer fires.

{ew

C. The **GetHandle** and **GetEventId** methods will be invoked when the timer is

{cmvimg,mvimage,!answer.bmp}

started at run time to supply the correct values for these parameters.

{ewcmvimg,mvimage,!answer.bmp}

D. These parameters are not used when specifying a callback routine.

For an overview of ActiveX component development, covered in Chapters 6 through 12, click this icon.
{ewc mvimg, mvimage,!exppov.bmp}

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

The Component Object Model (COM) is an object-oriented architecture for building applications, and the foundation on which ActiveX components are built. ActiveX components allow programmers to combine reusable pieces of code into applications and services.

You can create your own ActiveX components, or you can purchase them from Microsoft or third-party vendors to provide generic services, such as numerical analysis or user interface elements. When you create applications, you can combine ActiveX components with generic components to create robust client/server custom solutions.

This chapter is the first of four chapters that show you how to implement ActiveX clients and ActiveX component applications by using capabilities built into Visual Basic. This chapter looks at the client side; the next three chapters look at various implementations on the server side.

Objectives

By the end of this chapter, you will be able to:

® Explain the major constructs of COM.

® Explain the major constructs of Automation.

® Create a client application that can manipulate other applications through Automation, use multiple interfaces on an object, and receive notifications from a server.

For more information on creating and using code components, creating and using ActiveX controls, or creating and using ActiveX documents, see Chapter 7: Creating ActiveX Code Components, Chapter 8: Creating ActiveX Controls, and Chapter 10: Creating and Using ActiveX Documents, respectively.

To build effective client applications, you need to know about the server components used by the client and have an understanding of the underlying COM technology that connects the client to those components.

COM has two distinct aspects. First, the COM specification provides a definition (or model) for what an object is. Second, COM provides services for the creation of objects and the communications between client and server.

This section provides a high-level overview of COM. It explains why COM is a powerful addition to the developer's toolkit, and describes the default interfaces provided by COM objects, how objects communicate with clients, and how to use them when creating and using an instance of an object.

This section includes the following topics:

® Overview of COM

® Advantages of Using COM

® The COM Specification

® Using Interfaces

® The IUnknown Interface

® Client and Server Communication

This section describes how to use Visual Basic to implement a client application using Microsoft Excel.

It provides an overview of the objects exposed by Microsoft Excel, and how to:

® Use workbooks and charts, and call Microsoft Excel procedures.

® Use Automation to create an instance of Microsoft Excel, and then manipulate some of its object's methods and properties.

® Get and set property values to create and manipulate a chart object.

® Execute a Microsoft Excel procedure.

® Convert a procedure created with the Visual Basic Macro Recorder into a Microsoft Excel function.

To see a demonstration of how a client written in Visual Basic can control Microsoft Excel, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

This section includes the following topics:

® The Microsoft Excel Object Model

® Creating an Instance of Microsoft Excel

® Using Methods in Microsoft Excel

® Getting and Setting Values

® Using the Charts Collection

® Running a Microsoft Excel Procedure

Microsoft Excel exposes over a hundred Automation objects that you can use with Visual Basic. Each of these objects provides some aspect of Microsoft Excel, such as charting or drawing.

This illustration shows some of the objects in the Microsoft Excel object model.

{ewc mvimg, mvimage,!v06g025.bmp}

You can create the **Application**, **Chart**, and **Sheet** objects with the Visual Basic **New** keyword or with the **CreateObject** function. The other Microsoft Excel objects in the object model are dependent objects, which you can create by using the **Add** method.

You reference objects by navigating down the object model using the dot (**.**) operator. For example, the following code sets the value of cell A1 in the worksheet Sheet1 in workbook BOOK1.XLS:

```
xl.Workbooks("BOOK1.XLS").Worksheets("SHEET1") _
    .Range("A1").Value = 5
```

You can use the **Parent** property of an object to set a reference to an object that is one level higher in the object model, as shown in this code.

```
xlwb.Parent 'refers to the Application object
```

To set a reference to objects that you use repeatedly, you can create object variables.

To see a code sample that sets the object variable **xlsheet** to Sheet1 in Book1.xls, click this icon.
{ewc mvimg, mvimage,!code.bmp}

You can also set a reference to an object by name or by index. The following code sets a reference to the workbook MyBook.xls, part of a group, or a collection, of workbooks.

```
set xlwb = xl.Workbooks("MYBOOK.XLS")
```

The first item in a collection is index 1. The following code sets a reference to the first workbook in the **Workbooks** collection:

```
set xlwb = xl.Workbooks(1)
```

The first step in using Automation is to add a reference in your project to the type library of the component you want to use. For example, to add a reference to the type library for Microsoft Excel, you would use the following procedure.

To set a reference to the Microsoft Excel type library

1. On the **Project** menu, click **References**.
2. Select the **Microsoft Excel 5.0 Object Library**, and then click OK.

To create or set a reference to an instance of Microsoft Excel objects, you can use the Visual Basic **New** keyword, or the **CreateObject** or **GetObject** functions.

For more information about the differences between using the **New** keyword and the **CreateObject** function, see Creating Objects.

Using the New Keyword

The **New** keyword starts a new instance of Microsoft Excel. When you install Microsoft Excel, it registers **Excel.Application**, **Excel.Worksheet**, and **Excel.Chart** as classes, as shown in the following code.

```
Dim xl as Excel.Application
Dim xlchart as Excel.Chart
Dim xlsheet as Excedl.Worksheet

set xl = New Excel.Application
set xlchart = New Excel.Chart
set xlsheet = New Excel.Worksheet
```

Using GetObject

If you want to use an instance of Microsoft Excel that is already running, use the **GetObject** function.

The **GetObject** function accepts two arguments. The first is the full path and name of the file containing the object to retrieve. The second argument is a string representing the class of the object to be created. If the first argument is omitted, the second argument is required.

If the first argument is omitted, the **GetObject** connects to a running instance of Microsoft Excel. If Microsoft Excel is not active, **GetObject** returns a run-time error.

With a file name as the first argument, **GetObject** returns a reference to the first sheet in the workbook

---

Note When an instance of a Microsoft Excel object starts, it is not visible. To see it, you must set the **Visible** property of the **Application** object to **True**. This is the case for most Automation objects.

Scoping Rules

If you have set an object reference to a Microsoft Excel sheet or chart, and the object variable goes out of scope, the sheet or chart will close automatically.

If you have set a reference to the **Application** object, and the object variable goes out of scope, Microsoft Excel will not close automatically. To close Microsoft Excel, use the **Quit** method of the **Application** object.

The objects available from Microsoft Excel provide a number of useful methods and properties. This topic describes several of the methods available from the Microsoft Excel **Workbook** object and its collection.

To create new workbooks, open existing workbooks, and close workbooks, you use the **Workbooks** collection.

The following code shows how to add a new workbook to the **Workbooks** collection.

```
xl.Workbooks.Add ()
```

This code opens a specific workbook.

```
xl.Workbooks.Open "c:\book1.xls "
```

This code adds code that does minimal error handling when opening a workbook.

```
Dim xl as Excel.Application
Dim xlwb as Excel.WorkBook
On Error Resume Next
set xl = CreateObject("Excel.Application")
set xlwb = xl.Workbooks.Open ("c:\book1.xls")
If Err <> 0 Then
    Msgbox "Unable to open workbook."
    Unload Me
End If
```

This code shows how to use a **For...Each** statement to iterate through the **Workbooks** collection to determine whether or not a specific workbook is open.

```
For each xlwb in xl.WorkBooks
    if xlwb.Name = "MYBOOK.XLS" Then
        bookfound = True
        Exit For
    End If
Next xlwb
```

To close a workbook, you use the **Close** method of the **Workbook** object. If you do not use the **SaveChanges** argument to determine whether or not the workbook has been saved, the user will be prompted to save any changes to the workbook before it is closed.

The following code closes the active workbook.

```
xl.ActiveWorkbook.Close SaveChanges := False
```

This code closes the workbook Book1.xls.

```
xl.Workbooks("BOOK1.XLS").Close SaveChanges :=False
```

To set values for the cells of a worksheet, use the **Range** method of a **Worksheet** object. To specify a range in a worksheet, you can set a reference to a cell such as A1, or you can use a named range.

The following code sets a reference for the object variable Sheet1. It then sets the value of cell A1 and the value of the named range, myrange.

```
set xlsheet = _
  xlapp.Workbooks("Book1").Worksheets("Sheet1")
xlsheet.Range("A1").Value = 5
xlsheet.Range("myrange").Value = 10
```

The **Range** method returns a **Range** object. A **Range** object is a group of cells on a worksheet. This code shows various ways to set the value of a range.

```
set xl = CreateObject("Excel.Application")
' Explicitly state the exact hierarchy.
xl.Workbooks("x.xls").Worksheets("s").Range("A1").Value = 5

' Assume the current active workbook and worksheet.
xl.Range("A2").Value = 5

' Establish object variable to reference objects.
Set wb = xl.Workbooks("source.xls")
Set ws = wb.Worksheets("sheet1")
Set rg = ws.Range("A1")
rg.Value = 5
```

Using Arrays

To set values for a group of cells, you can use an array. This limits the number of calls to Microsoft Excel, and is more efficient that setting one value at a time.

The following code fills an array and sends the array to Microsoft Excel with one call.

```
Dim x(1 to 4, 1 to 2) As Integer
x(1,1) = 5
x(1,2) = 5
x(1,3) = 6
....'Add other lines to fill array.
xl.Range("A1:B4").Value = x 'Pass entire array.
```

You can retrieve a range of cells into a variant variable, and fill the variant with an array.

```
'Retrieve a range of values.
v = xl.Range("A1:B4").Value
MsgBox v(1,1)    'Variant holds array.
```

You can also loop through a range of cells by using a **For...Each** statement.

```
Rim r as Range
For each r in xl.Range("summary")
    r.value = 25
Next r
```

You can access the charting capabilities of Microsoft Excel by using Automation. To create, modify, or open existing charts, use the Microsoft Excel **Charts** collection.

The following code selects a series of cells on a Microsoft Excel worksheet and creates a chart based on the selection.

```
xlsheet.Range("A1:D4").Select
set xlc = xl.Charts.Add()
xlc.Type = xl3DColumn
```

This code sets a reference to an existing chart, and then changes the type of chart.

```
set xlc =xl.workbooks("bk1.xls").Charts("mycht")
xlc.Type = xl3DColumn
```

This code opens Microsoft Excel, sets values in a worksheet, and creates a chart based on those values.

```
Sub cmdCreateChart_Click()
    Dim xl as Excel.Application
    Dim xlc as Excel.Chart
    Set xl = CreateObject("Excel.Application")
    xl.Visible = True
    xl.Workbooks.Add
    xl.Range("A1").Value = 3
    xl.Range("A2").Value = 2
    xl.Range("A1:A2").Select
    Set xlc = xl.Charts.Add()
    xlc.Type = xl3DColumn
End Sub
```

If you have created procedures in Microsoft Excel, you can use Automation to run those procedures from Visual Basic.

To run a Microsoft Excel procedure from Visual Basic, you use the **Run** method of the **Application** object.

For example, assume that you have created a procedure in Microsoft Excel, as shown in the following code.

```
Sub ChangeNumbers ()
    For Each x In Worksheets("Sheet1").Range("A1:D10")
        x.Value = Rnd() * 100
    Next x
End Sub
```

You can run this procedure from Visual Basic, as shown here.

```
Private Sub Command1_Click()
    set xl = CreateObject ("Excel.Application")
    xl.Workbooks.Open ("c:\book1.xls")
    xl.Run "ChangeNumbers"
End Sub
```

The following code shows a Microsoft Excel procedure that returns a value.

```
Function MakeProper (MyString As String) As String
    MakeProper = Application.Proper (MyString)
End Function
```

You can run this procedure from Visual Basic by using the following code.

```
Private Sub Command1_Click()
    s = xl.Run ("MakeProper", "mY StRiNG")
    's would now = "My String"
End Sub
```

Performance Considerations

If your Visual Basic application sends multiple statements to Microsoft Excel, you can write a Microsoft Excel procedure that uses Visual Basic to run the procedure.

Using the Microsoft Excel Macro Recorder

An easy way to start writing Automation code in Visual Basic is to start with the Microsoft Excel Macro Recorder. You can then import the code you recorded from Microsoft Excel into a Visual Basic application.

If you use this approach, you will need to modify the code that you import. For example, in Visual Basic, you will need to create an instance of Microsoft Excel by adding a statement that uses the **New** keyword. You will also need to preface each Visual Basic statement with the Microsoft Excel object variable that refers to Microsoft Excel.

The following code shows an example of code generated by the Microsoft Excel Macro Recorder.

```
Sub Macro1
    ActiveWorkbook.SaveAs Filename:="TERMPAPR.XLS", _
        FileFormat:=xlNormal,_
        Password:="", WriteResPassword:="", _
        ReadOnlyRecommended:=False , CreateBackup:=False
End Sub
```

This code shows the modifications made for Visual Basic.

```
Dim xl As Excel.Application
```

```
Set xl = New Excel.Application
xl.Workbooks.Add
xl.ActiveWorkbook.SaveAs Filename:="TERMPAPR.XLS", _
    FileFormat:=xlNormal,_
    Password:="", WriteResPassword:="", _
    ReadOnlyRecommended:=False , CreateBackup:=False
```

1. Why is communication between a client and an in-process server more efficient than communication between a client and an out-of-process server?

{ewc mvimg, mvimage,!answer.bmp}
A. The server and client run in separate address spaces.

{ewc mvimg, mvimage,!answer.bmp}
B. Proxy code is used to facilitate communication between client and server.

{ewc mvimg, mvimage,!answer.bmp}
C. The server and client run in the same address space.

{ewc mvimg, mvimage,!answer.bmp}
D. Stub code is used to make communication more efficient.

2. Which COM interface is supported by every object?

{ewc mvimg, mvimage,!answer.bm
A. **IInvoke**

p}

{ewc mvimg, mvimage,!answer.bmp}

B. **IRelease**

{ewc mvimg, mvimage,!answer.bmp}

C. **IDispatch**

{ewc mvimg, mvimage,!answer.bmp}

D. **IUnknown**

3. When are a component's GUIDs created?

{ewc mvimg, mvimage,!answer.bmp}

A. When the component is created.

{ewc mvimg, mvimage,!answer.bmp}

B. When the component is compiled.

{ewc mvimg.mvimage.!answer.bmp}

C. When a client application references the component.

{ewc mvimg.mvimage.!answer.bmp}

D. When the component registers itself on a computer.

4. What is added to a COM-compliant server during compilation to enable communication with local and remote clients?

{ewc mvimg.mvimage.!answer.bmp}

A. Stub code

{ewc mvimg.mvimage.!answer.bmp}

B. A GUID for each interface

{ewc mvimg.mvimage.!answer.bmp}

C. Proxy code

{ew

D. The **AddRef** function

{\cmvimg,mvimage,!answer.bmp}

5. Which item is a feature of an in-process component?

{\cmvimg,mvimage,!answer.bmp} A. One instance serves multiple clients.

{\cmvimg,mvimage,!answer.bmp} B. Global data can be shared.

{\cmvimg,mvimage,!answer.bmp} C. The component and client run in the same address space.

{\cmvimg,mvimage,!answer.bmp} D. The client is insulated from problems with the server.

6. Which interface allows you to implement Automation by changing the properties and methods

supported by the object without modifying the interface?

{ewc mvimg, mvimage,!answer.bmp}

A. **IDispatch**

{ewc mvimg, mvimage,!answer.bmp}

B. **IUnknown**

{ewc mvimg, mvimage,!answer.bmp}

C. **IInvoke**

{ewc mvimg, mvimage,!answer.bmp}

D. **IAddref**

7. What must you do before you can use the Object Browser to learn about the objects, interfaces, methods and properties exposed by a component?

{ewc mvimg, mvimage,!answer.bmp}

A. Locate the component using the Object Browser.

{ewc mvimg.mvimage.!answer.bmp}

B. Set a reference to the component's type library.

{ewc mvimg.mvimage.!answer.bmp}

C. Declare object variables as data type **Object**.

{ewc mvimg.mvimage.!answer.bmp}

D. Declare object variables as data type **Variant**.

8. Which type of binding results in the fastest execution?

{ewc mvimg.mvimage.!answer.bmp}

A. Late binding

{ewc mvimg.mvimage.!answer.bmp}

B. dispID binding

{ewc mvimg.mvimage.!answer.bmp}

C. vtable binding

{ewc

{mvimg,mvimage,!answer.bmp}

{ewc mvimg,mvimage,!answer.bmp}

D. Early binding

9. What must you do to make sure the Visual Basic compiler can validate the syntax of a call made to an object at design time?

{ewc mvimg,mvimage,!answer.bmp}

A. Declare the object variable as data type **Object**.

{ewc mvimg,mvimage,!answer.bmp}

B. Declare the object variable as data type **Variant**.

{ewc mvimg,mvimage,!answer.bmp}

C. Set the server's **Instancing** property to SingleUse.

{ewc mvimg,mvimage,!answer.bmp}

D. Declare the object variable as an explicit object type.

{ewc mvi

mg.
mvi
ma
ge.!
ans
wer
.bm
p}

10. You have set the variable xlwb to refer to the third workbook in the Workbooks collection with the following code:

```
set xlwb = xl.Workbooks(3)
```

To which object does xlwb.Parent refer?

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

A. The object containing the data used to populate the third workbook in the **Workbooks** collection.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

B. The second workbook in the **Workbooks** collection.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

C. The first worksheet in the third workbook in the **Workbooks** collection.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

D. The **Application** object.

11. What is the component's usage count after the following code runs?

```
Dim check as Speller.Check
Dim mng as IManage

Set check = New Speller.Check
Set mng = check
```

{ewc mvimg, mvimage,!answer.bmp} A. 0

{ewc mvimg, mvimage,!answer.bmp} B. 1

{ewc mvimg, mvimage,!answer.bmp} C. 2

{ewc mvimg, mvimage,!answer.bmp} D. 3

12. Which term best describes the behavior of a server that generates events, as opposed to a server that implements callback functionality?

{ewc mvimg, A. Handshake

{mvimage,!answer.bmp}

{ewc mvimg, mvimage,!answer.bmp} B. Targeted

{ewc mvimg, mvimage,!answer.bmp} C. Broadcast

13. What is the result of the following code if Microsoft Excel is not active?

```
set xl = GetObject (, "Excel.Application")
```

{ewc mvimg, mvimage,!answer.bmp} A. A run-time error will occur.

{ewc mvimg, mvimage,!answer.bmp} B. A compile-time error will occur.

{ewc mvimg, C. A new instance of Microsoft Excel will be started with the Hide **property** set to **True**.

{mvimage,!answer.bmp}

{ewc mvimg, mvimage,!answer.bmp}

D. A new instance of Microsoft Excel will be started with the **Visible** property set to **False**.

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

With Visual Basic version 5.0, you can create ActiveX code components (formerly called OLE servers). An ActiveX code component is a server application that exposes functionality which can be used and reused by other applications through Automation.

Code components provide powerful functionality. For example, you can create a code component that implements business rules, or you can package a complex routine into a simple component that other programmers can use.

This chapter provides an overview of ActiveX code components and explains how to use Visual Basic to create them.

Objectives

By the end of this chapter, you will be able to:

® Use a class module to create an object within a Visual Basic project.

® Create an ActiveX code component that exposes properties, events, and methods.

® Create a client application that uses your code component.

® Debug and test your code component.

® Raise events from your code component.

® Call your code component asynchronously by using events.

Advantages of using ActiveX code components include:

® Clearly defined interfaces

Functionality implemented in an ActiveX code component is accessed through properties, methods, and events. These interfaces are self-documenting in that they are automatically exposed by a code component. You can also add descriptive information and context-sensitive Help for each interface that will be stored with the code component. Users can browse through the descriptive information for each interface by using the Object Browser or other tools.

® Reduced complexity

You can create an ActiveX code component to hide programming complexity from other programmers. Other programmers need only know what information to provide to your object, and what information to retrieve.

® Easier updating

You can create an ActiveX code component that encapsulates business rules. If the business rules change, you update the just the code component, and not all the applications that use the code component.

® Information sharing

You can create an ActiveX code component that displays forms. Other applications can then call a method of the component to display a form and retrieve information.

Creating ActiveX code components involves working with class modules to define the objects that your component provides. Classes encapsulate the internal functionality of a component, and clients create objects from instances of classes at run time.

This section discusses how to use class modules to define and create an instance of a class. It also explains how to work with properties, methods, and events for objects that are created when you instantiate a class.

Before creating an ActiveX code component, you'll learn how to create and use class modules within a single Visual Basic project. Then, you'll learn how to turn these classes into ActiveX code.

This section includes the following topics:

® What Is a Class Module?

® Creating an Instance of a Class

® Class Module Events

® Creating Methods

® Creating Properties

® Using Property Procedures to Create Properties

® Using the Class Builder

When creating ActiveX code components, the first step is to define objects within your Visual Basic project. These objects can then be used within your Visual Basic project or exposed to other applications. To define an object for your component, you use a class module.

Purpose of a Class Module

A class module is a type of code module that Visual Basic provides. Each class module acts as a blueprint for an object. In other words, each class module defines one type of object. You may have several class modules in an application. At run time, you create an object by creating an instance of a class.

You will find class modules useful when creating any kind of ActiveX code component. Class modules represent a logical object in your component. For example, you might create an Employee class that has properties such as **Employee.LastName** and **Employee.FirstName**, and methods such as **Employee.Hire**. Your application can work with multiple instances of an **Employee** object, and can create a collection of **Employee** objects.

Adding a Class Module to a Project

To insert a new class module in a project, click **Add Class Module** on the **Project** menu. You can add methods, properties, and events to classes manually, or you can use the Class Builder in Visual Basic. The Class Builder add-in automates the processes for adding properties, methods, and events to classes.

For more information about the Visual Basic Class Builder add-in, see Using the Class Builder.

Class modules differ from standard modules these two ways.

® Class modules must be explicitly created before they can be used.

® You can create multiple instances of a class module.

In the project that defines a class, you can create an instance of the class with the **Dim** and **Set** statements, as shown in the following code.

```
Dim objMyObject1 As Class1
Dim objMyObject2 As Class1

Set objMyObject1 = New Class1
Set obMyObject2 = New Class1
```

To see an animation of how to create an instance of a class, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

Note A class is a template for an object. An object is an instance of a class. In the example, **objMyObject1** and **objMyObject2** are objects of type Class1. You may want to use the more compact syntax **Dim objMyObject1 as New Class1**, but to clearly show where the instance of the class is instantiated, you should use the **Set** statement instead.

For information about using the **Set** statement, see Creating Objects in Chapter 6: Creating ActiveX Clients.

For information about the differences between class modules and standard modules, search for **Class Modules vs. Standard Modules** in Visual Basic Books Online, and then look in **Creating Your Own Classes**.

Class modules come with two built-in events: Initialize and Terminate.

To add code to the class module events, open a code window for the class, and click **Class** in the **Object** drop-down list box.

Initialize Event

The Initialize event occurs when an instance of a class is created, but before any properties have been set. You use the Initialize event to initialize any data used by the class, as shown in the following code.

```
Private Sub Class_Initialize()
    'Initialize data.
    iDept = 5
End Sub
```

You can also use the Initialize event to load forms used by the class.

Terminate Event

The Terminate event occurs when the object variable goes out of scope or is set to **Nothing**. Use the Terminate event to save information, unload forms, or perform tasks that should occur when the class terminates.

```
Private Sub Class_Terminate()
    'Any termination code.
End Sub
```

For information about coding Initialize and Terminate events, search on **Coding Robust Initialize and Terminate Events** in Visual Basic Books Online, and then click **Adding Classes to Components**.

To create a method for an object, you create **Public Sub** or **Function** procedures within a class module. When you create an instance of the class module, the **Public Sub** and **Function** procedures are available as methods of the resulting object.

The following code creates a method that displays the date.

```
Public Sub ShowDate()
    MsgBox "Date is: " & Now()
End Sub
```

This code creates a method that accepts a number and returns the number squared.

```
Public Function SquareIt (Num As Integer) _
    As Integer
    SquareIt = Num * Num
End Function
```

This code creates an instance of the class and calls the **ShowDate** and **SquareIt** methods.

```
Dim Demo1 As Class1
Set Demo1 = New Class1
Demo1.ShowDate
i = Demo1.SquareIt (Num:=5)
```

You can create methods manually, or you can use the Visual Basic Class Builder add-in. You can also add methods by opening a code window, and then clicking **Add Procedure** on the **Tools** menu.

You can use the Object Browser to view the properties and methods you have defined for a class.

For information about using the Object Browser, see Automation Objects in Chapter 6: Creating ActiveX Clients.

There are two ways to define a property for your object.

® Define public variables.

–or–

® Create public property procedures within your class module.

Public variables provide a variable to hold a property value. Property procedures enable you to run code when a property is set or retrieved.

This topic describes how to define a property by using public variables, and explains how to create a default property. The next topic in this section explains how to define a property by creating a property procedure.

Using Public Variables to Create Properties

The following code defines the string property **User**.

```
Public User As String
```

You define public variables in the General Declarations section of a class.

This code creates an instance of Class1, and sets the **User** property.

```
Dim Demo1 As Class1
Set Demo1 = New Class1
Demo1.User = "Joe"
```

Creating a Default Property

You can set the value of a default property for an object without explicitly referring to the object itself. For example, the **Text** property is the default property for a **TextBox** control in Visual Basic, an example of which is shown in the following code.

```
Text1.Text = "hello"
Text = "hello"
```

u To create a default property

1. Open the class module containing the property.
2. On the **Tools** menu, click **Procedure Attributes**.
3. In the **Procedure Attributes** dialog box, click the **Advanced** button.
4. In the **Name** drop-down list box, select the property you want to set as the default.
5. In the **Procedure ID** drop-down list box, select **(Default)**.

The following illustration shows a default property set in the **Procedure Attributes** dialog box.

{ewc mvimg, mvimage,!v07g010.bmp}

In this section, you will learn how to expose class modules to make them available to client applications such as ActiveX EXEs and ActiveX DLLs. When you make class modules available to external client applications, you create ActiveX code components.

This section includes the following topics:

® Choosing the Type of Code Component

® Setting Project Properties

® Setting Class Module Properties

® Compiling a Component

® Registering a Component

When you create a new ActiveX DLL or ActiveX EXE project, you can set a number of other properties for your project. These properties affect how your code component will run.

You set properties for the project in the **Project Properties** dialog box, which you access from the **Project** menu. All of the options described in this topic are found on the **General** tab of the **Project Properties** dialog box.

Project Type

The **Project Type** field provides four options: Standard EXE, ActiveX EXE, ActiveX DLL, and ActiveX Control. When you create a new ActiveX DLL or ActiveX EXE project, Visual Basic automatically sets the **Project Type** property.

The project type determines how some of the other project options can be used. For example, options on the **Component** tab are not available when the project type is set to Standard EXE.

Startup Object

For most ActiveX EXEs and ActiveX DLLs, you set the **Startup Object** field to **(None)**. If you want code to run when the component is loaded, rather than when an instance of a class is created, you can set the **Startup Object** property to **Sub Main**. You must then add a standard module to your project, and add a **Sub** procedure named **Main** to that module.

Project Name

The **Project Name** field indicates the name that the client application uses to refer to the component.

To see a code sample that shows the code from a client application that creates an instance of a class named clsPassWord in a code component named Server2, click this icon.
{ewc mvimg, MVIMAGE,!code.bmp}

Project Description

The **Project Description** field enables you to enter a brief description of the ActiveX code component and the objects that it provides. The contents of this field appear in the **References** dialog box for any client application that is selected in the **Available References** list. This string also appears in the Description pane at the bottom of the Object Browser.

Unattended Execution

The **Unattended Execution** check box indicates that the component is intended to be run without user interaction. Unattended components do not have a user interface. Any run-time functions, such as messages that normally result in user interaction, are written to an event log.

After setting the project properties for an ActiveX code component, you can set properties for each class module in the component to determine how that class will act.

Name Property

The **Name** property is used by client applications when creating an instance of the class. For example, if a class is named MyClass and is in a project named Project1, a client could use the following code to create an instance of that class.

```
Dim x as Project1.MyClass
Set x = New Project1.MyClass
```

Instancing Property

The **Instancing** property determines if applications outside a project can create new instances of a class, and if so, how those instances are created.
{ewc mvimg, mvimage,!tip.bmp}

The available instancing options are different in ActiveX EXE and ActiveX DLL components, as shown in the following illustration.

{ewc mvimg, mvimage,!v07g020.bmp}

This table defines each of the **Instancing** property settings.

Setting	Description
Private	Other applications are not allowed access to type library information about the class, and cannot create instances of it. **Private** objects are used only within a component.
	Applies to ActiveX EXE, ActiveX DLL, ActiveX Control, and Standard EXE components.
PublicNotCreatable	Other applications can use objects of this class only if a component creates the objects first. Other applications cannot use the **CreateObject** function or the **New** operator to create objects of this class. You set the **Instancing** property to this value when you want to create dependent objects.
	Applies only to ActiveX EXE, ActiveX DLL, and ActiveX Control components.
SingleUse	Allows other applications to create objects from the class. Every object of this class that is created by a client will start a new instance of the component.
	Applies only to ActiveX EXE projects.
GlobalSingleUse	Similar to the SingleUse setting, except that properties and methods of the class can be invoked as though they were global functions.
	Applies only to ActiveX EXE components.
MultiUse	Allows other applications to create objects from the class. One instance of a component can provide any number of objects created in this way, regardless of how many applications request them.
	Applies only to ActiveX EXE and ActiveX DLL components.
GlobalMultiUse	Similar to the MultiUse setting, except that properties and methods of the class can be invoked as though they were global functions. Explicitly creating an instance of the class first is not necessary because one will automatically be created.

Applies only to ActiveX EXE and ActiveX DLL components.

Once you have built an ActiveX code component, you are ready to compile it. Your component is compiled as either an .exe or .dll file, depending on the settings in the **Project Properties** dialog box.

Type Library ID

When you compile an ActiveX EXE or ActiveX DLL component, Visual Basic creates a unique type library ID. This ID is entered into the registry when you register your code component.

The **Project Compatibility** field in the **Project Properties** dialog box is selected by default. This setting tells Visual Basic to reuse the same type library ID each time you compile a code component.

If you make a change that is incompatible with the version of your server specified in the **Project Compatibility** field (for example, if you delete a property), Visual Basic displays a warning and generates a new type library ID.

To set the **Project Compatibility** field, click **Project Properties** on the **Project** menu, and then click the **Component** tab. On the **Component** tab, click **Project Compatibility**.

For information about the types of compatibility available in Visual Basic, see Version Compatibility.

Before an ActiveX code component can be used, it must be registered.

An ActiveX code component is temporarily registered when you run it from Design mode. The component is permanently registered when you:

® Run it for the first time.

® Install it by using a Setup program.

® You run it with the **/Regserver** command-line argument. This ends the component immediately after registering it.

Visual Basic registers an ActiveX DLL file when you compile it. You can also use Regsvr32.exe to register the DLL file, as shown in the following code.

```
Regsvr32.exe mydll.dll
```

A client application can use the **References** dialog box to set a reference to your DLL file.

To remove the registry entry for an ActiveX EXE, run it with the **/UnRegserver** command-line argument, as shown in this code.

```
myServer.Exe /UnRegserver
```

To remove an ActiveX DLL entry from the registry, you can run Regsvr32.exe with the **/u** option and the name of the DLL file, as shown in this code.

```
Regsvr32.exe /u mydll.dll
```

Regsvr2.exe is located on the Visual Basic CD-ROM in the \Tools folder.

This section describes various ways to enhance and extend the functionality of code components.

For more information about implementing the features shown in this section, refer to Visual Basic Help.

This section includes the following topics:

® Using Named Constants
® Component Information and Help
® Global Data Scoping
® Declaring Friend Methods
® Creating an Object Model
® Storing Objects in a Collection
® Version Compatibility
® Handling a Busy Component

To make classes in your code components easier to use, you can add descriptive text and context-sensitive Help for each of the member functions in a class.

To provide this information, you fill in the fields in the **Procedure Attributes** dialog box, as shown in the following illustration.

{ewc mvimg, mvimage,!v07g030.bmp}

This information will then be available as context-sensitive Help in the Object Browser of any client application that uses the code component.

Note You cannot set descriptions for properties defined as **Public** variables, but you can set descriptions for properties defined by using property procedures.

---

u To set Help information for methods and properties

1. Open the project for your ActiveX code component.
2. Open the code window for the class module.
3. On the **Tools** menu, click **Procedure Attributes**.
4. Fill in the fields **Description** and **Help Context ID** for each procedure.

For information about the Object Browser and its relation to type libraries, see Automation Objects in Chapter 6: Creating ActiveX Clients.

In a Visual Basic application, data declared as **Public** in a standard module is global data, and is available to the entire application.

Global data is shared between client application depending on whether you compile the server as an executable file or a DLL, and on the value of the **Instancing** property of the class.

ActiveX Executable

When you create an ActiveX executable file and set the **Instancing** property of a class to **MultiUse**, the executable file can service multiple clients.

The first time you request an instance of that class, the server will be executed. The next time you request an instance of that class, the server will already be running, and will merely create a new class instance. Both class instances will access the same global data.

If you create an ActiveX executable file and set the **Instancing** property of a class to **SingleUse**, a new copy of the server will be executed each time you request a new instance of that class. Global data cannot be shared between instances of the executable file.

ActiveX DLL

If you create an ActiveX DLL, you must set the **Instancing** property to **MultiUse** (or **PublicNotCreatable**). The component runs in the same process space as the client application.

A separate instance of the DLL is used for each client application, and global data cannot be shared between client applications. However, if one client application creates several instances of a class, all instances will use the same instance of the DLL, and will share any global data.

This topic discusses how to raise and trap run-time errors by using the **Raise** method of the **Err** object. It also details how options that you set for Break mode can affect how errors are handled when raising errors to the client.

Raising an Error in the Client

When you create an ActiveX component, you can provide error messages to the client application. To pass an error back to a client application, you use the **Raise** method in a component. For the error to be raised in the client, you must call **Raise** from your method's error-handling routine or with error handling disabled.

The **Raise** method has the following syntax.

ERR.Raise (*Number*, *Source*, *Description*, *HelpFile*, *HelpContext*)

The error number is generated by adding the intrinsic constant **vbObjectError** to the error number. The resulting number is returned to the client application. This ensures that your error numbers do not conflict with the built-in Visual Basic error numbers.

To see a code sample that shows error handling from a component, click this icon.
{ewc mvimg, mvimage,!code.bmp}

Breaking on Errors

You can change the way Visual Basic enters Break mode when an error occurs in your code component. To do this, you set **Error Trapping** options on the **General** tab of the **Options** dialog box. It is important to be aware of these options, because they can cause your application to break on an error, even if you have created an error handler.

Visual Basic provides three options for setting breakpoints, as shown in the following table.

Option	Description
Break on All Errors	Any error causes the project to enter break mode, whether or not an error handler is active, and whether or not the code is in a class module.
Break in Class Module	Any unhandled error that has been produced in a class module will cause the project to enter Break mode at the line of code in the class module that produced the error. When you debug an ActiveX server project by running an ActiveX client test program from another project, set this option in the ActiveX server project to break on errors in its class modules, instead of returning the error to the test program.
Break on Unhandled Errors	If an error handler is active, the error is trapped without entering Break mode. If there is not any active error handler, the error causes the project to enter Break mode. An unhandled error in a class module will cause the project to enter Break mode on the line of code that invoked the offending procedure of the class.

An object model defines the relationship between objects in a code component. Some objects are dependent on other objects. These dependent objects exist only within the context of another object.

For example, in Microsoft Excel, a **Button** object cannot exist on its own. It exists only within the **Worksheet** object.

A client application cannot use the **CreateObject** function or the **Set** *object* = **New** class statement to create a **Button** object. The **Worksheet** object provides an **Add** method that creates a new **Button** object.

You can create an ActiveX code component that provides an object model. By defining the relationships between the objects that you use in your program, an object model organizes the objects in a way that makes programming easier.

To create an object model for an ActiveX component, you use the following steps.

1. Create a top-level object by setting the **Instancing** property of the class to **SingleUse** or **MultiUse**.
2. Create dependent objects by setting the class **Instancing** property to **PublicNotCreatable**.

You can also provide a method for the top-level object that creates a dependent object and returns a reference to the newly created object. For example, an Employees class could have an **Add** function that creates a new instance of a dependent class and returns an object variable that points to the newly created dependent object.

For information about creating object models, search on **Organizing Objects: The Object Model** in Visual Basic Books Online, and then click **General Principles of Component Design**.

Creating a collection of objects enables you to store related items together. In Visual Basic, the **Collection** object is a predefined object. It can be created in the same way as other objects, as shown in the following code.

```
Dim Employees As New Collection
```

The **Collection** data type has predefined **Add** and **Remove** methods and a **Count** property.

The following code creates a new **Employee** object and adds the object to the **Employees** collection.

```
Public Sub AddEmp (s As String, i As Integer)
    Dim Emp As New Employee
    Emp.FName = s
    Emp.ID = i
    Employees.Add Item:=Emp, Key:=Emp.ID
End Sub
```

Storing related objects in a collection enables you to work with those objects as a group. For example, you could use the **Employees** collection created in the previous code to print a list of all employees, as shown in this code.

```
Private Sub cmdPrintEmployees_Click()
    For Each x In Employees
        Print x.FName, x.ID
    Next x
End Sub
```

If you want a code component to perform a lengthy operation, you may want to implement the operation as an asynchronous method.

Typically, when a client runs a method in a component, the client must wait for the component to finish processing before it can continue processing. When an asynchronous method is used, the client can continue working while the component runs in the background.

To enable an asynchronous method

1. Create a method in a component that enables a timer on a form.

   This will return control immediately to the client application, because no processing is done in the procedure.
2. In the Timer event, call another method in the component that performs whatever operation is required by the component.
3. When the method used to perform the operation finishes processing, raise an event to the client to notify it that the method has completed.
4. In the client, add code to the return event.

   The client application invokes a server method to start a background process. The client then receives an event from the code component when the processing is finished.

To see a demonstration of an application that implements an asynchronous method, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

{ewc mvimg, mvimage,!tip.bmp}

1. Which project template would you select to build an in-process code component?

{ewc mvimg, mvimage,!answer.bmp}

A. ActiveX Control

{ewc mvimg, mvimage,!answer.bmp}

B. ActiveX DLL

{ewc mvimg, mvimage,!answer.bmp}

C. ActiveX EXE

{ewc mvimg, mvimage,!answer.bmp}

D. Standard EXE

2. Your project is a component that exists only when providing objects to other applications. In the Visual Basic development environment, how can you keep the code component running while you start a client application to test the server?

{ewc mvimg, mvimage,!answer

A. Select **Project Compatibility** from the **Version Compatibility** dialog box.

.bmp}

{ewc mvimg, mvimage,!answer.bmp}

B. Set the **StartMode** property of the **App** object to ActiveX component(1).

{ewc mvimg, mvimage,!answer.bmp}

C. Set the startup form to **Sub Main**.

{ewc mvimg, mvimage,!answer.bmp}

D. Set the **StartMode** property of the **App** object to Standalone(0).

3. Which action results in the temporary registration of a code component?

{ewc mvimg, mvimage,!answer.bmp}

A. The code component is run for the first time.

{ewc mvimg, mvimage,!answer.bmp}

B. The code component is installed on a computer using Setup.

{ewc mvimg.mvimage,!answer.bmp}

C. The code component is run from design mode.

{ewc mvimg.mvimage,!answer.bmp}

D. The code component is run with the /REGSERVER command-line argument.

4. How can you create a read-only property for a class?

{ewc mvimg.mvimage,!answer.bmp}

A. Define a **Property Get** procedure and define a **Property Let** procedure with no arguments.

{ewc mvimg.mvimage,!answer.bmp}

B. Define a **Property Get** procedure and a **Property Let** procedure.

{ewc mvimg.mvimage,!answer.bmp}

C. Define a **Property Get** procedure without a **Property Set** or **Property Let** procedure.

{ewc mvimg.mvimage,!answer.bmp}

D. Define a **Property Set** or **Property Let** procedure without a **Property Get** procedure.

{ewc

{mvimg.mvimage.!answer.bmp}

5. How can you create a property that is an Object data type?

{ewc mvimg.mvimage.!answer.bmp} A. Define a **Property Get** procedure and a **Property Let** procedure.

{ewc mvimg.mvimage.!answer.bmp} B. Define a **Property Set** procedure and a **Property Let** procedure.

{ewc mvimg.mvimage.!answer.bmp} C. Define a **Property Get** procedure without a matching **Property Set** or **Property Let** procedure.

{ewc mvimg.mvimage.!answer.bmp} D. Define a **Property Get** procedure and a **Property Set** procedure.

6. The following lines of code have the same behavior:

```
MyIndex.CardPage = PageRef
```

```
CardPage = PageRef
```

What do you know about the CardPage property?

{ewc mvimg, mvimage,!answer.bmp}
A. The **CardPage** property was created using a **Public** variable.

{ewc mvimg, mvimage,!answer.bmp}
B. The **CardPage** property is the default property of the **MyIndex** object.

{ewc mvimg, mvimage,!answer.bmp}
C. The **CardPage** property was created using a property procedure.

{ewc mvimg, mvimage,!answer.bmp}
D. Code can be executed when the **CardPage** property is set or retrieved.

7. The code sample below shows parameters passed to a client by value and by reference:

```
Event InProgress(ByVal pd as Long, _
        ByRef cancel as Boolean)
```

What is the result if the client changes these parameters?

{ewc mvimg,
A. The change in the parameter passed by value will be seen by the server.

{mvimage,!answer.bmp}

{ewc mvimg,mvimage,!answer.bmp}

B. The change in the parameter passed by value will not be seen by the client.

{ewc mvimg,mvimage,!answer.bmp}

C. The change in the parameter passed by reference will not be seen by the server.

{ewc mvimg,mvimage,!answer.bmp}

D. The change in the parameter passed by reference will be seen by the server.

8. You are writing code to allow a client to cancel an asynchronous method, and the component code shown below is flawed:

```
Dim bCancel as Boolean
Event InProgress(ByVal pd as Long, _
            ByVal cancel as Boolean)
Private Sub ...(...)
...
    bCancel = False
    RaiseEvent InProgress(percent, bCancel)
    if bCancel then
        ' cancel the task
    End If
    ...
End Sub
```

How should you change this code?

{ew

A. The pd argument should be passed by reference.

{ewc mvimg, mvimage,!answer.bmp}

B. The cancel argument should be passed by reference.

{ewc mvimg, mvimage,!answer.bmp}

C. bCancel should be set to **True**.

{ewc mvimg, mvimage,!answer.bmp}

{ewc mvimg, mvimage,!answer.bmp}

D. The pd argument should be passed as an integer.

{ewc mvimg, mvimage,!answer.bmp}

```
'Client Application that supports type libraries
Dim x As Server2.clsPassWord
Set x = New Server2.clsPassWord

'Client Application that does not support type libraries
Dim x as Object
Set x = CreateObject ("Server2.clsPassWord")
```

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

When you distribute an application, you must distribute your application's executable file, as well as any supporting files that your application requires, such as DLLs, custom controls, and Automation servers. To do this, you should provide a Setup program for your users.

This chapter describes how to create a Setup program. It also explains how to optimize the usability and performance of an application, and how to use the tools provided with Visual Basic to develop applications more efficiently.

Objectives

By the end of this chapter, you will be able to:

- Create a Setup program, either manually or by using the Setup Wizard.
- Describe a variety of techniques for optimizing the performance of an application.
- Use Microsoft Visual SourceSafe.
- Use resource files.
- Use the **GetSetting** and **SaveSetting** statements to save application-specific information to an .ini file or to the registry.
- Use the **App** object to log error, warning, and information events.

When you distribute your application, you should provide users with a Setup program to perform the following tasks.

® Copy the necessary files to the user's system.

® Place the files in the appropriate folders.

® Register files.

® Create a Program Manager group or a **Start** menu item or group.

You can use the Setup Wizard provided by Visual Basic to create a Setup program. Visual Basic also includes a Setup Toolkit that you can use in conjunction with the Setup Wizard to create a custom Setup program.

You can also create your own Setup program, determine which files are needed, compress the files, and manually copy the files to a Setup disk or folder.

This section describes how to create a Setup program by using the Setup Wizard and the Setup Toolkit in Visual Basic. It also explains how to use the application removal utility that is included with Setup programs created by the Setup Wizard.

This section includes the following topics:

® Using the Setup Wizard

® Setup Files

® Using the Setup Toolkit

® Removing an Application

For information about creating a Setup program for an ActiveX control, see Packaging a Control in Chapter 9: Using ActiveX Components on a Web Page.

The easiest way to create a Setup program is to use the Visual Basic Setup Wizard. The Setup Wizard prompts you for information about your application, and automatically determines which files need to be distributed, compresses the files, copies them to disk, and creates the Setup program.

The Setup Wizard supports the following options for distributing your application.

® Multiple floppy disks. The Setup Wizard can split files that are too large to fit onto a single floppy disk.

® Copying your files to a hard disk directory for distribution over a network or onto a CD-ROM.

® Distributing your application across the Internet by using automatic code download from Microsoft Internet Explorer version 3.0.

You can start the Setup Wizard by clicking either **Application Setup Wizard** on your **Start** menu, or selecting the file Setupwiz.exe from the folder \Setupkit\kitfil32 where you installed Visual Basic.

To determine which files to include in your Setup program, the Setup Wizard checks the references and custom controls that you have loaded in your project. The Setup Wizard then reads the file VB5Dep.ini in the Windows folder to determine which files are required for each reference or custom control.

For example, if you have set a reference to the **Microsoft Common Dialog** control, the Setup Wizard includes the appropriate .ocx file in your Setup program.

Note When you create a new project, Visual Basic adds several controls to the Toolbox and a reference to data access objects (DAO). The Setup Wizard recognizes controls and references, whether you use them in your application or not.

If controls that you have not used in your application appear in the **Confirm Dependencies** step, or if the **Data Access Object** step is displayed, and you know you did not add any references to data access objects, you should delete the unused files. Either remove the tools and references from your source files manually, or clear the check boxes in the list box.

The Setup Wizard enables you to add or remove files from the list of files that need to be distributed.

To see a demonstration of how to use the Setup Wizard to create a Setup program, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

The Setup disks or hard disk folder created by the Setup Wizard contain all of the files necessary to install your application on a user's system.

Here are some of the necessary files.

® Setup.exe

The Setup Wizard copies Setup.exe from a Visual Basic folder to your Setup disks or hard disk folder. The user runs Setup.exe to install your application.

Setup.exe copies the bootstrap files, and then executes the main Setup program (usually named Setup1.exe) listed in Setup.lst.

® Setup.lst

The Setup Wizard creates the Setup.lst file. This file contains the list of files required by your application, and contains general information, such as default folders and the required disk space.

A sample Setup.lst file might look like the following code.

```
[Bootstrap]
File1=1,,setup1.ex_,setup1.exe,$(WinPath),...
..
[Files]
File1=1,,GRID32.OC_,GRID32.OCX,$(WinSysPath),,$(Shared)..
..
[Setup]
Title=LoanSheet
DefaultDir=$(ProgramFiles)\Loan
```

® Setup1.exe

The Setup Wizard copies the main Setup program, Setup1.exe, from a Visual Basic folder to your Setup disks or hard disk folder. Setup1.exe copies and registers files, and creates startup icons. The Setup program also increments the reference count for shared files in the registry.

The setup program creates an application log file (St5Unst.log) that is copied to the application folder, and includes the following information.

- Folders that were created.
- Files that were installed. The log indicates if a file was not copied because a newer version of the file was already found on the disk.
- Registry entries that were created.
- Program Manager groups or **Start** menu entries that were created.
- DLLs, EXEs, or OCXs that were self-registered.

The Setup program also copies an application removal utility (St5Unst.exe) to the Windows folder. For information about this utility, see Removing an Application.

With the Visual Basic Setup Toolkit, you can create a custom Setup program to enable features that are not provided automatically by the Setup Wizard. For example, you can use the Setup Toolkit to create a Setup program that displays custom Setup forms and dialog boxes, or enables a user to install optional features of your application.

The Setup Toolkit consists of utilities, DLLs, and a sample Setup program written in Visual Basic.

You can modify the sample Setup program, Setup1.vbp, in the folder \Vb\SetupKit\Setup1, and create an executable file named Setup1.exe. You can then use the Setup Wizard to create the actual Setup disks or Setup folder. The Setup Wizard copies Setup1.exe from the Visual Basic folder Setupkit\Setup1 to your Setup folder.

Note Be sure to save the original version of Setup1.vbp.

For information about modifying Setup1.vbp, search for **Using the Setup Wizard with the Setup Toolkit** in Visual Basic Books Online.

When a user runs a Setup program created by the Setup Wizard, the program installs an application removal utility (St5Unst.exe) to the Windows folder of the user's system.

To remove an application with either Microsoft Windows NT or Windows 95, click the **Add/Remove Programs** icon in the Control Panel.

The application removal utility removes the application files and icons or groups, and it decrements the reference count for shared components. If a reference count is zero, the utility prompts the user to remove the shared component.

Note The application removal utility depends on an accurate log file and an accurate registry entry to perform its function.

If your application uses a shared component that you know is already on the user’s system, you should still include the component with your Setup program. This ensures that the reference count for the component is incremented when the user installs your application.

Microsoft Visual SourceSafe is a project-oriented version control system for team development of software applications. Like a library, Visual SourceSafe enables users to check files in and out from a general repository.

This system tracks and stores changes to files so that developers can review a file's history and revert to earlier versions of a file as they develop an application.

Installing Visual SourceSafe

To use Visual SourceSafe with Visual Basic, you must first install the support files and source code control database on a server that is accessible to all users. The Visual Basic Setup program provides an option to install the SourceSafe server.

After the server setup has been completed and Visual Basic has been installed, you must install the SourceSafe client software on each system that will use Visual SourceSafe.

To install the client software, click **SccAddin.SourceCodeControlAddin** on the **Add-Ins** menu. The SourceSafe add-in adds the **SourceSafe** menu to the Visual Basic **Add-Ins** menu, and enables the **Get**, **Check Out** and **Check In** commands on the **Tools** menu in the Visual Basic environment.

Adding a Project to Visual SourceSafe

To add a project to Visual SourceSafe, click **Add Project to SourceSafe** on the **SourceSafe** menu. The files are then copied to the SourceSafe project.

Modifying a File

To modify a file that has been added to a project, you must first check out the file. To check out a file, click **Check Out** on the Visual Basic **Tools** menu. SourceSafe copies the latest version of the file to your computer. Once you have finished modifying the file, click **Check In** to copy the file back to the server and remove the local version.

To optimize an application, you need to determine which code runs most often, and then optimize that code.

The Visual Basic Code Profiler is an add-in that you can use to determine which areas of your application to optimize. The Code Profiler tells you what code is being executed in an application, how many times it is executed, and how long it takes to execute.

The following illustration shows the Code Profiler dialog box.

{ewc mvimg, mvimage,!v12g005.bmp}

The Code Profiler is included in the Professional Edition of Visual Basic in the Tools folder. This folder contains various unsupported tools and accessories. The folder is not installed when you install Visual Basic. You can copy files from this folder, or you can work with the files on the Visual Basic CD-ROM.

To see a demonstration of how to use the features of the Code Profiler, click this icon.

{ewc mvimg, mvimage,!democlip.bmp}

For information about the utilities provided in the Tools folder, see **Contents of Your Compact Disc** in Visual Basic Help.

If you are planning to distribute your application to an international market, you can reduce the amount of time it takes to localize your application. Rather than using literal data in your code, you can store data such as strings and bitmaps in a resource file. Your application can use functions to load data at run time. You can easily change the data in the resource file without changing the actual code.
{ewc mvimg, mvimage,!tip.bmp}

For more information on using resource files, see "Using Resource Files for Localization" in Visual Basic Books Online.

Creating Resource Files

To create a resource file

1. Create a resource definition file (*.rc) that contains all the string resources of your application. The syntax for creating the resource definition file is documented in Resource.txt in the \Tools folder of the main Visual Basic folder.

   You associate an identifier (ID) with each resource, and then reference the ID in your code. (The resource ID 1 is reserved for the application icon. You cannot use the resource ID 1.)

   The easiest method of creating the resource definition file is to use a resource editor such as that included with Microsoft Visual C++.
2. Use the resource compiler to convert the resource definition file into a resource file (*.res). You can use the resource compiler (Rc.exe) that is included with the Visual Basic Professional and Enterprise editions.

To add a resource file to your project

1. On the **Project** menu, click **Add File**.
2. In the **Add File** dialog box, select **Resource Files (*.RES)** in the **Files of type** box.
3. Select the resource file you want to add to the project, and then click **Open**.

Using Resource Functions

Visual Basic provides several functions that enable you to load information from a resource file at run time, as shown in this table.

Function	Purpose
LoadResString	Loads a string.
LoadResPicture	Loads a picture. The second argument to **LoadResPicture** determines the type of data to load (0=bitmap, 1=icon, 2=cursor).
LoadResData	Loads general data, such as a .wav file.

You can make your applications easier to use by saving information about user activity or preferences each time a user runs your application. You can then use this information in subsequent sessions.

For example, you can save the name of the last database the user opened, and then use that name as the default database the next time your user opens a database.

In Windows 3.1 and earlier, program settings were commonly stored in .ini files. In Windows NT and Windows 95, program settings are stored in the registry.

To save application settings, you can use the Visual Basic statements **GetSetting** and **SaveSetting** or the Windows API routines.

Using GetSetting and SaveSetting

Use the **GetSetting** function to read a setting from the registry. The syntax for **GetSetting** is as follows.

GetSetting (*appname*, *section*, *key*,[*default*])

The following code obtains the customer ID information from the appropriate key in the Windows 95 registry.

```
Dim strCustID as Integer
Private Sub Form_Load()
    strCustID = GetSetting _
        ("OrderApp", "CustSection", "CustID", "0")
End Sub
```

Use the **SaveSetting** function to write an entry to the registry. The syntax for **SaveSetting** is as follows.

SaveSetting *appname*, *section*, *key*, *setting*

The following code saves the customer ID information to the appropriate key in the Windows 95 registry.

```
Dim strCustID
Private Sub_Form_Unload()
    strCustID = txtCustID.Text
    SaveSettings _
        "orderApp", "CustSection", "CustID", strCustID
End Sub
```

Using the Win32 API

Windows provides several procedures you can use to read and write to .ini files.

In Windows 95 and Windows NT, you should always store application-specific information in the registry by using Visual Basic statements. However, there may be times when you need to use Windows functions.

For example, you might want to retrieve information from the Win.ini file.

This table lists Windows procedures that you can use to save application settings.

Procedure	Purpose
GetPrivateProfileString	Reads a string entry from a private .ini file.
GetPrivateProfileInt	Reads a numeric entry from a private .ini file.
WritePrivateProfileString	Writes an entry to a private .ini file.
GetProfileString	Reads a string entry from Win.ini.
GetProfileInt	Reads a numeric entry from Win.ini.

For information about working with DLLs, see Chapter 5: Using Dynamic-Link Libraries.

1. **You have created a Setup program for your Windows 95 application using the SetupWizard. Which file creates the startup icons?**

{ewc mvimg, mvimage,!answer.bmp}
A. SETUP.EXE

{ewc mvimg, mvimage,!answer.bmp}
B. SETUP1.EXE

{ewc mvimg, mvimage,!answer.bmp}
C. SETUP132.EXE

{ewc mvimg, mvimage,!answer.bmp}
D. SETUP.LST

2. **Your application uses a complicated graphic to display order status statistics for the production department of your company, and the display is updated whenever an order is entered, constructed, packaged, or shipped. What can you do to improve the performance of your application's display operations?**

{ewc mvimg, mvimage,!ans
A. Use a picture box instead of an image control.

wer
.bm
p}

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

B. Set **AutoRedraw** to **True** so your status graphic is redrawn automatically.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

C. Make sure you unload rather than hide the status graphic when it is not being used so it will display more quickly when it is needed again.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

D. Set **AutoRedraw** to **False** and add code to refresh your status graphic.

3. You application retrieves data from text files in the form of many long strings. How can you reclaim the memory used by these strings when the data is no longer needed by your application?

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

A. Set the strings to “”.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer

B. Use **Erase** to remove the strings from memory.

.bmp}

{ewc mvimg, mvimage,!answer.bmp}

C. Use **Remove** to erase the strings from memory.

{ewc mvimg, mvimage,!answer.bmp}

D. Use **PageStrings** to free memory and write the strings to a swap file.

4. How can you install the SourceSafe add-in in the Visual Basic design environment?

{ewc mvimg, mvimage,!answer.bmp}

A. On the **SourceSafe** menu, click **Add project to SourceSafe**.

{ewc mvimg, mvimage,!answer.bmp}

B. Add code to the FormLoad event of your project by using the **GetSourceSafe** command.

{ewc mvimg, mvimage,!answer.bmp}

C. On the **Add-Ins** menu, click **SccAddin.SourceCodeControlAddin**.

5. A developer is using a resource file in her project and calls the LoadResPicture function to load a picture. If the second argument of the function is 2, what type of data is being loaded?

{ewc mvimg, mvimage,!answer.bmp}

A. bitmap

{ewc mvimg, mvimage,!answer.bmp}

B. gif

{ewc mvimg, mvimage,!answer.bmp}

C. icon

{ewc mvimg, mvimage,!answer.bmp}

D. cursor

6. How can you retrieve a string from Win.ini in your Windows NT application?

{ewc mvimg, mvimage,!answer.bm

A. Use the **GetSetting** function.

p}

{ewc mvimg, mvimage,!answer.bmp}

B. Use the **GetPrivateProfileString** function.

{ewc mvimg, mvimage,!answer.bmp}

C. Use **GetPublicProfileString** function.

{ewc mvimg, mvimage,!answer.bmp}

D. Use **GetProfileString** function.

This section provides pointers to a number of Web sites, which provide information useful to a Visual Basic Developer.

Note At the time this course was released, all Web sites listed in this section were valid and fully functional. However, as the World Wide Web evolves, you may find that one of the Internet jumps provided in this title is no longer valid. If this happens, try searching the Web for the site, or connect to the Mastering Series Web page on the Microsoft Web site to locate updated Uniform Resource Locators (URLs).

Jump to one of these sites from this section:

® Advanced Visual Basic
® Ask the VB Pro
® Chris & Tim's Rapid Application Development Home Page
® Entisoft Home Page
® Jens Balchen Jr.-Visual Basic Home Page
® Microsoft Mastering Series
® Microsoft Site Builder Workshop
® Microsoft Solutions Framework Home Page
® Microsoft TechNet
® Microsoft Visual Basic Case Studies
® Microsoft Visual Basic Community Resources
® Microsoft Visual Basic Cool Links Page
® Microsoft Visual Basic Events
® Microsoft Visual Basic Feature Area
® Microsoft Visual Basic Free Downloads
® Microsoft Visual Basic Home Page
® Microsoft Visual Basic Links
® Microsoft Visual Basic Product Documentation
® Microsoft Visual Basic Technical Information
® Microsoft Visual Basic Technical Support
® Microsoft Visual Basic Tip of the Week
® Microsoft Visual Basic Training
® Microsoft Visual Basic What's New
® Mike Dixon's QAID Pages
® Paul Treffers Visual Basic Home Page
® TegoSoft Self-Study Home Page
® The BASIC Archives
® The Phil Weber Home Page
® The Toolbox Visual Basic Home Page
® The Visual Basic Resource List
® VB4UandMe Web Page
® VBxtras, Inc. Home Page
® Visual Basic Education & Certification Page
® Visual Basic for Applications Home Page
® Visual Basic Programmer's Journal Web Site
® Visual Basic Starting Point

® Wild Web of Visual Basic

This section contains articles from various sources within Microsoft and from third party publishers:

® Adding Icons to the Taskbar Notification Area
® Building Document-Centered Interfaces with VB 4.0
® Sorting Easily with Version 4.0's DBGrid Control
® Taking Advantage of VB 4.0 Compiler Directives
® Visual Basic Naming Conventions
® Working with DLLs

This section contains whitepapers that provide in-depth discussion on technical topics related to Visual Basic 5.0.

® Accessing the World of Information: ODBC

® Deploying Controls on the Web

® Microsoft Visual Basic for Applications 5.0 Reviewer's Guide

® Microsoft Visual Basic Control Creation Edition version 5.0

Click here to connect to the Support Online page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft offers technical support and services, ranging from no-cost and low-cost online information services (available 24 hours a day, 7 days a week), to annual support plans. The Microsoft Support Online Web site contains the following sources of support:

® "How to" information

® Solutions to problems

® Drivers, patches, and code samples

® Answers to frequently asked questions

® Opportunities to share information and expertise with other customers

® Information about other service offerings

The *Mastering Visual Basic 5* Help file also contains information about Microsoft Technical Support. See the last topic in the Help file, "When You Need Technical Support."

Microsoft provides a variety of training options for developers, including classroom training at Microsoft Authorized Technical Education Centers (ATECs), online training, and self-paced training such as this and other Mastering Series titles.

® Face-to-Face Training

® Microsoft Online Institute

® Self-Paced Training

This section includes product descriptions and marketing and ordering information for various publishers of technical information for software developers.

® Microsoft Developer Network (MSDN)

® Microsoft Press

® Microsoft Systems Journal

® The Cobb Group

® Fawcette Technical Publications

® Pinnacle Publishing

This section includes information about User Groups and Mailing Lists that may be of interest to Visual Basic developers.

® Events

® User Groups

® Mailing Lists

Click here to connect to the Microsoft Press page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft Press is the independent publishing division of Microsoft Corporation—your source of inside information and unique perspectives about Microsoft products and related technologies. We've been around since 1984, and we offer a complete line of computer books—from self-paced tutorials for first-time computer users to advanced technical references for professional programmers.

Microsoft Press Interactive, our new multimedia publishing group, creates exciting CD-ROM based interactive training and reference products for business and home use. Microsoft Press International distributes Microsoft Press books worldwide in English language editions and in 29 local languages worldwide. In short, when you need to know more about Microsoft products, come to us. After all, who knows more about Microsoft?

In the Books section, check out the following MS Press-related material:

® Books Available from Microsoft Press

® Inside COM

For more information on how to order from MS Press, go to the next section.

® Ordering Information

Microsoft Press books and CD-ROM titles can be purchased at bookstores, software resellers, or directly from Microsoft Press. Click here to connect to the Microsoft Press page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft *Press*

PO Box 141875
Austin, TX 78714-9745
Phone: (800)MSPRESS
Fax: (800) 826-5399
CompuServe: GO MSP
MSN: GOto MSPress

In Canada, contact:

Macmillan Canada
Attn: Microsoft Press Dept.
164 Commander Blvd.
Agincourt, Ontario
Canada MIS 3C7
Phone: 1-800-667-1115
Fax: (416) 293-0846)

For information about Microsoft Press publications outside the United States and Canada, please contact your local Microsoft subsidiary, see the international distributor list on the Microsoft Press Web site, or contact Microsoft Press by mail or fax:

Microsoft Press
International Coordinator
One Microsoft Way
Redmond, WA
98052-6399
Fax: 1 (206) 936-7329

The following sample chapter from Microsoft Press publications can be found in the Books section:

® Inside COM

Click here to connect to The Cobb Group Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Cobb Group is the world's leading publisher of software-specific journals, with over 60 titles and more than 450,000 subscribers. The Cobb Group's "how-to" publications provide a constant flow of fresh ideas, timesaving tips, and innovative techniques for applications, networking, and operating systems, languages and entertainment packages.

The Cobb Group's Microsoft Desktop Products Team publishes on the Microsoft Windows 95 operating system and many Microsoft desktop applications, including Microsoft Office, Office Professional, Excel, Access, Word, PowerPoint, Works, Project, and Internet tools. The Cobb Group also publishes products on Visual Basic, Visual C++, Fox Pro 2.x, Visual FoxPro, and Windows NT.

You'll find The Cobb Group on MSN in The Cobb Group's World of Windows. On AOL, connect with keyword COBB.

Corporate Backround

Founded in 1984, The Cobb Group successfully tapped into the rapidly growing market of PC users seeking information. In 1990, *Inc.* magazine named The Cobb Group one of the 500 fastest-growing private companies in the country. In 1991, The Cobb Group was sold to Ziff-Davis Publishing Company, and today operates as an autonomous subsidiary, with headquarters in Louisville, Kentucky. The Cobb Group continues to produce publications valued for their clarity, accuracy, and timeliness.

How To Contact Us

For subscriptions to journals, CD-ROMs, or resource disks, contact:

The Cobb Group
Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
1-800-223-8720
Fax 502-491-8050
customer_relations@merlin.cobb.zd.com

You can also order sample issues at the Web site shown above.

You'll find several sample articles from The Cobb Group in the Technical Articles section:

® Adding Icons to the Taskbar Notification Area

® Building Document-Centered Interfaces with VB 4.0

® Sorting Easily with Version 4.0's DBGrid Control

® Taking Advantage of VB 4.0 Compiler Directives

® Working with DLLs

Click here to connect to the Pinnacle Publishing Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Pinnacle Publishing is a leading publisher of technical newsletters, in-depth special reports, and CD-ROMs on a variety of software products. Our information products give you time-saving tips, practical techniques, and up-to-the-minute solutions to help you get the most from your software in the least amount of time. Each of the products is highly respected for the depth, quality, and readability of its contents.

Our newsletters and special reports are written by experts—software professionals, trainers, consultants, and developers—who know their software products inside and out. They share the techniques they've learned from extensive experience, helping you to be more productive.

> *"You can examine your first issue of any Pinnacle newsletter FREE. If you don't like it, return your invoice marked 'Cancel' and pay nothing. If you like it, simply return payment with your invoice to receive the remaining 11 months of your subscription. And if, at any point during your subscription, you decide you're not completely satisfied, we'll refund your money. That's our promise to you."* —Mickey Friedman, Publisher

Get More Information

You can get more detailed information about any Pinnacle Product via our web site at **http://www.pinpub.com.** You can also get more information through our Information Fax Service (IFS) by calling toll-free 800-788-1900. When prompted, enter the desired IFS numbers along with your fax number.

We Value Our Customers

Pinnacle Publishing is known for outstanding customer service and superior technical support. We do not believe service ends when a sale is made–we have a commitment to your future success. We do whatever we can to help you get the most from your Pinnacle products and the host products they support, and to keep you apprised of new tools and resources that will make your job even easier. To put it simply, our goal is to exceed your expectations every step of the way.

You can count on Pinnacle to provide useful technology and expert information now and in the future. We have plans underway for new software and publications, and we're beginning to distribute our products via on-line services. If we can do anything to make your work more successful, please let us know.

And remember: Pinnacle products are guaranteed! If a product or publication does not meet your expectations, we'll refund your money. There's absolutely no risk—and no hassle—when you rely on Pinnacle.

How to Contact Us

Pinnacle Publishing, Inc.
PO Box 888
18000 72nd Ave S., Suite 217
Kent, WA 98035
(206) 251-1900 voice
(206) 251-5057 fax
(206) 251-6218 BBS
Email: ppi@pinpub.com
Web site: http://www.pinpub.com
CompuServe: GO PINNACLE

Click here to connect to the Microsoft Training and Certification Web page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Certified Professional (MCP) program provides the best method to prove your command of current Microsoft products and technologies. Microsoft, an industry leader in certification, is on the forefront of testing methodology. Our exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop or implement and support solutions with Microsoft products and technologies. Computer professionals who become Microsoft Certified Professionals are recognized as experts and are sought after industry-wide.

The Microsoft Certified Professional program offers four certifications, based on specific areas of technical expertise:

- **Microsoft Certified Systems Engineer.** MCSEs are qualified to effectively plan, implement, maintain, and support information systems in a wide range of computing environments with Microsoft Windows NT Server and the Microsoft BackOffice integrated family of server products.
- **Microsoft Certified Solution Developer.** MCSDs are qualified to design and develop custom business solutions with Microsoft development tools, technologies, and platforms, including Microsoft Office and Microsoft BackOffice.
- **Microsoft Certified Product Specialist.** MCPSs demonstrate in-depth knowledge of at least one Microsoft operating system. Candidates may pass additional Microsoft certification exams to further qualify their skills with Microsoft BackOffice products, development tools, or desktop applications. See **MCPS Areas of Specialization**.
- **Microsoft Certified Trainer.** MCTs are instructionally and technically qualified by Microsoft to deliver Microsoft Official Curriculum through Microsoft Solution Provider Authorized Technical Education Centers (ATECs) and Authorized Academic Training Program institutions (AATPs).

Who Should Become a Microsoft Certified Professional?

Anyone who must demonstrate technical expertise with Microsoft products should consider the program, including systems engineers, professional developers, support technicians, system and network administrators, consultants, and trainers.

What are the Benefits of Becoming a Microsoft Certified Professional?

The Microsoft Certified Professional program offers a number of immediate and long-term benefits, including the following:

- **Recognition.** Microsoft Certified Professionals are instantly recognized as experts with the technical knowledge and skills needed to design and develop or implement and support solutions with Microsoft products. Microsoft helps build this recognition by promoting the expertise of Microsoft Certified Professionals within the industry and to customers and potential clients.
- **Access to technical information.** Microsoft Certified Professionals gain access to technical information directly from Microsoft. Depending on the certification, Microsoft Certified Professionals also receive a prepaid trial membership to the Microsoft TechNet Technical Information Network, or a discount for the Microsoft Developer Network; are entitled to free incidents with Microsoft Product Support Services; and are eligible to participate in the Microsoft Beta Evaluation program.
- **Global community.** Microsoft Certified Professionals join a worldwide community of technical professionals who have validated their expertise with Microsoft products. Microsoft Certified Professionals are brought together through dedicated forums on MSN, The Microsoft Network; receive a free subscription to *Microsoft Certified Professional Magazine*, a career and professional development magazine created especially for Microsoft Certified Professionals; and, depending on their certification, are eligible to join the Network Professional Association, a worldwide association of computer professionals.

What are the Requirements for Becoming a Microsoft Certified Professional?

The certification requirements differ for each certification and are specific to the products and job functions addressed by the certification.

To review the requirements, see the description of each certification.

How Can I Get More Information About the Microsoft Certified Professional program?

The Microsoft Training and Certification Web site (see above) offers you and your customers all the information you need regarding Microsoft Official Curriculum courses, Microsoft Certified Professional certifications and exam requirements, and how education supports your customers' certification goals.

Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification) is a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Exam preparation guides (included in the Microsoft Train_Cert Offline) are available from the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

Sample assessment exams (included in the Microsoft Train_Cert Offline) are available by calling (800) 636-7544 (United States and Canada).

The Exam Study Guide provides information about how Microsoft Certified Professional exams are developed. It is available from the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

Click here to connect to the Microsoft Training and Certification page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Certified Professional (MCP) program provides the best method to prove your command of current Microsoft products and technologies. Microsoft, an industry leader in certification, is on the forefront of testing methodology. Our exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop or implement and support solutions with Microsoft products and technologies. Computer professionals who become Microsoft Certified Professionals are recognized as experts and are sought after industrywide.

For network professionals, Microsoft offers the Microsoft Certified Systems Engineer (MCSE) credential. Microsoft Certified Systems Engineers are qualified to effectively plan, implement, maintain, and support information systems in a wide range of computing environments with Microsoft Windows NT Server and the Microsoft BackOffice integrated family of server products.

To become a Microsoft Certified Systems Engineer, you must pass a series of rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise. These exams are developed with the input of professionals in the industry and reflect how Microsoft products are used in organizations throughout the world. The exams are administered by an independent organization—Sylvan Prometric—at more than 800 Authorized Prometric Testing Centers around the world.

Benefits of Becoming a Microsoft Certified Systems Engineer

Why should you become a Microsoft Certified Systems Engineer? To demonstrate to your customers and colleagues that you have what it takes to plan, implement, and support business solutions with Microsoft Windows NT and Microsoft BackOffice. As a Microsoft Certified Systems Engineer, you receive the following benefits:

® Industry recognition of your knowledge and proficiency with Microsoft products and technologies.

® Access to technical information directly from Microsoft. Microsoft Certified Systems Engineers receive a complimentary one-year subscription to the Microsoft TechNet Technical Information Network, providing valuable information via monthly CDs. If you currently have a Microsoft TechNet subscription, an additional year will be added to your existing subscription, free of charge.

® One free Priority Comprehensive support 10 pack from Microsoft and a 25 percent discount on purchases of additional Priority Comprehensive support incident 10 packs. You gain access to support for any Microsoft product, 24 hours a day, seven days a week. This provides the opportunity—when you need it—to resolve customer issues with the help of Microsoft support engineers.

® A one-year subscription to the Microsoft Beta Evaluation program. This benefit provides you with up to 12 free monthly CDs containing beta software (English only) for many of Microsoft's newest software products, allowing you to become familiar with new versions of Microsoft products before they are generally available.

® A free annual subscription to *Microsoft Certified Professional Magazine*, a career and professional development magazine created especially for Microsoft Certified Professionals.

® Microsoft Certified Systems Engineer logos and other materials to let you identify your Microsoft Certified Systems Engineer status to colleagues or clients.

® A Dedicated forum on MSN, The Microsoft Network, that allows Microsoft Certified Professionals to communicate directly with Microsoft and one another.

® *Certification Update* bimonthly newsletter from the Microsoft Certified Professional program, keeping you informed of changes and advances in the program and exams.

® Invitations to Microsoft conferences, technical training sessions, and special events.

Microsoft Certified Systems Engineer Requirements

Microsoft Certified Systems Engineers are required to pass four operating system exams and two elective exams. The operating system exams require candidates to prove their expertise with desktop, server, and networking components. The elective exams require proof of expertise with Microsoft BackOffice products.

For specific MCSE requirements, please visit the Certified Systems Engineer Web page (see below) or see the

Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification), a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Click here to connect to the Microsoft Certified Systems Engineer page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Process for Becoming a Microsoft Certified Systems Engineer

® **Preparation.** Exam preparation guides with outlines of exam topics and lists of relevant training resources are available on the Microsoft Training and Certification Web site (see top of page) and included in Microsoft Train_Cert Offline. Sample assessment exams, which enable individuals to evaluate skills and practice answering questions similar to those on actual exams, are also available.

® **Training.** Microsoft Official Curriculum courses can help you gain the skills needed for certification. These courses are available through Microsoft Solution Provider Authorized Technical Education Centers (ATECs) in both instructor-led and self-paced formats. For referrals to Microsoft ATECs, call (800) SOL-PROV.

® **Exams.** Sylvan Prometric administers all Microsoft Certified Professional exams. You must register with Sylvan Prometric prior to scheduling an exam. Exams may be taken at any of more than 800 Authorized Prometric Testing Centers around the world. To register for an exam, please call Sylvan Prometric at (800) 755-EXAM.

For More Information

For additional information on becoming a Microsoft Certified Systems Engineer, or on Microsoft Training and Certification in general, visit the Microsoft Training and Certification Web site (see top of page) or get a copy of Microsoft Train_Cert Offline. Both resources include complete information about the Microsoft Certified Professional program and Microsoft Official Curriculum and sample exams for the MCP program.

Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Exam preparation guides (included in the Microsoft Train_Cert Offline) are available from the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

Sample assessment exams (included in the Microsoft Train_Cert Offline) are available by calling (800) 636-7544 (United States and Canada).

The Exam Study Guide, which provides information about how Microsoft Certified Professional exams are developed, is available by using the Microsoft Sales Fax Service: (800) 727-3351 (request document catalog #1000-0733).

Click here to connect to the Microsoft Training and Certification page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Certified Professional (MCP) program provides the best method to prove your command of current Microsoft products and technologies. Microsoft, an industry leader in certification, is on the forefront of testing methodology. Our exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop or implement and support solutions with Microsoft products and technologies. Computer professionals who become Microsoft Certified Professionals are recognized as experts and are sought after industrywide.

For developers, Microsoft offers the Microsoft Certified Solution Developer (MCSD) credential. Microsoft Certified Solution Developers are qualified to design and develop custom business solutions with Microsoft development tools, technologies, and platforms, including Microsoft Office and Microsoft BackOffice.

To become a Microsoft Certified Solution Developer, you must pass a series of rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise. These exams are developed with the input of professionals in the industry and reflect how Microsoft products are used in organizations throughout the world. The exams are administered by an independent organization—Sylvan Prometric—at more than 800 Authorized Prometric Testing Centers around the world.

Benefits of Becoming a Microsoft Certified Solution Developer

Why should you become a Microsoft Certified Solution Developer? To demonstrate to your customers and colleagues that you have what it takes to design and develop superior custom solutions with Microsoft tools and technologies. As a Microsoft Certified Solution Developer, you receive the following benefits:

® Industry recognition of your knowledge and proficiency with Microsoft products and technologies.

® Microsoft Developer Network subscription discount. Microsoft Certified Solution Developers receive a discount they may apply toward a one-year Microsoft Developer Network (MSDN) Library Subscription or Professional Subscription, providing valuable information on quarterly CDs.

® One free 10 pack of Priority Development with Desktop support incidents and a 25 percent discount on purchases of additional Priority Developer and Desktop support incident 10 packs. You gain access to support for any Microsoft product, seven days a week, 24 hours a day. This provides the opportunity-when you need it-to resolve customer issues with the help of Microsoft support engineers.

® A one-year subscription to the Microsoft Beta Evaluation program. This benefit provides you with up to 12 free monthly CDs containing beta software (English only) for many of Microsoft's newest software products. This allows you to become familiar with new versions of Microsoft products before they are generally available.

® A free annual subscription to *Microsoft Certified Professional Magazine*, a career and professional development magazine created especially for Microsoft Certified Professionals.

® Microsoft Certified Solution Developer logos and other materials to help you identify your Microsoft Certified Solution Developer status to colleagues or clients.

® A dedicated forum on MSN, The Microsoft Network, that enables Microsoft Certified Professionals to communicate directly with Microsoft and one another.

® *Certification Update* bimonthly newsletter from the Microsoft Certified Professional program, keeping you informed of changes and advances in the program and exams.

® Invitations to Microsoft conferences, technical training sessions, and special events.

Microsoft Certified Solution Developer Requirements

Microsoft Certified Solution Developers are required to pass two core-technology exams and two elective exams. The core technology exams require candidates to prove their understanding of Windows 32-bit architecture, OLE, UI design, and Windows Open Services Architecture components. The elective exams require proof of expertise with Microsoft development tools.

For specific MCSD requirements, please visit the Certification Solution Developers Web page (see below) or see Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification), a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-

7544 (United States and Canada).

Click here to connect to the Certified Solution Developers Web page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Process for Becoming a Microsoft Certified Solution Developer

® **Preparation.** Exam preparation guides with outlines of exam topics and lists of relevant training resources are available on the Microsoft Training and Certification Web site (see top of page) and included in Microsoft Train_Cert Offline. Sample assessment exams, which enable individuals to evaluate skills and practice answering questions similar to those on actual exams, are also available.

® **Training.** Microsoft Official Curriculum courses can help you gain the skills needed for certification. These courses are available through Microsoft Solution Provider Authorized Technical Education Centers (ATECs) in both instructor-led and self-paced formats. For referrals to Microsoft ATECs, call (800) SOL-PROV.

® **Exams.** Sylvan Prometric administers all Microsoft Certified Professional exams. You must register with Sylvan Prometric prior to scheduling an exam. Exams may be taken at any of more than 800 Authorized Prometric Testing Centers around the world. To register for an exam, please call Sylvan Prometric at (800) 755-EXAM.

For More Information

For additional information on becoming a Microsoft Certified Solution Developer, or on Microsoft Training and Certification in general, visit the Microsoft Training and Certification Web site (see top of page) or get a copy of Microsoft Train_Cert Offline. Both resources include complete information about the Microsoft Certified Professional program and Microsoft Official Curriculum and sample exams for the MCP program.

Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Exam preparation guides are available by using the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

Sample assessment exams are available by calling (800) 636-7544 (United States and Canada).

The Exam Study Guide, which provides information about how Microsoft Certified Professional exams are developed, is available by using the Microsoft Sales Fax Service: (800) 727-3351 (request document catalog #1000-0733).

Click here to connect to the Microsoft Training and Certification Web page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Certified Professional (MCP) program provides the best method to prove your command of current Microsoft products and technologies. Microsoft, an industry leader in certification, is on the forefront of testing methodology. Our exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop or implement and support solutions with Microsoft products and technologies. Computer professionals who become Microsoft Certified Professionals are recognized as experts and are sought after industrywide.

For individuals who would like to demonstrate their expertise with a particular Microsoft product, Microsoft offers the Microsoft Certified Product Specialist (MCPS) credential. Microsoft Certified Product Specialists have demonstrated in-depth knowledge of at least one Microsoft operating system. Candidates may pass additional Microsoft certification exams to further qualify their skills with Microsoft BackOffice products, development tools, or desktop applications.

To become a Microsoft Certified Product Specialist, you must pass one or more rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise. These exams are developed with the input of professionals in the industry and reflect how Microsoft products are used in organizations throughout the world. The exams are administered by an independent organization—Sylvan Prometric—at more than 800 Authorized Prometric Testing Centers around the world.

Benefits of Becoming a Microsoft Certified Product Specialist

Why should you become a Microsoft Certified Product Specialist? To demonstrate to your customers and colleagues that you have the specialized knowledge required to use or support specific Microsoft products. As a Microsoft Certified Product Specialist, you receive the following benefits:

® Industry recognition of your knowledge and proficiency with Microsoft products and technologies.

® Access to technical information directly from Microsoft. Microsoft Certified Product Specialists receive a complimentary Microsoft TechNet CD, plus a 50 percent discount toward a one-year membership to the Microsoft TechNet Technical Information network, providing valuable information on monthly CDs.

® A free annual subscription to *Microsoft Certified Professional Magazine*, a career and professional development magazine created especially for Microsoft Certified Professionals.

® Microsoft Certified Product Specialist logos and other materials to enable you to identify your Microsoft Certified Product Specialist status to colleagues or clients.

® A dedicated forum on MSN, The Microsoft Network, that enables Microsoft Certified Professionals to communicate directly with Microsoft and one another.

® *Certification Update* bimonthly newsletter from the Microsoft Certified Professional program, keeping you informed of changes and advances in the program and exams.

® Invitations to Microsoft conferences, technical training sessions, and special events.

Microsoft Certified Product Specialist Requirements

Microsoft Certified Product Specialists are required to pass one operating system exam, proving their expertise with a current Microsoft Windows desktop or server operating system.

For specific MCPS requirements, please visit the Certified Product Specialist Web page (see below) or see the Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification), a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Click here to connect to the Certified Product Specialist Web page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Process for Becoming a Microsoft Certified Product Specialist

- **Preparation.** Exam preparation guides with outlines of exam topics and lists of relevant training resources are available at the Microsoft Training and Certification Web site (see top of page). Sample assessment exams, which enable individuals to evaluate skills and practice answering questions similar to those on actual exams, are also available. In addition, an Exam Study Guide is available that provides information about how Microsoft Certified Professional exams are developed. Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification) is a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).
- **Training.** Microsoft Official Curriculum courses can help you gain the skills needed for certification. These courses are available through Microsoft Solution Provider Authorized Technical Education Centers (ATECs) in both instructor-led and self-paced formats. For referrals to Microsoft ATECs, call (800) SOL-PROV.
- **Exams.** Sylvan Prometric administers all Microsoft Certified Professional exams. You must register with Sylvan Prometric prior to scheduling an exam. Exams may be taken at any of more than 800 Authorized Prometric Testing Centers around the world. To register for an exam, please call Sylvan Prometric at (800) 755-EXAM.

For More Information

The Microsoft Training and Certification Web site (see top of page) offers you and your customers all the information you need regarding Microsoft Official Curriculum courses, Microsoft Certified Professional certifications and exam requirements, and how education supports your customers' certification goals.

Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification) is a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Exam preparation guides (included in the Microsoft Train_Cert Offline) are available from the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

Sample assessment exams (included in the Microsoft Train_Cert Offline) are available by calling (800) 636-7544 (United States and Canada).

The Exam Study Guide provides information about how Microsoft Certified Professional exams are developed. It is available from the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

An area of specialization should focus on a specific Microsoft product or technology. For an area of specialization, the exam combination should include a minimum of one required operating system exam and one elective exam. The attached document, Microsoft Certified Product Specialist Examples of Areas of Specialization, contains some examples of areas of specialization identified by Microsoft that can be used to guide you further.

Below is a list of areas of specialization identified by Microsoft that can be used to guide you in combining exams. These specializations are categorized into three tracks:

- **Systems Engineer** combinations are chosen from Microsoft Certified Systems Engineer requirements.
- **Solution Developer** combinations are chosen from Microsoft Certified Solution Developer requirements.
- **Desktop Support** combinations include a desktop operating system exam and any desktop application exam.

Systems Engineer Areas of Specialization

Networking—Leads to Systems Engineer

Operating System Exam

- Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

  OR

- Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exams (two exams: one desktop operating system exam and one networking exam)

Choose one desktop operating system exam:

- Exam 70-63: Implementing and Supporting Microsoft Windows 95
- Exam 70-30: Microsoft Windows 3.1
- Exam 70-48: Microsoft Windows for Workgroups 3.11—Desktop
- Exam 70-73: Implementing and Supporting Microsoft Windows NT Workstation 4.0

  OR

- Exam 70-42: Implementing and Supporting Microsoft Windows NT Workstation 3.51

Choose one networking exam:

- Exam 70-58: Networking Essentials
- Exam 70-47: Networking with Microsoft Windows 3.1
- Exam 70-46: Networking with Microsoft Windows for Workgroups 3.11—Desktop

Microsoft TCP/IP—Leads to Systems Engineer

Operating System Exam

- Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

  OR

- Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exam

- Exam 70-53: Internetworking Microsoft TCP/IP on Microsoft Windows NT (3.5–3.51)

Microsoft Mail for PC Networks 3.2—Leads to Systems Engineer

Operating System Exam

- Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

  OR

- Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exam

® Exam 70-37: Microsoft Mail for PC Networks 3.2—Enterprise

Microsoft SQL Server 4.2—Leads to Systems Engineer

Operating System Exam

® Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

OR

® Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exams (two exams)

® Exam 70-26: System Administration of Microsoft SQL Server 6

® Exam 70-27: Implementing a Database Design on Microsoft SQL Server 6

OR

® Exam 70-21: Microsoft SQL Server 4.2 Database Implementation

® Exam 70-22: Microsoft SQL Server 4.2 Database Administration for Microsoft Windows NT

Microsoft Systems Management Server 1.0—Leads to Systems Engineer

Operating System Exam

® Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

OR

® Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exam

® Exam 70-14: Implementing and Supporting Microsoft Systems Management Server 1.0

Microsoft SNA Server—Leads to Systems Engineer

Operating System Exam

® Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

OR

® Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exam

® Exam 70-12: Microsoft SNA Server

Microsoft Exchange—Leads to Systems Engineer

Operating System Exam

® Exam 70-67: Implementing and Supporting Microsoft Windows NT Server 4.0

OR

® Exam 70-43: Implementing and Supporting Microsoft Windows NT Server 3.51

Elective Exam

® Exam 70-75: Implementing and Supporting Microsoft Exchange

Solution Developer Areas of Specialization

Microsoft Visual Basic 3.0 for Windows—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I

® Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

® Exam 70-50: Microsoft Visual Basic 3.0 for Windows—Application Development

Microsoft Access 2.0 for Windows—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I

®____Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

® Exam 70-51: Microsoft Access 2.0 for Windows—Application Development

Microsoft Excel Application Development—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I

® Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

® Exam 70-52: Developing Applications with Microsoft Excel 5.0 Using Visual Basic for Applications

Microsoft Visual FoxPro—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I

® Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

® Exam 70-54: Programming in Microsoft Visual FoxPro 3.0 for Windows

Implementing OLE in MFC Applications—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I

® Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

® Exam 70-25: Implementing OLE in Microsoft Foundation Class Applications

Microsoft Visual Basic 4.0—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

- ® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I
- ® Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

- ® Exam 70-65: Programming with Microsoft Visual Basic 4.0

Microsoft Access for Windows 95—Leads to Solution Developer

Operating System Exam

Choose one Microsoft Windows architecture exam:

- ® Exam 70-150: Microsoft Windows Operating Systems and Services Architecture I
- ® Exam 70-151: Microsoft Windows Operating Systems and Services Architecture II

Elective Exam

- ® Exam 70-69: Microsoft Access for Windows 95 and the Microsoft Access Developer's Toolkit

Desktop Support Areas of Specialization

Microsoft Word 6.0 for Windows—Leads to Desktop Support

Operating System Exam

Choose one desktop operating system exam:

- ® Exam 70-63: Microsoft Windows 95
- ® Exam 70-30: Microsoft Windows 3.1
- ® Exam 70-48: Microsoft Windows for Workgroups 3.11—Desktop
- ® Exam 70-73: Implementing and Supporting Microsoft Windows NT Workstation 4.0

  OR
- ® Exam 70-42: Implementing and Supporting Microsoft Windows NT Workstation 3.51

Elective Exam

- ® Exam 70-49: Microsoft Word 6.0 for Windows

  OR
- ® Exam 70-66: Microsoft Word for Windows 95

Microsoft Excel 5.0 for Windows—Leads to Desktop Support

Operating System Exam

Choose one desktop operating system exam:

- ® Exam 70-63: Microsoft Windows 95
- ® Exam 70-30: Microsoft Windows 3.1
- ® Exam 70-48: Microsoft Windows for Workgroups 3.11—Desktop
- ® Exam 70-73: Implementing and Supporting Microsoft Windows NT Workstation 4.0

  OR
- ® Exam 70-42: Implementing and Supporting Microsoft Windows NT Workstation 3.51

Elective Exam

- ® Exam 70-39: Microsoft Excel 5.0 for Windows

Microsoft Project 4.0 for Windows—Leads to Desktop Support

Operating System Exam

Choose one desktop operating system exam:

- ® Exam 70-63: Implementing and Supporting Microsoft Windows 95
- ® Exam 70-30: Microsoft Windows 3.1
- ® Exam 70-48: Microsoft Windows for Workgroups 3.11—Desktop
- ® Exam 70-73: Implementing and Supporting Microsoft Windows NT Workstation 4.0

OR

- ® Exam 70-42: Implementing and Supporting Microsoft Windows NT Workstation 3.51

Elective Exam

- ® Exam 70-38: Microsoft Project 4.0 for Windows

Logo Recognition for Areas of Specialization

Candidates who pass a minimum of one required operating system exam and one elective are encouraged to display their specialty as an additional tagline below their Microsoft Certified Product Specialist logo. Examples of the MCPS logo including Area of Specialization taglines include:

Area of Specialization
Networking

{ewc mvimg, mvimage,! mscert.bmp}
Product Specialist
Networking

Area of Specialization:
Microsoft Word 6.0 for Windows

{ewc mvimg, mvimage,! mscert.bmp}
Product Specialist
Microsoft Word 6.0
for Windows

Area of Specialization
Microsoft Visual Basic 3.0

{ewc mvimg, mvimage,! mscert.bmp}
Product Specialist
Microsoft Visual Basic 3.0

Area of Specialization:
Multiple Specializations

{ewc mvimg, mvimage,! mscert.bmp}
Product Specialist
Networking
Microsoft Visual Basic 3.0
Microsoft Word 6.0
for Windows

Click here to connect to the Microsoft Training and Certification Web page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft Certified Trainers (MCTs) play an important role in Microsoft's Education and Certification process. MCTs are qualified instructionally and certified technically by Microsoft to deliver Microsoft Official Curriculum instructor-led courses to computer professionals. Only trainers who will to deliver Microsoft Official Curriculum instructor-led courses at Microsoft Authorized Technical Education Centers are eligible to become Microsoft Certified Trainers. Once you have been approved as an MCT, your MCT credential will be accepted in all Microsoft education channels.

Microsoft Official Curriculum courses, developed by Microsoft product groups, educate computer professionals who develop, support, and implement solutions that use Microsoft technology.

Microsoft authorized education channels include Microsoft Authorized Technical Education Centers (ATECs), Microsoft Authorized Academic Training Program (AATP), and Microsoft Online Institute (MOLI).

For more information about the process of becoming an MCT, refer to the Microsoft Certified Trainer Guide available for downloading at the Microsoft Training and Certification Web site (see top of page) or by calling (800) 636-7544. After you have reviewed this document, complete a Microsoft Certified Trainer Application. To request an MCT application kit, please call Microsoft at (800) 688-0496.

Benefits of Becoming a Microsoft Certified Trainer

Why should you become a Microsoft Certified Trainer? To qualify to deliver Microsoft Official Curriculum at Microsoft authorized education sites. Only trainers who intend to deliver Microsoft Official Curriculum at Microsoft authorized education sites are eligible to become Microsoft Certified Trainers. As a Microsoft Certified Trainer, you receive the following benefits:

® Industry recognition of your knowledge and proficiency with Microsoft products and technologies, and expertise delivering Microsoft Official Curriculum.

® Access to technical information directly from Microsoft. MCTs receive a complimentary Microsoft TechNet CD, plus a 50 percent discount toward a one-year membership to the *Microsoft TechNet* Technical Information Network, providing valuable information on monthly CDs.

® Microsoft Certified Trainer logos and other materials to let you identify your Microsoft Certified Trainer status to colleagues or clients.

® A dedicated MCT Forum on MSN, The Microsoft Network, that enables Microsoft Certified Trainers to communicate directly with Microsoft trainers and course developers, and one another.

® Annual subscription to *Microsoft Certified Professional Magazine,* a career and professional development magazine created especially for Microsoft Certified Professionals.

® *Education Forum* monthly newsletter from Microsoft Education and Certification, keeping you informed of new courses, new exams, trainer certification requirements, course schedule, and program updates.

® *Certification Update* bimonthly newsletter from the Microsoft Certified Professional program, keeping you informed of changes and advances in the program and exams.

® Invitations to Microsoft conferences, technical training sessions, and special events.

Microsoft Certified Trainer Requirements

Microsoft Certified Trainers are certified on a course-by-course basis. For specific trainer certification requirements for each Microsoft Official Curriculum refer to the *Microsoft Certified Trainer Course Requirements* in the *Microsoft Certified Trainer Information Kit* or use the Microsoft Sales Fax Service at (800) 727-3351.

Process for Becoming a Microsoft Certified Trainer

® **Apply.** Submit a Microsoft Certified Trainer application with proof of your instructional skills and the name of the Microsoft authorized education site that will employ you (or to which you are under contract).

® **Attend the Course.** Attend the Microsoft Official Curriculum course at a local Microsoft ATEC or attend a Microsoft Certified Trainer course at Microsoft. For referrals to Microsoft ATECs, call (800) SOLPROV. Fax your student course certificate to Microsoft at (801) 578-1429 to demonstrate you have attended the course.

- ® **Prepare for the Exam.** Exam preparation guides with outlines of exam topics and lists of relevant training resources are available on the Microsoft Training and Certification Web site (see top of page) and included in Microsoft Train_Cert Offline. Sample assessment exams, which enable individuals to evaluate skills and practice answering questions similar to those on actual exams, are also available.
- ® **Pass the Exam(s).** Sylvan Prometric administers all Microsoft Certified Professional exams and Microsoft trainer exams. When there is no Microsoft Certified Professional exam for a Microsoft Official Curriculum course, a trainer exam is created. You must register with Sylvan prior to scheduling an exam. You may take exams at any of more than 700 Sylvan Authorized Testing Centers around the world. To register for an exam, call Sylvan at (800) 755-EXAM.
- ® **Purchase and Review the Trainer Kit.** Prepare to deliver the Microsoft Official Curriculum course by calling Microsoft at (800) 688-0496 to purchase the Microsoft Trainer Kit. Outside the United States and Canada, trainers can purchase trainer kits through their local Microsoft ATEC. Review all Microsoft Trainer Kit materials and course materials, and complete all labs and demonstrations.

For More Information

For additional information on becoming a Microsoft Certified Trainer or to receive a *Microsoft Certified Trainer Information Kit,* call Microsoft at (800) 688-0496. For information regarding Microsoft Training and Certification in general, visit the Web site (see top of page).

Microsoft Train_Cert Offline (formerly Microsoft Roadmap to Education and Certification) is a copy of the Microsoft Training and Certification Web site on CD-ROM. Microsoft Train_Cert Offline is available by calling (800) 636-7544 (United States and Canada).

Exam preparation guides are available separately by using the Microsoft Sales Fax Service: (800) 727-3351 (refer to the document catalog).

Sample assessment exams are available by calling (800) 636-7544 (United States and Canada).

An Exam Study Guide, which provides information about how Microsoft Certified Professional exams are developed, is available by using the Microsoft Sales Fax Service: (800) 727-3351 (request document catalog #1000-0733).

This section contains information about the Microsoft Logo Programs. The Microsoft logo assures end users that your product takes advantage of the new technologies integrated into all Microsoft products.

® Powered by Microsoft BackOffice Logo Program

® Windows 95 Logo Program

For up-to-date information, see the Powered by BackOffice page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Until recently, Web pages were mainly static HTML pages, providing a way for businesses to simply present information on products or services. Today, businesses recognize that web sites can do much more. Innovative Web sites allow people to dynamically interact with information, find what they want, and enjoy the experience. In short, the best Web sites are destinations worth visiting over and over.

In its commitment to enable the evolution of the Internet, Microsoft delivers powerful solutions for businesses to customize, manage and present dynamic information on the Web while meeting customers' demands for speed and stability. Through the power of the Microsoft BackOffice suite of integrated server products including the Internet Information Server, businesses now have the best platform for the new generation of dynamic web applications.

Web sites bearing the Powered by Microsoft BackOffice logo utilize the versatility of these powerful solutions, setting a new standard for excellence in web development. Below are some examples of sites Powered by BackOffice on the Web today.

Database-driven Publishing

The Internet Database Connector in Microsoft's Internet Information Server provides seamless access to ODBC databases, such as Microsoft SQL Server and Microsoft Access for the fastest way to publish database information on the Internet. Microsoft's Online Events Calendar demonstrates how easy this can be when your site is Powered by BackOffice.

Web sites bearing the Powered by Microsoft BackOffice logo utilize the versatility of these powerful solutions, setting a new standard for excellence in Web development.

Click here to connect to the Designed for Windows 95 Logo page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Designed for Windows 95 logo was developed to help end users easily identify hardware and software products that were designed specifically for the Microsoft Windows 95 operating system. Users can mix and match hardware and software products designated with the Windows 95 logo and be assured that the products will take advantage of the new technologies integrated into the Microsoft Windows 95 operating system. Licensing the logo enables you to use the logo on your product packaging, advertising, collateral, and other marketing materials, which signals to customers that you designed your product to give them extra benefits when it is run on Windows 95.

Because both hardware and software are now tested before rights to use the logo are granted, customers have a high regard for the logo. Hardware is tested by the Microsoft Windows Hardware Quality Labs (whqlinfo@microsoft.com). Software applications are tested by a fully independent laboratory, VeriTest, Inc. (logolab@VeriTest.com).

Important Notes

The logo indicates that a software product provides all the features outlined in these guidelines; it is not a "quality assurance" seal. VeriTest will not be testing the quality of your product or ensuring that it is "bug-free"—VeriTest's job is just to make sure that your product has full functionality, that the logo features are present, and that your product is not generating frequent faults or system crashes.

Please note that the license agreement states:

You may only use the Logo as a symbol that your Product is compatible with the Microsoft Windows 95 operating system. You may not explicitly state or imply that Microsoft in any way endorses your product. Also, the Windows Logo program is not intended to be a "certification" program, i.e., the Logo does not represent that Microsoft certifies your product in any way.

Please note further that VeriTest's role in the process is to test applications for compliance with the logo criteria. A passing test result is not a certification or endorsement of your product(s) by VeriTest.

Click here to connect to the Microsoft Solution Provider Program page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft Solution Providers (SPs) are independent organizations that have teamed with Microsoft to use technology to solve business problems for companies of all sizes and industries. SPs use the Microsoft Solutions Platform of products as building blocks and offer various value-added services, such as integration, consulting, software customization, developing turn-key applications, and technical training and support. All Solution Providers have at least one Microsoft Certified Professional on staff who has demonstrated expertise in developing, implementing, and supporting Microsoft solutions.

Microsoft SP's understand how to help organizations take advantage of today's powerful new graphical client-server applications to make sound business decisions. By partnering with Microsoft, Solution Providers can combine their own high-quality services and expertise with Microsoft's powerful technologies, products and information.

The Microsoft Solution Provider team is a full-service commitment to your organization. We offer the following advantages:

Understanding of Your Business

Solution Providers—working with Microsoft—have the knowledge and expertise to match technology to the exact needs of any business. Some SPs focus on a specific vertical market; others serve organizations in a wide range of industries. All SPs are carefully screened and qualified to ensure consistent, high-quality capabilities.

World-Wide Diversity

The Microsoft Solution Provider program includes thousands of Solution Providers (and tens of thousands of technical experts) throughout the world. Often, multiple Solution Providers, each with a specific area of expertise, will team up to provide a comprehensive solution to a customer's needs.

Responsive, Wide-Ranging Service

Solution Providers offer a level of expertise, support, and training not always available or affordable within a single organization, with services ranging from computer network design consulting to installation and highly specialized custom application development, to on-site training and support. Many SPs can also integrate and develop solutions using multiple vendors' products and tools.

The Solution Provider Exchange Specialist Initiative

One of the benefits offered through the Solution Provider program is the ability to participate in various "specialist" programs. As a Solution Provider Exchange Specialist, you will be given technical training, resources, and customer leads. To qualify, your organization must be an active member of the Solution Provider program, and technical staff must pass an examination on Microsoft Exchange Server.

Other Microsoft Solution Provider Specialist Programs include: Windows 95 and Office for Windows 95 Migration Specialist, Internet Specialist, and 32-bit Migration Specialist.

Enrolling in the Solution Provider Program

To obtain an application to become a Microsoft Solution Provider in the United States and Canada, call Microsoft toll-free at: (800) SOLPROV [(800) 765-7768].

Outside the U.S. or Canada, please contact your local subsidiary. Customers who are deaf or hard of hearing can reach Microsoft text telephone (TT/TDD) services by calling (800) 892-5234 in the United States or (905) 568-9641 in Canada.

There are a large number of events that feature Microsoft Visual Basic as a key part of the program. The four that are of greatest interest to experiences Visual Basic programmer are VBits, TechEd, DevCon, and DevDays. These events occur throughout the year and in many different countries around the world.

Click here to connect to the Microsoft Visual Basic page of the Microsoft Web Site.
{ewc mvimg, mvimage,!intjump.bmp}

As a member of one or more of these mailing lists you can engage in discussions with diverse groups of developers. Representatives from Microsoft will also be available on these lists to help answer questions and keep you up to date on the latest developments.

Examples of technologies that are represented on mailing lists include Microsoft Visual Basic Scripting Edition, Microsoft Internet Explorer, the Microsoft Crypto, WinInet, and WebPost APIs, Microsoft ActiveX Controls, Microsoft ActiveX Scripting, Microsoft Internet Information Server, and many more.

Each Microsoft technology mailing list is available in either digest or non-digest form. The digest form compiles all postings to a particular mailing list, and sends the compilation out in a single piece of e-mail: each day. You select the form you wish to receive when you subscribe. Note that you can change the form in which you receive the list at any time.

To subscribe to a mailing list, send e-mail to the mailing list address and, in the message body, type

Subscribe *listname username@email.address*

listname is the name of the list to which you want to subscribe; *username* is your e-mail name; and *email.address* is your e-mail address.

To receive list mailings in digest form, type

Digest *listname username@email.address*

The list below is a sample of some of these Microsoft mailing lists. For the latest information on what mailing lists are available, connect to the Mailing list information on the Resources page of the Microsoft Internet Workshop. {ewc mvimg, mvimage,!intjump.bmp}

Note that some of the lists below provide an e-mail address you can contact for more information.

ActiveXControls@lists.msn.com

Development of ActiveX Controls and Containers (formerly OLE Controls and Containers), COM Objects for the Internet, and OLE Controls 96.

E-mail:: oleidea@microsoft.com

ActiveXScript@lists.msn.com

ActiveX Scripting mailing list, covering development of OLE scripting engines, and the use of OLE scripting languages (other than VBScript).

E-mail:: oleidea@microsoft.com

CodeSign@lists.msn.com

Microsoft's Windows Trust Verification Services, a set of API's which determine whether a software component contains digital certificates that identify it as being authentic software released by a publisher trusted on the local user's system.

E-mail:: safecode@microsoft.com

DevWire@microsoft.nwnet.com

DevWire is a list server for developers. You can subscribe to this free service to receive headlines of developer-related news, events, and information from Microsoft over electronic mail, plus occasional Flash messages on time-sensitive topics. The DevWire Page contains the latest edition of the DevWire electronic newsletter (excluding Flash messages) with pointers to other pages within our Web site and elsewhere on the World Wide Web.

DocObjects@lists.msn.com

Development of OLE Document Objects (DocObjs) and DocObj container applications.

To submit suggestions, send e-mail to the OLE Suggestion alias: oleidea@microsoft.com

ISAPI@lists.msn.com

ActiveX Server Framework mailing list, focusing on the development of ISAPI applications, server-side scripts, and ISAPI filters.

ISAPI-L - listserv@peach.ease.lsoft.com

For developers using or interested in ISAPI.

ISAPI-Server-Porters@lists.msn.com

For web server vendors supporting ISAPI in their server.

Note This mailing list is restricted to Web server vendors.

OfficeIA@lists.msn.com

Discussions on the Internet Assistants and Wizards of Microsoft Office, including Word, Excel, PowerPoint, etc.

OLEHyperlinking@lists.msn.com

OLE hyperlinking services and interfaces, technologies which allow Windows applications to integrate with one another and with Web browsers using popular forward and back hyperlinking navigation.

E-mail: oleidea@microsoft.com

VBScript@lists.msn.com

Microsoft Visual Basic Scripting Edition (VBScript) mailing list, covering all aspects of VBScript development.

E-mail: vbwish@microsoft.com

WinInet@lists.msn.com

Internet extensions to the Microsoft Win32 application programming interface (API). These extensions give Win32 applications easy access to common Internet protocols such as HTTP, FTP, and Gopher.

WinNTNews-Admin@winnews.microsoft.com

Microsoft's WinNT News Electronic Newsletter. Issues contain information on various pertinent topics, such as Subscription information, News and Events, FAQs, Hints and Tips, User Group News, Third Party News, and Microsoft News and Product Information.

Microsoft Official Curriculum is also available in self-paced training formats for those who prefer to learn on their own, at their own pace. These materials are available in book, computer-based training (CBT), and mixed-media (book and video) formats. A Solution Provider ATEC in your area can provide you with self-paced training materials, or they can be purchased wherever Microsoft Press books are sold.

Additional self-paced training materials are developed by third-party companies partnering with Microsoft. Microsoft Certified Professional (MCP) Approved Study Guides have been carefully and rigorously reviewed and approved by Microsoft as an effective way to prepare for Microsoft certification. Look for the Microsoft Certified Professional Logo prominently displayed on all MCP Approved Study Guides, found at retail bookstores.

The next section provides details on all of the currently available Mastering Series titles:

® The Microsoft Mastering Series

Click here to connect to the Mastering Series page on the Microsoft Web site. Use this web site to get up-to-date information on new and upcoming releases.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Mastering Series titles are content-rich, self-paced, interactive CD-ROM–based learning tools that are designed to help you quickly master Microsoft development tools. The courses combine high-level technical content with an intuitive learning methodology, so you can learn in the way that best suits your schedule and study habits.

Mastering Visual Basic 5 is the latest in the Mastering Series of titles. The other Mastering Series titles currently available are:

® Mastering Microsoft Visual FoxPro

® Mastering Microsoft Visual C++ 4

® Mastering Microsoft Exchange Development

® Mastering Internet Development with Microsoft ActiveX Technologies

® Mastering Microsoft Office 97 Development

All Mastering titles include interactive lessons, narrated demonstrations, hands-on labs, sample code, and a library of reference material to supplement courses and exercises. A flexible browsing and searching system allows you to navigate directly to the course material or library resources that you need.

Most of the Mastering Series courses are Microsoft Official Curriculum and can be used to gain the skills necessary for Microsoft Certification. Microsoft Mastering Series products are not introductions to programming. It is assumed that you have experience using Microsoft development tools to create solutions for Microsoft Windows.

Course No. 676

This course is intended for developers and system administrators responsible for implementing applications for Microsoft Exchange.

This course teaches developers to design and build sophisticated workgroup applications around Microsoft Exchange using the Microsoft Visual Basic programming system. It also teaches nonprogrammers, such as system administrators, to build simpler form and folder applications on Microsoft Exchange without programming.

The topics covered in this course are:

® Designing a Microsoft Exchange application

® Implementing public folders

® Creating Microsoft Exchange Forms Designer applications

® Using Visual Basic to extend a form

® OLE Messaging

® OLE scheduling

® Intergrating Microsoft Office and Microsoft Exchange

At Course Completion

At the end of the course, students will be able to design a Microsoft Exchange application, implement public folders, create Forms Designer applications, extend form applications using Visual Basic, use OLE messaging, use OLE scheduling, and integrate Microsoft Exchange applications with Microsoft Office.

Microsoft Certified Professional Exams

Information on Microsoft Certified Professional exam preparation provided by this course is yet to be decided.

Prerequisites

Students should have a basic understanding of using an electronic mail system.

To create basic form and folder applications, students do not need any programming experience.

To complete the more advanced sections of the course, students should have intermediate experience using Visual Basic. Students should be able to:

® Create an application in Visual Basic that uses multiple forms.

® Use OLE Automation to control other applications.

® Retrieve and update database information using data access objects (DAOs).

Chapter 1: Designing Solutions

Topics

® Development overview

® Development tools

® Creating solutions

Skills

Students will be able to:

® Describe the major components of Microsoft Exchange.

® Describe the types of solutions that you can build using Microsoft Exchange.

® Describe the Microsoft Exchange development tools.

® Choose the right tools to build a Microsoft Exchange solution.

® Describe the difference between a Microsoft Exchange form application and an external mail-enabled application.

Chapter 2: Implementing Folders

Topics

® Creating folders

® Using forms in folders

® Creating views

® Folder permissions

® Setting rules

® Using a folder offline

® Replicating folders

® Managing folders

Lab

Implementing folders

Skills

Students will be able to:

® Create a new public folder or copy an existing folder.

® Associate forms with a folder and remove forms from a folder.

® Define views for a folder.

® Grant permissions on a public folder.

® Set rules for a folder.

® Make a folder available for use offline and synchronize an offline folder.

® Explain the fundamentals of replication.

® Manage folders.

Chapter 3: Creating Basic Form Applications

Topics

® Forms Designer

® Creating forms

® Creating fields

® Saving and installing forms

Lab

Creating basic form applications

Skills

Students will be able to:

® Use Forms Designer to create a basic Post and Send form.

® Install a Microsoft Exchange form.

® Differentiate between global forms, private forms, and forms associated with a folder.

® Differentiate between a Post form and a Send form.

® Create a form that enables a user to include an attachment.

® Perform form administration tasks such as determining which forms are associated with a folder, copying, moving, deleting, and adding forms to folders.

® Describe how forms are registered and activated.

Chapter 4: Creating Advanced Form Applications

Topics

® Adding a second window to a form

® Using custom response events

® Adding Help information

Lab

Creating advanced form applications

Skills

Students will be able to:

® Differentiate between a window and a form.

® Create a form that contains a Read window and a Compose window.

® Create a form with multiple custom responses.

® Share fields between forms.

® Add Help information to a form, a window, and fields.

® Modify standard menus on a form.

® Create an application that consists of multiple forms.

Chapter 5: Extending Forms: Basic Modifications

Topics

® Understanding the Visual Basic–based project

® Common modifications

® Testing your form

Lab

Extending forms: Basic modifications

Skills

Students will be able to:

® Determine when you need to extend a form generated by Forms Designer.

® List the files generated by Forms Designer.

® List the typical files and routines that are modified to extend a form application.

® Use Visual Basic to modify a project created by Forms Designer.

® Recompile and reinstall a modified form.

® Define the purpose of a .CFG file.

Chapter 6: Extending Forms: Advanced Modifications

Topics

® Accessing databases

® Adding controls

® Creating a routing form

® Adding a file

Lab

Extending forms: Advanced modifications

Skills

Students will be able to:

® Extend a form to read and write database information.

® Add new controls to a form.

® Add routing features to a form.

® Specify files, such as a Help file, to install with your form.

Chapter 7: Using OLE Messaging

Topics

® Creating a messaging application programming interface (MAPI) session

® Working with messages

® Working with addresses

® Working with folders

Lab

Using OLE messaging

Skills

Students will be able to:

® Use basic OLE Automation.

® Describe how OLE messaging is used.

® Describe the OLE messaging object model.

® Create a Visual Basic–based application that:

- Uses OLE messaging to send a message with and without a Compose dialog box.
- Uses OLE messaging to read messages.
- Sends and retrieves an attachment in a message.
- Searches for a specific message.
- Adds custom properties to a message.
- Reads custom properties from a message.
- Deletes a message.
- Moves a message from one folder to another folder.
- Uses the personal address book to address a message.
- Sends a message to a public folder.

Chapter 8: Using OLE Scheduling

Topics

® Working with appointments

® Working with projects and tasks

® Working with contacts

® Distributing an application

Lab

Using OLE scheduling

Skills

Students will be able to:

® Describe the OLE scheduling object model.

® Create a Visual Basic–based application that:

- Creates and views an appointment.
- Creates and views a task.
- Adds and views a contact.
- Reads all appointments.

Chapter 9: Integrating Microsoft Exchange and Microsoft Office

Topics

® Posting Office documents

® Sending and routing documents

® Office features

Lab

Integrating Microsoft Exchange and Microsoft Office

Skills

Students will be able to:

® Send, route, and post documents manually or programmatically from within the Microsoft Office programs.

® Create custom document properties.

® Create a linked custom property.

® Group and sort on custom properties.

® Describe how to use Microsoft Word as an e-mail editor.

® Describe how your personal address book can be used in a Word mail merge.

® Describe how project scheduling can be integrated with Schedule+.

Course No. 629

This course is intended for programmers with a basic working knowledge of the C++ programming language and experience developing Microsoft Windows–based or MS-DOS–based applications.

This self-paced CD-ROM–based course teaches experienced programmers how to create sophisticated Microsoft Foundation Class (MFC) Library–based applications designed to solve business problems. You will learn how to use, modify, and extend MFC to create reliable desktop and enterprise solutions.

The course also covers how to create and integrate OLE controls into applications, and create database applications using Open Database Connectivity (ODBC) and Data Access Objects (DAO).

At Course Completion

At the end of the course, students will be able to create user interface constructs; display text and simple graphics device interface (GDI) elements; implement persistence; integrate user-defined classes into the document-view architecture; build dynamic-link libraries (DLLs) that link to MFC; implement database support using ODBC and DAO; create and use OLE controls; add Help to applications; and extend printing and print preview capabilities of applications.

Microsoft Certified Professional Exams

This course helps you prepare for the *Developing Applications with C++ Using the MFC Library* Microsoft Certified Professional exam.

Prerequisites

® Programming experience in C++

® Familiarity with the Windows operating system event-driven programming model

The course materials are in English. To benefit fully from our instruction, students need an understanding of the English language and completion of the prerequisites.

Course Materials and Software

All course materials are included on the Mastering Visual C++ CD-ROM.

System Requirements

® Personal computer with a 486DX or higher processor running the Microsoft Windows 95 operating system or Microsoft Windows NT Workstation operating system version 3.51 or later

® 8 MB of memory (16 MB recommended) for Windows 95; 12 MB of memory (20 MB recommended) for Windows NT Workstation

® 10 MB of available hard-disk space

® MPC2-compatible CD-ROM drive

® Super VGA or higher-resolution video adapter capable of displaying 256 colors or greater

® Microsoft Mouse or compatible pointing device

® MPC2-compatible audio board required for audio and video instruction

® Installation of Microsoft Visual C++ is required to run lab exercises.

Chapter 1: Windows Fundamentals and Architecture

Topics

® Win32-based architecture

® Anatomy of a Windows-based application

® Displaying and using windows

® Event-driven programming

Skills

Students will be able to:

® Describe the overall architecture of Windows-based applications.

® Define the terms *process* and *thread*, and describe their characteristics in the Win32-based environment.

® List the various graphical user interface (GUI) parts of a Windows-based application.

® Explain the function of messages, diagram their flow, and explain the event-driven paradigm.

® Describe important Windows-based programming issues such as activation and focus, and window invalidation and repainting.

® Explain the general structure and features of memory management in the Microsoft Windows 95 and Windows NT operating systems.

Chapter 2: MFC Architecture

Topics

® Classifications of MFC

® Classes in a minimal MFC application

® Document/view architecture

® Non-document/view architecture

Skills

Students will be able to:

® List the benefits provided to the majority of Microsoft Foundation Classes by deriving from **CObject**.

® List the classifications of classes as defined by MFC.

® Define the base classes in a typical MFC application.

® Describe the purpose of and the justification for document/view architecture.

® Describe the benefits and costs of document/view, non-document/view, and dialog-based application architectures.

Chapter 3: Creating a Visual C++ Project

Topics

® Developer Studio

® AppWizard, ClassWizard

® Component Gallery

® Resource editors

® Browser

® Debugging

Labs

® Building an AppWizard application

® Hand-coding a minimal MFC application

® Creating a dialog-based application

Skills

Students will be able to:

® Use the tools in Microsoft Visual C++ development system version 4 (Microsoft Developer Studio).

- Use AppWizard to build a Single Document Interface (SDI) application.
- Use ClassWizard to add a handler.
- Use several of the resource editors.
- Explain the purpose of the Project Workspace window.
- Describe the structure and function of the Component Gallery.

® Access online information, including Visual C++ Help and Books Online.

® Build and run a simple application from within Visual C++.

® Use Browser to display information about symbols, class, and function relationships in the project.

® Describe debugging capabilities in Microsoft Developer Studio.

Chapter 4: Handling Messages

Topics

® Windows messages

® Message map

® Using ClassWizard to manage message handlers

® Using WizardBar to handle messages

® Advanced message handling

Lab

Simple message handling

Skills

Students will be able to:

® Define the term *message* as it applies to the Microsoft Windows operating system.

® List the types of MFC messages.

® Describe how messages are handled by application framework.

® Describe the purpose of a message map and explain how one is declared and implemented.

® Use ClassWizard and WizardBar to add or delete an event's message handler.

® Implement a handler member function (for instance, implement **OnRButtonDown** to respond to a right mouse-button click).

® Explain the purpose of user-defined messages and how they are added to an application.

® Define a system-registered message, and explain the difference between asynchronous messaging and synchronous messaging.

Chapter 5: Graphics

Topics

® Writing output to a device

® CDC class

® Displaying text to the view

® GDI objects

® Graphic output functions

® Transformations

® Special visual effects

Lab

Creating a simple graph

Skills

Students will be able to:

® Explain the general purpose of a device context and the associated Microsoft Foundation Classes.

® Describe painting issues.

® Display text to the view window.

® List and describe standard GDI objects.

® Use stock objects effectively in a program.

® Draw simple graphics objects.

® Describe and use mapping modes to perform transformations between logical and physical drawing space.

® Describe Raster Operation (ROP) codes and give examples of their use.

Chapter 6: Menus, Toolbars, and Status Bars

Topics

® Building menus

® Adding an accelerator key to a menu

® Updating the appearance of menus

® Creating shortcut menus

® Adding toolbars

® Implementing status bars

® Advanced techniques

Labs

® Static drop-down menus

® Changing menu text

® Enabling menu items

® Adding a progress control to the status bar

® Creating a shortcut menu

® Adding a graphic to a status bar

Skills

Students will be able to:

® Add menus, accelerators, status bar menu prompts, and toolbar buttons to an application.

® Explain the differences between a window message and a command message.

® Explain the routing of a command message.

® Dynamically change the state of a menu item.

® Describe Ownerdraw menus.

® Incorporate a context or shortcut menu into an application.

® Add additional panes and graphics to a status bar.

Chapter 7: Creating and Using Dialog Boxes

Topics

® Designing and creating dialog boxes

® Initializing list and combo boxes

® Working with modeless dialog boxes

® Advanced dialog box handling

Labs

® Adding and invoking modal dialog boxes

® Adding and invoking modeless dialog boxes

® Adding property pages and sheets

Skills

Students will be able to:

® Define the various types of dialog boxes.

® Use the dialog box editor to create dialog box templates.

® Use ClassWizard to create dialog box classes.

® Write code to manage dialog data exchange (DDX) and dialog data validation (DDV).

® Use advanced techniques for placing data in controls of dialog boxes.

® Invoke and display modal and modeless dialog boxes.

® Customize common dialog boxes.

® Create tabbed dialog boxes and property sheets.

Chapter 8: Views

Topics

® Working with views

® Creating an application with interrelated views

Labs

® Splitter windows

® Adding a rich edit view to an application

Skills

Students will be able to:

® Describe the purpose of documents, views, templates, and frames within the document/view architecture, and explain how they interact.

® Describe the various types of view classes in MFC.

® Implement applications that use **CScrollView**, **CListView**, **CSplitterWnd**, **CTreeView**, **CEditView**, and **CRichEditView**.

® Use two interrelated views in an application.

Chapter 9: Persistence

Topics

® Registry

® Serialization

® Universal naming convention (UNC) and long filename support

Lab

Adding serialization with version handling to an application

Skills

Students will be able to:

® Explain the purpose of the registry.

® View and modify the registry.

® Programmatically update the registry.

® Define serialization and describe support provided for it in MFC.

® Describe how MFC supports persistent storage.

® Create serializable objects.

® Handle version issues related to serializable objects.

® Describe guidelines for UNC and long filenames.

Chapter 10: User-Defined Classes and the Document/View Architecture

Topics

® Data in document/view architecture

® Data support in MFC

® C++ class templates

® MFC collection classes

® Serializing a collection

Skills

Students will be able to:

® Integrate application-specific (solution domain) classes into document/view architecture.

® Describe the advantages of deriving a class from **CObject**.

® Describe and use templated and nontemplated collection classes (lists, maps, and arrays).

® Create type-safe collections of objects.

® Serialize collections of type **CArray**, **CList**, or **CMap**.

Chapter 11: Dynamic-Link Libraries

Topics

® Dynamic-Link Libraries (DLLs) that use MFC

® DLLs using the static version of MFC

® DLLs using the shared version of MFC

® Extension DLLs

® Run-time linking of regular DLLs

Lab

Creating a DLL from existing classes

Skills

Students will be able to:

® Describe DLLs and list the advantages of their use.

® List the three types of C++ DLLs that link with MFC, and describe their characteristics, advantages, and disadvantages.

® Build and use each of the three types of DLLs that link with MFC:

- A regular DLL that statically links to MFC.
- A regular DLL that links to the shared version of MFC.
- An extension DLL.

® Define run-time linking and describe the advantages of its use.

® Build an application that uses run-time linking with a regular DLL.

Chapter 12: Implementing Database Support Using ODBC and DAO

Topics

® The recordset

® Introduction to database access with MFC

® Open Database Connectivity (ODBC)

® Data Access Objects (DAO)

® Implementing data access with MFC

® Customizing a query

® Using QueryDefs

® Parameterizing a query

® Finding records within a recordset

® Advanced data access with MFC

Labs

® Building a database viewer with DAO

® Building a database editor with DAO

® Building a two-tier database browser with DAO

Skills

Students will be able to:

® Explain the differences between ODBC and DAO.

® Describe the role of ODBC in applications that interact with databases.

® Identify and describe the most important issues that a simple ODBC-compliant application must address.

® Use the ODBC Database Manager and understand its use of the registry.

® Develop a forms-based database browser.

® Use the basic features of the MFC database classes.

® Implement an ODBC connection to a local database.

® Describe and use the basic features of the DAO database classes.

® Use DAO to interact with a Microsoft Access–based database.

Chapter 13: Adding Context-Sensitive Help

Topics

® Help file architecture

® WinHelp function

® Using AppWizard to add Help

® F1 Help

® Context-sensitive Help mode

® Adding context-sensitive Help to an existing MFC application

Skills

Students will be able to:

® Describe the functionality available through WinHelp.

® Explain the MFC messaging associated with F1 and SHIFT+1.

® Explain the structure of a Windows-based Help file.

® Use MFC tools to create an application that implements context-sensitive Help.

Chapter 14: Printing and Print Preview

Topics

® Adding printer support to an MFC application

® Printing process

® Retrieving and setting printing information

® Managing the printing process

® Document versus printer pages

Lab

Adding Print and Print Preview to Textview

Skills

Students will be able to:

® Describe default printing capabilities provided by MFC in an AppWizard-generated application.

® Describe document-oriented versus page-oriented output, and customize screen-directed and printer-directed output.

® Explain how to get printer-specific information at run time and incorporate it into an application.

Chapter 15: Implementing and Using OLE Controls

Topics

® Creating an OLE control with ControlWizard

® Implementing an OLE control container

® OLE control properties

® OLE control methods

® OLE control events

® Communicating errors in OLE controls

® Implementing OLE control property pages

® Implementing per-property browsing

® Data binding in an OLE control

Lab

Building an OLE control from an existing class

Skills

Students will be able to:

® Describe the advantages of OLE control technology.

® Explain the elements of an OLE control.

® Explain the purpose of the ODL file.

® Explain the features of ControlWizard in creating an OLE control.

® Describe the primary tasks of an OLE control container.

® Explain the interaction between an OLE control container and an OLE control.

® Use AppWizard to create an OLE control container and use the Component Gallery to incorporate an OLE control into the application.

® Use ControlWizard to create skeletal code for your OLE control.

® Use ClassWizard to add both stock and custom properties, and methods to an OLE control.

® Use ClassWizard to define stock and custom events that the OLE control will use to communicate with its container.

Course No. 625

This course is intended for experienced database developers who create and distribute applications.

Microsoft Mastering Series titles are available through major bookstores, software resellers, and through Microsoft Authorized Technical Education Centers (ATECs) For a referral to an ATEC in your area, call (800) SOLPROV.

Delivered on CD-ROM, this course is the first in a series supporting developers who create mission-critical applications. Focusing on the newest release of the Microsoft Visual FoxPro database management system (DBMS), this course offers in-depth interactive training for experienced developers.

At Course Completion

At the end of the course, students will be able to utilize the full functionality of object-oriented programming; use the core set of commands and functions in object-oriented programming; apply the hierarchy of the classing structure; design and create additional functionality for forms; add functionality to the grid; consider multiuser issues during the application development process; manage and control data in multiuser situations; use client/server support to work with remote data; use OLE objects with Visual FoxPro version 3.0 and Microsoft Office products; and apply basic conceptual architecture when creating wizards.

Microsoft Certified Professional Exams

This course helps you prepare for the Microsoft Certified Trainer exam for Visual FoxPro.

Prerequisites

® Experience working with the Microsoft Windows operating system

® Knowledge of relational database design concepts

® Fundamental programming skills

® The ability to apply the fundamentals described in the Visual FoxPro User's Guide

The course materials, lectures, and lab exercises are in English. To benefit fully from our instruction, students need an understanding of the English language and completion of the prerequisites.

Course Materials and Software

This course is licensed for use by a single user on a single computer. Multiuser usage is prohibited. All content on the CD-ROM can be printed.

Module 1: Project Manager

Topics

® Overview

® Creating applications

® Converting FoxPro 2.*x* project files

® Converting catalog files

Skills

Students will be able to:

Use Project Manager to build new applications, manage project files, and convert existing project or catalog files.

Module 2: Wizards and Builders

Topics

® Overview

® Wizards

® Builders

® Architecture

Skills

Students will be able to:

® Use Wizards to accomplish tasks.

® Use Builders to add controls to forms.

® Understand architecture of wizards and builders.

Module 3: Form Designer

Topics

® Overview

® Designing forms

® Form controls

® Data environment

® Data sessions

® Form sets

® Setting properties

® Working with the grid

® Page frames

® OLE Controls

Labs

® Working with grid positioning

® Manipulating the grid

® Adding a control programmatically

® Adding a control visually

® Using the sparse property

® Calling a method programmatically

® Changing grid color and bringing up a browse

® Changing column colors and reset

® Changing the color of the active cell

® Common tasks

® Creating a form set that updates and skips through records

® Creating a one-to-many form

® Data environment

® Drag and Drop

® Grid cells with different colors

® Multiple sessions

® OLE Controls

® Page frames

® User-defined controls

® Working with a grid through code

Skills

Students will be able to:

® Use Form Designer and available controls.

® Use Data Environment Designer.

® Create form sets.

® Set properties.

® Manage controls and user interfaces with page frames.

® Display and manipulate data with the Grid tool.

® Understand the benefits of OLE Controls.

Module 4: Query Designer

Topics

® Overview

® Query Designer

Skills

Students will be able to:

® Use Query Designer to extract records.

® Specify criteria to extract records.

Module 5: Report Designer

Topics

® Overview

® Accessing Report Designer

® User interface

® Ease-of-use features

Skills

Students will be able to:

Use Report Designer to summarize, format, and print data.

Module 6: Error Management

Topics

® Overview

® Types of errors

® Debugging tools

® Error handling

Labs

® Checking values in the debug window

® Creating an error-handling routine

Skills

Students will be able to:

® Use command, debug, and trace windows to resolve errors.

® Use an error-handling routine to resolve run-time errors.

Module 7: Object Model

Topics

® Overview

® What is object-oriented technology?

® Terminology

® Properties, events, and methods

® Scoping of variables

® Object-oriented language

Labs

® Using the (::) scope resolution operator

® ParentClass property

® Creating a form programmatically

® Using the DEFINE CLASS command to create a subclass

® Adding controls to a form using the AddObject() method

® Creating a custom function

Skills

Students will be able to:

® Understand benefits of object-oriented programming.

® Understand hierarchical structure and related terminology.

® Define properties.

® Scope variables.

® Work with the Visual FoxPro language.

Module 8: Visual Classes

Topics

® Overview

® What are classes?

® Types of classes

® Creating classes and subclasses

® Class browser

Labs

® Creating a form class

® Creating and registering a button class

® Subclassing from an existing class

® Modifying an existing class
® Creating a container class
® Creating a control class
® Creating a custom class
® Toolbar class

Skills

Students will be able to:

® Understand benefits of classes.
® Understand types and hierarchies of classes.
® Create classes and subclasses.
® Use the Class browser.

Module 9: OLE Automation

Topics

® Overview
® What is OLE?
® OLE features
® Using OLE Automation objects
® Controlling Microsoft Excel from Visual FoxPro
® Controlling Microsoft Word from Visual FoxPro

Skills

Students will be able to:

® Understand benefits of OLE.
® Understand how Visual FoxPro stores and converts OLE objects.
® Understand the OLE features in Visual FoxPro.
® Use OLE Automation to create and manipulate objects.
® Control Microsoft Excel and Word.

Module 10: Shared Access to Data

Topics

® Overview
® Controlling multiuser access
® Buffering data
® Managing data and conflicts
® Transaction support

Skills

Students will be able to:

® Control access to data.
® Buffer data.
® Manage conflicts.

® Use transactions to maintain data integrity.

Module 11: Client/Server

Topics

® Overview

® Establishing a connection

® SQL pass-through connections

® What is a view?

® Local and remote views

® Fetching data

® Updating, deleting, and inserting data on a remote server

Skills

Students will be able to:

® Connect to a remote server using SQL pass-through connections.

® Retrieve an updatable result set using View Designer.

® Explain local and remote views.

® Fetch data.

® Manipulate and update data on a remote server.

® Use Upsizing Wizard.

Module 12: Dynamic-Link Libraries (DLLs)

Topics

® Overview

® What DLL procedures can I call?

® Using DLLs

Labs

® Nulls

® Passing arguments by value or by reference

® Windows directory (listed under Labs for the Table of Contents but not in the course chapter)

® Using an alias for application programming interface (API) functions

® Handles

® Hiding the main FoxPro window

® Using a form to hide the main FoxPro window

Skills

Students will be able to:

® Use DLLs.

® Know sources of DLLs.

Microsoft Official Curriculum courses are available through Microsoft authorized training sites. Microsoft Solution Provider Authorized Technical Education Centers (ATECs) are commercial training organizations offering Microsoft Official Curriculum over consecutive days, and Microsoft Authorized Academic Training Program (AATP) institutions offer Microsoft Official Curriculum over an academic term. These professional education centers deliver consistent, high-level, hands-on technical training on the full range of Microsoft productivity, networking, operating system, and application development products. All Solution Provider ATECs and AATP sites have met stringent guidelines to receive their Microsoft authorization. These guidelines govern curriculum, class sizes, training facilities, and, most importantly, the educational and technical expertise of the Microsoft Certified Trainers. This means you receive the highest level of curriculum and education services available in the industry. For example:

- ® Only Microsoft authorized training sites may use Microsoft Certified Trainers to deliver Microsoft Official Curriculum.
- ® Microsoft Certified Trainers undergo rigorous training and testing by Microsoft to become certified to teach Microsoft Official Curriculum. They are experts in providing training on Microsoft operating systems, the Microsoft BackOffice family of server software, and Microsoft desktop and development tools.
- ® Solution Provider ATECs and AATPs receive early releases of Microsoft software and course materials so they can begin classroom instruction as soon as Microsoft products are released to the public.

For full course descriptions and a referral to a Microsoft Solution Provider Authorized Technical Education Center, call (800) SOLPROV in the United States and Canada. In other countries, contact your local Microsoft office.

Ask for the Roadmap to Microsoft Education and Certification, an online windows-based information tool. Or see E&CMAP.ZIP from Library 5 of the Solution Provider forum on CompuServe (GO MSEDCERT).

Check out the details on Microsoft's Certified Professional Programs.

Click here to connect to the Microsoft Online Institute campus on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Microsoft Online Institute (MOLI) is an online learning environment that brings in-depth Microsoft technical training directly to your desktop. Utilizing the latest in Internet technology, the Microsoft Online Institute pairs self-study learning materials with highly-qualified instructors, providing an effective, flexible, and low-cost training solution for busy professionals.

The Microsoft Online Institute offers a growing list of classes to help you gain the latest knowledge and skills on Microsoft products and technologies, and to help you pursue your Microsoft Certified Professional goals.

Best of all, you can progress at your own pace, on your own schedule. Available 24 hours a day, 7 days a week, the Microsoft Online Institute gives you the training you need when and where you need it.

And when you need help, your instructor will be available online. Take advantage of instructors with "real world" experience who will put a practical spin on course material. Send your questions via e-mail and get a personal reply. Join scheduled real-time chat sessions with your instructor and other students.

By attending a virtual classroom on the Web, you stay in touch with a real instructor and keep motivated to move along with the course materials.

Click here to connect to find out about User Groups on the Mindshare page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

No matter how much—or little—you know about computers, a user group can help you get more out of Microsoft software. Computer user groups can take any form, from an informal group of 10 individuals meeting over a laptop and take-out food, to a nonprofit corporation with a membership the size of a small town, a board of directors, and a telephone directory-sized newsletter. User groups can serve whole communities, educational institutions, corporations, professions, or associations. They can center around a single software program or an operating platform. They can be exclusive to in-house information system managers or more general to, for example, home users who want to have more fun with their computers. Or, they can be so large as to encompass all of these things at once.

Whatever their size or form, user groups share in common the commitment to providing a platform for sharing experience, insight, and knowledge about computing, for the personal and professional enrichment of their members.

This appendix presents a set of suggested coding conventions for Visual Basic programs.

Coding conventions are programming guidelines that focus not on the logic of the program but on its physical structure and appearance. They make the code easier to read, understand, and maintain. Coding conventions can include:

® Naming conventions for objects, variables, and procedures.

® Standardized formats for labeling and commenting code.

® Guidelines for spacing, formatting, and indenting.

In the sections that follow, each of these areas are discussed, along with examples of good usage.

® Why Coding Conventions?

® Object Naming Conventions

® Constant and Variable Naming Conventions

® Structured Coding Conventions

The main reason for using a consistent set of coding conventions is to standardize the structure and coding style of an application so that you and others can easily read and understand the code.

Good coding conventions result in precise, readable, and unambiguous source code that is consistent with other language conventions and as intuitive as possible.

Minimal Coding Conventions

A general-purpose set of coding conventions should define the minimal requirements necessary to accomplish the purposes discussed above, leaving the programmer free to create the program's logic and functional flow.

The object is to make the program easy to read and understand without cramping the programmer's natural creativity with excessive constraints and arbitrary restrictions.

To this end, the conventions suggested in this appendix are brief and suggestive. They do not list every possible object or control, nor do they specify every type of informational comment that could be valuable. Depending on your project and your organization's specific needs, you may wish to extend these guidelines to include additional elements, such as:

® Conventions for specific objects and components developed in-house or purchased from third-party vendors.

® Variables that describe your organization's business activities or facilities.

® Any other elements that your project or enterprise considers important for clarity and readability.

Objects should be named with a consistent prefix that makes it easy to identify the type of object. Recommended conventions for some of the objects supported by Visual Basic are listed below.

® Suggested Prefixes for Controls

® Suggested Prefixes for Data Access Objects (DAO)

® Suggested Prefixes for Menus

® Choosing Prefixes for Other Controls

Control type	Prefix	Example
3D Panel	pnl	pnlGroup
Animated button	ani	aniMailBox
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Communications	com	comFax
Control (used within procedures when the specific type is unknown)	ctr	ctrCurrent
Data control	dat	datBiblio
Data-bound combo box	dbcbo	dbcboLanguage
Data-bound grid	dbgrd	dbgrdQueryResult
Data-bound list box	dblst	dblstJobType
Directory list box	dir	dirSource
Drive list box	drv	drvTarget
File list box	fil	filSource
Form	frm	frmEntry
Frame	fra	fraLanguage
Gauge	gau	gauStatus
Graph	gra	graRevenue
Grid	grd	grdPrices
Horizontal scroll bar	hsb	hsbVolume
Image	img	imgIcon
Key status	key	keyCaps
Label	lbl	lblHelpMessage
Line	lin	linVertical
List box	lst	lstPolicyCodes
MAPI message	mpm	mpmSentMessage
MAPI session	mps	mpsSession
MCI	mci	mciVideo
MDI child form	mdi	mdiNote
Menu	mnu	mnuFileOpen
MS Flex grid	msg	msgClients
MS Tab	mst	mstFirst
OLE	ole	oleWorksheet
Outline	out	outOrgChart
Pen BEdit	bed	bedFirstName
Pen HEdit	hed	hedSignature
Pen ink	ink	inkMap
Picture	pic	picVGA
Picture clip	clp	clpToolbar
Report	rpt	rptQtr1Earnings

Shape	shp	shpCircle
Spin	spn	spnPages
Text box	txt	txtLastName
Timer	tmr	tmrAlarm
UpDown	upd	updDirection
Vertical scroll bar	vsb	vsbRate
Slider	sld	sldScale
ImageList	ils	ilsAllIcons
TreeView	tre	treOrganization
Toolbar	tlb	tlbActions
StatusBar	sta	staDateTime
ListView	lvw	lvwHeadings
ProgressBar	prg	prgLoadFile
RichTextBox	rtf	rtfReport

Use the following prefixes to indicate Data Access Objects.

Database object	Prefix	Example
Container	con	conReports
Database	db	dbAccounts
DBEngine	dbe	dbeJet
Document	doc	docSalesReport
Field	fld	fldAddress
Group	grp	grpFinance
Index	idx	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	rec	recForecast
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
User	usr	usrNew
Workspace	wsp	wspMine

Some examples:

```
Dim dbBiblio As Database
Dim recPubsInNY As Recordset, strSQLStmt As String
Const DB_READONLY = 4                ' Set constant.
'Open database.
Set dbBiblio = OpenDatabase("BIBLIO.MDB")
' Set text for the SQL statement.
strSQLStmt = "SELECT * FROM Publishers WHERE _
    State = 'NY'"
' Create the new Recordset object.
Set recPubsInNY = db.OpenRecordset(strSQLStmt, _
    dbReadOnly)
```

Applications frequently use many menu controls, making it useful to have a unique set of naming conventions for these controls. Menu control prefixes should be extended beyond the initial "mnu" label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string. The following table lists some examples.

Menu caption sequence	Menu handler name
File Open	mnuFileOpen
File Send Email	mnuFileSendEmail
File Send Fax	mnuFileSendFax
Format Character	mnuFormatCharacter
Help Contents	mnuHelpContents

When this naming convention is used, all members of a particular menu group are listed next to each other in Visual Basic's Properties window. In addition, the menu control names clearly document the menu items to which they are attached.

For controls not listed above, you should try to standardize on a unique two or three character prefix for consistency. Use more than three characters only if needed for clarity.

For derived or modified controls, for example, extend the prefixes above so that there is no confusion over which control is really being used. For third-party controls, a lower-case abbreviation for the manufacturer could be added to the prefix. For example, a control instance created from the Visual Basic Professional 3D frame could uses a prefix of fra3d to avoid confusion over which control is really being used.

In addition to objects, constants and variables also require well-formed naming conventions. This section lists recommended conventions for constants and variables supported by Visual Basic. It also discusses the issues of identifying data type and scope.

Variables should always be defined with the smallest scope possible. Global variables can create enormously complex state machines and make the logic of an application extremely difficult to understand. Global variables also make the reuse and maintenance of your code much more difficult.

Variables in Visual Basic can have the following scope:

Scope	Declared in	Visible in
Procedure-level	Event procedure, sub, or function	The procedure in which it is declared
Form-level, Module-level	Declarations section of a form or code module (.frm, .bas)	Every procedure in the form or code module
Global	Declarations section of a code module (.bas, using Global keyword)	Everywhere in the application

In a Visual Basic application, global variables should be used only when there is no other convenient way to share data between forms. When global variables must be used, it is good practice to declare them all in a single module, grouped by function. Give the module a meaningful name that indicates its purpose, such as Global.bas.

It is good coding practice to write modular code whenever possible. For example, if your application displays a dialog box, put all the controls and code required to perform the dialog's task in a single form. This helps to keep the application's code organized into useful components and minimizes its run-time overhead.

With the exception of global variables (which should not be passed), procedures and functions should operate only on objects passed to them. Global variables that are used in routines should be identified in a general comment area at the beginning of the routine. In addition, you should pass arguments to subs and functions using ByVal, unless you explicitly need to change the value of the passed argument.

Variable Scope Prefixes

As project size grows, so does the value of recognizing variable scope quickly. A one-letter scope prefix preceding the type prefix provides this, without greatly increasing the size of variable names.

Scope	Prefix	Example
Global	g	gstrUserName
Module-level, form-level	m	mblnCalcInProgress
Local to procedure	None	dblVelocity

A variable has **global** scope if it is declared Public in a standard module or a form module. A variable has *module-level* or *form-level* scope if declared Private in a standard module or form module, respectively.

Note Consistency is crucial to productive use of this technique; the syntax checker in Visual Basic will not catch module-level variables that begin with "g."

Constants

The body of constant names should be mixed case with capitals initiating each word. Although standard Visual Basic constants do not include data type and scope information, prefixes like i, s, g, and m can be very useful in understanding the value and scope of a constant. For constant names, follow the same rules as variables. For example:

```
mnUserListMax        'Max entry limit for User list
                         '(integer value,local to module)
gsNewLine                'New Line character
                         '(string, global to application)
```

Variables

Declaring all variables saves programming time by reducing the number of bugs caused by typos (for example, aUserNameTmp vs. sUserNameTmp vs. sUserNameTemp). On the Editor tab of the Options dialog, check the Require Variable Declaration option. The Option Explicit statement requires that you declare all the variables in your Visual Basic program.

Variables should be prefixed to indicate their data type. Optionally, especially for large programs, the prefix can be extended to indicate the scope of the variable.

Variable Data Types

Use the following prefixes to indicate a variable's data type.

Data type	Prefix	Example
Boolean	bln	blnFound
Byte	byt	bytRasterData
Collection object	col	colWidgets
Currency	cur	curRevenue
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFName
User-defined type	udt	udtEmployee
Variant	vnt	vntCheckSum

Descriptive Variable and Routine Names

The body of a variable or routine name should use mixed case and should be as long as necessary to describe its purpose. In addition, function names should begin with a verb, such as InitNameArray or CloseDialog.

For frequently used or long terms, standard abbreviations are recommended to help keep name lengths reasonable. In general, variable names greater than 32 characters can be difficult to read on VGA displays.

When using abbreviations, make sure they are consistent throughout the entire application. Randomly switching between Cnt and Count within a project will lead to unnecessary confusion.

User-Defined Types

In a large project with many user-defined types, it is often useful to give each such type a three-character prefix of its own. If these prefixes begin with "u," they will still be easy to recognize quickly when you are working with a user-defined type. For example, "ucli" could be used as the prefix for variables of a user-defined Client type.

In addition to naming conventions, structured coding conventions, such as code commenting and consistent indenting, can greatly improve code readability.

Code Commenting Conventions

All procedures and functions should begin with a brief comment describing the functional characteristics of the routine (what it does). This description should not describe the implementation details (how it does it) because these often change over time, resulting in unnecessary comment maintenance work, or worse yet, erroneous comments. The code itself and any necessary inline comments will describe the implementation.

Arguments passed to a routine should be described when their functions are not obvious and when the routine expects the arguments to be in a specific range. Function return values and global variables that are changed by the routine, especially through reference arguments, must also be described at the beginning of each routine.

Routine header comment blocks should include the following section headings.

Section heading	Comment description
Purpose	What the routine does (not how).
Assumptions	List of each external variable, control, open file, or other element that is not obvious.
Effects	List of each affected external variable, control, or file and the effect it has (only if this is not obvious).
Inputs	Each argument that may not be obvious. Arguments are on a separate line with inline comments.
Returns	Explanation of the values returned by functions.

Remember the following points:

- Every important variable declaration should include an inline comment describing the use of the variable being declared.
- Variables, controls, and routines should be named clearly enough that inline commenting is only needed for complex implementation details.
- At the start of the .bas module that contains the project's Visual Basic generic constant declarations, you should include an overview that describes the application, enumerating primary data objects, routines, algorithms, dialogs, databases, and system dependencies. Sometimes a piece of pseudocode describing the algorithm can be helpful.

Formatting Your Code

Because many programmers still use VGA displays, screen space should be conserved as much as possible while still allowing code formatting to reflect logic structure and nesting. Here are a few pointers:

Standard, tab-based, nested blocks should be indented four spaces (the default).

The functional overview comment of a routine should be indented one space. The highest level statements that follow the overview comment should be indented one tab, with each nested block indented an additional tab. For example:

```
'*****************************************************
' Purpose:  Locates the first occurrence of a
'               specified user in the UserList array.
' Inputs:
'   strUserList():  the list of users to be searched.
'   strTargetUser:  the name of the user to search for.
' Returns:  The index of the first occurrence of the
'               rsTargetUser in the rasUserList array.
'               If target user is not found, return -1.
'*****************************************************
```

```
Function intFindUser (strUserList() As String, strTargetUser As _
    String)As Integer
    Dim i As Integer            ' Loop counter.
    Dim blnFound As Integer     ' Target found flag.
    intFindUser = -1
    i = 0
    While i <= Ubound(strUserList) and Not blnFound
        If strUserList(i) = strTargetUser Then
            blnFound = True
            intFindUser = i
        End If
    Wend
End Function
```

Grouping Constants

Variables and defined constants should be grouped by function rather than split into isolated areas or special files. Visual Basic generic constants should be grouped in a single module to separate them from application-specific declarations.

& and + Operators

Always use the & operator when linking strings and the + operator when working with numerical values. Using the + operator to concatenate may cause problems when operating on two variants. For example:

```
vntVar1 = "10.01"
vntVar2 = 11
vntResult = vntVar1 + vntVar2    'vntResult = 21.01
vntResult = vntVar1 & vntVar2    'vntResult = 10.0111
```

Creating Strings for MsgBox, InputBox, and SQL Queries

When creating a long string, use the underscore line continuation character to create multiple lines of code so that you can read or debug the string easily. This technique is particularly useful when displaying a message box (MsgBox) or input box (InputBox) or when creating an SQL string. For example:

```
Dim Msg As String
Msg = "This is a paragraph that will be " _
& "in a message box. The text is" _
& " broken into several lines of code" _
& " in the source code, making it easier" _
& " for the programmer to read and debug."
MsgBox Msg

Dim QRY As String
QRY = "SELECT *" _
& " FROM Titles" _
& " WHERE [Year Published] > 1988"
TitlesQry.SQL = QRY
```

Course No. 677

This course is intended for content developers who are responsible for creating Web pages, and programmers who want to make their existing Visual Basic–based and Visual C++–based applications Internet-aware.

This course will teach developers how to publish content on the Internet and create applications that connect with the Internet by focusing on authoring active Web content and developing Internet-aware applications.

At Course Completion

At the end of the course, students will be able to install an Internet server with Microsoft Internet Information Server, and author a Hypertext Markup Language (HTML) page (create static content) using Internet Assistant for Microsoft Word for Windows 95. They will also be able to add code to an HTML page (create active content) using Microsoft ActiveX controls and the Visual Basic programming system, Scripting Edition, and use the Microsoft Internet Explorer object model from a Visual Basic–based or Visual C++–based application. In addition, students will be able to read and write information to an Open Database Connectivity (ODBC) database from a Web page using the Internet server application programming interface (ISAPI) extension provided with Internet Information Server, and call a method of an OLE Automation server from a Web page using the ISAPI extension. Finally, students will be able to write their own ISAPI server extensions and filters using the Visual C++ development system.

Microsoft Certified Professional Exams

This course helps you prepare for the following Microsoft Certified Professional exams:

® To be determined

Prerequisites

This course assumes basic experience installing the Microsoft Windows NT Server network operating system, using the Visual Basic or Visual C++ to create applications for Windows, and using Word to create a document. This course does not assume any prior experience or knowledge of the Internet.

Potential students should be able to accomplish the following tasks before taking this training:

® Install Windows NT Workstation and Windows NT Server.

® Set up security for Windows NT Server.

® Install Transmission Control Protocol/ Internet Protocol (TCP/IP).

® Set up a system ODBC data source.

Content Developers:

® Compose a new document based on a template.

® Open a document of one type and save it in a different format.

® Create a table in a document.

® Insert a graphic into a document.

Programmers:

® Use Visual Basic to add controls to a form, place code in the appropriate events, and create an executable.

or-

® Use Visual C++ and wizards to create a Single Document Interface (SDI) application.

® Add ActiveX controls to a Visual Basic–based or Visual C++–based application.

The course materials are in English. To benefit fully from the course, students need an understanding of the English language and completion of the prerequisites.

Course Materials and Software

This course is licensed for use by a single user on a single computer. Multiuser usage is prohibited. All content on the CD-ROM can be printed.

Module 1: Architecture of the Internet

Topics

® History

® Architecture of the Internet

® Intranet versus Internet (including limitations of intranet structure)

® Using person-to-resource and person-to-person services on the Internet

® Using Uniform Resource Locators (URLs)

® Introduction to the World Wide Web

Labs

® Exploring Internet services

® Using URLs on the Web

Skills

Students will be able to:

® Describe why protocols are important for communicating on the Internet.

® Describe the function of the Internet backbone.

® Define a URL.

® Discriminate between the Internet and an intranet.

® Name three Internet services.

® Define the role of an Internet service provider.

® Describe the purpose of a “newsgroup.”

® Describe the purpose of HTML.

® Discriminate between an active and a static Web page.

® Use FTP to download files.

® Search the Web for a given topic.

® Add a URL to a Windows-based application.

Module 2: Installing an Internet Server

Topics

® Overview

® Microsoft Internet Information Server

® Extending the server

Labs

® Setting up Microsoft Internet Information Server

® Defining security for your Web server

Skills

Students will be able to:

® Explain the reasons for setting up an Internet server.

® Install Microsoft Internet Information Server.

® Configure Web services on Internet Information Server with Information Server Manager.

® Add security to an Internet server.

Module 3: Authoring a Static Web Page

Topics

® Designing and managing a Web site

® Introduction to HTML

® Creating an HTML page with Internet Assistant for Word

® Converting an existing document to HTML format

® Adding advanced features such as frames, image maps, and server-side includes to an HTML page

Labs

® Creating an HTML page

® Converting an existing document to HTML format

® Adding links

® Using frames, image maps, and server-side includes

Skills

Students will be able to:

® List the elements that make up the basic structure of an HTML file.

® Compare and contrast the Microsoft authoring tools that are available and select the appropriate tool for a given task.

® Use Internet Assistant for Microsoft Word, Microsoft Excel, or Microsoft Access to convert an existing document to HTML format.

® Using Internet Assistant for Word, create a new static HTML page that incorporates formatted text, links, graphics, and other special effects.

® Add Microsoft Internet Explorer–specific tags to an HTML page to expose features not supported by Internet Assistant for Word.

® Explain why server-side includes are useful.

® Use a server-side include, image maps, and frames on a Web page.

Module 4: Adding Controls and Other Objects to a Web Page

Topics

® Overview

® Adding standard controls with Internet Assistant for Word

® Adding ActiveX controls and Java “applets”

Labs

® Adding standard HTML controls

® Adding ActiveX controls

Skills

Students will be able to:

® Explain the difference between a static and an active Web page, and explain the role of controls in creating active Web content.

® Explain the purpose of an HTML form in an HTML page.

® Add a standard control to an HTML form.

® Add an ActiveX control to an HTML page.

Module 5: Adding Client-side Scripts with Visual Basic, Scripting Edition

Topics

® Overview of client-side scripting

® What is Visual Basic, Scripting Edition?

® Controlling controls with Visual Basic, Scripting Edition

® Using the Microsoft Internet Explorer object model for scripting

Labs

® Writing control event procedures

® Communicating with an ActiveX control on the client

® Writing validation code

Skills

Students will be able to:

® Create an event procedure for a standard control using Visual Basic, Scripting Edition.

® Create an event procedure for an ActiveX Control using Visual Basic, Scripting Edition.

® Control Microsoft Internet Explorer from a Web page.

® Add page initialization code to a Web page with Visual Basic, Scripting Edition.

Module 6: Communicating with the Server

Topics

® Overview

® Linking to an application on the server

® Sending information to the server with a form

® Using server-extension sample applications

® Calling an OLE Automation server from a Web page

Labs

® Using the FormDump server extension

® Using the ReDir server extension

® Accessing an OLE Automation server

Skills

Students will be able to:

® Explain the process by which a Web page communicates and interacts with the Internet server.

® Create an HTML page that is able to send information from a form to the server.

® Explain the role of the OLE Automation server in creating active content on the Web.

® Create an HTML page that can communicate with an OLE Automation server located on the Internet server.

Module 7: Enabling Data Access Using ODBC

Topics

® Overview of the Internet Database Connector server extension

® Constructing a simple query

® Constructing a query with parameters

® Writing information from the user into a database on the server

Labs

® Reading information from an ODBC database

® Passing parameters to the database query using a URL

® Passing parameters to the database query using a form

® Returning links from the database

® Writing information into the ODBC database

Skills

Students will be able to:

® Explain the role of the ODBC database in creating active content on the Web.

® Create an HTML page that can read and write information to an ODBC data source located on the server.

Module 8: Automating Microsoft Internet Explorer

Topics

® Applications and the Internet

® Microsoft Internet Explorer 3.0 object model

® Creating an instance of Microsoft Internet Explorer from a Visual Basic–based application

® Using the Microsoft Internet Explorer control from a Visual Basic–based application

® Working with the Microsoft Internet Explorer objects

Labs

® Automating Microsoft Internet Explorer

® Using the WebBrowser control

® Downloading a file

Skills

Students will be able to:

® Discuss the implications of creating an Internet-aware application, and list some features typically found in this type of application.

® Using Visual Basic, create an application that views HTML pages on the Web.

Module 9: Mail-enabling an Application

Topics

® Overview of mail technologies

® Adding simple messaging application programming interface (Simple MAPI) to a Visual Basic–based or Visual C++–based application

® Adding Extended MAPI to an application using OLE messaging

® Working with the OLE messaging objects

Lab

® Using MAPI to mail-enable an application

Skill

Students will be able to:

® Create a Visual Basic–based or Visual C++–based application that sends and receives mail through MAPI.

Module 10: Developing Server-side Extensions

Topics

® Overview

® Building an ISAPI filter

® Building an ISAPI application

Labs

® Creating an ISAPI filter

® Creating an ISAPI application (dynamic-link library)

Skills

Students will be able to:

® Describe the role of server-side Internet components.

® Explain how server-side preprocessing works.

® Explain when ISAPI filters are called.

® Create a simple ISAPI filter.

® Create an ISAPI application (DLL).

With Visual Basic, you can create components that range from code libraries to Automation-enabled applications. You can create and distribute ActiveX control packages with full licensing capabilities. You can also create Internet applications with ActiveX documents that display themselves in Internet browsers.

ActiveX technology lets you assemble reusable components into applications and services.

This section defines the three major types of ActiveX components, and reviews the advantages of using ActiveX technology.

This section includes the following topics:

® Introduction to ActiveX Components

® Advantages of Using ActiveX Code Components

An ActiveX component is a unit of executable code, such as an .exe, .dll, or .ocx file, that follows the ActiveX specification for providing objects. An ActiveX component exposes objects that can be used by other applications.

There are three types of ActiveX components you can create with Visual Basic: ActiveX controls, ActiveX documents, and ActiveX code components.

ActiveX Controls

ActiveX controls (formerly known as OLE controls) are standard user-interface elements that allow you to rapidly assemble reusable forms and dialog boxes.

For a description of how to write ActiveX controls, see Chapter 8: Creating ActiveX Controls.

ActiveX Documents

ActiveX documents (formerly known as document objects) are ActiveX components that must be hosted and activated within a client application. ActiveX document technology is an extension to OLE documents. It expands the functionality of visual editing of embedded objects, enabling generic shell applications, such as Internet Explorer, to host different types of documents.

For information about how to develop ActiveX documents, see Chapter 10: Creating and Using ActiveX Documents.

ActiveX Code Components

Code components (formerly known as OLE servers) are libraries of objects. Client applications use code components by creating objects from classes provided by the component. Clients call the properties, methods, and events provided by the object.

Any application that supports standard Automation can use an ActiveX code component created in Visual Basic. This includes Visual Basic, Microsoft Excel, Microsoft Access, Microsoft Project, Microsoft Visual FoxPro, and Microsoft Visual C++.

Visual Basic handles much of the complexity of creating an ActiveX code component, such as creating a type library and registering the component automatically.

Code components use events to notify clients that an action has occurred. To establish this relationship, the client must specify that the component provide information when a specific action occurs.

For example, a client may want to be notified when a database value has changed or when a message has arrived. The component can then send a notification that the action has occurred.

This section describes how to work with events in a code component.

This section includes the following topics:

® Adding Events to a Class

® Handling Events in a Client

® Making Asynchronous Calls with Events

® Canceling a Procedure Within an Event

You use these following steps to raise an event from a code component.

1. Declare the event.
2. Raise the event.

Declaring an Event

You declare an event in the Declarations section of a class module by using the **Event** keyword. This makes the event visible to clients that use your component, as shown in the following code.

```
Public Event Status(ByVal StatusText As String)
```

Raising an Event

When you want an event to be raised to the client application, you call the **RaiseEvent** statement, and pass the event name and any arguments that the event takes.

The following code raises the **Status** event.

```
Public Sub SubmitOrder()
    'Method to simulate processing an order.

    RaiseEvent Status("Checking credit...")
    'Simulate credit check delay.
    EndTime = Timer + 2
    Do While Timer < EndTime
        DoEvents
    Loop

    RaiseEvent Status("Processing Order...")
    'Simulate processing.
    EndTime = Timer + 2
    Do While Timer < EndTime
        DoEvents
    Loop
End Sub
```

You can use this approach to pass status information back to a client application during a **Long** method.

When you want the objects of a code component to communicate with each other, but you do not want them to make them available to client applications, you can use the **Friend** keyword to declare a method so its scope will be available only to the other objects in the component.

The **Public** methods in a class can be called by other objects, but they can also be called by clients. The **Private** methods cannot be called from outside a component, but neither are they visible to other objects within the component.

Methods declared with the **Friend** keyword are visible to other objects in your component, but are not visible to clients because **Friend** methods are not added to the type library or to the public interface.
{ewc mvimg, mvimage,!tip.bmp}

In the following code, the **Hello** method in Class1 is called by the procedure in Class2.

```
'Code in Class1
'can be called from other class modules but not
'from outside the project.
Friend Sub Hello()
    MsgBox "Hello"
End Sub

'Code in Class2.
Public Sub Hi()
    Dim x as New Class1
    x.Hello
End Sub
```

Microsoft Visual Basic offers two scenarios for testing and debugging components: one for in-process components and one for out-of-process components. In the first scenario, you test a component within a single instance of Visual Basic. In the second scenario, you must run two copies of the development environment.

This section describes how to test an ActiveX DLL and an ActiveX EXE, and explains the advantages of setting a reference to a type library. It also explains how to raise errors to a client application to provide the client with error information.

This section includes the following topics:

® Setting Up a Test Project

® Setting a Reference to a Type Library

® Debugging a Component

® Error Handling Styles

® Raising Run-Time Errors

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

In this chapter, you will learn to create ActiveX controls (formerly known as OLE controls). Any ActiveX controls created in Visual Basic can be used in an ActiveX client application, such as another Visual Basic application, Microsoft Office, and Internet Explorer.

Objectives

By the end of this chapter, you will be able to:

® Describe the benefits of using ActiveX controls.

® Describe how an ActiveX control differs from an ActiveX Automation server.

® Create an ActiveX control.

® Test and debug an ActiveX control.

® Expose properties, methods, and events of an ActiveX control.

® Create and enable the property page for a control.

® Enable the data-binding capabilities of an ActiveX control.

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

The Component Object Model (COM) provides support for ActiveX documents (formerly known as OLE document objects. An ActiveX document is a specific type of ActiveX object that can be placed and activated within ActiveX document containers, such as Microsoft Internet Explorer.

ActiveX documents are an extension of compound document technology (object linking and embedding). However, ActiveX documents include new capabilities that let you extend your applications to take advantage of the Internet and provide users with increased user interface functionality.

This chapter describes the features of Visual Basic that let you create and use ActiveX documents.

Objectives

At the end of this chapter, you will be able to:

® Describe how ActiveX documents differ from embedded objects, and when to use them.

® Describe the function of an ActiveX document container.

® Create an ActiveX project with one or more **UserDocument** objects.

® Use the **Hyperlink** object to gain access to the Internet or to an intranet.

® Persist data for an ActiveX document.

® Automate an ActiveX document.

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

In this chapter, you will learn to use ActiveX controls on a Web page. By adding ActiveX controls to Web pages, you can add more functionality and performance to Internet applications than with standard Hypertext Markup Language (HTML).

Objectives

After completing this chapter, you will be able to:

® Use the Application Setup Wizard to create an Internet download setup.

® Use an ActiveX control on a Web page.

® Create and test a package file for licensing.

® Add Visual Basic Scripting Edition (VBScript) code to an HTML page.

® Use safety control options.

For an overview of the topics discussed in this chapter, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

You can use Visual Basic to create client applications that provide access to the Internet. These client applications can support a variety of capabilities for Web users, including the ability to view Web pages, participate in chat sessions, and move files between computers that run under different operating systems.

This chapter describes how to create client applications that provide these capabilities. It also explains how to install and manage a Personal Web Server, so that you can easily test the client applications you create.

Objectives

At the end of this chapter, you will be able to:

® Enable communication between clients and servers across the Internet, or across an intranet by using the **Winsock** control.

® Enable HTTP and FTP connections to the Internet.

® Use the **WebBrowser** control to create applications that browse HTML pages.

® Use Internet Explorer as an Automation server from a Visual Basic application.

As you develop Visual Basic applications, it is important to debug code you have written, and to handle any errors might occur. It is also important to prevent as many of these errors as possible by validating input to your application.

This section covers debugging, preventing errors, and trapping errors in applications.

This section includes the following topics:

® Tools for Debugging

® Validating Field Information

® Validating Form Information

® Handling Run-Time Errors

® Disabling Error Handling

ActiveX documents constitute a core element of a set of technologies known collectively as ActiveX. While ActiveX documents and embedded objects may seem very similar, it is important to understand some key distinctions.

This section introduces you to ActiveX documents and ActiveX document containers. It also explains how ActiveX documents differ from embedded objects, and discusses how they are implemented in Visual Basic.

This section includes the following topics:

® Introduction to ActiveX Documents

® ActiveX Documents vs. Embedded Objects

® Advantages of Using ActiveX Documents

® Common ActiveX Document Containers

® ActiveX Documents in Visual Basic

To create an ActiveX document, you work in an ActiveX project. Visual Basic includes templates for creating a new ActiveX project, or you can convert an existing Standard EXE project to an ActiveX project.

Users can gain access to an ActiveX document from different entry points and from a variety of container applications. Therefore, you must write error-handling code in ActiveX document applications. Writing code for the errors resulting from how a user gains access to the application requires an understanding of how events work during the lifetime of an ActiveX document.

This section describes how to build an ActiveX project for an ActiveX document, and which events to use.

This section includes the following topics:

® Elements of an ActiveX Document Project

® Creating an ActiveX Document Project

® Compiling an ActiveX Document Project

® Siting an ActiveX Document

® User Document Event Behavior

® Differences Between Document Containers

® Determining the Document Container Type

You can create an ActiveX document as either an .exe or a .dll file. In either case, the ActiveX project for the ActiveX document must contain at least one **UserDocument** object by default. To create either of these type of files, you add controls, any additional forms, and code modules to the **UserDocument** object, and then add code for the controls.

When you compile the ActiveX project, an .exe or a .dll file will be created, as well as a Visual Basic document (.vbd) file. To open the ActiveX document in a browser such as Internet Explorer, users must be able to navigate to the .vbd file.

The following two types of files apply to user documents.

® .dob files

Visual Basic stores user documents in text files with a .dob extension. These files contain the source code and property values of the **UserDocument** object and its controls.

® .dox files

Visual Basic stores any graphical elements in files with a .dox extension. These files, such as bitmaps, are used for the controls of a **UserDocument** object.

The .dob and .dox files define the appearance and interface of an ActiveX document, including its properties, events, and methods. These two files are analogous to the Standard EXE project files .frm and .frx files, respectively.

To build an ActiveX document in Visual Basic, you can either create a new ActiveX project or convert an existing Standard EXE project.

Creating a New ActiveX Project

Visual Basic provides templates for creating a new ActiveX project. To create an ActiveX document, you use either the ActiveX Document DLL template or the ActiveX Document EXE template.

The ActiveX projects used for ActiveX documents differ from Standard EXE projects in two ways. ActiveX projects are compiled into .exe or .dll files, and contain a default **UserDocument**, or in the case of an ActiveX control project, a default **UserControl**.

To see an illustration of the **New Project** dialog box with all of the available project templates displayed including ActiveX Document DLL or ActiveX Document EXE templates, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Converting an Existing Standard EXE Project

To convert an existing Standard EXE project to an ActiveX document project, you can use the ActiveX Document Migration Wizard. When you first run the wizard, it converts the selected project forms to ActiveX documents, and changes the project type to either ActiveX EXE or ActiveX DLL.

You can use the ActiveX Document Migration Wizard to convert project forms other than Standard EXE projects to an ActiveX document project, but the option to convert the project type will be unavailable.

u **To convert an existing project to an ActiveX project**

1. On the **Add-Ins** menu, click **ActiveX Document Migration Wizard**.

   If this command is unavailable, use the **Add-In Manager** to add it to the menu.
2. If you want to skip the introduction screen for future use, click **Skip this screen in the future**, and then click **Next**.
3. In the **Form Selection** screen, select the form that you want to convert, and then click **Next**.
4. In the **Options** screen, select the following options, and then click **Next**:
   a. **Comment out invalid code**
   b. **Remove original forms after conversion**
   c. **Convert to an ActiveX EXE**
5. In the **Finished** screen, click **No** when asked if you want to see a Summary Report after the wizard is done, click **Save current settings as default**, and then click **Finish**.

To see a demonstration of how to use the ActiveX Document Migration Wizard to convert a Standard EXE project to an ActiveX document project, and how to convert a form in an ActiveX project to an ActiveX document, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Testing and debugging ActiveX documents involve some additional issues to consider for those creating a standard project. The primary issue is that a user has a variety of ways to view and work with an ActiveX document, as well as choices and some limitations regarding the document container.

This section discusses how you can prepare for situations that result from a user's interaction with an ActiveX document. It also explains how to place a standard project in Break mode when Internet Explorer is the container, and describes how you can use the debugging tools in Visual Basic.

This section includes the following topics:

® Running the Project Within Visual Basic

® Debugging an ActiveX Document

When you design an ActiveX document, you have two choices. You can either design the ActiveX document to run in a separate container by default, or you can have the application provide the container for displaying its document objects. These choices are useful because you do not have control over which container will be used.

To ensure that users who do not have a document container application will still be able to use your application, and to be able to test it in a container, you can provide browsing capabilities in your Visual Basic ActiveX document.

Testing an ActiveX Document in Its Container

To test an ActiveX document in its container application, you must use its .vbd file to run the application. You cannot open an ActiveX document directly from Visual Basic.

To run an ActiveX document in its container

1. Create a file association for the .vbd file and the container application.
2. Switch to Visual Basic, and run the application.

   User documents and forms will not be visible because the output from the project is in the .vbd file.
3. Switch to either **My Computer** or the **Windows Explorer**, and run the .vbd file.

   This causes the .vbd file to be loaded into the container application.
4. After making sure that the .vbd file is loaded correctly in the container application, close the container, switch to Visual Basic, and explicitly end the application.

Determining the Application's Run-Time Environment

To test whether the application is being started in stand-alone mode, you can add a **Sub Main** procedure to a project's standard module, and then take appropriate action.

The following code uses the **StartMode** property of the **App** object to check whether the application is starting in stand-alone mode. If so, the application displays a form.

```
If App.StartMode = vbModeStandalone Then
    frmContainer.Show
End If
```

The previous code opens the application in stand-alone mode. However, based on the result of testing the application's start mode, you could also provide the application with the ability to act as a document container.

The **WebBrowser** control lets you add browsing capabilities to an application.

To test the project in stand-alone mode

1. On the **Project** menu, click **Properties**.
2. On the **General** tab, set **Startup Object** to **Sub Main**.
3. On the **Component** tab, set **Start Mode** to **Standalone**, and then click OK.
4. Run and test the application.

To debug an ActiveX document, you use the same Visual Basic debugging tools you use for other ActiveX components, including setting breakpoints, stepping through code, and using watch expressions. For information about debugging a Visual Basic component, see Tools for Debugging in Chapter 1: Visual Basic Review.

Avoiding Errors in the Container Application

The container that hosts an ActiveX document is the document's client, and uses the objects provided by the ActiveX document.

If you stop running an ActiveX project while the container is hosting the ActiveX document, an error will occur in the container. To prevent this from happening, close the container before you stop the project. This releases the reference to the ActiveX document.

Using Break Mode to Debug an ActiveX Document

If you are running a project and are viewing it in Internet Explorer, you can put it into Break mode by pressing CTRL+BREAK without causing any errors in the container application.

When you press CTRL+BREAK to put a project into Break mode, you can enter a **Stop** statement to halt execution at a particular line of code, as shown in the following code.

```
Private Sub cmdMoveTo_Click()
    Stop ' Add this line to the procedure
    Hyperlink.NavigateTo txtLocation.Text
End Sub
```

In the previous code, clicking the **cmdMoveTo** button causes the program to halt just before the line that uses the **NavigateTo** method of the **Hyperlink** object. Visual Basic will then display the **Code** window with the **Stop** statement highlighted. At this point, you can continue to run the program by pressing F5.

1. What is the function of Word when you start a Word document object from Internet Explorer 3.0?

{ewc mvimg, mvimage,!answer.bmp} A. ActiveX document container

{ewc mvimg, mvimage,!answer.bmp} B. Stand-alone application

{ewc mvimg, mvimage,!answer.bmp} C. Document subcontainer

{ewc mvimg, mvimage,!answer.bmp} D. Server

2. When you create an ActiveX project, which type of file is used by Visual Basic to store the graphical elements of a User Document?

{ewc mvimg, mvimage,!answer.bm A. .dox

p}

{ewc mvimg,mvimage,!answer.bmp}

B. .dob

{ewc mvimg,mvimage,!answer.bmp}

C. .dll

{ewc mvimg,mvimage,!answer.bmp}

D. .vbd

{ewc mvimg,mvimage,!answer.bmp}

3. When you compile an ActiveX project named Myproj as an out-of-process component, what file is produced in addition to Myproj.exe?

{ewc mvimg,mvimage,!answer.bmp}

A. Myproj.dob

{ewc mvimg,mvimage,!answer.bmp}

B. Myproj.dll

{ewc mvimg,mvimage,!answer.bmp}

{ewc mvimg.mvimage.!answer.bmp}

C. Myproj.vbd

{ewc mvimg.mvimage.!answer.bmp}

D. Myproj.dox

4. If your container application is the Internet Explorer, which method of the Hyperlink object should replace the XXXXXXXXXX below to create a procedure to move to a document named NewDoc in the c:\documents directory?

```
Private Sub cmdMoveTo_Click()
    UserDocument.Hyperlink.XXXXXXXXXX _
    "c:\documents\NewDoc.vbd"
End Sub
```

{ewc mvimg.mvimage.!answer.bmp}

A. **NavigateTo**

{ewc mvimg.mvimage.!answer.bmp}

B. **GetObject**

{ewc mvimg.mvimage.!answer.bmp}

C. **GetDocument**

{ewc mvimg.mvima

ge,!answer.bmp}

{ewc mvimg,mvimage,!answer.bmp}

D. **MoveTo**

5. What happens if you attempt to hyperlink from a document contained in the Microsoft Binder to another document?

{ewc mvimg,mvimage,!answer.bmp}

A. A run-time error will occur.

{ewc mvimg,mvimage,!answer.bmp}

B. The link will be successfully navigated because the Microsoft Binder supports hyperlinks.

{ewc mvimg,mvimage,!answer.bmp}

C. An alternative application will be started to handle the request because the Microsoft Binder does not support hyperlinks.

{ewc mvimg,mvimage,!answer.bmp}

D. The request will be ignored because the Microsoft Binder does not support hyperlinks.

{ewc mvimg,mvimage,!

{answer.bmp}

6. What is the first step in saving a new property value in an ActiveX document?

{ewc mvimg, mvimage,! answer.bmp}

A. Read the current property value with the **ReadProperty** method of the **PropertyBag** object.

{ewc mvimg, mvimage,! answer.bmp}

B. Write the new property value to the document using the **WriteProperty** method of the **PropertyBag** object.

{ewc mvimg, mvimage,! answer.bmp}

C. Notify the container that a property value has changed with the **PropertyChanged** method of the **UserDocument** object.

{ewc mvimg, mvimage,! answer.bmp}

D. Notify the **PropertyBag** object that a property value has changed with the **PropertyChanged** method of the **PropertyBag** object.

7. At which UserDocument event would you place code to determine an ActiveX document's container?

{ewc mvimg, mvima

A. Initialize

ge,!
ans
wer
.bm
p}

{ew B. Show
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

{ew C. ReadProperties
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

{ew D. Hide
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

In this exercise, you will enhance the ActiveX documents that you created in the previous exercise.

You will add a custom property to the Orders document to store the current Order ID and make the current Order ID externally available. You will then add the necessary code to the **UserDocument** object to make that property value persistent. When users open the user document again, they should see the same record that was displayed when the document was closed.

u Create the custom property OrderID for the Orders document

1. Add the procedure **FindOrderByOrderID** to the Orders document.

   This procedure should take OrderID as a parameter and move to the associated record, if that record exists.
2. Declare the function as **Private**. The function should also take a variant as a parameter and return a **Boolean**.
3. If the Order ID is a negative number, move to the first record in the recordset.
4. Use the **FindFirst** method of the DAO **Recordset** object to find the associated record. If no match exists, use a bookmark to return to the current record, and return a value of **False**.
   {ewc mvimg, mvimage,!tip.bmp}

u Add the OrderID property to the user document

1. Add the **OrderID** property to the docOrders document, and include code in the **PropertyGet** procedure that returns the current Order ID.
2. Add code to the **PropertyLet** procedure to move to the associated order, if it exists, and then update the document with the new order information.
3. On the **Tools** menu, click **Add Procedure**, and add the public property OrderID.
4. In the **PropertyGet** procedure, simply return the current value as displayed in the Order ID text box.
5. In the **PropertyLet** procedure, call FindOrderByOrderID, and pass the new property value.

u Store properties for the ActiveX document

1. Add code to store the properties for the Orders document, so when the Orders document is reinitialized, it displays the appropriate record(the record displayed just before exiting and saving the document).
2. In the WriteProperties event for the docOrders document, invoke the **PropertyBag** object's **WriteProperty** method to store the current Order ID value into the .vbd file. Use a default value of –1, as shown in the following code:

   ```
   PropBag.WriteProperty "OrderID", txtOrderID.Text, -1
   ```

3. In the ReadProperties event for the docOrders document, invoke the **PropertyBag** object's **ReadProperty** method to read the stored OrderID property value. Use –1 as a default value, as shown in the following code:

   ```
   Me.OrderID = PropBag.ReadProperty("OrderID", -1)
   ```

4. In both the ReadProperties and WriteProperties events, use the **Debug** object to display an appropriate notification in Visual Basic in the **Immediate** window.
5. In the Changed event of the **Order ID** text box, use the **PropertyChanged** method of the user document to prompt the user to save changes.

u Test the document

1. After running the project in Visual Basic, use Windows Explorer to note the file size of the docOrders.vbd file in the Visual Basic folder.
2. Invoke the Home document, and then switch to the Orders document.
3. Move to the last record in the recordset and note the Order ID.
4. Close Internet Explorer. When prompted with the message "Do you want to save the changes?", click **Yes**.

   Note the file size of docOrders.vbd. (You may need to refresh Windows Explorer to see the updated file size).
5. Launch the Orders document in the browser, and verify that it starts up with the appropriate Order ID.

In this exercise, you will extend the user interface of the container based on the requirements of the **Document** object. You will also change the position of the document's controls from the upper-left corner of Internet Explorer to the center of a browser, regardless of the size of the window.

You will add a menu associated with the document to the container's menubar when the **Document** object is active. You will also specify the container size of the document so that when either the container's width or height is less than the document size, the appropriate scrollbars will be displayed.

u Center the document in the browser

In this exercise, you will add functionality to maintain the document's central position in the browser, regardless of the current size of the browser.

1. Select all of the controls on the docHome document, and click **Cut** on the **Edit** menu to move them to the Clipboard.
   {ewc mvimg, mvimage,!tip.bmp}
2. Maximize the document window, and use a **Frame** control to fill up the window.
3. Name the control fraMain, and set the caption of the frame to NULL.
4. While the frame is selected, paste the controls back onto the document, thus making the frame the parent of the pasted controls. Resize the frame and center the controls.
   {ewc mvimg, mvimage,!tip.bmp}
5. In the Resize event for the **UserDocument** object, add code to center the frame (fraMain) by setting the **Left** and **Top** properties.
   {ewc mvimg, mvimage,!tip.bmp}
6. Save and test the document for the desired behavior.

u Add a Record Navigation menu to the Orders document

1. Using the Menu Editor, add a **Record** menu to the Orders document. Add the commands to the menu for **MoveFirst**, **MovePrevious**, **MoveNext**, and **MoveLast**.
2. Specify the position of the menu to appear in the middle of the menu for the container.
3. Add code to each of the menu items for the appropriate functionality.

u Set the document's minimum width and height

1. Use the size of the **Frame** control on the Main document, set the document's **minWidth** and **minHeight** properties.

u Test the project

1. Open the Main document and resize the container until the width and/or height are less than the frame control on the document, fraMain.

   The appropriate scrollbars should be displayed in the window of the document.
2. Switch to the Orders document, and verify that the **Record** menu is displayed and functional on the container.

Most ActiveX document projects consist of two or more related documents. These documents must provide logical navigation among themselves, so it is important to determine how to move between the documents in a particular container.

Determining a Document's Navigational Requirements

As you design an ActiveX document, you should establish how the document will use the features of its intended container. For example, an ActiveX document designed to run in Internet Explorer will probably use its ability to navigate between HTML pages.

An ActiveX document stored in the Microsoft Office Binder does not operate in the same way as a document that runs in Internet Explorer. Instead of navigating between documents or pages, the Binder navigates between sections.

Note You can add a new **UserDocument** object to a project by clicking **Add UserDocument** on the **Project** menu to display the **Add UserDocument** dialog box, and then double-clicking the **User Document** icon.

Examining a Simple Multiple Document Scenario

If users of your ActiveX document will be working with two documents, you may you want the user to be able to open the second document by clicking a button on the first document.

If Internet Explorer is the container application, you must use the **NavigateTo** method of the **Hyperlink** object to move from one document to another, as shown in following code:

```
Private Sub cmdMoveTo_Click()
    'assuming the document is named MyDoc2.
    UserDocument.Hyperlink.NavigateTo _
    "c:\projects\MyDoc2.vbd"
End Sub
```

Note If the container does not support hyperlinks (for example, Microsoft Binder does not), the system registry determines an application that does support hyperlinks, and starts the alternative application to handle the request.

In Visual Basic, you can build code components to run in-process (ActiveX DLL) or out-of-process (ActiveX EXE).

In-Process vs. Out-of-Process Components

In-process components enable faster access to their objects than do out-of-process components. This is because the property and method called do not have to be marshalled across process boundaries. Out-of-process components use separate execution threads than their clients. They can be more flexible because they run outside of the client's process space.

For more information about the differences between in-process and out-of-process components, see Determining Where Server Components Run in Chapter 6: Creating ActiveX Clients.

Component Project Templates

When you create a new project in Visual Basic, you can choose among a number of project templates on which to base the new project. To create a code component, you double-click either ActiveX EXE or ActiveX DLL in the **New Project** dialog box. Selecting either type sets a number of default values that are important to creating code components.
{ewc mvimg, mvimage,!tip.bmp}

The following illustration shows the standard Visual Basic templates.

Visual Basic provides version compatibility that enables you to enhance components while maintaining compatibility with programs compiled with earlier versions of a component.

In Visual Basic, you select version compatibility options by clicking **Project Properties** on the **Project** menu, and then clicking the **Component** tab of the **Project Options** dialog box.

Under **Version Compatibility**, you can select one of three options, as shown in the following table.

Option	Result
No Compatibility	Each time you compile a component, new type library information is generated, including new class IDs and new interface IDs. There is no relationship between versions of a component. Applications that were compiled to use one version cannot use subsequent versions. Use this compatibility option when you start a new project, or when you do not want to enforce compatibility with an earlier version of the component.
Project Compatibility	Each time you compile a component, new type library information is generated, but the type library identifier remains so that your test projects can maintain their references to the component project. Use this compatibility option when you start a new project.
Binary Compatibility	When you compile a component, Visual Basic creates new class IDs and interface IDs only if necessary, and preserves the class ID and interface ID information from the previous version. Programs compiled with the earlier version will continue to work. Use this compatibility option when you want to ensure that an upgrade to a component will work with clients compiled against an earlier version of the component. Visual Basic also warns you when changes to your code would make the new version incompatible with previously compiled applications.

For information about how to use these compatibility options, search on **When Should I Use Version Compatibility?** in Visual Basic Books Online. Select **Version Compatibility for ActiveX Components** to locate a jump to the topic. For information about the levels of binary version compatibility, see **Levels of Binary Version Compatibility** in the same source.

For information about interface IDs, see Client and Server Communication in Chapter 6: Creating ActiveX Clients.

When a code component provides asynchronous methods, you may also want to provide a way for the user to cancel the processing of those methods before they complete.

To enable a client to cancel a lengthy asynchronous method, you can provide a status or progress method that receives a **Cancel** argument. If the client sets the **Cancel** argument to **False** while the status method is running, the component stops the asynchronous method.

For example, you can add the following code to a client.

```
Dim WithEvents x as SomeObject

Sub....
    ...
    x.DoSomeAsynchronousTask
    CancelDialog.Show    ' Display form
                         ' with Cancel button.
    ...
End Sub

Sub x_InProgress(PercentDone, bCancel)
    ' Update progress bar.
    ...
    bCancel = CancelDialog.bCancel
    ...
End Sub
```

In the component, the corresponding code would look like this:

```
Dim bCancel as Boolean
Event InProgress(ByVal pd as Long, _
            ByRef cancel as Boolean)

Private Sub ...(...)
...
    bCancel = False
    RaiseEvent InProgress(percent, bCancel)
    if bCancel then
        ' Cancel the task.
    End If
    ...
End Sub
```

The previous code assumes that the user interface of the client application includes a dialog box containing a **Cancel** button. In both code examples, the **Boolean** variable **bCancel** is used to set a flag that determines if a user has clicked the **Cancel** button.

If the user clicks the **Cancel** button at any time, the **bCancel** flag set in the private **Sub** procedure is set to **True**, and the **If** statement logic cancels the task with the InProgress event.

Note In the InProgress event, using the **ByVal** keyword to pass the parameter means that any change that the client makes to the parameter will not be seen by the component. Using the **ByRef** keyword to pass the parameter means that the component will see the change in the value.

You use these following steps to handle an event that is provided by an ActiveX code component.

1. Declare a **WithEvents** variable.
2. Create an event procedure to handle the event.
3. Create an instance of the class that is defined with the **WithEvents** variable.

To see a demonstration of how to create and use events, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Declaring a WithEvents Variable

You declare a **WithEvents** variable to contain a reference to the object that will provide the notification events. To do this, you add a standard **Dim** statement and the **WithEvents** keyword, as shown in the following code.

```
Dim WithEvents Order As Project1.Class1
```

After you declare the **WithEvents** variable, it will appear in the code window as an object, along with its associated events, as shown in the following illustration.

{ewc MVIMG, MVIMAGE,!v07g025.bmp}

Writing Event Code

In the event procedures associated with the **WithEvents** variable, you write code to handle the notifications you want, as shown in the following code.

```
Private Sub Order_Status(ByVal StatusText As String)
    Debug.Print StatusText
End Sub
```

Creating an Instance of the Object

When you create an instance of an object that provides the events, and store that instance in the variable you dimensioned by using the **WithEvents** keyword, any events fired by the object will automatically be passed to the appropriate event procedure.

The following code creates an instance of the **Order** object, and calls the **SubmitOrder** method, which then raises the events.

```
Private Sub Command1_Click()
    Set Order = New Project1.Class1
    Order.SubmitOrder
End Sub
```

For information about using asynchronous notifications in clients, see Server Notifications Using Events in Chapter 6: Creating an ActiveX Client.

Interfaces are groups of functions that provide connection points through which clients and server components communicate. Interfaces provide standardized access to the methods and properties (functionality) available from servers.

Providing this functionality involves three tasks:

1. Creating an abstract class.
2. Developing a component that uses the abstract class.
3. Developing the client application that uses the component.

This section explains how to define and implement interfaces. It also describes what interfaces must provide so that a component will continue to work with the clients that use it when the component is upgraded.

This section includes the following topics:

® Using Interfaces

® Creating an Interface

An interface is a group of property procedures and methods that a class exposes for use by clients. An interface defines what behavior to expect from an object created from the class. However, it does not define how this behavior is implemented.

To define an interface, you use an abstract class. To implement the behavior, you provide the interface to additional classes, which implement the interface in different ways.

Using Abstract Classes

The purpose of an abstract class is to provide the template for an interface you add to other classes. An abstract class does not contain any implementation code.

You include the **Implements** keyword and the name of the abstract class in the Declarations section of other class modules. The actual implementation of the interface occurs within those other classes.

Why Implement an Interface?

You implement an interface for the following reasons.

® It enables the reuse of code.

® It enables different implementations for similar kinds of objects.

® It enables objects to begin small, with minimal functionality, and over time acquire additional features, as it becomes clear from actual use what those features should be.

Supporting Later Versions of Components

When you use an abstract class to create an interface, you specify the functionality that an object will offer.

When you provide an interface to a code component, you guarantee that client applications written for the component will always work because the original interface will be maintained.

You can extend the original interface, and the original functionality will remain available to the client. This protects client authors from problems caused by component upgrades, while it enables developers to enhance components.

For information about issues related to versioning, see Version Compatibility.

For information about how interfaces work, see Using Interfaces in Chapter 6: Creating ActiveX Clients.

You use the following three phases in the development and use of interfaces and abstract classes.

1. Create the abstract classes.
2. Develop a code component that implements those abstract classes.
3. Develop an application that uses the component.

u To create abstract classes for Visual Basic interfaces

1. Open a new ActiveX EXE or ActiveX DLL project.
2. Add the properties and methods you want the interface to include in the class module.
3. Specify a class name for the interface.
4. On the **File** menu, click **Make <Project>** to build the project into an executable file.

   The type library for the resulting .exe file or DLL will contain the information required by the **Implements** statement in the classes of your component.

   The following code provides two custom methods for a class.

```
' Code for the abstract IPayroll class module.
Public Sub CalculateWithholdings()
    ' No implementation code here.
End Sub

Public Sub DoDirectDeposit()
    ' No implementation code here.
End Sub
```

u To develop a component that uses abstract classes

1. Add a reference in the component to the type library for the abstract classes.
2. Use the **Implements** statement to provide the interface defined in the abstract class to other classes.
3. In the secondary classes, add implementation code to the property or methods that use the abstract class.
4. Compile the component.

   The following code contains the implementation code for a class that uses the custom classes that were declared in the previous code.

```
' This class offers objects that implement
' the IPayroll abstract class
Implements IPayroll

Private Sub IPayroll_CalculateWitholdings()
    ' This code implements the class defined
    ' in the IPayroll abstract class.
End Sub

Private Sub IPayroll_DoDirectDeposit()
    ' More implementation code specific to this type
    ' of object.
End Sub
```

You could develop different classes that would implement the custom classes in different ways, but would use the same abstract class.

u To develop a client application that uses the component

1. In the client application, add a reference to the component.
2. In the client application, add a reference to the library that contains the abstract classes.
3. Compile the application.

4. Create a Setup program that will install the client, the component, and the type library, if the component runs out-of-process.

You use property procedures if you want to run code when a property is set or retrieved. With property procedures, you can perform the following tasks.

® Run a procedure when a property value is changed or read.

® Constrain a property to a small set of valid values.

® Expose a property that is read-only.

You create property procedures in pairs. For example, you can create a **Set** or **Let** procedure to assign a value to the property, and then create a **Get** procedure to return the value. You give the two procedures the same name, which is also the name of the property.

To see a demonstration of how to create and use classes, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Creating a Property

The following code creates a **Property Let** and **Property Get** procedure that assigns a string value and returns the value for the **User** property.

```
Private gsUser As String
Public Property Let User (s As String)
    gsUser = Ucase(s)
End Property

Public Property Get User () As String
    User = gsUser
End Property
```

Setting and Retrieving a Property

This code creates an instance of Class1, and sets and retrieves the **User** property:

```
Dim Demo1 As Class1
Set Demo1 = New Class1
Demo1.User = "Joe" 'Calls Let procedure.
Print Demo1.User   'Calls Get procedure.
```

To create a property that returns a standard data type, define a **Property Get** procedure and a **Property Let** procedure. To create a property that is an **Object** data type, define a **Property Get** procedure and a **Property Set** procedure.

When you read the property, the **Property Get** procedure runs. When you set the property, the **Property Let** procedure runs.

The **Property Let** procedure always contains at least one argument, which is the value for the property. This argument should always be the last argument in the argument list.

To create a read-only property, define a **Property Get** procedure without a matching **Property Let** or **Property Set** procedure.

For information about property procedures, search for **Adding Properties to a Class** in Visual Basic Books Online, and then click **Programming with Objects**.

A named constant is a named item that retains a constant value throughout the execution of a program, and can be used in place of literal values.

For example, if you want to create a message box that includes **Yes** and **No** buttons, you can use the named constant **vbYesNo** for the argument instead of a literal value of **4**. Named constants make your code easier to read and maintain.

Creating Named Constants

To define your own set of named constants, you create an enumeration to include the constants.

The following code shows how to create an enumeration and use a named constant with a simple method.

```
Public Enum Temp
    Cold = -50
    Warm = 70
    Hot = 100
End Enum

Public Sub ReportTemp(x As Temp)
    If x = Cold Then
        MsgBox "Take a coat."
    End If
End Sub
```

You can make the members of your enumeration available to users of a component by marking the enumeration **Public**, and including it in any public module that defines a class.

Although the enumeration appears in a module that defines a class, it has global scope in the type library.
{ewc mvimg, mvimage,!tip.bmp}

Note To avoid enumeration member name conflicts, prefix the member name with lowercase characters that identify the type and the component to which it belongs. For example, the built-in enumeration VbDayofWeek contains numeric constants with the names vbMonday, vbTuesday, and so on.

```
Option Explicit
Public CardNumber As Integer
Public ExpireDate As Date
Private msngAmount As Single

Public Property Get PurchaseAmount() As Single
    PurchaseAmount = msngAmount
End Property

Public Property Let PurchaseAmount(ByVal sngAmount As Single)
    If sngAmount > 0 Then
        msngAmount = sngAmount 'save in private module variable
    Else 'purchase less than zero is invalid
        msngAmount = 0
    End If
End Property


Public Function Approve() As Boolean
    'dummy logic for approving credit card
    If msngAmount < 1000 And ExpireDate > Now() Then
        Approve = True
    Else
        Approve = False
    End If
End Function
```

Once you have set up a test project and established a reference to the component, you can debug the component just as you would any Visual Basic application.

You can step between the client application and component, and use the standard debugging tools to set break points, step through code, use watch expressions, and so on.

Handling Missing References

In the course of debugging components, you may see the following error message: **Connection to type library or object library for remote process has been lost. Press OK for dialog to remove reference.**

This message indicates that the globally unique identifier (GUID) of the component's type library has changed, and the test project cannot locate the type library.

To clear the missing reference, click OK. In the **References** dialog box, the word **MISSING** will appear next to the name of the component. Clear the check mark next to the name of the missing component, and then click OK.

For information about type libraries and object browsers, see Automation Objects in Chapter 6: Creating ActiveX Clients.

For information about how COM uses GUIDs, see Client and Server Communication in Chapter 6: Creating ActiveX Clients.

The first step in testing an ActiveX code component is to create a test project that will use the component.

This topic explains how to test ActiveX DLL and ActiveX EXE code components.

Testing an ActiveX DLL

To test an ActiveX DLL, you can add a Standard EXE project to the ActiveX DLL's project group, and test the in-process component within a single instance of Visual Basic. You can then step directly from the test component code to the in-process component code.

Testing an ActiveX EXE

To test an ActiveX EXE, you must run the component within one instance of Visual Basic, and run the test application in a second instance of Visual Basic. In this case, you cannot step directly from one component to the other, but you can use all of the other Visual Basic debugging tools.

To see a demonstration of how to create and test a basic code component, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Handling errors requires close communication between clients and components. If possible, a component and its clients should handle errors, rather than displaying a message to the user.

When a client application calls a method of an object provided by a component, the method can use the following two ways of providing error information.

® Raise an error that the client application must handle. In this case, the client uses conventional error-handling statements to handle errors that are raised.

–or–

® Return an error code that the client application must test for and handle.

Using a Raised Error

In a code component, the method that was called by the client application uses the **Raise** method of the **Err** object to raise an error to the client application. The client implements an error handler to trap the error.

Developers of client applications can either implement in-line error handling by using the **One Error Resume Next** statement, or by writing error-handling routines by using the **On Error Goto** statement.

Using a Return Value

In a code component, the method called by the client application returns a value to the client. The client examines the return value to determine if an error has occurred, and if so, which one.

With this type of error handling, developers of client applications must use in-line error handling. Code must always test the return value after calling a method.

For information about error handling in components and clients, search on **Debugging, Testing, and Deploying Components** in Visual Basic Books Online, and then click **Topics on Debugging**.

Once you have finished creating an application, you can create an executable file for your users. This section describes how to compile your project.

This section includes the following topics:

® Compiling an EXE File

® Compiling to Native Code

This topic describes how ActiveX documents are developed, and explains how they operate within ActiveX container applications.

Background

Initially, ActiveX documents were a proprietary Microsoft technology used by the Binder application included with Microsoft Office for Windows 95. This application enabled users to group a set of related Office documents together in one application, regardless of the Office application or type of data.

In conjunction with the underlying COM technology, ActiveX objects and ActiveX document containers support additional interfaces that enable you to create document objects (or forms).

ActiveX Document Containers

The underlying OLE document technology on which ActiveX documents are based enables objects to be hosted within a container application. A container application is also referred to as an ActiveX document container.

The primary aspect of ActiveX document functionality is its effect on the user interface. An ActiveX document fills the display area of the container, and gives the container the ability to transform itself to look and act just like the server application.

For example, a Microsoft Excel worksheet object can be called from within Internet Explorer, or the container application.

The following illustration shows how an ActiveX document can be called from a component and displayed in its document container.

{ewc MVIMG, MVIMAGE,!v10g005.bmp}

You must view ActiveX documents in an ActiveX document container. An ActiveX document is a component that exists within a client.

An ActiveX document container is usually a stand-alone application, such as Internet Explorer. For example, when you start Microsoft Word, you are really launching the Word application as a container for its document object. When you start a Word document object from Internet Explorer, Word is launched as a server that provides the document to another container.

For information about the types of document containers available, see Common ActiveX Document Containers.

This topic describes how Internet Explorer version 3.0 and Microsoft Office Binder can be used as document containers for ActiveX documents. It also previews the features of Internet Explorer version 4.0.

Internet Explorer 3.0

Microsoft Internet Explorer version 3.0 can host an ActiveX document so that the user's working environment is similar to an existing application, but with added Internet functionality.

For example, users can view and update an order entry form contained within Internet Explorer, just as they would a form in a stand-alone Visual Basic application. Users can also navigate to another form or page by typing a different Uniform Resource Locator (URL) in the Address box, or by clicking a button or a hyperlink.

The following illustration shows an ActiveX document, in this case a Visual Basic form, that has been activated within Internet Explorer 3.0.
{ewc mvimg, mvimage,!illust.bmp}

Microsoft Office Binder

The Microsoft Office Binder is a multipurpose container. Analogous to an electronic paper clip, it enables users to group and edit related documents.

Users can work with an ActiveX document to create active content in the Office Binder . For example, a Binder may contain a set of project documents that include Word documents, financial analysis worksheets, flowcharts, and schematics. All of the documents are saved in a single Binder file, where users can open each document type individually, without having to launch separate applications. The individual documents can also be printed as one print job.

Microsoft Internet Explorer 4.0

Internet Explorer version 4.0 will combine the elements of the current Windows 95 Explorer with features of Internet Explorer 3.0 into a shell that resembles the Windows Explorer. This shell, combined with new Internet tools, will provide integration between data on a local computer and data on the Internet. Users will no longer have to use one interface for the Windows Explorer and another interface for the Internet.

The Windows Explorer combined with Internet Explorer will be an ActiveX document container. Users will be able to open any applications that support ActiveX document interfaces directly from Windows Explorer.

For example, users will be able to open HTML browsers, such as Internet Explorer, from the Windows Explorer. A Web page written in HTML will be considered a type of document in the same manner as a Word document or a Microsoft Excel workbook.

In Visual Basic, you create an ActiveX document by adding a **UserDocument** object to the designer provided by Visual Basic for ActiveX documents. The **UserDocument** object is the foundation for all ActiveX documents.

The UserDocument Object

The **UserDocument** object is similar in functionality to the **UserControl** object, including its support for properties and methods. Like the **UserControl** object, the **UserDocument** object can fire events, support callbacks, and supply asynchronous server notification.

For information about the **UserControl** object, see Introduction to Controls in Chapter 8: Creating ActiveX Controls. For information about server notification, see Receiving Notifications from Servers in Chapter 6: Creating ActiveX Clients.

The **UserDocument** object also implements interfaces that enable it to be hosted within a document container.

The following illustration shows an ActiveX project viewed in the Visual Basic Project Explorer.

{ewc MVIMG, MVIMAGE,!v10g030.bmp}

This project consists of a user document, which is the default Visual Basic **UserDocument** object added to a new ActiveX document project. You can add additional user documents to a project. The project also consists of forms and modules.

The ActiveX Document Designer

There are several designers in Visual Basic that are used to design classes, forms, ActiveX controls and ActiveX documents. When you open a user document in design mode, the document appears in a window referred to as a designer.

Working with a designer is similar to working in a form. You can use the designer to create the visual aspects of an ActiveX document, such as placing controls on the **UserDocument** object.

To see an illustration of a user document inside its designer, along with the code module for the document, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

During the lifetime of a **UserDocument** object, the events that occur and the order in which they occur depends on a number of factors, including which container is used, whether the document has been sited, and what actions the user takes within the container.

Many different containers can host an ActiveX document. This topic focuses on the events that occur when using Internet Explorer as the container.

The following list provides examples of the events that can occur during the lifetime of an ActiveX document.

- Initialize

  The Initialize event occurs each time an instance of an ActiveX document is created or recreated. It is always the first event that occurs during the lifetime of an ActiveX document.

- InitProperties

  The InitProperties event occurs when an ActiveX document has been sited in the document's container, but none of the ActiveX document's property values have been saved. Once a property value is saved, the InitProperties event is replaced by the ReadProperties event.

- ReadProperties

  The ReadProperties event occurs when a property is saved by using the **PropertyBag** object.

- Terminate

  The Terminate event occurs just before the ActiveX document is destroyed. In Internet Explorer 3.0, an ActiveX document is stored in a cache of four documents.

  When a user loads or navigates to a fifth document, the ActiveX document is terminated. To remove any object references, you can set all global object references to Nothing by using the Terminate event.

- Show

  The Show event occurs in one of the following two situations.

  - When a user opens an ActiveX document that has been sited in its container.
  - When a user clicks the **Back** or **Forward** button to return to an ActiveX document in the History list of Internet Explorer.

- Hide

  The Hide event occurs in one of the following two situations.

  - When a user navigates off a document, immediately before the Terminate event.
  - When Internet Explorer is terminated while the document is being viewed, or is still within the cache of active documents.

The functionality of the Show and Hide events depend on the particular container. For information about these different behaviors, see Differences Between Document Containers.

One major difference between forms and ActiveX documents is that you do not know how a user might invoke an ActiveX document. In Internet Explorer, for example, a user can open an ActiveX document from various entry points, such as the Favorites list or from a desktop shortcut.

You can use global variables to control the way in which ActiveX documents communicate with each another. Global variables also enable you to make public properties of one document available to another document.

Setting a Global Variable

To determine how a user has launched an ActiveX document, you must establish a way for the document to identify itself to another ActiveX document in the same suite of documents. One way to enable an ActiveX document to identify itself to another ActiveX document is to use a global variable. Set the variable as an object reference to an ActiveX document so that it acts as a signal between documents.

The following illustration shows the global variable gVar that acts an object reference to Doc1 and Doc2.

{ewc MVIMG, MVIMAGE,!v10g070.bmp}

Setting a global variable involves declaring it in a code module. The following code shows the global variable gHomeDoc that functions as an object reference to the document HomeDoc:

```
Public gHomeDoc as String
```

To check for and destroy any global references set by other ActiveX documents, you can use the Show and Hide events. Destroying global references frees the memory and system resources used to retain the reference.

Detecting an Object Reference

You can make the public properties of one ActiveX document available to other ActiveX documents. To make the properties globally available, you must test that a global variable has been set to an object that contains the public property representing the ActiveX document.

The following code uses the **Is Nothing** statement to check that the global variable gHomeDoc is set to an object. If it is, the public properties of the ActiveX document are available, and the caption of the label is set to the **strMyProp** property from the **HomeDoc** document. If it is not set to an object, the object reference is destroyed, and an appropriate message is displayed.

```
Private Sub UserDocument_Show()
    If Not gHomeDoc Is Nothing Then
        lblCaption.Caption = gHomeDoc.strMyProp
        Set gHomeDoc = Nothing ' IMPORTANT! Set
        'the variable to nothing to destroy the
        'reference to the global variable
    Else
        MsgBox "Sorry, please open UserDocument HomeDoc first."
    End if
End Sub
```

Using the Hide Event to Destroy References

In Internet Explorer, the Hide event occurs in one of two circumstances. It occurs whenever a user navigates off a document, or immediately before the Terminate event.

You use the Hide event to destroy any global object references before navigating to another document, as in the following code:

```
Private Sub UserDocument_Hide()
    If Not gHomeDoc Is Nothing Then
        Set gHomeDoc = Nothing ' IMPORTANT! Set
        'the variable to nothing to destroy the
        'reference to the global variable
    End if
```

```
End Sub
```

Data persistence is the ability of a component to store and retrieve data. By storing properties of the ActiveX document you create, you can provide users with the convenience of not having to repeatedly enter the same data into an application.

Certain containers, such as Internet Explorer and Microsoft Office Binder, let you persist data by writing to an interface of the application. To save data to a file (either the .vbd file or some other kind of file, depending on the application), you use the **PropertyBag** object.

The **PropertyBag** object is exposed as part of the WriteProperties and ReadProperties event declarations and has two methods, which are described in this table.

Method	Action
WriteProperty	Writes a value to be persisted to a **PropertyBag** object.
ReadProperty	Returns a persisted value from a **PropertyBag** object.

Notifying the Container of Changes

The first step in saving a property is to let the container know that the property value has changed. You can accomplish this by using the **PropertyChanged** method of the **UserDocument** object.

The following code shows the **PropertyChanged** method in the Change event of a **TextBox** control. It notifies the container that the property value has changed, and in the case of Internet Explorer, prompts the user to save changes before it terminates.

```
Private Sub txtHomeDoc_Change()
    PropertyChanged
End Sub
```

In general, whenever a property value changes and is persisted to the **PropertyBag**, you should call **PropertyChanged** to ensure that the new value is saved.

Writing a Property to the .Vbd File

In response to the **PropertyChanged** method, a flag is set to provide a notification to the container, and the resulting WriteProperties event occurs. You can use the **WriteProperty** method to save the property to the .vbd file, or to whatever location the container provides. The following code shows this process:

```
Private Sub UserDocument_WriteProperties _
    (PropBag As VB.PropertyBag)
    'Write the property to the Property Bag
    PropBag.WriteProperty "txtHomeDoc", _
    txtHomeDoc.Text, "You are in the Home Document"
End Sub
```

Reading Properties

The InitProperties event occurs only as long as none of the ActiveX document's properties have been saved using the **PropertyBag**. In the case of the previous code example, the next time the ActiveX document is opened, the ReadProperties event would occur, and through the **PropertyBag** object, you would retrieve the saved data using the **ReadProperty** method.

The following code shows how the data stored in the previous code example is retrieved:

```
Private Sub UserDocument_ReadProperties _
    (PropBag As VB.PropertyBag)
    'Read the property back into the TextBox control.
    txtHomeDoc.Text = ReadProperty("txtHomeDoc", _
    "You are in the Home Document")
End Sub
```

Working with an ActiveX document is very much like working with a form. To implement functionality, you use all of the same properties as you do with a form.

This section describes how you can enhance the user interface of ActiveX documents. These enhancements to the user interface include how to work with scaling properties that control screen displays in an ActiveX document, as well as suggestions for including menus.

This section includes the following topics:

® Setting User Document Properties

® Adding Menus to a User Document

A **UserDocument** object is a special type of form. It includes all of the same properties available for a form, and more. The additional properties enable behavior that is specific to working within a document container.

This topic focuses on the **UserDocument** object properties that enable you to control the user's view of an ActiveX document within its container.

Setting MinHeight and MinWidth

With an ActiveX document, all of the same scaling properties are available, working just as they do with a form.

Users can resize the dimensions of a container application, such as Internet Explorer, so you will not have any control over the amount of screen space a document container gives to your ActiveX document.

You can, however, determine the minimum space requirements for your ActiveX document, and cause scroll bars to appear when the document exceeds those requirements within the container.

To specify when scroll bars are displayed, you set the **MinHeight** and **MinWidth** properties of the **UserDocument** object, as shown in the following code:

```
Private Sub UserDocument_Resize()
    UserDocument1.MinHeight = 4000
    UserDocument1.MinWidth = 9000
End Sub
```

In general, you should specify the **MinHeight** and **MinWidth** properties. If you do not set these properties, and use the default height and width settings of the **UserDocument** object, scroll bars will still be displayed when users resize the document. Because these are usually not desirable values, you will generally want to specify the **MinHeight** and **MinWidth** properties.

Scaling Properties

When working with a standard form, the **ScaleMode** property will return or set a value that indicates the unit of measurement for the individual coordinates. These measurements can include twips, points, pixels, and so on. By using the related **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties, you can create a custom coordinate system.

For information about scaling, see Visual Basic 5 Help.

You can create your own menus for an ActiveX document by using the Menu Editor provided in Visual Basic.

Using the **About** box as an example, this topic describes how to add menus, focusing on how custom Help menus are implemented in container applications.

Providing an About Box

You should provide users with an **About** box for all ActiveX documents. The information in this box lets users know about vendor and version information, copyright details, and any licensing issues.

In Visual Basic, you add an **About** box through the **Add Form** dialog box.

The following illustration shows the default form that is added to a project when you double-click the **About Dialog** icon in the **Add Form** dialog box.
{ewc mvimg, mvimage,!illust.bmp}

Merging Help Menus

When you create a custom Help menu for your ActiveX document, the Help menu is automatically merged with the container. By setting the value of the **NegotiatePosition** property, you can determine where the Help menu of an ActiveX document is displayed within the container's Help menu.

To create a Help menu and submenu for the **About** command, you use the Visual Basic Menu Editor.

You also add the following code to the Click event for the **About** menu:

```
Private Sub mnuAbout_Click()
    frmAbout.Show vbModal
End Sub
```

The position of a control on a user document is based on the **ScaleMode** property of the user document, just as a control is placed on a form.

The following settings center the **Panel** control on the user document:

```
pnlMain.Left = (ScaleWidth - pnlMain.Width) /2
pnlMain.Top = (ScaleHeight - pnlMain.Height) /2
```

When calling a code component, a client application may receive a message that the server is busy. For example, this occurs when a component is completing a long operation and the client attempts to perform an action.

You can use the properties of the **App** object (for example, **OLEServerBusyRaiseError**, **OLEServerBusyTimeout**, or **OLEServerBusyMsgText**) to customize how this message will be displayed to users.

The following code shows how to set a custom title and message text. These settings override the settings in the **Component Busy** dialog box.

```
Public Const App_Title = "Employee Expenses Application"

App.OLEServerBusyMsgTitle = App_Title
App.OLEServerBusyMsgText = "The Employee Expenses  __
   application has not responded because of higher _
   than usual network usage. If you have been _
   waiting more than five minutes, you may wish _
   to cancel this request and try it later."
```

For information about the properties of the **App** object, see Visual Basic Help.

This section provides an introduction to using ActiveX controls, and describes how to create an ActiveX control with Visual Basic. It also discusses how to distribute controls either as compiled code or as source code.

This section includes the following topics:

® Introduction to Controls

® Options for Distributing Controls

® Steps for Creating an ActiveX Control

As with any standard control in a Visual Basic application, an ActiveX control provides functionality by exposing its properties, methods, and events. Creating properties, methods, and events for a control is similar to creating them for code components.

You can use the ActiveX Control Interface Wizard to perform many of the tasks associated with exposing properties, methods, and events, including delegation and mapping to constituent controls.

This section includes information about how to use property procedures to create properties, and to map properties to multiple controls. It also describes how to store and retrieve the values of properties.

This section includes the following topics:

® Adding Properties

® Adding Methods

® Using Ambient Properties

® Storing and Retrieving Property Values

® Exposing Named Constants

® Raising Control Events

® Using the ActiveX Control Interface Wizard

This section explains how to add property pages to an ActiveX control.

Property pages provide an alternative to organizing ActiveX control properties. With property pages, a user can view related properties in a tabbed dialog box.

For example, if you create a **Font** property that has several different values, you can create a property page for each of them instead of displaying the values in a drop-down list box in the **Properties** window.

This section includes the following topics:

® Creating the Property Page Interface

® Coding Property Page Behavior

® Establishing Property Page Relationships

® Using Standard Property Pages

This section discusses issues related to testing an ActiveX control.

When you test your ActiveX control, you must test the control's design-time functionality. Developers who use your ActiveX control must be able to work in the control's Design mode. Because a control cannot run by itself, you must create another project that you can use to test it.

The specific events in an ActiveX control will behave differently, depending on how the control is instantiated. Understanding this behavior will help you test your controls properly.

This section includes the following topics

® Establishing a Basis for Testing

® Control Instancing and Events

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **90 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 8.

Exercise 1: Creating a Control

In this exercise, you will create a simple ActiveX control and test that control from a Standard EXE project.

Exercise 2: Adding Properties and Methods

In this exercise, you will create properties and a method and add them to the Stock ActiveX control created in the previous exercise.

Exercise 3: Saving and Loading Properties

In this exercise, you will add code to save and restore the **UserControl** object property values.

Exercise 4: Raising an Event

In this exercise, you will add code to raise the TickerKeyPress event from the Stock control.

Exercise 5: Creating a Property Page

In this exercise, you will create a property page for the Stock ActiveX control.

Exercise 6: (Optional) Creating a Data-Bound Control

In this exercise, you will create an ActiveX control that can be bound to a data source.

In this lab, you will build and test an ActiveX control.

After completing this lab, you will be able to:

® Create a basic ActiveX control with Visual Basic.
® Add properties and methods to the control.
® Save and load control properties.
® Raise an event from the control.
® Create a property page for the control.
® Create a data-bound control.

Before working on this lab, you should be familiar with the following concepts:

® Creating ActiveX code components

® The contents of this chapter

To complete this lab, you need the following setup:

® Visual Basic version 5.0 or later

This section describes how to license controls in Visual Basic so you can distribute them safely. It also explains how to use the Application Setup Wizard to package all the required components of an application for distribution.

For information about distributing ActiveX controls over the Internet, see Chapter 9: Using ActiveX Components on a Web Page.

This section includes the following topics:

® Licensing Controls

® Creating a Control Setup Program

ActiveX controls were formerly known as OLE controls. With Visual Basic version 5.0, you can create ActiveX controls for use in any ActiveX host, including Visual Basic and Internet Explorer.

What Is a Control?

Controls are reusable objects that include visual elements and code. With Visual Basic, you can use controls to quickly create forms and dialog boxes.

The Control Toolbox in Visual Basic contains all of the built-in controls, so you can develop applications quickly and easily. Controls must be placed in some type of container, such as a form or an application.

Control Classes

A control you create in Visual Basic is known as a control class, which acts as a template for that control. When you place a control on a form, you create an instance of that control, as shown in the following illustration.

{ewc mvimg, mvimage,!v08g005.bmp}

Control Components

Controls can be compiled into control components, also known as .ocx files. A control component can provide more than one kind of control.

A Visual Basic ActiveX control project contains one or more .ctl files, each of which defines a separate control class. When you compile a control project, an .ocx file is created for the control component.

A single .ocx file can contain multiple controls, as shown in the following illustration.

{ewc mvimg, mvimage,!v08g010.bmp}

Controls vs. Code Components

Controls differ from code components. A code component is an application that exposes functionality and can be used and reused by other applications through Automation. Control components contain visual elements that can generate events based on user actions.

To see an illustration that compares control and code components, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

The UserControl Object

The **UserControl** object is the foundation for building controls. Every ActiveX control that you create with Visual Basic contains a **UserControl** object..

UserControl objects contain code modules and visual designers. When you open a **UserControl** in design mode, the object is displayed in a visual designer window. You can use the visual designer to place additional controls on the **UserControl** object, just as you would on a Visual Basic form.

To see an illustration of the **UserControl** visual designer with a code window, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Files Associated with the UserControl Object

The source code and property values for a **UserControl** object are stored in text files with a .ctl extension. A .ctl file is equivalent to the Visual Basic .frm file that is used to store this information in a form.

Graphical elements, which cannot be stored as text, are stored in files with the .ctx extension. A .ctx file is equivalent to a Visual Basic .frx file that is used to store graphical elements in forms.

The way in which you distribute a control depends on how the control will be used. You can either distribute it as a compiled component, or include the source code of the control as part of your application.

An ActiveX control can be used in many different types of applications, such as Microsoft Office applications, Internet Explorer, and Visual Basic forms. You can use ActiveX controls with local projects, or you can download them onto a Web page.

Distributing Controls as Compiled Components

To distribute a compiled control component (.ocx file) with an application, you must create ActiveX controls as public classes. You can include the compiled control component through the Setup program of your application.

Distributing Controls as Source Code

You can also include a .ctl file in any Visual Basic project. When the application is compiled, the source code of the control is compiled as part of the application's executable file.

The advantages of distributing controls as source code are:

® There is no .ocx file to distribute.

® Debugging is easier because you only need to debug features of the specific application.

® There is version independence because a version of the control’s source code is compiled into the application.

The disadvantages of distributing controls as source code are:

® Fixing bugs in the control’s source code requires recompiling the entire application.

® Multiple applications require additional disk space because each application includes all of the source code for a control.

® The source code can be changed, so version control may be required.

In this section, you will learn how to create the user interface for a control.

Creating the user interface for an ActiveX control is similar to creating a standard Visual Basic form. You draw controls, and then provide the code that defines the behavior of those controls.

Unlike a form, however, a control acts as a component within an application. When you start designing a control, you can look at how existing controls have been created. For example, if you are designing a control that takes input, you can view the properties, methods, and events provided by the standard **TextBox** control.

This section includes the following topics:

® Adding Constituent Controls

® Sizing a Control

® Creating a Container Control

® Adding an About Box

® Specifying a Toolbox Bitmap

To test controls, it is useful to understand the event behavior of the **UserControl** object.

An ActiveX control is a client that runs in a form or in some other type of container. Therefore, its events behave somewhat differently than control events on a standard .exe project form.

Events in Instances of Controls

When you create controls, the **UserControl** object acts as the blueprint for any control placed on a form or other container. You use this blueprint to create an instance of a control. When a control is created, a series of control events will occur.

When testing a control, it is important to understand when and how an instance of the control is generated. The specific events and the order in which they occur depend on the particular user or system actions that take place during the lifetime of the control instance.

Summary of Instancing and Events

The following table summarizes how controls are created in response to a control's actions, and which events occur when the control is created.

User action	Type of control instance	Events
Places the control on a form.	Design-time instance is created.	Initialize InitProperties Resize, Paint
Runs an application containing the control at design time.	Design-time instance is destroyed, and run-time instance is created.	WriteProperties Terminate Initialize ReadProperties Resize, Paint
Quits an application containing the control at design time.	Run-time instance is destroyed, and design-time instance is created.	Terminate Initialize ReadProperties Resize, Paint
Closes the form containing the control.	Design-time instance is destroyed.	WriteProperties Terminate
Runs the compiled application.	Run-time instance is created.	Initialize ReadProperties Resize, Paint

In this exercise, you will create an ActiveX control that can be bound to a data source.

u Create the control

1. Create a new ActiveX control, as shown in the following illustration.

   {ewc mvimg, mvimage,!v08g075.bmp}
2. Create **CompanyName** and **Address** properties for the control, and make them bindable.
3. Save the project.

u Test the control

1. Create a client project to test the control.
2. Bind the **CompanyName** and **Address** properties to the Company Name and Address fields of the Publishers table located in the Biblio.mdb database.

   The file Biblio.mdb is located in the Visual Basic install directory.
3. Test to see if the data is bound, and that it updates correctly.

In this exercise, you will create a property page for the **Stock** ActiveX control.

u Add a custom property page

1. Add a new property page to the Stock project.

   Do not use the Property Page Wizard.
2. Set the property page **Name** property to **Active**.
3. Add a **CheckBox** control to the property page, as shown in the following illustration.

   {ewc mvimg, mvimage,!v08g065.bmp}
4. In the Click event of the **CheckBox** control, set the **Changed** property to **True**.
5. In the **PropertyPage_SelectionChanged** event procedure, set the **Value** property of the **CheckBox** control, as shown in the following table.

If SelectedControls(0).Active =	The Value property will be
True	vbChecked
False	vbUnchecked

   This will set the state of the **CheckBox** control, based on the value of the **Active** property of the selected control.

   For information about property pages, see Coding Property Page Behavior.
6. In the **PropertyPage_ApplyChanges** event procedure, set the **Active** property of SelectedControls(0), as shown in the following table.

If CheckBox Value property =	The Active property will be
vbChecked	**True**
vbUnchecked	**False**

   This will set the **Active** property of the **Stock** control, based on the selection made in the **Property** dialog box.
7. In the **Connect Property Pages** dialog box of the **Stock** control, add the **Active** property page, as shown in the following illustration.

   {ewc mvimg, mvimage,!v08g070.bmp}
8. In the **Procedure Attributes** dialog box, associate the **Active** property page with the **Active** property.

   For information about associating a property page with a property, see Establishing Property Page Relationships.
9. Save the project.

u Test the control

1. Close all of the windows associated with the Stock project.
2. Open the Container form.
3. Right-click the **StockPrice** control, and then click **Properties**.
4. In the **Property Pages** dialog box, click the **Active** tab.

   The active check box should match the value of the **Active** property.
5. Change the active check box, and then click OK.

   The **Active** property should appear changed in the **Properties** window.
6. In the **Properties** window, click the ellipsis (**…**) button next to the **Active** property.

   Clicking this button will also display the **Property Pages** dialog box. Notice, however, that only the **Active** tab is displayed in the **Property Pages** dialog box.

In this exercise, you will add code to raise the TickerKeyPress event from the **Stock** control. This event will be raised from the KeyPress event of the ticker text box to enable developers to add code to the event.

u **Raise a control event**

1. In the **UserControl** object code window, declare the event TickerKeyPress to take the single argument **KeyAscii** of type **Integer**.
2. In the KeyPress event of the ticker text box, raise the TickerKeyPress event and pass the **KeyAscii** argument.

   This will simulate the KeyPress event for the **UserControl** object.
3. Save the project.

u **Test the event**

1. Close all of the windows associated with the Stock project.
2. Open the Container form.

   The **StockPrice** control should now contain the TickerKeyPress event.
3. In the TickerKeyPress event, add the following line of code:

   ```
   KeyAscii = Asc(UCase(Chr(KeyAscii)))
   ```

   This code will convert any character type into the uppercase value of that character.
4. Run the project.
5. Type **msft** (lowercase) into the ticker text box.

   The text should be converted to uppercase as you type.

   If you are having problems with your solution, click this icon to see the **UserControl** object code from the lab solution.

   {ewc mvimg, mvimage,!code.bmp}

In this exercise, you will create properties and a method, and add them to the **Stock** ActiveX control created in the previous exercise.

u Create properties and a method for a control

1. Using property procedures, add the property **Active** to the **UserControl** object. Set the property type to **Boolean**.
2. In the **Active Property Get** procedure, return the value of the **Timer** control's **Enabled** property.
3. In the **Active Property Let** procedure, set the **Enabled** property of the **Timer** control to the value passed to the **Active** property.

   This will let you control the **Stock** control's refresh behavior.
4. Using property procedures, add the property **Font** to the **UserControl** object. Set the property type to **Font**.

   Because this is an object property, use a **Property Set** procedure instead of a **Property Let** procedure. This property will be used to set the **Font** object for all of the **TextBox** and **Label** controls on the **UserControl** object.
5. In the **Font Property Get** procedure, return the **Font** object of the **Stock Price** text box.

   When returning the **Font** object, be sure to use a **Set** statement to set **Font** is an object.
6. In the **Font Property Set** procedure, assign the **Font** value to the **Font** property for all of the **TextBox** and **Label** controls on the **UserControl** object.
7. Using a **Public Sub** procedure, add the method **Refresh** to the **UserControl** object.
8. In the **Refresh** method, call the **Timer** control's Timer event procedure.

   This will update the stock price when the **Refresh** method is called.

u Enable a property page for the Font property

1. In the **Procedure Attributes** dialog box, set the Property Page attribute for the **Font** property to StandardFont.

   This will let you use the **Standard Font** dialog box to change the property.
2. In the **Connect Property Page** dialog box, select the StandardFont property page.

   Your **Font** property can now be set by using the StandardFont property page.
3. Save the project.

u Test the control

1. Close all of the windows associated with the Stock project.
2. Open the Container form and select the **StockPrice** control.
3. In the **Properties** window, select the **Font** property and click the ellipsis (**…**)button.

   This should display the **Font** property page.
4. Change the font to **Bold**, and then click **OK**.

   The **TextBox** and **Label** controls should be bold.
5. Add two command buttons to the Container form, as shown in the following illustration.

   {ewc mvimg, mvimage,!v08g060.bmp}
6. Add code to the command buttons, as shown in the following code.

```
Private Sub cmdActivate_Click()
    'toggle Active property
    StockPrice1.Active = Not StockPrice1.Active
End Sub

Private Sub cmdRefresh_Click()
    StockPrice1.Refresh
End Sub
```

7. Run the Stock project, and type **MSFT** in the **Stock Ticker** text box.

   When you run the project, notice that the **Label** and **TextBox** controls are no longer bold. This is because the properties were not saved for the form.

8. Test each of the command buttons.

   The **Activate** button should refresh the **Stock** controls on and off.

   The **Refresh** button should refresh the stock price.

   If you are having problems with your solution, click this icon to see the **UserControl** object code from the lab solution.

   {ewc mvimg, mvimage,!code.bmp}

In this exercise, you will create a simple ActiveX control and test the control from a Standard EXE project. The ActiveX control will simulate returning stock price information.

u Create the ActiveX control

1. Create a new ActiveX Control project.
2. Name the project **Stock**.
3. Name the **UserControl** object **StockControl**.
4. Save the **UserControl** object and project files as Stock.ctl and Stock.vbp, respectively.
5. Add two **Label** controls, two **TextBox** controls, and one **Timer** control to the **UserControl** object, as shown in the following illustration.

   {ewc mvimg, mvimage,!v08g055.bmp}
6. Set the **Timer** control's **Interval** property to **1000**.
7. In the **Timer** control's Timer event, add the following code:

```
If txtStockTicker = "MSFT" Then
    'Return simulated stock price.
    txtStockPrice.TEXT = Rnd() * 200
Else 'unknown ticker
    txtStockPrice.TEXT = 0
End If
```

   Your text boxes may be named differently, so you should adjust your code accordingly.
8. Save the project.

u Test the ActiveX control

1. Add a new Standard EXE project to the project group.
2. Save the new form and project as Contain.frm and Contain.vbp, respectively.
3. Add a **StockControl** to the form.

   If the **Toolbox** icon for the **StockControl** is disabled, make sure all of the windows associated with the Stock project are closed.

   Notice that when the control is placed on the form, and the form is running, the number zero should appear in the **Stock Price** text box.
4. Run the .exe project.
5. In the **Stock Ticker** text box, enter **MSFT** (all caps).

   You should see new random values every two seconds in the **Stock Price** text box.

To build an ActiveX control, you perform the following steps.

1. Create the user interface for the control.
2. Provide the properties and methods of the control.
3. Define how the control will react to events.
4. Add property pages for the control.
5. Debug and test the control.

To see a demonstration of how to build a simple ActiveX control, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

In this exercise, you will add code to save and restore the **UserControl** object's property values.

Save and load property values

1. In the **UserControl_WriteProperties** event procedure, use the **WriteProperty** method to save the property values, as shown in the following table.

Property	Value	Default value
Active	Timer1.Enabled	**True**
Font	txtStockPrice.Font	(None)

2. In the **Property Let** procedures for the **Active** and **Font** procedures, call the **PropertyChanged** method.

   This will ensure that the properties are correctly saved.

3. In the **UserControl_ReadProperties** procedure, use the **ReadProperty** method to load the **Active** and **Font** properties.

   For more information about saving and loading properties, see Storing and Retrieving Property Values.

4. Save the project.

Test property save and load functionality

1. Close all of the windows associated with the Stock project.
2. Open the Container form and select the **StockPrice** control.
3. Set the **Active** property to **False**.
4. Set the **Font** property to **Bold**.
5. Run the project.

   The **Active** and **Font** properties should maintain their values.

   If you are having problems with your solution, click this icon to see the **UserControl** object code from the lab solution.

   {ewc mvimg, mvimage,!code.bmp}

If you plan to distribute an ActiveX control component created with Visual Basic, you need to create a setup program.

Using the Application Setup Wizard

The easiest way to create a setup program is to use the Application Setup Wizard because it ensures that all the necessary files to run the application are packaged and distributed safely.

The Application Setup Wizard works in the same way whether you create a setup program for an ActiveX component or for any other Visual Basic component. If you want to provide licensing support, you may need to include additional support files for any constituent controls, as well as a .vbl file.

If your application includes a large number of files, tracking those files can become a complex process. The Application Setup Wizard provides a simple and convenient way to track those files and create a Setup program.

To create a Setup program for ActiveX control components

1. In the properties for a control, specify whether or not you want to include licensing support for the control.
2. Compile the .ocx file.
3. Run the Application Setup Wizard.

How Setup Uses the Licensing File

If you have specified licensing support for your control, the Application Setup Wizard creates a Setup program that automatically registers any .vbl files in the system registry. The wizard also prompts you for any .vbl files on the user's computer that you want to install.

If you want to install a .vbl file, the Application Setup Wizard provides a default option for the destination folder. The default option is **Do not install this file**, because you usually want the user to have the license file registered but not installed on their computer. This option ensures that developers can include your controls in any application they distribute, while users of the application will only be able to access the run-time version of the controls.

To protect your ActiveX controls, Visual Basic provides licensing capabilities. Licensing your controls prevents other developers from using an instance of your control to create their own control.

How Licensing Works

Visual Basic provides the ability to include licensing support for your control by compiling a license key into the .ocx file and having the .ocx Setup program write the same key into the system registry.

When a user attempts to create an instance of a control at design time, the Setup program checks the registry for the value of the license key. If the Setup program does not find the key, it will not create a design-time instance of the control. However, if a compiled application includes a licensed control, a user can create a run-time version of the control without a license key.

Using Licensed Constituent Controls

To use licensed constituent controls as a part of a control, you must require users of your control to have the controls installed, or ask that the control vendor include the licensing key with the control's Setup program.

Adding Licensing to a Control

To add licensing to a control, you select the **Require License Key** option in the **Project Properties** dialog box, as shown in the following illustration.

{ewc mvimg, mvimage,!v08g050.bmp}

When you compile the .ocx file, Visual Basic creates a .vbl file that contains the licensing key for your component control. The Application Setup Wizard adds this key.

An ActiveX control includes a **UserControl** object and its constituent controls.

What Is a Constituent Control?

A constituent control is an instance of a control that you place on a **UserControl** object. When you place an ActiveX control on a form, an instance of the **UserControl** object is created, along with instances of any constituent controls you placed on the **UserControl** object.

The following illustration shows a **UserControl** object on which several constituent controls have been placed.

{ewc mvimg, mvimage,!v08g015.bmp}

Using Constituent Controls

You can use any standard Visual Basic control on the **UserControl**, except the OLE Container control. You add constituent controls to a **UserControl** in the same way you add controls to a standard Visual Basic form.

If you want to use controls that are not built into Visual Basic, you must include those controls on the **UserControl** object, and have the appropriate licensing rights to distribute the controls.

For information about distributing controls, see Distributing a Control.

Some controls can act as containers for other controls. For example, you can use a Visual Basic **Frame** control as a container for any constituent controls. If you select the **Frame** control and then move it, all of its constituent controls will move with the container.

To use an instance of an ActiveX control as a container for another developer's controls, you set the **ControlContainer** property of the **UserControl** object to **True**.

Container Control Performance

When you enable other developers to place controls on instances of your ActiveX control, it significantly affects the performance of the control. Contained controls must appear on top of all the constituent controls that are part of the **UserControl** object.

Because the ActiveX control is a container, any controls placed on it must be clipped. An outline of the ActiveX control is created in memory, and the Windows operating environment uses the outline paint different elements of the ActiveX control.

For these reasons, you should carefully evaluate whether you want to make your ActiveX control available as a container control.

ContainedControls Collection

An instance of an ActiveX control that acts as a container for other controls uses the **ContainedControls** collection to perform operations on any of the contained controls.

The **ContainedControls** property returns a collection of controls that a developer or an end user can add to the ActiveX control at run time. This property is not available when the control is created, and is displayed as read-only at run time.

The following code shows how to use the **ContainedControls** collection to return the names of controls contained in a control:

```
Dim CtrContained as Object
For Each CtrContained In _
        MyControl.ContainedControls
    MsgBox CtrContained.Name
Next
```

In this case, an instance of an ActiveX control contains two command buttons. If the control's **ControlContainer** property is set to **True**, the control will display the command buttons for this instance only.

The container for an ActiveX control supplies a number of default properties, so you should decide which additional properties you need for the control.

Creating a Property

To create a property for a control, you use property procedures. Creating a property for a control is similar to creating a property for a code component. For more information about creating properties, see Using Property Procedures to Create Properties in Chapter 7: Creating ActiveX Code Components.

With a control, you cannot use public variables to hold a property value. Instead, you must let Visual Basic know when a property value changes. To indicate that a property value has changed, you use the **PropertyChanged** method of the **UserControl** object within a **PropertyLet** or **PropertySet** statement.

The following code shows a property procedure that defines the **Name** property:

```
Public Property Get Name()As String
    Name = txtName.Text
End Property

Public Property Let Name(ByVal NewName As String)
    txtName.Text = NewName
    PropertyChanged "Name"
End Property
```

Exposing Properties of Constituent Controls

When a user works with your control, the properties of the **UserControl** object and of any constituent controls are not available at design time by default. You can, however, make the properties available through existing properties of the **UserControl** object or of any constituent controls you have placed on the object.

For example, if you create a control that contains a **Label** control and a **TextBox** control, you can expose the properties of either the **UserControl** object or the constituent controls by using property procedures.

The following code shows how to expose the **Caption** property of a **Label** control, which is a constituent control on a **UserControl** object:

```
Public Property Get Caption()As String
    Caption = lblName.Caption
End Property

Public Property Let Caption(ByVal NewCaption As String)
    lblName.Caption = NewCaption
    PropertyChanged "Caption"
End Property
```

When an instance of the **UserControl** object is placed on the form, the **Caption** property becomes available in the **Properties** window for that instance. A developer that uses your control can then set the value of the text that will appear in the label.

Mapping a Property to Multiple Controls

You can map more than one constituent control property to a property of your control through delegation.

For example, if you create a custom foreground color for the **UserControl** object, you can match this color in constituent controls, as shown in the following code:

```
Public Property Get ForeColor() As OLE_COLOR
    ForeColor = MyControl.ForeColor
End Property

Public Property Let ForeColor(ByVal NewfColor As OLE_COLOR)
```

```
    Dim ctlElement As Object
    MyControl.ForeColor = NewfColor
    For Each ctlElement In Controls
        If(TypeOf ctlElement Is Label) _
        Or (TypeOf ctlElement Is CheckBox) _
        Then ctlElement.ForeColor = NewfColor
    Next
    PropertyChanged "ForeColor"
End Property
```

As the author of a control, you cannot always predict which container will be used. Therefore, you should design your control so that it can function appropriately in a variety of containers. To ensure consistency between your control and its container, you can use the container's ambient properties.

What Is an Ambient Property?

Containers provide ambient properties to suggest behavior to controls that is appropriate for the container. Ambient properties ensure that the user of a control is provided with the expected visual appearance and behavior, regardless of the container.

For example, **BackColor** is one of the standard ambient properties of a Visual Basic form. The container is suggests the background color for the control so it matches the color of the container.

The AmbientChanged Event

The AmbientChanged event occurs when an ambient property is changed. To enable a control to respond to any changes made to the ambient properties of the container at design time, you place code in the AmbientChanged event. This event has one argument, **PropertyName**, that identifies which property has changed.

The Ambient Object

The **Ambient** object lets you read the values of the various ambient properties.

For example, the following code shows how to use the **Ambient** object in the AmbientChanged event:

```
Private Sub UserControl_AmbientChanged(PropertyName As String)
    If PropertyName = "BackColor" Then
        UserControl.BackColor = Ambient.BackColor
    End If
End Sub
```

If the developer who uses your control changes the **BackColor** property of the container, the background color of the control instance will match that of the container.

Supporting an Enabled Property

The **Enabled** property is an ambient property. To correctly support this property of the **UserControl** object, you must write code for the **Enabled** property procedures, and then associate the code with the **Enabled** property.

Because the container is responsible for enabling and disabling controls, you assign the **Enabled** property to the **Enabled** procedure ID, ensuring that the control does not interfere with other properties.

To assign the **Enabled** property to the **Enabled** procedure ID, you click **Procedure Attributes** on the **Tools** menu, and then select **Enabled** from the **Procedure ID** list.

Members of an enumeration do not need to be sequential or contiguous.

ActiveX controls can have an **About** property at the top of the **Properties** window, as shown in the following illustration.

{ewc mvimg, mvimage,!v08g020.bmp}

To add an About box to your control

1. Add an About form to the project.
2. Add a procedure to display the form.
3. Set the **Procedure ID** to **AboutBox**.

To display an **About** box for the control, you click the ellipsis button (**...**) next to the **About** property.

You can specify a custom bitmap to appear in the Visual Basic Control Toolbox, as shown in the following illustration.

{ewc mvimg, mvimage,!v08g025.bmp}

Toolbox bitmaps are 16 by 15 pixels in size. If you specify a bitmap of a different size, it will be scaled to 16 by 15 pixels, which will usually distort the image.

To specify a Toolbox bitmap, set the **ToolboxBitmap** property of the **UserControl** object to which you want to add the control.

With Visual Basic, you can add data-aware controls to a form. To display and edit the associated database, you set properties that bind controls to the data control. You can also support data binding by using ActiveX controls. For more information about using data controls, see Chapter 2: Using the Data Control.

Visual Basic lets you set options that mark control properties as bindable. Bindable properties provide a user with access to the data source; however, you must first add code that enable changes to be written to the database.

This section includes the following topics:

® Making a Control Bindable

® Updating Bound Properties

You can change the project type later by changing the project properties.

Visual Basic provides data-binding options for individual control properties, including the ability to let developers make your control properties bindable and your control with different data sources.

Marking a Property as Bindable

To enable data-binding options for properties, you use the **Procedure Attributes** dialog box. In the dialog box, you can select only one property to be bound to the field specified in the **DataField** property.

To see an illustration of how to use the data-binding options in the **Procedure Attributes** dialog box, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Binding a Property in the Client

Although you can bind only one property to the field specified in the **DataField** property, you can also designate properties of an ActiveX control as bindable by using the **Procedure Attributes** dialog box.

Making additional properties as bindable enables developers to place an instance of your control on a client, and bind the additional properties to a different data source.

Some properties of a control are provided by the container rather than the control. These properties are known as extender properties. The **DataBindings** collection, which is an extender property in a Visual Basic client, provides developers with access to the list of bindable properties of your control.

The **DataBindings** collection becomes available at design time when you have enabled the data-binding options of properties by using the **Procedure Attributes** dialog box.

You can enable developers who use your control to bind the properties of your control to a data source by marking the properties as bindable. However, binding the properties to a new data source will not enable users to update the data source.

In addition to marking properties as bindable, you must also add code so that users will be able to update those bound properties.

To see a demonstration of how to build a bound control that updates correctly, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Using Constituent Controls

If you are binding to a constituent control, you delegate to that control in the associated property's **Property Get** and **Property Let** procedures.

For example, if you have a **Name** property that applies to a text box on an **Address** control, you would map the **Name** property to the text box control.

CanPropertyChange Method

A constituent data-bound control should always call the **CanPropertyChange** method before changing the value of a property. In Visual Basic, this method always returns **True**, even if a database field is read-only. Visual Basic will not raise an error if a user tries to change a field to read-only.

PropertyChanged Method

In addition to calling the **CanPropertyChange** method, you must also call the **PropertyChanged** method in the constituent control's Change event. This will ensure that your control is bound for updating, and the data is written to the database.

The following code shows how the **LastName** property delegates to the txtLastName text box control, calls the **CanPropertyChange** method in the **Property Let** statement, and then calls the **PropertyChanged** method in the text box control's Change event:

```
Public Property Get LastName() As String
    LastName = txtLastName.Text
End Property

Public Property Let LastName(ByVal _
 NewLastName As String)
    If CanPropertyChange("LastName") Then
        txtLastName.Text = NewLastName
    End if
End Property

Private Sub txtLastName_Change()
    PropertyChanged "LastName"
End Sub
```

In this exercise, you will use the ActiveX Document Wizard to convert an existing forms-based project to an ActiveX document server.

You will convert the Main form and Order Details form in an existing database application to ActiveX documents. You will then test the new ActiveX document files by associating them with Internet Explorer, and invoking them.

u Run the ActiveX Document Wizard

1. In the Visual Basic environment, open the project in Lab10.
2. Use the ActiveX Document Wizard to create an ActiveX EXE project and convert the form frmHome to an ActiveX document.

   In the Project Explorer, the project should now consist of two forms (frmOrders and frmOrderDetails) and one user document. Modify the name of the user document to docHome.

u Modify the user document

1. Remove the **Exit Application** button from the docHome document.

   The **Exit Application** button does not provide any appropriate behavior in this context. If you view the underlying code for the Click event for the button's click event, you will see that the ActiveX Document Wizard has commented out the invalid code.
2. Save the project.

u Test the ActiveX Document in the container

1. Associate Visual Basic ActiveX (.vbd) files with Internet Explorer.
   a. Open either **My Computer** or the Windows Explorer.
   b. On the **View** menu, click **Options**, and then select the **File Types** property page.
   c. Add a new file type that associates .vbd files with Internet Explorer (IExplore.exe).
2. Switch to Visual Basic and run the project.

   There are not any visible forms because the output from running this project is the .vbd file.

Note While running the project in Visual Basic, the .vbd files are created in the Visual Basic folder, and when running exe or .dll files, the .vbd files are created in the .Exe or .Dll folders.

---

3. In the **My Computer** or Windows Explorer window, switch to the Visual Basic folder and run the ActiveX file docHome.vbd.

   This should cause the file to be loaded into your document container.
4. Once the .vbd file has been loaded properly, exit Internet Explorer, return to Visual Basic, and then explicitly end the program.

u Convert the Order Details form to an ActiveX document

1. Add a new module to the project.
2. Declare the public variable sDocPath to hold the document path.
3. In the docHome user document, add code to the Initialize event, and set the sDocPath variable to point to the Visual Basic folder (for example, C:\Program Files\Vb\).
4. Using the ActiveX Document Wizard, follow the same steps as before to convert the frmOrders form into an ActiveX document, and rename the document to docOrders.

u Add code for controls to navigate between the two documents

In the new document, the **Close** button is not appropriate in the current context. In this next procedure, you will change this button so that users can return to the docHome user document only if they have first invoked the docOrders document from the docHome document.

1. Change the **Caption** property of the **Close** button (cmdClose) to **Home**.
2. Add code to the user document for the InitProperties event to enable the **Home** button only if the path variable sDocPath has been initialized.

3. In the Click event for the **Home** button, use the **NavigateTo** method of the document's **Hyperlink** object to return to the docHome document.
4. Switch to the docHome document, and then update the Click event for the **Orders** button to navigate to the docOrders document.

u **Test the functionality of the application**

1. Click the **Orders** button on the main form to invoke the docOrders document.
2. Make sure that the database functionality still works correctly, including moving through, adding, modifying, and deleting records.
3. Invoke the Details form for any order.
4. Use the **Home** button to return to the docHome document.

The **Instancing** property in Visual Basic 5.0 includes the functionality of the **Public** property in Visual Basic 4.0.

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **45 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Note The database for this lab, Nwind.mdb, is located in the <*Install Folder*>\Labs\Lab10 directory and is accessed by the lab through a relative path. To ensure that the lab solution finds the database (that is, that the solution directory is the current directory), open the solution project in Visual Basic 5 by double-clicking the .vbp file in Explorer. If you start Visual Basic and then open the project, the Visual Basic installation directory will be the current directory and the database file will not be found.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 10.

Exercise 1: Converting Forms to ActiveX Documents

In this exercise, you will use the ActiveX Document Wizard to convert an existing forms-based project to an ActiveX document server.

Exercise 2: Adding Properties to ActiveX Documents

In this exercise, you will enhance ActiveX documents.

Exercise 3: Extending User Interface Functionality

In this exercise, you will enhance the user interface of the container based on the requirements of the document object. You will also modify one of the documents so that it is always centered within the container.

Exercise 4: (Optional) Creating a Document Container

In this exercise, you will modify the project to allow the executable file to run as a stand-alone application and to display the document objects within their own context.

By the end of this lab, you will be able to:

® Use the ActiveX Document Wizard to convert forms into an ActiveX document server.

® Add properties to an ActiveX document.

® Enhance the user interface of an ActiveX document.

® Create a document container.

Before working on this lab, you should be familiar with the following concepts:

- ActiveX document project fundamentals

To complete this lab, you need the following:

® Visual Basic 5.0 or later

® Internet Explorer 3.0 or later

The exercises in this lab use Internet Explorer, but they will also work with any valid document container.

When you debug a control (as opposed to another type of object), some of the control's code is actually running while an instance of the control is generated at design time.

There are several reasons for allowing code to run in Design mode.

® A developer can set the properties of your control.

® Property values can be read and saved.

® The design-time appearance of the control's instance can be displayed.

Creating a Test Project

Because an ActiveX control cannot run by itself, you must create another .ocx project to test the control project. Visual Basic enables you to load two or more projects into a project group, which is saved with the extension .vbg. You use a form in another project to run and test the control.

u **To add a test project to a project group**

1. On the **File** menu, click **Add Project**.

   Do not click **Open Project** or **New Project** because these commands will close your control project.

2. In the **Add Project** dialog box, double-click the **Standard EXE** icon to add an .exe project to your project group.

   You can now see both projects in the **Project** window. The caption of the **Project** window indicates the project group name, as shown in the following illustration.

   {ewc mvimg, mvimage,!v08g030.bmp}

To run a control's code at design time, you must close the control's visual designer. This causes Visual Basic to display the icon for the control in the Control Toolbox, so you can place instances of the control on a container form.

Debugging and Error Handling

Once you have created a test project for your ActiveX control, you can test and debug the control in the same way as you debug an in-process component. The difference between debugging an ActiveX control project and an in-process component is that the component code can run while the client is in Design mode.

In both cases, a seamless debugging process occurs between the component and the client within the same integrated development environment (IDE). You can step through code between the component and the client.

For information about debugging an ActiveX control, see Testing ActiveX Code Components in Chapter 7: Creating ActiveX Code Components.

You cannot guarantee how users of your control display the control, so you should consider various sizing scenarios when writing code for a control.

How a Control Is Displayed

The factors that determine how a control is displayed include the amount of available screen space, user preference, and the container used to display the control.

The following scenarios describe common ways in which controls are resized:

- A user resizes the control and its constituent controls at run time.
- The developer of a form that includes your control resizes the control and its constituent controls at design time.

A user may also resize a form that contains a **UserControl** object consisting of a command button, several text boxes, and a list box. Unless you have added the appropriate code to the Resize event, the text boxes and list box may not be completely visible, particularly if the user has resized the form to be smaller.

Note Containers cannot handle errors when the Resize event occurs, and will cause your the application using the control to fail. Therefore, you should provide error handling in Resize event procedures.

Adjusting the User Interface

You can use the Resize event to handle the resizing of controls and adjust the appearance of controls. The sizing of constituent controls is adjusted according to the current property settings for the **UserControl** object.

The following code sample adjusts the width of a text box when the **UserControl** is resized:

```
Private Sub UserControl_Resize()
    Const MinWide = 2000
    If UserControl.ScaleWidth < MinWide Then
       Text1.Width = 1000
    Else
        'scale textbox to size of usercontrol.
        Text1.Width = UserControl.ScaleWidth - Text1.Left
    End If
End Sub
```

Instances of controls are continually created and destroyed, so you must ensure that the property values can be preserved. When you create a control, you must include code that saves and retrieves property values of the control.

To see a demonstration of how to save and retrieve control properties, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Saving Property Values

To store and retrieve information each time an object is called, you use the **PropertyBag** object. To save and restore the state of an object, the **PropertyBag** object is passed through the **ReadProperties** and **WriteProperties** events.

The following code shows how to write current property values for the **WriteProperty** method with the **PropertyBag** object:

```
Private Sub UserControl_
  WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Caption", _
        Caption, Username
End Sub
```

Reading Property Values

To read property values, you use the ReadProperties event with the **ReadProperty** method of the **PropertyBag** object.

The **ReadProperty** method takes two arguments: a string designating the property name and a default value. If a property value has been saved, the **ReadProperty** method returns the value. If a property value has not been saved, the method returns the default value.

The following example shows how to use the **ReadProperty** method to return the saved value of the **Caption** property:

```
Private Sub UserControl _
  ReadProperties(PropBag As PropertyBag)
'Trap for invalid property values
    On Error Resume Next
    Caption = PropBag.ReadProperty "Caption", _
        Username
'...return values of additional properties..
End Sub
```

Default Property Values

When you read and write property values, it is important to provide default values. Visual Basic writes a line of code in the source file (.frm, .dob, .pag, or .ctl) of the control's container only if the property value differs from the default value you have provided. As a result of providing a default value, the file size is reduced and application performance is improved.

Initial Property Values

The first time an instance of a control is placed on a container, it receives the InitProperties event. Thereafter, only the ReadProperties event occurs.

When future instances of the control are placed on the container, you set the initial values for a property in the **InitProperties** event, and use the same default value that you provide with the **WriteProperty** and **ReadProperty** methods to save and retrieve the property value.

Visual Basic includes the ActiveX Control Interface Wizard, which simplifies the task of creating a control. This wizard lets you determine which properties, methods, and events will constitute the interface definition of your control.

The ActiveX Control Interface Wizard maps functionality from your control to functionality of the **UserControl** object or constituent controls. You can change the default mappings later in the process.

In addition, this wizard generates the underlying interface code, including:

® Property procedure code to implement procedures.

® **Sub** and **Function** procedures to implement methods.

® Code to raise the events you have selected.

The wizard generates the correct arguments and data types for standard events and event forwarding code for all your event mappings.

To see a demonstration of how to use the ActiveX Control Interface Wizard, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

With Visual Basic version 5.0, you can use enumerations to provide named constants for a component. Enumerations provide a convenient way to group a set of related named constants and associate them with constant values. To display constants, you can use enumerations to write code for a control so it appears in the **Auto List Members** drop-down list box.

You make the members of an enumeration available to users of your control by declaring the enumeration as **Public**. Include the enumeration in any public module that defines a class, such as a class module, UserControl, or UserDocument.

The following code defines an enumeration that contains named constants representing the Celsius boiling temperatures of various substances:

```
Public Enum TempBoilCelsius
    msubWater = 100
    msubIron = 2750
    msubNitrogen = -195.8
    msubGold = 2807
End Enum
```

You can use this enumeration as the data type of the property **MyTemp**, as shown in the following illustration.

{ewc mvimg, mvimage,!v08g035.bmp}

In this exercise, you will modify a project so the executable file can run as a stand-alone application, and display document objects within their own context.

The **WebBrowser** control included with Visual Basic lets you add browsing capabilities to your application. This includes the ability to act as a **Document** object container.

u Add capabilities to a Document object container

1. Create a form as the parent of the **WebBrowser** control. Display this form only if the application is being started in stand-alone mode.
2. Add a form named frmContainer to the project.
3. Add the **WebBrowser** control to the form.

   If the **WebBrowser** control is not available in the Toolbox, click **Components** on the **Project** menu, and then select **Microsoft Internet Controls** on the **Controls** tab.
4. In the Form_Load event, use the **WebBrowser** control to navigate to the docHome document object in the Visual Basic folder, as shown in the following code:

   ```
   WebBrowser1.Navigate "c:\program files\devstudio\vb\docHome.vbd"
   ```

5. In the form's Resize event, move the **WebBrowser** control to fill the client area of the form.

u Determine the Startup mode of the application

Add a **Sub Main** procedure to the Standard module of a project to test whether or not the application is being started in stand-alone mode, and take an appropriate action.

1. In stand-alone mode, display the application in the new container form.
2. Add a **Sub Main** procedure to the Standard module of the project.
3. In the procedure, use the App's **StartMode** property to check whether the application is starting in standalone mode, in which case you should display the container form.

u Test the project in stand-alone mode

1. Open the Project Settings property page, and then modify the project so it will run in stand-alone mode.
2. On the **General** tab, set the Start Object to **Sub Main**.
3. On the **Components** tab, set the Start Mode to **Standalone**.
4. Run and test the application.

You can implement custom methods or delegate to existing methods of constituent controls, just as you can with properties.

Creating a Method

Creating methods for controls is the same as creating a method for a code component. You declare a **Public Sub** or **Function** procedure, and call it in the code module of the **UserControl** object. For more information about creating a method, see Creating Methods in Chapter 7: Creating ActiveX Code Components.

The following code creates a method that displays the date:

```
Public Sub ShowDate()
    MsgBox "Date is: " & Now()
End Sub
```

Exposing Methods of Constituent Controls

You can also expose methods of any constituent controls that are part of your control; the container application can call the methods.

The following code creates the method **IDFocus**, and invokes it in the Click event of a button on a container form:

```
Public Sub IDFocus()
    txtEmpID.SetFocus
End Sub

Private Sub cmdSetIdFocus_Click()
    MyControl.IDFocus
End Sub
```

In this code, the **TextBox** control txtEmpID is a consituent control of the **UserControl** object. When a user clicks the button on the container form, the focus will be automatically set to txtEmpID on the **UserControl** object.

You can raise an event for a control just as you would for a code component. The events your control can raise include Click, DblClick, KeyDown, KeyPress, KeyUp, MouseDown, MouseMove, MouseUp, and any other control specific events. For information about raising events, see Using Events in Chapter 7: Creating ActiveX Code Components.

Received Events vs. Raised Events

There is an important distinction between the events received by your **UserControl** object (or by its constituent controls) and the events your control raises. You use the events your control receives to add functionality to your control.

A developer that uses your control can use events the control raises to add functionality to the control.

Raising a Control Event

To raise an event from a control, you first declare the event, and then use the **RaiseEvent** statement to call the event.

The following code adds a Click event to a **UserControl** object:

```
'Declare a public Click
'event with no arguments.
Public Event Click()
```

The following line of code raises the Click event:

```
RaiseEvent Click
```

Exposing Constituent Control Events

Unlike exposing properties and methods by delegation, you expose an event in a constituent control by raising your own event.

To expose the events of a control on a UserControl object

1. Declare a new event in the **UserControl** object.
2. In the constituent control's event, raise your own event.

In the following code, the KeyPress event from a **TextBox** control is exposed. This is the code for **TextBox** control:

```
Public Event MyKeyPress(KeyAscii As Integer)

Private Sub Text1_KeyPress(KeyAscii As Integer)
    RaiseEvent MyKeyPress (KeyAscii)
End Sub
```

Notice that arguments to the event are also passed to the client.

This is the code for the client of the control:

```
Private Sub MyControl_MyKeyPress(KeyAscii As Integer)
    'Convert the character to uppercase.
    'Since KeyAscii is passed ByRef it can be changed
    'in the client.
    KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

Property pages enable you to define a custom interface for setting properties. This gives you more flexibility than using the **Properties** window.

The following illustration shows a custom **Property Pages** dialog box.

{ewc mvimg, mvimage,!v08g040.bmp}

To create a property page, you first draw the elements of an interface one page at a time. The tabs and buttons in the **Property Pages** dialog box are created for you.

You add a property page to a project by following these steps.

u **To add a property page to your control project**

1. On the **Project** menu, click **Add Property Page**.
2. On the **New** tab of the **Add Property Page** dialog box, double-click the **Property Page** icon.

   This displays a visual designer, which you use to draw controls as you would on a standard form.

The following illustration shows a sample property page with some controls drawn on it.

{ewc mvimg, mvimage,!v08g045.bmp}

A control can use one or more property pages in an ActiveX control project. However, you must first establish a connection between the control and the property page.

To see a demonstration of how to create and use a property page, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Assigning Property Pages to a Control

To assign one or more property pages to a control, you use the **Connect Property Pages** dialog box. To display the **Connect Property Pages** dialog box, you double-click the **PropertyPages** property or the ellipsis (**...**) button for the control in the **Properties** window.

To see an illustration of how to gain access to the **Connect Property Pages** dialog box from the **Properties** window, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

Associating a Property Page with a Property

Some properties are objects with properties of their own. Other properties consist of an array of values. When a property is too complex to set from the **Properties** window, you can use a property page to display all of the values for the property.

By associating a property with a property page, you can make all the options that use the property more available. You can identify a property that is associated with a property page by the ellipsis (**...**) button next to the property in the **Properties** window. When you click this button, it opens the property's associated property page.

u **To associate a property page with a property**

1. In the **Project** window, right-click the **UserControl** object, and then click **View Code** to open the code window.
2. On the **Tools** menu, click **Procedure Attributes**, and then click **Advanced**.

   This opens the **Procedure Attributes** dialog box to display additional options.
3. In the **Name** box, select the property you want to associate with a property page.
4. In the list **Use this Page in Property Browser**, select the property page, and then click the **Apply** button or **OK**.

Standard interfaces can be developed in either one of two ways: You can define them either by creating a type library with the MIDL compiler, or by compiling a Visual Basic project containing abstract classes.

```
Private Sub Command2_Click()
    xl.Range("Growth").Value = Text1.Text
    xl.Range("Inflation").Value = Text2.Text
End Sub
```

```
Private Sub Command1_Click()
    set xl = CreateObject("Excel.Application")
    xl.Workbooks.Open App.Path & "\Earn.xls"
    xl.Visible = True
End Sub
```

```
Private Sub Command3_Click()
    xl.Range("Net_Profit").Select
    Set xlChart = xl.Charts.Add()
    xlChart.Type = xl3DColumn
End Sub
```

Automation is a technology for manipulating objects that are defined by an application or library from outside the application. While COM is both a specification and a type of implementation, Automation is the part of COM that enables objects to be created and programmed by clients in a standard way.

Automation works through a standard interface known as **IDispatch**. This interface exports any number of properties and methods supported by an object. In general, an Automation client can use any component that exposes an **IDispatch** interface.

This section discusses Automation objects. It describes:

® The relationship between objects and type libraries.

® How to view the contents of a type library by using an Object Browser.

® How objects can be arranged into logical structures called Object models.

® Which interfaces can be used to access properties and invoke the methods of objects.

® The way in which you connect (or bind) a client to an object by using an object variable.

® Why you should use the **Set** statement or the **CreateObject** function, rather than declaring and defining a variable.

This section includes the following topics:

® Automation Objects

® Type Libraries

® The Object Browser

® Object Models

® The IDispatch Interface

® Dual Interfaces

® Introduction to Binding

```
Dim xl as Excel.Application
Dim xlSheet as Excel.Worksheet
Set xl = CreateObject("Excel.Application")
Set xlSheet = xl.Workbooks("BOOK1.XLS").Worksheets("SHEET1")
xlSheet.Range("A1").Value = 5
```

```
Dim xl as Excel.Application
Dim xlwb as Excel.Workbook
On Error Resume Next
set xl = CreateObject("Excel.Application")
set xlwb = xl.workbooks.Open("c:\book1.xls")
If Err <> 0 The
    Msgbox "Unable to open workbook"
    Unload Me
End If
```

```
For each xlwb in xl.Workbooks
    if xlwb.Name = "MyBook.xls" Then
        bookfound = True
    End If
Next xlwb
```

```
set xl = CreateObject("Excel.Application")
'Explicitly state the object hierarchy
xl.Workbooks("x.xls").Worksheets("s").Range("A1").Value = 5

'Assume the current active workbook and worksheet
xl.Range("A1").Value = 5

'Establish object variables to reference objects
Set wb = xl.Workbooks("source.xls")
Set ws = wb.Worksheets("sheet1")
Set rg = ws.Range("A1")
rg.Value = 5
```

```
Dim r as Range
For each r in xl.Range("summary")
    r.value = 25
Next r
```

```
Sub cmdCreateChart_Click()
    Dim xl as Excel.Application
    Dim xlc as Excel.Chart
    Set xl = CreateObject("Excel.Application")
    xl.Visible = True
    xl.Workbooks.Add
    xl.Range("A1").Value = 3
    xl.Range("A2").Value = 2
    xl.Range("A1:A2").Select
    Set xlc = xl.Charts.Add
    xlc.Type = xl3DColumn
End Sub
```

Component technology helps you create applications more efficiently, because it provides advantages such as code compatibility, reusability, and version control.

Binary Compatibility and Cross-Platform Development

The first major advantage of component technology is code compatibility across computing platforms. Since becoming an industry-wide standard, COM-compliant client applications and servers are independent of the development language and operating system.

COM ensures complete binary compatibility between the client you develop in Microsoft Visual Basic and the components developed in Microsoft Visual C++ or Microsoft Visual J++, or components running on a Macintosh or a UNIX server.

Code Reusability

The second major advantage of component technology is an enhancement of DLL technology. COM components expose groups of methods, known as interfaces, through which clients interact with objects. Because these interfaces are documented, the code that creates those objects can be reused by many clients for a variety of different purposes.

Version Control

The last major advantage resolves issues related to version control. COM-compliant components are said to be self-versioning. This means that new functionality can be added to a component (by adding or changing the implementation of an interface) without affecting clients that already use the component. Functionality is not lost when components are upgraded; it is always enhanced or added.

Because COM is built into the operating system, it implements communication between client applications and servers that provide components. As an industry standard, COM is built into several operating systems and compilers.

This topic discusses the mechanisms that enable communication between clients and servers, and the role of the system registry in this communication.

Communication Between a Client and an In-Process Server

At the operating system level, the speed of communication between a client and server components varies, depending on whether the server is in-process (a DLL) or out-of-process (an .exe file).

Communication between a client and an in-process server component provides the most efficient exchange of information. Because the client and the server component share the same address space, calls are made directly between the two.

Communication Between a Client and an Out-of-Process Server

Communication between a client and an out-of-process server component is more complex than an in-process server, because it has to cross process boundaries. Communication between a client and an executable file on a remote computer is even more complex.

To manage this complexity, a COM-enabled compiler inserts proxy code in a client to handle any server calls and their return values. When an author compiles a COM-compliant component server, stub code is inserted to handle calls from the client proxy code and any return values to clients.

This handling of parameter and return values is known as marshalling. Marshalling is the packaging of all parameters and return values and the sending of them back and forth across process boundaries.

The following illustration shows the flow of information from a client to a server through proxy and stub code.

{ewc mvimg, mvimage,!v06g003.bmp}

This implementation of interprocess communication is platform-independent because it is based on the Distributed Computing Environment's Remote Procedure Call (DCE-RPC) specification. Calls between COM-compliant components can be made between Windows 95, Windows NT, UNIX, and Macintosh environments, as well as other environments that support DCE-RPC.

For more information about how COM is implemented, see The Component Object Model Specification in the development section of Microsoft.com.
{ewc mvimg, mvimage,!intjump.bmp}

COM, GUIDs, and the Registry

When a component is compiled, the compiler creates globally unique identifiers (GUIDs) for each public class and interface in the component. These GUIDs are 128-bit numbers that are generated using an algorithm designed to ensure uniqueness worldwide.

For a component to become available for clients, it must register itself with the operating system. It does this by updating the HKEY_CLASSES_ROOT section of the system registry with information about itself, including its program ID (ProgID), which is a human readable name for the component, its class ID (ClassID), its location, and the location of its type library.

When you compile a client application that references the server component, the class IDs and interface IDs of any objects that the component creates are included in the executable file. At run time, the client (in conjunction with COM) uses the information in the registry to call the server component and request that the component create an object that returns a default interface to the client.

For more information about the relationship among client applications, COM, the registry, and object creation, see Creating Objects and Type Libraries.

To see a demonstration that shows the subdirectories (also known as hives) relevant to COM and the creation of classes and ClassIDs, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Automation objects are any type of code object that offers client applications access to its properties and methods.

Properties describe the characteristics of an object, such as its name, width, or background color. You create properties by using public variables or property procedures.

Methods are requests to an object to perform an action. For example, to add an item to a list box, you use the **AddItem** method of the list box.

Note Although Automation objects can have events, they are not discussed in this topic. Automation involves a client communicating to an object through interaction with the object's properties and methods.

Events, on the other hand, involve communication from the object back to the client. This communication is a distinctly different issue and is addressed in the topics Notifying Clients and Server Notifications Using Events.

The dispatch interface, **IDispatch**, is a standard Automation interface designed explicitly for exposing component methods and properties to a client. This interface contains four functions: **GetTypeInfoCount**, **GetTypeInfo**, **GetIDsOfNames**, and **Invoke**.

The functions **GetTypeInfoCount** and **GetTypeInfo** obtain information from the component's type library about the interfaces, methods, and properties that it supports.

GetIDsOfNames takes one or more properties and/or methods, and returns their dispatch ID (**dispID**) values as defined in the type library. **Invoke** takes a **dispID** and a pointer to an array of parameters, and causes the object to execute the associated property or method.

The following illustration shows the functions contained in **IDispatch**, along with additional functions supported by the interface.

{ewc mvimg, mvimage,!v06g012.bmp}

IDispatch enables a client to perform the following tasks.

® Query an object from its type library.

® Call a property or method name in the form of a string to obtain the associated dispatch ID (**dispID**).

® Specify the value of a dispatch ID to call the properties and methods of the object.

As with all COM interfaces, **IDispatch** supports all the functionality of the **IUnknown** interface.

The following illustration shows how the **IDispatch** interface works.

{ewc mvimg, mvimage,!v06g035.bmp}

For more information about **IDispatch** and **dispIDs**, see Introduction to Binding and Using Interfaces.

For a client to use a method or property of an object, it must be bound to the object. Binding defines how a client connects to a particular object.

How a client is bound to an object is primarily determined by how you as the developer of a client application declare object variables.

Binding can take place either at run time or at compile time, so there are two types of binding: late binding and early binding.

Late Binding

Late binding occurs when the compiler does not know about an object until run time.

When you use late binding, the client must make two calls to the object to access any method or property. The first call is made to the **GetIDsOfNames** method of the **IDispatch** interface to query the names of the methods and properties that are available from the object. The second call is made to **Invoke** to perform the desired action.

To see an animation on how late binding works, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

Early Binding

In contrast, early binding occurs when Visual Basic knows at compile time exactly which interface will be used with an object variable.

Because the compiler has access to the object's type library, it can check the syntax of any calls that use that object variable, and notify you of any syntax errors at design time.

The compiler can also modify your code so that it optimizes execution speed when accessing the object. This optimization can be accomplished in one of two ways: by **dispID** binding or by **vtable** binding.

DispID Binding

With **dispID** binding, the compiler obtains the **dispID** from the object's type library and modifies all calls to the object by calling **Invoke** with the appropriate **dispID**. Therefore, the executable eliminates the calls to **GetIDsOfNames** for each method or property that is accessed.

To see an animation on how **dispID** binding works, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

Vtable Binding

A faster form of early binding is known as **vtable** binding. **Vtable** binding is a more efficient form of binding, because it avoids making the two calls to **GetIDsOfNames** and **Invoke**. Visual Basic does not make any function calls at all. Instead, it uses an offset (or a pointer) to a virtual function table (**vtable**) to access methods and properties of an object directly.

In the COM specification, servers provide **vtable** binding by default. If a client application declares variables by using explicit class names, Visual Basic will always use **vtable** binding. Using **vtable** binding to call a method or property in an in-process Visual Basic component does not require any additional overhead than calling a function in a DLL.

To see an animation on how **vtable** binding works, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

For more information about the effects of different types of binding on performance, see Declaring Object Variables and Creating Objects.

When a server is developed, the author determines how many instances of the server component will be created at run time and where the server will be run. These decisions can significantly influence the performance and security of your client.

This section explains aspects of server components that you should understand when developing effective client applications.

This section includes the following topics:

® Setting the Instancing Property

® Determining Where Server Components Run

® Notifying Clients

When a component is designed, its author must set its **Instancing** property to specify whether or not the object will be available for use by client applications, and whether or not multiple instances of the object can be created.

The following table defines the different **Instancing** property settings that are available for components that client applications will use.

Setting	Result
MultiUse	A new instance of the component starts each time a client requests services.
SingleUse	Multiple clients are served by one instance of the component.

Although a client does not have any control over the instancing of the component, the setting for the **Instancing** property can significantly influence the performance of the client. For example, if your client must share an instance of a component, its performance may be slower.

The last feature of components that you should understand is their ability to communicate with clients asynchronously. This is a new feature for the components in Visual Basic. Earlier versions of Visual Basic enabled only synchronous processing.

When a component used to retrieve property values and invoke methods, the component immediately returned values to the client. However, if you wanted the component to perform a task that you knew will need several seconds or minutes to complete, you would notify the client asynchronously when the server completed some task.

In this version of Visual Basic, when a component needs to communicate with a client, the component uses an outgoing interface. This means that the component defines the interface, but the client implements it. The server communicates with the client by retrieving a pointer to the client's implementation of this interface, and then makes the appropriate calls.

The following illustration shows the communication between a component and a client.

{ewc mvimg, mvimage,!v06g038.bmp}

Outgoing communication can take the form of firing events or calling a specific function in the client. This is known as a callback function.

For more information about how events are managed by a client, or how callback functions are implemented, see Server Notifications Using Events or Server Notifications Using Callbacks.

This section describes how to create a component. It also includes information about how to retrieve references to additional interfaces on a component.

COM components are created with tools such as Visual Basic, Microsoft Visual C++, and Microsoft Visual J++. Visual Basic offers a rich feature set, while maintaining its rapid application development environment roots.

Creating a client application in Visual Basic requires four simple steps.

1. Set a reference to the type library of a component.
2. Declare an object variable.
3. Create an object.
4. Use the object's methods and properties.

This section includes the following topics:

® Setting References

® Declaring Object Variables

® Creating Objects

® Using Automation Objects

® Getting to Additional Interfaces

When you create a client application, you connect the client to a component. The first step in connecting a client to a component is to set a reference to the component's type library.

u To set a reference to a type library

1. On the **Project** menu, click **References**.
2. In the **References** dialog box, select the type library you want to connect to, and then click OK. If the reference is not listed in the dialog box, and the component is available, click the **Browse** button to add the reference.

Once the component is referenced, you can open the Object Browser to learn about the objects, interfaces, methods, properties, events, and constants exposed by the component.

For more information about type libraries and object browsers, see Implementing Automation.

ActiveX documents offer many advantages and increased functionality to both the user and the application developer.

Advantages to Users

ActiveX documents enable a generic container application to host a variety of different document types. With ActiveX documents, there is less of a distinction between a document and an application. This means that users do not need to know which applications operate on which types of data. Because this aspect is transparent to users, they can focus on the tasks that they want to accomplish, rather than the application itself.

Web browsers, such as Internet Explorer, are widely available as document container applications for ActiveX documents, which makes it easy for users to access the documents.

ActiveX documents can run on local computers. This enables compute-intensive processing to take place locally, and reduces the processing operations on a network.

The Web plays an important role in extending the use of ActiveX documents. Using ActiveX documents on the Web, users work with information in any number of formats through common browser functionality.

Advantages to Developers

The most important advantage for developers in using ActiveX documents is the ability to apply existing programming skills. Developers can create ActiveX documents by using the programming language and environment with which they are already familiar. For example, developers can build ActiveX documents with the programming languages Microsoft Visual Basic 5.0, Microsoft Visual Java ++, and Microsoft Visual C++.

When you create an ActiveX document in Visual Basic, you receive immediate visual feedback about the layout of elements in your application. You use the same forms-based technology as you do in a stand-alone Visual Basic application. You can apply your existing knowledge of Visual Basic to create an ActiveX document, instead of writing code for an HTML document.

Interfaces are groups of functions that provide connection points through which clients and server components communicate. Interfaces provide standardized access to the methods and properties (functionality) available from servers. Further, they are a contract between the component author and the client developer that ensures consistent access to functionality. Finally, they structure that access so that servers are easier to use.
{ewc mvimg, mvimage,!tip.bmp}

The following illustration shows you how clients and server components communicate through interfaces.

{ewc mvimg, mvimage,!v06g033.bmp}

An object can have multiple interfaces that provide access to different functionality for different clients.

This illustration shows a credit card object with two interfaces, **IRetail** and **IManager**, that provide different functionality to a retail store client and a credit manager client, respectively.

{ewc mvimg, mvimage,!v06g040.bmp}

Once a client uses a particular interface, and that interface is not changed, the client will continue to communicate, even if the component has been modified.

If you want a component to provide different functionality, you must use a new interface. Clients will not cease to communicate, even when new or upgraded clients take advantage of new features.

Callbacks provide the same kind of functionality as events, but they give the component a great deal more control over clients.

Before you use a callback function, you should understand the **Implements** keyword, which is new to Visual Basic 5.0. Callback routines on the component side are created by the component developer, who creates an empty interface template that will be implemented by the developer who creates the client.

Note Standard interfaces are created by compiling abstract classes (class modules with procedure declarations but no implementation) in ActiveX DLLs or EXEs, or with the Microsoft Interface Definition Language (MIDL) compiler, which is included in the Tools folder of Visual Basic.

Experienced Visual C++ users may prefer using the MIDL compiler to define an interface. Basic programmers may want to create an interface by using a Visual Basic class module.

Using the Implements Keyword

To implement the interface definition in the component, you first set a reference to that component.

After a reference is set, you can use the Object Browser to find the interface definition. You can then use the **Implements** keyword in a class module to declare an additional interface.

For example, assume you want to implement the interface **ISomeInterface** with the methods **Method1** and **Method2**. The following code shows how you would add these methods to a class module.

```
Implements ISomeInterface    ' This declares the interface.

Public Sub ISomeInterface_Method1()
    ' Provide some implementation for method1.
End Sub

Public Sub ISomeInterface_Method2()
    ' Provide some implementation for method2.
End Sub
```

Implementing Callback Functionality

Callback notification requires the client to implement an explicit interface defined by the component, and to pass a reference to that interface back to the component.

The server component provides the following two pieces of code.

® The name of a custom interface whose functions will act as callbacks to the client.

® A public function (generally a write-only property) through which the client will pass a reference to the interface that contains the callbacks.

To receive callback notifications from a component, the client application must take the following steps.

1. Ensure that the outgoing interface from the component is recognized. In Visual Basic, a reference must be set to the component.
2. In the client application, add a class module that implements the server's outgoing interface by using the **Implements** keyword, and provide implementation for each of the functions defined in the interface.
3. In an appropriate module, add code that declares an object variable to hold the component and create an instance of it.
4. Add code that declares an object variable of the class in which the outgoing interface was implemented and creates an instance of that class.
5. Pass to the object (by using an appropriate method or property) the reference to the class that implements the outgoing interface. This gives the component the ability to call back to the client through the interface implemented in the class module.

The following code, contained in the class module **CNotifyee**, implements the outgoing interface **INotifyProc**.

```
Option Explicit

Implements INotifyProc       ' Interface defined by server.

Public Sub INotifyProc_SleepEntered()
  MsgBox "Callback - Sleep Entered"
End Sub

Public Sub INotifyProc_SleepEnded()
  MsgBox "Callback - Sleep Ended"
End Sub
```

To complete the callback functionality, the following code creates the component object, creates the callback object, and passes an appropriate reference to the component.

```
Option Explicit
  ' Declare the notification object.
  Dim objNotifyee As Notifyee
  ' Declare the notifier object.
  Dim objNotifier As CNotifier

Private Sub Form_Load()
  Set objNotifier = New CNotifier
  Set objNotifyee = New Notifyee
  ' Pass a reference to the callback object to the server.
  objNotifier.NotifyProc = objNotifyee
End Sub

Private Sub cmdExit_Click()
  Unload Me
End Sub
Private Sub cmdSleep_Click()
  objNotifier.Interval = txtInterval
  objNotifier.Slumber
End Sub
```

Most objects provide more than one interface. Components often implement multiple interfaces, where each interface provides additional functionality.

To retrieve additional functionality, you declare an additional variable of the interface type, and then set the new variable by using the existing object variable.

For example, assume that a spelling-checker component named Speller supports two interfaces: **Check** and **IManage**. **Check** is the Automation interface used to check spelling and suggest fixes. **IManage** is used to add and delete words from the dictionary. The client accesses both interfaces with the following code.

```
Dim check as Speller.Check
Dim mng as IManage

Set check = New Speller.Check  ' This creates the object and
                               ' increments its usage count.
Set mng = check                ' This returns the IManage
                               ' interface and increments the
                               ' usage count to 2.
```

How do you know what events are available in a component? You find them in the events procedures listbox of a code window after you have declared a variable of a given type using the **WithEvents** keyword. This keyword declares a hidden or implicit events interface.

In general, events are easier to implement than callback routines, but callbacks provide greater control. This topic describes the strengths and weakness of each.

The primary difference between events and callback routines is that an event is like an anonymous broadcast, while a callback is like a direct handshake. An event in a component does not know anything about the client applications using it, while a callback routine on a component knows a great deal about the client using it.

The following list contrasts characteristics of events and callbacks.

® Any object that has a reference to a component that provides an event can handle the event by adding **WithEvents** to the variable declaration. The component that contains that event has no idea what other objects may be handling its events because it is broadcasting to an unknown audience.

  By contrast, a component that contains callback routines must provide a reference to every object that it will call. The component knows exactly how many objects are referencing it.

® A component that provides events does not have any control over the order in which client applications receive its events.

  By contrast, a component that uses callbacks can control the order in which it calls client applications back. For example, the component might give some clients a higher priority.

® When a component raises an event, all of its clients handle the event before the server regains control.

  By contrast, a component that uses callbacks regains control after each call to a client.

® If an event contains a **ByRef** argument, that argument can be modified by any client that handles the event. The component will see only the last changes made by a client because the component does not regain control until all clients have handled the event.

  By contrast, a component that provides a callback routine can see any changes made to **ByRef** arguments after each call to a client, and can pass the next client fresh values for those arguments.

® If an error occurs in a client's event handler, the component that provided the event will be unaware of the error.

  By contrast, a component that provides a callback routine can receive errors that occur in the callback routine of the client, and can be prepared to handle them.

In conclusion, you should use events instead of callback routines when the component providing the notifications:

® Can broadcast them anonymously.

® Does not care in what order clients receive them.

® Does not need to get control back between calls to the clients.

® Can use the last known value of any **ByRef** argument.

® Does not have to handle errors from the client.

If any one of these conditions are not met, then the component should provide notifications by using callback routines.

When performance is critical, you may also want to use callbacks instead of events. Callback routines can be faster than events because callbacks, unlike events, can be **vtable** bound. This will be much more noticeable with an in-process component.

If your ActiveX document project uses a browser as its container, you must provide navigational capabilities between multiple documents.

This section describes how to provide document navigation functionality for your ActiveX documents. It also explains how to set up a notification system that identifies which document is opened so your ActiveX document application can take an appropriate action.

In addition, the section covers how to create and refer to a public property in an ActiveX document, and how to ensure that global references are handled properly and do not consume resources. The section also explains how use the **PropertyBag** object to persist data between documents.

This section includes the following topics:

® Navigating Between Documents

® Using Global References

® Adding Properties

® Persisting Properties

You may not know which container a user will view an ActiveX document with, so you should plan accordingly. For example, you may need to notify users that the selected container lacks certain features required by the ActiveX document, and suggest an alternative container.

The following list provides some important differences between containers.

® Internet Explorer caches documents in memory, while Microsoft Office Binder does not.

® In the Office Binder, adding a new section to a binder creates a new object from the ActiveX document class. By adding a new section to a binder, the InitProperties event will always occur, rather than the ReadProperties event.

® In Internet Explorer, the Show and Hide events occur as described in the previous topic.

This topic describes how to write code for the events used by the property page you create.

Property pages and forms are similar in appearance, and you create them in a similar fashion. However, property pages and forms use different events, and as a result they function differently.

SelectionChanged Event

The SelectionChanged event occurs whenever a property page is opened, and when the list of currently selected controls changes. You use this event to set the values of the controls that display property values, and to edit them.

The following code assigns the value of a **Timer** control's **Interval** property to a text box on the property page:

```
Private Sub PropertyPage_SelectionChanged()
     txtTimerInterval.Text = _
         SelectedControls(0).Interval
End Sub
```

SelectedControls Collection

The **SelectedControls** collection contains all controls in a container that have been currently selected by the user of the control. The collection may contain several instances of your control.

To determine whether multiple controls have been selected, you use the **Count** property with the **SelectedControls** collection. A return value greater than 1 indicates that multiple controls have been currently selected in the container.

Changed Property

To notify Visual Basic of changes to properties on a property page, you set the **Changed** property of the **PropertyPage** object to **True**. This also enables the **Apply** button in the **Property Pages** dialog box.

For example, the following code notifies Visual Basic that the value of the **Interval** property displayed in a text box has been changed on the property page:

```
Private Sub txtTimerInterval_Change()
    Changed = True
End Sub
```

ApplyChanges Event

To write any changed property values back to the currently selected controls, you use the **ApplyChanges** event.

The **ApplyChanges** event occurs in the following situations.

® The **OK** button is clicked to dismiss the **Property Pages** dialog box.

® The **Apply** button is clicked.

® Another tab in the **Property Pages** dialog box is selected.

The following code declares the variable objControl to refer to whichever control is currently selected, and applies changes to the **Interval** property of the control's property page:

```
Private Sub PropertyPage_ApplyChanges()
    Dim objControl As Variant
    For Each objControl In SelectedControls
        objControl.Interval = txtTimerInterval.Text
    Next
End Sub
```

Standard property pages let you use existing functionality to provide a familiar user interface. The interface enables users to work with the more complex properties, such as those pertaining to font style and color.

Visual Basic provides the following standard property pages:

® StandardFont

® StandardColor

® StandardPicture

When you create properties of type Font, OLE_COLOR, or Picture, the **Properties** window automatically associates these properties with the appropriate standard property page. In the **Properties** window, an ellipsis (**...**) button next to any one of these properties indicates that a standard property page will be displayed when a user clicks the button.

If you want standard Visual Basic property pages to be displayed in the **Property Pages** dialog box, you must use the **Connect Property Pages** dialog box to add them. When you use a standard property page for multiple properties, the properties will be displayed in a list on the property page.

As with the **UserControl** object, you can use property procedures to create public properties for **UserDocument** objects. This topic provides an example of how to create a public property and refer to the property from a different ActiveX document.

Creating Public Properties

When you create a public property, you expose the property so other applications can set or get its values.

The following code shows the property **strMyProp** exposed as a public property of the **HomeDoc** ActiveX document. It authorizes the **Text** property of the **TextBox** control to control how the string will be stored and displayed.

```
Public Property Get strMyProp() As String
    strMyProp = txtHomeDoc.Text
End Property

Public Property Let strMyProp(ByVal _
NewstrMyProp As String)
    txtHomeDoc.Text = NewstrMyProp
End Property
```

Referencing a Public Property

The advantage to creating a global object variable to refer to a particular ActiveX document is that you can use this reference to gain access to the public properties and methods of the document.

In the following code, the HomeDoc ActiveX document contains the **Transfer** button. In the Click event for the **Transfer** button, the object variable gHomeDoc is set to **Me**, creating a reference to HomeDoc. The HomeDoc document has the **User** read-only property named **User** that contains the name of the current user. The Satellite ActiveX document can then use the gHomeDoc reference to gain access to the **User** property of the HomeDoc ActiveX document.

```
Private Sub cmdTransfer_Click()
    Set gHomeDoc = Me
    Hyperlink.NavigateTo _
    App.Path & "\Satellite.vbd"
End Sub

Private Sub UserDocument_Show()
    Dim strUser As String

    strUser = gHomeDoc.User
    'Display the user name as desired.
End Sub
```

Visual Basic version 5.0 provides two ways of handling asynchronous notification to a client: events and callbacks. Both ways are performed with outgoing interfaces. For more information about outgoing interfaces, see Notifying Clients.

With events, implementation of the outgoing interface is built into the client. In this case, you do not need to write code for the client application to respond to the event. With callbacks, the developer of the client application must explicitly implement the outgoing interface.

Both of these methods achieve the same goal, but require two different programming techniques. In most cases, the method used is determined by the author of the server component.

In this section, you will learn how to write code for handling events and callbacks from the client perspective, and explore the advantages and disadvantages of both approaches to asynchronous notifications.

This section includes the following topics:

® Server Notifications Using Events

® Server Notifications Using Callbacks

® Events vs. Callbacks

The third step in creating a client application is to create an object to assign to object variables, so you can use the available methods and properties.

There are four ways to create an object in Visual Basic.

® The **CreateObject** function

® The **New** keyword with a **Set** statement

® The **As New** statement in the Declaration section of a module

® The **GetObject** function

Each of these four methods has its advantages for creating an object. In general, the method you select will depend on whether you are creating an instance of an object from another component, or an instance of an object that belongs to your Visual Basic project. Whichever method you use, there are two steps that occur during run time.

® The object is created (or instantiated).

® The pointer to the object's default implementation interface is assigned to an object variable.

The implementation of COM provides services for creating an object. To see an animation of how to implement the steps that take place when a client application requests these services, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

The following discussion describes the methods used to create an **Automation** object, and when to use each method.

Using the CreateObject Function

One method for creating an object is to use the **CreateObject** function. Whether you use this function to create external objects or instances of classes that are part of your project, **CreateObject** always uses COM object creation services.

To use the **CreateObject** function, you use the following code syntax.

```
Dim objMyObject as Project1.Class1
Set objMyObject = CreateObject("Project1.Class1")
```

Using the New Keyword with a Set Statement

Another common method for creating objects is to use the **New** keyword with a **Set** statement. When you use the **New** keyword to create an instance of a class that is external to your application, the COM creation services are used.

In this case, no difference exists between the **CreateObject** function and the **New** keyword. However, if you use the **New** keyword to create an object that is part of your project, Visual Basic will use a more efficient internal object creation method.

When you create instances of classes in your project, the following code syntax is the recommended.

```
Dim objMyObject as Project1.Class1
Set objMyObject = New Project1.Class1
```

To see code samples that use this syntax, see Creating an Instance of Microsoft Excel.

Note When creating an object, you must use the **Set** statement to assign the object to an object variable. This is because the object must have a usage count that enables it to destroy itself when it is no longer in use. The **Set** statement ensures that the **AddRef** method is called when the object interface pointer is returned. The object's internal usage count is incremented, and the object exists until the **Release** function is called and its usage count is decremented.

For more information about **AddRef** and **Release**, see Using Interfaces.

Using As New in the Declaration Section

To create an object with the **As New** statement, use the following syntax.

```
Dim objMyObject as New Class1
```

Using the **As New** statement to declare an object variable defers object creation until the object is actually referenced. The compiler must add code before each object reference to determine if the object variable has been set, and if it hasn't, create the object.

Try to avoid using the **As New** statement because it produces less efficient code. For example, when you use the following code:

```
Dim MyObjectVariable as New Class1
   If MyFunc() Then
      MyObjectVariable.Foo = 5
   End if
   MyObjectVariable.Bar = 6
```

Visual Basic generates code as though you had coded the following:

```
Dim MyObjectVariable as New Class1
   If MyFunc() Then
      If Not IsObject(MyObjectVariable)  ' Added by the compiler
         Set MyObjectVariable = New Class1
      End if                          ' Added by the compiler.
          MyObjectVariable.Foo = 5
    End if
    If Not IsObject(MyObjectVariable)    ' Added by the compiler.
        Set MyObjectVariable = New Class1
       End if                           ' Added by the compiler.
    MyObjectVariable.Bar = 6
```

New vs. CreateObject

When you create a single object, there isn't any significant difference in performance between using the **New** keyword and the **CreateObject** function. However, when you create multiple objects, there is a significant difference in performance. **CreateObject** uses standard COM services to create an instance of a class, while **Set x = New Y** and **Dim x As New Y** use an optimized internal version of object creation services.

When you use **CreateObject**, the underlying code performs at least two lookups in the registry. First, it looks for the **ProgID** for the class (for example, **Excel.Application**), and determines its associated ClassID.

Next, it looks up that ClassID in the **ClsID** section of the registry, and extracts the information it needs specific to that server. For example, the **ClsID** retrieves the location of the server and its type library.

These lookups will not affect performance when creating a single object, but will when creating multiple objects because the code must perform lookups for each **CreateObject** call.

By contrast, when you use the **New** keyword, the underlying code can optimize your code (using early binding) because there is only one lookup for the **ClsID**. All of the server information (for example, the location of the type library) is also cached in memory and reused for subsequent calls.

When creating objects, try to use the **New** keyword instead of the **CreateObject** function.

Using the GetObject Function

The primary reason for using the **GetObject** function is to create an instance of an object that already contains data from a file. For example, the following code returns a Microsoft Excel workbook object with the Finances.xls worksheet already open.

```
Dim xl as Object
Set xl = GetObject("c:\excel\finances.xls")
```

Because there isn't any class specified in the call, Visual Basic creates the object by using the component associated with the .xls file extension.

You can also use **GetObject** to determine whether or not a component is currently running or, if the data file supports it, to create an object of a specified class.

The fourth and final step in creating an Automation client is to invoke the methods and properties of the object you have created, which is similar to programming controls by using the dot (.) operator.

Because controls are enhanced Automation objects, the following syntax applies to creating any Automation object.

```
object.[method]
```

To view an example of implementation, see Creating a Client that Uses Microsoft Excel.

At design time, a server component author can specify a set of notifications that the component will fire when certain events occur. Since these notifications are visible in the type library, they define a outgoing interface on the component.

When building the client component, you need to know which events and associated parameters are generated by the component, and implement each event with an event handler.

You implement an event handler by declaring the event interface in Visual Basic. This is accomplished by extending an object variable declaration with the **WithEvents** keyword, as shown in the following code.

```
Public WithEvents obj as Server.Object
```

The second step is to implement the event handlers in the client application, as shown in this code.

```
Private Sub obj_Event1()
    ' Take some action based on Event1.
End Sub

Private Sub obj_Event2(vNewValue As Variant)
    ' Take some action based on Event2.
End Sub
```

In this chapter the terms programmable object and Automation object are synonymous.

A socket is a bidirectional endpoint through which two applications or processes can pass data. For example, sockets are used to implement chat applications.

This section introduces you to the details of socket architecture and shows you how to enable communication between servers and clients by using sockets.

This section includes the following topics:

® Introduction to the Winsock Control

® Basic Operations of the Winsock Control

® Coding a Winsock Server

® Coding a Winsock Client

® Sending and Retrieving Data

You can use the **WebBrowser** control to create a browser that can view HTML files or host ActiveX controls.

The **WebBrowser** control is an in-process COM server. It must be hosted within some other process, but it can host Automation documents. The **WebBrowser** control implements hyperlinks and provides COM class objects that can be used by other components.

This section introduces the **WebBrowser** control and discusses its properties, methods, and events.

This section includes the following topics:

® Introduction to the WebBrowser Control

® Basic Operations of the WebBrowser Control

® Coding a Web Browser

Using the **WebBrowser** control enables you to control every aspect of the interface for your browser. However, in many situations, you may want to create an instance of Internet Explorer to avoid the code required by the **WebBrowser** control.

This section introduces you to the **InternetExplorer** object, and shows you how to use it through Automation.

Because the **InternetExplorer** object is very similar in functionality to the **WebBrowser** control, this section assumes that you have read the three topics in Using the WebBrowser Control. This section focuses on the additional functionality available from the **InternetExplorer** object.

The section contains the following topics:

® Introduction to the InternetExplorer Object

® Using the InternetExplorer Object

The File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) are two of the most widely used protocols on the Internet. FTP allows users to locate and retrieve files of all types across the Internet, regardless of differences among operating systems. Similarly, HTTP allows users to locate, view, and retrieve HTML documents from any host.

In this section, you will learn how to use the **Internet Transfer** control to implement both FTP and HTTP connections in a client application. You will also learn how to code both asynchronous and synchronous connections.

This section includes the following topics:

® Introduction to the Internet Transfer Control

® Basic Operations of the Internet Transfer Control

® Coding for Asynchronous Operation

® Coding for Synchronous Operation

1. **The following If statement checks for an open port on a server, then uses the Accept method:**

```
If Index = 0 Then
        intMax = intMax +1
        Load tcpServer(intMax)
        tcpServer(intMax).LocalPort = 0
        tcpServer(intMax).Accept requestID
    End If
```

Which event should be handled to run this code?

{ewc mvimg, mvimage,!answer.bmp} A. DataArrival

{ewc mvimg, mvimage,!answer.bmp} B. ConnectionRequest

{ewc mvimg, mvimage,!answer.bmp} C. Listen

{ewc mvimg, mvimage,!answer.bmp} D. Form_Load

2. **When coding a client for HTTP functionality, which method is used to open a connection to an HTTP host?**

{ew A. **GetChunk**

{ewc mvimg, mvimage,!answer.bmp}

B. **Execute**

{ewc mvimg, mvimage,!answer.bmp}

C. **Accept**

{ewc mvimg, mvimage,!answer.bmp}

D. **OpenURL**

{ewc mvimg, mvimage,!answer.bmp}

3. **In the following code for an FTP client, the StillExecuting property is used to monitor the state of a control:**

```
While Inet1.StillExecuting
    DoEvents
Wend
```

What is the purpose of DoEvents?

{ewc mvimg, mvimage,!ans

A. Retrieves data from the controls buffer.

{ewc mvimg.mvimage.!answer.bmp}

B. Initiates a request to a remote server.

{ewc mvimg.mvimage.!answer.bmp}

C. Passes control to the operating system so other events in the application's message queue can be processed while the control is busy.

{ewc mvimg.mvimage.!answer.bmp}

4. You are creating a Web browser and use the Form_Load event as shown below:

```
Private Form_Load()
    WebBrowser.XXXXXXXXX _
        URL:="http://yourdirectory/yourpage.htm"
End Sub
```

Which WebBrowser method would you use in place of XXXXXXXXX to specify the client's home page?

{ewc mvimg.mvimage.!answer.bmp}

A. **GoSearch**

{ewc mvimg.mvimage.!answer.bmp}

B. **Navigate**

{ewc mvimg.mvimage.!answer.bmp}

C. **GoHome**

{ewc

{mvimg,mvimage,!answer.bmp}

{ewc mvimg,mvimage,!answer.bmp}

D. **LocationURL**

5. You want to code a command button on your Web Browser to reload the currently displayed page from the server where it originated. Which method would you invoke to do this?

{ewc mvimg,mvimage,!answer.bmp}

A. **Navigate**

{ewc mvimg,mvimage,!answer.bmp}

B. **Refresh**

{ewc mvimg,mvimage,!answer.bmp}

C. **Refresh2**

{ewc mvimg,mvimage,!answer.bmp}

D. **GoBack**

{ewc mvi

mg.
mvi
ma
ge.!
ans
wer
.bm
p}

6. **You have set a reference to the Microsoft Internet Explorer Controls and added the following code to your project:**

```
Dim browser as InternetExplorer
Set browser = CreateObject("InternetExplorer.Application")
browser.Navigate "http://www.Microsoft.com"
```

Why can't you see the Internet Explorer?

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

A. You must set the **LocationURL** property in the NavigationComplete event to make the Internet Explorer visible.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

B. You must use the **GoHome** method to enable the Internet Explorer to display the initial page in your project.

{ew
c
mvi
mg.
mvi
ma
ge.!
ans
wer
.bm
p}

C. You must set the **Visible** property of the Internet Explorer instance to **True**.

{
e
w
c
m
vi
m
g.
m
vi

D. You must use the **Navigate** method to specify a home page and make it visible.

image_answer.bmp}

For background information on this lab, click each one of these topics:

Objectives

Prerequisites

Lab setup

The exercises in this lab use Personal Web Server, however, they will also work with any Web server.

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **45 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder *<Install Folder>*\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 11.

Exercise 1: Building the Browser Application

In this exercise, you will use the WebBrowser control to create a browser that is capable of viewing files, HTML pages, and ActiveX documents.

Exercise 2: Building the FTP Download Component

In this exercise, you will build a reusable component that provides FTP downloading services for a client.

Exercise 3: Adding Browser Capabilities to the Component

In this exercise, you will extend the FTPService component in the previous exercise to provide users with the ability to view files on a remote computer and select one for downloading.

Exercise 4: Coding a Chat Session Application

In this exercise, you will create an application that uses the Winsock control to conduct a peer-to-peer chat session with another application on the same or remote computer.

By the end of this lab, you will be able to:

® Build a Web browser client by using the **WebBrowser** control.

® Build a component that provides downloading capabilities.

® Create the client and server code for a peer-to-peer chat application.

Before starting this lab, you should be familiar with the following concepts:

® Socket technology and the **Winsock** control

® The File Transfer Protocol (FTP)

® the Hypertext Transfer Protocol (HTP)

® The contents of this chapter

To complete this lab, you will need the following:

® Visual Basic 5.0 or later

You can distribute ActiveX controls so they can be used as stand-alone applications, or you can download them from the Internet to a Web page.

This section discusses the advantages of using ActiveX controls on a Web page. It discusses how to use ActiveX controls to provide users with Web pages that contain interesting and interactive content, and describes the process used to add downloading capability to an ActiveX control.

This section includes the following topics:

® Static vs. Active Web Pages

® Advantages of Using Controls on a Web Page

Visual Basic provides the Application Setup Wizard, which enables you to create an Internet setup for your ActiveX control. By using this wizard, you can also license your control to prevent others from creating controls by using a design-time instance of your control.

This section describes how to download an ActiveX control to a Web page, and explains how to license the control for use on the Internet.

This section includes the following topics:

® Steps for Downloading a Control

® Packaging a Control

® Adding a Control to an HTML Page

® Including Licensing with Your Controls

You can set initial properties for an ActiveX control before it is displayed on a Web page., or set properties using Visual Basic Scripting Edition.

Using the <PARAM> Tag

When you create ActiveX controls in the Visual Basic environment, you can set initial properties at design time by using the Properties window. The Properties window is not available when you write VBScript code, but you can use the <PARAM> tag to create initial properties for the object. Each object property uses a separate <PARAM> tag.

For example, this code shows how to set the initial value for three properties of a control:

```
<OBJECT
    classid="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"     id=lblActiveLbl
>
<PARAM NAME="Angle" VALUE="90">
<PARAM NAME="Alignment" VALUE="2">
<PARAM NAME="Caption" VALUE="Hello, World!">
</OBJECT>
```

Microsoft Visual Basic Scripting Edition (VBScript) is a subset of the Visual Basic language. You can use Visual Basic Scripting Edition to manipulate and extend the functionality of ActiveX controls from within an HTML page.

When you write VBScript code, you add tags to an HTML page for your code. Otherwise, the code syntax is essentially the same as Visual Basic.

This section provides a brief introduction to Visual Basic Scripting Edition. It also describes some of the programming tasks you can accomplish by using VBScript, including working with properties, methods, and events.

This section includes the following topics:

® Introduction to Visual Basic Scripting Edition

® Adding Code to an HTML Page

® Using Properties and Methods

® Creating Procedures

® Setting Initial Control Properties

One of the larger questions facing the software industry is how users can trust code that is published on the Internet. Currently, most Web pages contain only static information, but soon they will be filled with controls and applications that are downloaded and run locally on the user's computer.

When you create ActiveX controls for use on the Internet, you must ensure that the controls are secure. Your controls should not compromise the security of users who download your controls on their computers.

This section discusses the security options you can use with ActiveX controls that will be distributed over the Internet.

This section includes the following topics:

® Component Security Options

® Signing a Control

® Code Safety

When you add ActiveX controls to a Web page written in HTML, it offers these advantages.

® Increased functionality

  Using Visual Basic to create an ActiveX control provides you with user interface options and application functionality that are not possible with standard HTML.

® Improved performance

  Executing application logic in a control instead of on the Web server can increase performance by reducing communication with the Web server. A compiled ActiveX control usually executes much faster than script or HTML code.

® Code privacy

  Because an ActiveX control is compiled, users will not be able to view your source code, as they can with HTML.

The following illustration shows an example of how you can use an ActiveX control on a Web page.

{ewc mvimg, mvimage,!v09g005.bmp}

This type of user interface would be very difficult to accomplish with standard HTML.

To enable an ActiveX control so that it can be downloaded to a Web page, you follow these steps:

1. Package the control and associated files into a .cab file by using the Application Setup Wizard in Visual Basic.
2. Add an <OBJECT> tag to your HTML page that specifies the control to be downloaded.

   A browser such as Internet Explorer uses the information contained in the <OBJECT> tag to download and install the control.

To see an animation of how to download an ActiveX control to a Web page, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

Visual Basic Scripting Edition (VBScript) is the newest member of the Visual Basic family of programming languages. It is a high-performance scripting language designed for creating interactive content on the Web. If you are familiar with Visual Basic, Visual Basic Scripting Edition is easy to learn.

You add VBScript code to HTML pages, and the code is then run in the browser. With VBScript, you can validate form data, generate custom pages, and manage ActiveX controls—all from your Web pages.

For more information about Visual Basic Scripting Edition, see the VBScript Web page on Microsoft.com.
{ewc mvimg, mvimage,!intjump.bmp}

Working with properties and methods in Visual Basic Scripting Edition is the same as in Visual Basic. You can use the Object Browser provided by Visual Basic to identify control properties and methods, and then call them from VBScript.

Setting and Returning Properties at Run Time

To set the value of a property at run time, you use this syntax:

```
Object.Property = Value
```

The **Value** property is commonly used for standard HTML controls. In the following code, the contents of a text box are changed to the selected value in a **Slider** control.

```
Sub sldTemperature_Change()
    Temp.Value = sldTemperature.Value
End Sub
```

To return the value of a property at run time, you use this syntax:

```
ReturnValue = Object.Property
```

Calling Methods

You call methods in VBScript code the same as you do in Visual Basic. In the following code, the **Refresh** method is used to update a control when the user clicks a button on a Web page.

```
Sub BtnRefresh_OnClick()
    MyControl.Refresh
End Sub
```

In Visual Basic Scripting Edition, you must place procedures within the HTML <SCRIPT> tag. This enables the procedures to be recognized by the script interpreter and other procedures in an HTML page.

This topic describes how to create event procedures with **Sub** and **Function** procedures with Visual Basic Scripting Edition.

Creating Event Procedures

When creating an event procedure in Visual Basic Scripting Edition, you use a **Sub** procedure with the convention ObjectName_Event. This is the same syntax used to define event procedures in Visual Basic.

For example, the **Button1_OnClick** procedure in the following code will run when the **Button1** control is clicked:

```
Sub Button1_OnClick
    MsgBox "hello"
End Sub
```

You can also define an event procedure in an HTML <INPUT> or <SCRIPT> tag. For more information about creating event procedures in Visual Basic Scripting Edition, see the VBScript Web page on Microsoft.com. {ewc mvimg, mvimage,!intjump.bmp}

Creating Sub and Function Procedures

Sub and **Function** procedures must also be placed within the <SCRIPT> tag, and are defined in the same way as Visual Basic procedures. All variables in Visual Basic Scripting Edition are of type **Variant**, so all arguments must be of the same type.

The following code shows a simple VBScript **Function** procedure, which can also run in Visual Basic:

```
Function Validate(dt)
    If IsDate(dt) Then
        Validate = True
    Else
        Validate = False
    End If
End Function
```

For background information on this lab, click each of these topics:

Objectives

Prerequisites

Lab Setup

To see a demonstration of the completed lab solution, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Estimated time to complete this lab: **90 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 9.

Exercise 1: Creating a Setup Package

In this exercise, you will use the Application Setup Wizard for the Stock project to package an ActiveX control for downloading to the Internet.

Exercise 2: Scripting a Control

In this exercise, you will modify the HTML file created by the Application Setup Wizard. You will set an initial property for the control, and add scripting to automate the control.

Exercise 3: (Optional) Adding Licensing.

In this exercise, you will add licensing support to your control, and create a licensing package file so the control can be downloaded from a Web page.

In this lab, you will use an ActiveX control on a Web page.

After completing this lab, you will be able to:

® Package an ActiveX control for Internet setup.
® Download an ActiveX control to a Web page.
® Set the behavior of a control behavior using Visual Basic Scripting Edition.
® Use a licensed control on the Internet.

Before working on this lab, you should be familiar with the following concepts:

® Creating ActiveX code components.

® The contents of this chapter.

To complete this lab, you should have the following setup:

® Visual Basic 5.0 or later

® Internet Explorer 3.0 or later

® Access to a Web server (optional)

When you use ActiveX controls on a Web page, you can implement the following security measures:

® Set the appropriate level of security in Internet Explorer.

® Provide users with information about the author of the control through digital code signing.

® Create an Internet download by marking your controls as safe for scripting and initialization.

To see a demonstration of the Internet Explorer security options, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

To make sure that your control can be downloaded from the Internet, you must package the control so it can be used in the HTML environment.

Creating a .Cab File

When you use an ActiveX control on a Web page, it must reside on a Web server, along with any dependent files and Setup instructions.

To simplify the distribution and installation of these files over the Internet, the ActiveX controls can be compressed into a single .cab file. The Application Setup Wizard in Visual Basic can create these .cab files for you.

Using the Application Setup Wizard

The Application Setup Wizard handles most of the work involved with placing a control on a Web page, including the following tasks:

® Creating a .cab file.

® Marking your control as safe for scripting and initializing.

® Building an HTML template for using your control.

For information about ensuring the safety of controls, see Code Safety.

To see a demonstration of how to create an Internet setup with the Application Setup Wizard, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

When you use the Application Setup Wizard to create an Internet download, an HTML template is created. This template enables you to specify your control by using the <OBJECT> tag .

The <OBJECT> tag is a new element of HTML 3.0 that provides a standardized way to add new types of media to an HTML page. The HTML pages created by the Application Setup Wizard contain the appropriate <OBJECT> tag for your ActiveX control.

For information about HTML syntax and other Internet related topics, see the Author/Editing page of Site Builder site. To go to this site, click this icon.
{ewc mvimg, mvimage,!intjump.bmp}

Parameters of the <OBJECT> Tag

The <OBJECT> tag contains a number of parameters that define an object, its location, and how it appears on an HTML page.

This following code shows an example of how to use the parameters of an <OBJECT> tag:

```
<OBJECT
    CLASSID="clsid:6A7A58F2-3DFC-11D0-A520-0080C776418A"
    WIDTH=200
    HEIGHT=200
    ID=MyControl
    CODEBASE="MySetup.CAB#version=1,0,0,0">
</OBJECT>
```

The following table lists some of the standard parameters used in the<OBJECT> tag.

Parameter	Definition
CLASSID	A unique class identifier used to identify the control, and stored in the system registry.
ID	Specifies the object name so you can refer to it from VBScript.
CODEBASE	A Uniform Resource Locator (URL) that points to a file containing the implementation of an object. For Visual Basic controls, you specify a URL that points to your .cab file, relative to the location of your HTML page. If the HTML page and .cab file are in the same folder on your Web server, the CODEBASE parameter refers to the name of the .cab file.

When you have determined that your control is safe, you can provide a guarantee for the control.

This topic discusses code safety and how you can use the Application Setup Wizard to mark components as safe.

ActiveX Code Safety Issues

To guarantee that a control's code is safe for an end user to operate, you can use code signing. This ensures that a component has been created from a reliable source.

Using ActiveX controls from scripts increases the need for additional security measures. Even if a control is known to be safe, it does not necessarily remain safe when it is used by a script that is not secure.

For example, Microsoft Word is a secure tool from a reputable source, but a malicious script could use Word with Automation to delete files or install macro viruses on a user's system.

Marking a Component as Safe

You can use the Application Setup Wizard to mark an ActiveX control as safe for scripting and initialization. This will ensure that the author of the control certifies that the control cannot do any damage, such as reformatting a hard drive, with an HTML script.

Marking an ActiveX control as safe also ensures that it will not delete files or cause system settings to change by any data passed during initialization. In addition, downloading your control will never corrupt a user's computer or obtain unauthorized information from the computer.

The following illustration shows the **Safety** dialog box in the Application Setup Wizard.

{ewc mvimg, mvimage,!v09g010.bmp}

In this dialog box, you can mark your component as safe for scripting and/or initialization.

For information about the Application Setup Wizard, see Packaging a Control.

In this exercise, you will use the Application Setup Wizard for the Stock project to package an ActiveX control for downloading to the Internet.

u Package the control

1. Run the Application Setup Wizard in Visual Basic.
2. Select the **Rebuild the Project** option.
3. When prompted for a project, select Stock.vbp, which is located in the <*Install Directory*>\Labs\Lab09 directory.
4. In the **Setup Options** group, click **Create Internet Download Setup**.
5. When prompted for a distribution location, specify the <*Install Directory*>\Labs\Lab09\Setup.
6. When prompted for a location for the run-time components, click **Download from the Microsoft Web Site**.
7. Mark the **Stock** control by clicking **Safe for initialization** and **Safe for scripting**.
8. When prompted to include the Property Page DLL, click **No**.

   For more information about marking code as safe, see Code Safety.
9. To complete the download setup process, click **Finish**.

u Test the control locally

1. Using Internet Explorer, open the file Stock.htm created by the Application Setup Wizard.
2. In the **Stock Ticker** text box, type **MSFT** (all caps).

   The text box control should display a random number that updates at frequent intervals, as shown in the following illustration.

   {ewc MVIMG, MVIMAGE,!v09g015.bmp}
3. To display the HTML that was created by the Application Setup Wizard, click **Source** on the **View** menu.

u Test the download for the control

These steps require access to a Web server. If you do not have access to a Web server, you can complete the steps by using the files on a local computer.

1. Copy Stock.htm and Stock.cab to a directory on your Web server.
2. Download Stock.htm from your Web server.

   The following illustration shows how the Stock.htm page is downloaded from a Web server.

   {ewc MVIMG, MVIMAGE,!v09g020.bmp}
3. Download the **Stock** control from your Web server with Internet Explorer by using Regsvr32.exe to unregister the control. Use the following command line:

   ```
   Regsvr32.exe /u C:\MVB5\Labs\Lab09\Stock.ocx
   ```

   The application Regsvr32.exe is located on the *Mastering Microsoft Visual Basic 5* CD-ROM in the \Tools\ Regutils directory.
4. In Internet Explorer, make sure that security is set to the highest level.

   a. On the **View** menu, click **Options**, and then click the **Security** tab.

   b. Click the **Safety Level** button, and then click **High**, as shown in the following illustration.

   {ewc MVIMG, MVIMAGE,!v09g025.bmp}
5. Refresh the Stock.htm page.

   Because the **Stock** control has not been signed and guaranteed to be safe, you will get an error message that your control could not be displayed.
6. Set Security to **Medium**, and refresh the page again.
7. When given the option of downloading the control, click **Yes**.
8. Make sure that the control is properly installed and operates correctly.

In this exercise, you will add licensing support to your control, and create a licensing package file so the control can be downloaded from a Web page.

u Add licensing to the control

1. In the <*Install Directory*>\Labs\Lab09\License folder, open the Stock.vbp project.
2. In the **Project Properties** dialog box, click the **Require License Key** property.
3. Recompile the Stock.ocx file.

   You should now see a licensing file with the extension .vbl in the <*Install Directory*>\Labs\Lab09\License for the Stock project folder.

u Package the control

1. Use the Application Setup Wizard to create a new Internet download setup for the Stock.ocx file.
2. Save the download setup in the <*Install Directory*>\Labs\Lab09\License\Setup folder.
3. When prompted by the Application Setup Wizard to add the licensing file, click **No**, and then complete the setup.

u Create the licensing file

1. Using the application Lpk_tool.exe, create a licensing package file named Stock.lpk for the Stock.ocx control.

   Lpk_tool.exe is located on the *Mastering Microsoft Visual Basic 5* CD-ROM in the \Tools\Lpk_tool directory. (This file is not installed with Visual Basic.)

   a. From the list of **Available Controls**, select **StockLicense.StockPrice**, and then click **Add** to add it to **Controls in License Package**.

   b. Save Stock.lpk in the <*Install Directory*>\Labs\Lab09\License\Setup directory.

u Test the download of the control

1. Delete the control license key from the registry.

   When you compile a control, the license information from the control is written to the registry. To test Internet licensing, you need to delete that licensing information from the registry.
2. To make sure that the control is installed from the .cab file, unregister the control.
3. Open the Stock.htm page.

   The control should not display because the licensing package file has not been specified in the HTML code.
4. Use Notepad (or any other text editor) to edit the Stock.htm page and change the licensing object tag so that it points to the Stock.lpk package file.
5. Uncomment the licensing object tag (-->) by moving it above the <OBJECT> tag, and then reload the Stock.htm page.
6. Reload the Stock.htm page.

   The Stock control should now load correctly.

In this exercise, you will modify the HTML file created by the Application Setup Wizard. You will set an initial property for the control, and add scripting to automate the control.

Set the initial control property

1. Use Notepad (or any other text editor) to open the local copy of Stock.htm.
2. Using the <PARAM> tag, set the initial **Active** property of the **Stock** control to zero (which is **False**).

   For more information about setting initial properties, see Setting Initial Control Properties.
3. Save the HTML file.
4. Load Stock.htm in Internet Explorer. (If it is open, reload the page.)

   The **Stock** control should no longer update automatically.

   To see the HTML code from this lab exercise, click this icon.
   {ewc mvimg, mvimage,!code.bmp}

Add the HTML input controls

By following these steps, you will add the HTML text box and command button controls to the Stock.htm page. These controls will be used to interact with the **Stock** ActiveX control.

1. After the StockPrice <OBJECT> tag, add the following lines of HTML code.

```
<INPUT TYPE=button VALUE="Refresh" NAME="cmdRefresh">
<INPUT TYPE=txt NAME="txtLastUpdate" SIZE="40">
```

2. Save the HTML file.
3. Load Stock.htm in Internet Explorer. (If it is open, reload the page.)

   The Stock.htm page should resemble the following illustration.

   {ewc mvimg, mvimage,!v09g030.bmp}

Use VBScript to add code to the HTML input controls

1. After the <HTML> tag at the beginning of the HTML file, add <SCRIPT> tags for Visual Basic Scripting Edition.
2. Add an OnClick event procedure for the **cmdRefresh** button.
3. To update the stock price, call the **Refresh** method in the event procedure of the **StockPrice** object.
4. Add the PriceUpdated event procedure for the Visual Basic **StockPrice** control.
5. To display the date and time the stock price was updated, set the **Value** property of the **txtLastUpdate** text box in the event procedure by using the following code:

```
"Last Updated: " & Now
```

6. Save the HTML file.
7. Load Stock.htm in Internet Explorer. (If it is open, reload the page.)
8. In the **Stock Ticker** text box, type **MSFT** (all caps), and then click **Refresh**.

   The stock price should update, and the current date and time should be entered in the text box, as shown in the following illustration.

   {ewc mvimg, mvimage,!v09g035.bmp}

   To see the HTML code for this exercise, click this icon
   {ewc mvimg, mvimage,!code.bmp}

To assure users that the author of a control can be identified, and that the code has not been modified since it was signed, you can add a digital signature to your code. You can add a digital signature by using the Authenticode technology, which is derived from public-key signature algorithms.

To sign your code, you work with a certificate authority (CA) to obtain a digital certificate, which provides users with information about the author of a control. The CA provides and renews your certificate, authenticates identity, and handles legal and liability issues when security is broken.

Signing Code

To sign code by adding a digital signature, you follow these steps:

1. Apply for a digital certificate over the Internet from a certificate authority.

   For instructions on how to obtain a certificate, see the Authenticode page of Microsoft.com:
   {ewc mvimg, mvimage,!intjump.bmp}
2. Install the ActiveX Software Development Kit (SDK) by downloading it from Microsoft.com:
   {ewc mvimg, mvimage,!intjump.bmp}

   The ActiveX SDK provides the tools and documentation necessary for code signing.
3. Sign your files.

   Use the Signcode.exe application included in the ActiveX SDK to apply your digital certificate to your code. Although you can sign most types of executable files, you will probably want to sign .cab files in particular because this applies the signature to the complete collection of compressed files.

You can include licensing protection for any controls you want to distribute on the Internet, as well as providing licensing for ActiveX controls distributed as stand-alone applications.

Licensing Controls for Standard Distribution

You can use Visual Basic to create licensed ActiveX controls. A licensed control contains a key that prevents it from being used at design time. This prevents other developers from including your control as part of their application without obtaining its license.

For information about licensing controls for standard distribution, see Licensing Controls in Chapter 8: Creating ActiveX Controls.

Licensing Controls for the Internet

Licensing a control for distribution on the Internet is different from licensing a control as part of a stand-alone application. Internet licensing involves the following two steps:

® Create a license package (.lpk) file.

When you license a control for use on a Web page, you create a licensing package file. This file stores the run-time licensing information for all components on the page.

® Create a reference to the licensing package file on your HTML page.

When a user attempts to use a licensed control on a Web page, the Internet Explorer License Manager checks the information in the .lpk file to ensure that the proper license is being used. If someone tries to use your control on another Web page that does not include the .lpk file, the control will not appear on the page.

Creating an .Lpk File

You create a licensing package files by using the utility program Lpk_Tool.exe. This program is located on the Visual Basic CD-ROM in the \Tools\Lpk_Tool folder.

The licensing information for all controls on a page must be in a single .lpk file.

To see a demonstration of how to create a licensing package file, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Creating a Reference to an .Lpk File

After you create the .lpk file, you must specify it by using the **License Manager** object in the HTML code of your Web page. The necessary <OBJECT> tag is added to the sample .htm page created by the Setup Wizard. Here is an example of an <OBJECT> tag for licensing:

```
<OBJECT CLASSID="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
    <PARAM NAME="LPKPath" VALUE="MyLicenseFile.LPK">
</OBJECT>
```

Testing an .Lpk File

When you compile a licensed ActiveX control on your computer, the license information is written to your system registry. The control will load properly on your computer without using the .lpk file.

To make sure your control's licensing functionality works correctly, you should test it on your local computer before distributing it over the Internet. To test the licensing functionality, you must first remove the license information from your computer's system registry.

To see a demonstration of how to remove a licensing key from the registry, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

You can also declare property procedures with the **Friend** keyword. The different property procedures that make up a property can have different levels of scope.

Microsoft Personal Web Server for Windows 95 turns any Windows 95 computer into a Web server, and enables easy publication of personal Web pages. Easy to install and manage, Personal Web Server (PWS) simplifies information sharing on corporate intranets, as well as on the Internet. PWS is well-suited for developing, testing, and staging Web applications.

Like Microsoft Internet Information Server, Personal Web Server supports all ISAPI extensions and CGI scripts. Personal Web Server has been optimized for interactive workstation use, but does not have the system requirements of a full Web server.

This section explains how to install and configure Personal Web Server.

This section includes the following topics:

® Installing Personal Web Server

® Configuring Personal Web Server

® Starting Internet Services

Click here to connect to the Microsoft Developer Network (MSDN) page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

MSDN is the official source of development toolkits, operating systems, and development-related technical, strategic, and resource information from Microsoft. It is the comprehensive source of programming information and toolkits for those who write applications for the Microsoft Windows and Windows NT operating systems, or use Microsoft products for development purposes. The Microsoft Developer Network is an annual membership program. Members are kept up to date through regular deliveries of information in the Development Library CDs, a newsletter, and other information sources.

By becoming a member of the Microsoft Developer Network, developers ensure that they receive the most current technical and strategic information right from the source. To subscribe to the Microsoft Developer Network, call (800) 759-5474.

Many of the whitepapers in the Technical Whitepapers section were provided by MSDN.

Course No. 778

This course is intended for corporate developers and solution providers who need to create and distribute custom solutions developed with Microsoft Office.

This course syllabus should be used to determine whether the course is appropriate for the student, based on current skills and technical training needs. Technical information is provided on the intended audience, course prerequisites, covered topics, lab exercises, course materials, and software.

Course content, prices, and availability are subject to change without notice.

The primary objective of this course is to teach students how to develop custom solutions that integrate components of the Microsoft Office 97 family.

At Course Completion

At the end of the course, students will be able to design a custom solution; use the Microsoft Visual Basic programming system environment in each of the Microsoft Office applications; use Visual Basic for Applications to write code; describe the purpose of object models and use automation to program applications; list and describe the object models provided in the Office Developer Edition; customize the user interface of Office applications; use data access objects (DAOs) and ODBCDirect in an Office application to programmatically access external data; add intranet links to Office applications and use the Office Web object model; add workgroup features to custom solutions; and distribute a custom solution.

Microsoft Certified Professional Exams

This course helps you prepare for the following Microsoft Certified Professional exams:

® To be determined

Prerequisites

® Be able to use basic features of the Microsoft Office applications

® Understand event-driven programming concepts

® Understand basic database concepts such as tables and queries

The course materials are in English. To benefit fully from the course, students need an understanding of the English language and completion of the prerequisites.

Course Materials and Software

The course CD-ROM is yours to keep. This course is licensed for use by a single user on a single computer. Multiuser usage is prohibited. All content on the CD can be printed.

You will be provided with the following software for use in the classroom:

® Microsoft Windows 95 operating system

® Microsoft Office 97 Developer Edition

Chapter 1: Designing a Custom Solution

Topics

® Microsoft Office 97 development features

® Design process

® Office 97 applications

® Developing with Office 97

® Sample custom solutions

Lab
Using the sample solution

Skills
Students will be able to:
® Determine the Office 97 application best suited to solve a business need.

® Describe the developer features found in Office 97.

® Describe the solution architecture of a custom solution.

® Describe where you can write code for your custom solution.

Chapter 2: Using the Visual Basic Editor

Topics
® Where code is stored

® Working with the Visual Basic editor

® Writing code

® Running code

® Reusing code

Lab
Using the Visual Basic editor

Skills
Students will be able to:
® Describe how projects relate to documents.

® List the components that you can include in a project.

® List where code is stored.

® Record code in Microsoft Excel, Word, and the PowerPoint presentation graphics program.

® Use the Visual Basic editor to view, modify, and run code.

® Assign code to a menu or a command button on a toolbar.

Chapter 3: Using Visual Basic for Applications

Topics
® Variables

® Procedures

® Controlling program execution

® Errors

® Debugging

Lab
Using Visual Basic for Applications

Skills
Students will be able to:
® Declare and use variables.

® Determine the appropriate data type to use in a particular scenario.

® Create and invoke *Sub* and *Function* procedures.

® Use decision control and looping statements to control program execution.

® Add error-handling statements to handle run-time errors.

Chapter 4: Using Forms and Controls

Topics

® Using controls on forms

® Working with forms

® Using controls on documents

Lab

Using controls

Skills

Students will be able to:

® Create and use user forms in Microsoft Excel, Word, and PowerPoint.

® Add controls to a Microsoft Office document.

® Link the contents of controls to worksheet ranges in Microsoft Excel.

Chapter 5: Object Models and Automation

Topics

® Navigating an object hierarchy

® Working with objects

® Automating Microsoft Office applications

Lab

Object models and automation

Skills

Students will be able to:

® Describe the purpose and benefits of using an object model.

® Use the object browser to view an object model.

® Describe the difference between objects, properties, and methods.

® Write code with Visual Basic for Applications to automate applications.

® List and describe the object models provided in Microsoft Office Developer Edition.

Chapter 6: Using Microsoft Excel Objects

Topics

® Using workbooks

® Using worksheets

® Creating charts

® Creating PivotTable dynamic views

Lab
Using Microsoft Excel objects

Skills
Students will be able to:
® Programmatically create and modify workbooks and worksheets.
® Programmatically add values and formulas to a worksheet.
® Write code to workbook and worksheet events.
® Programmatically create and modify a chart.
® Programmatically create and modify a PivotTable.

Chapter 7: Using Word Objects

Topics
® Word objects
® Navigating the Word object model
® Using events in Word
® Working with Word documents
® Working with areas of a document
® Working with text
® Working with Word forms
® Using Mail Merge

Lab
Using Word objects

Skills
Students will be able to programmatically:
® Create and modify Word documents.
® Create a document based on a Word template.
® Format text in a Word document.
® Create and use Word forms.
® Use Mail Merge.

Chapter 8: Using PowerPoint Objects

Topics
® Working with presentations
® Working with slides
® Working with a slide show

Lab
Using PowerPoint objects

Skills

Students will be able to:

® Create and modify a PowerPoint presentation.

® Add shapes to a slide.

® Use the *OnAction* property to determine how a shape responds to mouse actions during a slide show.

® Add speaker notes to a slide.

® Modify the master slide in a presentation.

® Create a custom show.

Chapter 9: Using Microsoft Access Objects

Topics

® Introduction to programming with Microsoft Access

® Using forms

® Using reports

Lab

Using Microsoft Access objects

Skills

Students will be able to:

® Run a Microsoft Access form or report from another Office 97 application.

® Write code for a form, report, and control events in Microsoft Access.

® Programmatically create forms and reports in Microsoft Access.

® Programmatically add a code module or code to an existing code module for a form or report in Microsoft Access.

® Programmatically add controls to a form or report in Microsoft Access.

Chapter 10: Shared Object Models

Topics

® Customizing menus and toolbars

® Programming the Assistant

® Searching for files

® Drawing objects

Lab

Shared object models

Skills

Students will be able to:

® Use the CommandBars object model to modify and create menus and toolbars.

® Use the Assistant object model to retrieve and display information.

® Use the FileSearch object model to search for files.

® Use the Drawing object model to create graphics.

Chapter 11: Using DAOs

Topics

® Microsoft Jet databases

® ODBCDirect Workspace

Lab

Using data access objects

Skills

Students will be able to use DAOs to:

® Open a Microsoft Jet database.

® Retrieve and update records.

® Create and execute a query.

® Refresh a linked table.

® Open a connection to an Open Database Connectivity (ODBC)–compliant data source.

® Retrieve records from an ODBC-compliant data source.

Chapter 12: Using Outlook Objects

Topics

® Designing Outlook forms

® Automating with Outlook

Lab

Using Outlook

Skills

Students will be able to:

® Create, publish, and use Outlook forms.

® Add code to an Outlook form.

® Use the Outlook object model programmatically to create and use electronic-mail messages, contacts, tasks, and appointments.

Chapter 13: Adding Intranet Features

Topics

® Working with hyperlinks

® Saving documents in hypertext markup language (HTML) format

® Using the WebBrowser control

Lab

Adding intranet features

Skills

Students will be able to:

® Describe the difference between the Internet and an intranet.

® Add hyperlinks to a document.

® List the advantages of Microsoft ActiveX technology.

® Save documents in HTML format.

® Use the WebBrowser control on a document or form.

Chapter 14: Distributing a Custom Solution

Topics

® Securing your custom solution

® Working with add-ins

® Using references

® Using Setup Wizard

Skills

Students will be able to:

® Set a password for a document, database, and Visual Basic for Applications project.

® Understand the security options in Microsoft Access.

® Load and unload an add-in in Microsoft Excel, Word, PowerPoint, and Microsoft Access.

® Describe the differences among add-ins in Microsoft Excel, Word, PowerPoint, and Microsoft Access.

® Set a reference to an Office document manually and programmatically.

® Use Setup Wizard in Office Developer Edition to create a Setup program for your custom solution.

Click here to connect to the Mastering Series page on the Microsoft Web site.
{ewc mvimg, mvimage,!intjump.bmp}

The Mastering Series Web site offers you the following information about the titles:

® What's New

® Titles Available

® Frequently Asked Questions

® Where to Buy

® International Update

The site also offers you a chance to submit feedback on the Series, and provides links to these other Microsoft Web sites:

® Support

® Knowledge Base

® For Developers Only

Click here to connect to the Microsoft Visual Basic Cool Links page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This page provides links to Visual Basic-related information, including articles and reviews, publications, and developer Web pages.

Click here to connect to the Microsoft Solutions Framework Home page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This page provides information about Microsoft Solutions Framework, delivered as facilitated training on the effective creation of distributed computing systems. MSF helps organizations align business and technology objectives, successfully deploy Microsoft technologies to streamline business processes, and reduce the life-cycle costs of new business solutions.

Click here to connect to the Microsoft TechNet page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft TechNet is the comprehensive information resource for anyone who evaluates, implements, or supports Microsoft business products. This page is a sampling of the TechNet content, which includes over 150,000 pages of in-depth information on CD-ROM.

This section describes various Microsoft programs of interest to Visual Basic developers. The Microsoft Certified Professional program is a series of exams that verify your knowledge of Microsoft products and technologies. By developing software that conforms to Microsoft standards, you can qualify to display Microsoft logos with your products. And, if you want to keep up with the latest Microsoft news and gain valuable business connections, consider joining the Microsoft Solution Provider program.

® The Microsoft Certified Professional Program

® Microsoft Logo Programs

® The Microsoft Solution Provider Program

Created: October 1992
Revised: July 1993

Abstract

Open database connectivity (ODBC) is Microsoft's strategic interface for accessing data in a heterogeneous environment of relational and non-relational database management systems. Based on the Call Level Interface specification of the SQL Access Group, ODBC provides an open, vendor-neutral way of accessing data stored in a variety of proprietary personal computer, minicomputer, and mainframe databases. ODBC alleviates the need for independent software vendors and corporate developers to learn multiple application programming interfaces. ODBC now provides a universal data access interface. With ODBC, application developers can allow an application to concurrently access, view, and modify data from multiple, diverse databases. ODBC is a core component of Microsoft Windows Open Services Architecture (WOSA). Apple has endorsed ODBC as a key enabling technology and has announced the ODBC software developer's kit for Macintosh developers. With growing industry support, ODBC has emerged as the industry standard for data access for both Windows-based and Macintosh-based applications.

Introduction

Providing data access to applications in today's heterogeneous database environment is very complex for software vendors as well as corporate developers. With ODBC, Microsoft has eased the burden of data access by creating a vendor-neutral, open, and powerful means of accessing database management systems (DBMSs).

Note In the context of this paper, DBMS refers to a database product. This may be a relational database, such as Oracle or DB2, or a file-based database, such as dBASE.

® ODBC is *vendor neutral*, allowing access to DBMSs from multiple vendors.

® ODBC is *open*. Working with ANSI standards, the SQL Access Group (SAG), X/Open, and numerous independent software vendors, Microsoft has gained a very broad consensus on ODBC's implementation, and it has become the dominant standard.

® ODBC is *powerful*—t offers capabilities critical to client/server on-line transaction processing (OLTP) and decision support systems (DSS) applications, including system table transparency, full transaction support, scrollable cursors, asynchronous calling, array fetch and update, a flexible connection model, and stored procedures for "static" SQL performance.

Benefits

ODBC provides many significant benefits to developers, end users, and the industry by providing an open, standard way to access data.

® ODBC allows users to access data in more than one data storage location (for example, more than one server) from within a single application.

® ODBC allows users to access data in more than one type of DBMS (such as DB2, Oracle, Microsoft SQL Server, DEC Rdb, Apple DAL, and dBASE) from within a single application.

® ODBC greatly simplifies application development. It is now easier for developers to provide access to data in multiple, concurrent DBMSs.

® ODBC is a portable application programming interface (API), enabling the same interface and access technology to be a cross-platform tool.

® ODBC insulates applications from changes to underlying network and DBMS versions. Modifications to networking transports, servers, and DBMSs will not affect current ODBC applications.

® ODBC promotes the use of SQL—the standard language for DBMSs—as defined in the ANSI 1989 standard. It is an open, vendor-neutral specification based on the SAG Call Level Interface (CLI).

® ODBC allows corporations to protect their investments in existing DBMSs and protect developers' acquired DBMS skills. ODBC allows corporations to continue to use existing diverse DBMSs, while continuing to "rightsize" applications.

ODBC is the database access component of Windows Open Services Architecture (WOSA), Microsoft's strategic architecture for delivering on "Information At Your Fingertips." This paper describes how ODBC fits into

the WOSA framework, the challenges facing developers and users in today's complex computing environment, how ODBC meets these challenges, and why ODBC has become the dominant solution for data access.

The WOSA Solution

In the absence of a formal way to connect front-end applications to various back-end services, application developers are forced to incorporate support for vendor-specific APIs in their applications. Supporting additional services requires that application developers either build new applications or modify existing ones to accommodate diverse APIs—this process is both labor-intensive and expensive.

WOSA provides a single, system-level interface to enterprise computing environments, while hiding the complexities of heterogeneous environments from end users and developers. By taking advantage of the WOSA interface, a Windows-based desktop application does not need to contain special code for any of the types of network in use, the types of computers in the enterprise, or the types of back-end services available in order to gain seamless access to available information. As a result, even if the network, computers, or services should change, the desktop application does not need to be modified. In other words, WOSA enables Windows-based applications to connect to all the services across multiple computing environments.

The Windows operating system presents end users with a single application interface. Once users learn how to use one application, they can quickly learn others. Similarly, WOSA presents developers of distributed applications with a single interface for communicating with back-end services such as DBMSs and messaging. Instead of having to learn a different API for each implementation of a service, developers building applications with WOSA only need to learn a single API for *all* implementations of a particular service. Furthermore, when an existing service is modified or replaced, the front-end application is unaffected as long as the new back-end service communicates through the WOSA interface.

WOSA makes it possible for corporate developers to build stable, long-term enterprise solutions using various combinations of off-the-shelf products and custom packages, while presenting end users with a single, consistent interface. End users are spared having to learn a new application for each new service or each alteration of an existing service. Developers are spared having to constantly modify their applications to communicate with new services. WOSA makes the Windows operating system the single, reliable, strategic platform for end users, application developers, and MIS managers.

What WOSA Does

WOSA provides a single system-level interface for connecting front-end applications with back-end services. Application developers and end users alike do not have to worry about using numerous different services, each with its own protocols and API, because making these connections is the business of the operating system, not of individual applications.

WOSA provides an extensible framework in which Windows-based applications can seamlessly access information and network resources in a distributed computing environment. WOSA accomplishes this magic by making a set of common APIs available to all applications.

ODBC and MAPI (Messaging API) are two of the key components of WOSA, along with the Windows Sockets Library, the Licensing API, and Remote Procedure Calls (RPCs). ODBC addresses database connectivity technology, while MAPI addresses electronic mail and workgroup productivity applications. MAPI, ODBC, and the other WOSA components will become part of the standard Windows and Windows NT operating systems sometime in the future.

The Need for Database Connectivity

Information has become a key asset to corporate competitiveness. To be competitive in the 1990s, corporations need access to accurate and timely information. Companies are striving to achieve a higher level of accuracy and effectiveness in areas such as pricing, quality control, market analysis, capacity planning, inventory management, customer service, and billing. At the same time, users are demanding better tools for accessing this information. Users demand graphical user interfaces, leading edge analytical tools, easy ways of accessing and viewing information—all without having to know the structure or language of the underlying DBMS or the issues unique to their network.

To achieve this goal, corporations must provide better tools for end users to access existing information while providing a migration path for the data and the applications as they evolve or are "rightsized" to the optimum platform. This presents a difficult challenge, given the heterogeneous nature of most corporations' current

information technology.

Heterogeneous Database Environments

Historically, database applications have been built to access a single source of data. The range of applications varies from mainframe-based, batch-oriented DBMSs, to terminal-based, interactive applications, to personal computer–based, single-user DBMSs, to the more recent client/server DBMSs. Data typically resides in a variety of file formats, such as VSAM and ISAM, as well as in hierarchical and relational DBMSs.

Corporations typically have applications and data residing on diverse platforms and DBMSs for historical, strategic, and technological reasons. Corporations often have legacy systems that must be maintained because they contain key corporate data. Corporate mergers often bring together diverse information technologies. Systems were often developed using technology that met a specific requirement, such as an engineering application. Departmental users developed their own workgroup and single-user personal computer databases. Over time, data on any of these systems might be summarized for consumption by analysts, copied and distributed geographically, or have many snapshots taken of it.

There is a strong requirement for a common method of accessing, managing, and analyzing data. The heterogeneous nature of database environments is a problem corporations are faced with today. Many firms have discovered that much of the cost associated with application maintenance is related to data access problems. One of the first steps to meeting the needs of information users is database connectivity.

Database Connectivity Components

Database connectivity allows an application to communicate with one or more DBMSs. Database connectivity is a requirement whether the application uses a file-based (ISAM) approach, a client/server model, or traditional mainframe connectivity. The requirement for database connectivity has been hastened by client/server computing. As users increasingly use graphical, personal computer–based tools to analyze, prepare, and present data, they require greater access to the vast volumes of existing corporate data. In the most general sense, client/server computing means that some portion of the application runs on a personal computer. At the very least, this computer is responsible for screen presentation and gathering user input. The server (or host) is responsible for responding to queries; managing concurrency, security, backup and recovery; transaction processing; and so on. This differs from centralized, host-based applications where the entire application runs on the host platform.

{ewc mvimg, mvimage,!707_01.bmp}

Some of the key components involved with database connectivity are explained below.

Application

Allows users to perform a set of functions, such as queries, data entry, and report generation. Examples are Microsoft Excel, Microsoft Works, Aldus PageMaker, and internally developed applications such as an executive information system or a reporting system.

Client system

The physical system where the client portion of an application runs. In the personal computer world, this may be an IBM PC or compatible or an Apple Macintosh.

Data access software

A service layer on client systems that provides a direct interface for applications. This “middleware” or enabling software plays a key role in client/server data access. This layer accepts data retrieval and update requests directly from the application, and transmits them across the network. This “middleware” also is responsible for returning results and error codes back to the application.

Data source

The data and method of data access. The data may exist in a variety of hierarchical or relational DBMSs, or in a file with a format such as ISAM or VSAM.

Network

The physical connection of the client to the server system.

Network software

The software protocols that allow the client to communicate with the server system.

Server system

The physical system where the DBMS resides (also known as the host system). For example, the server system could be an IBM PC or compatible, a DEC VAX, or an IBM mainframe.

The Challenge of Database Connectivity

One of the challenges of database connectivity is accessing multiple, heterogeneous data sources from within a single application. A second challenge is flexibility—the application should be able to directly access data from a variety of data sources without modification to the application. For example, an application could access data from dBASE in a stand-alone, small office environment, and from SQL Server or Oracle in a larger, networked environment. Due to these challenges, some Fortune 500 firms have as little as 1 percent of enterprise data in a form that is truly accessible.

{ewc mvimg, mvimage,!707_02.bmp}

These challenges are common to developers of off-the-shelf applications, and to corporate developers attempting to provide solutions to end users or to migrate data to new platforms. These challenges grow exponentially for developers and support staff as the number of data sources grows.

The problems of database connectivity are apparent in the differences among the programming interfaces, DBMS protocols, DBMS languages, and network protocols of disparate data sources. Even when data sources are restricted to relational DBMSs that use SQL, significant differences in SQL syntax and semantics must be resolved.

The primary differences in the implementation of each of these components are:

® Programming interface. Each vendor provides his/her own proprietary programming interface. One method of accessing a relational DBMS is through embedded SQL. Another method is through an API.

® DBMS protocol. Each vendor uses proprietary data formats and methods of communication between the application and the DBMS. For example, there are many different ways to delineate the end of one row of data and beginning of the next.

® DBMS language. SQL has become the language of choice for relational DBMSs, but many differences still exist among actual SQL implementations.

® Networking protocols. There are many diverse local area network (LAN) and wide area network (WAN) protocols in networks today. DBMSs and applications must coexist in these diverse environments. For example, SQL Server may use DECnet on a VAX, TCP/IP on UNIX, and Netbeui or SPX/IPX on a PC.

To access various database environments, an application developer would have to learn to use each vendor's programming interface, employ each vendor's SQL, and ensure that the proper programming interface, network, and DBMS software were installed on the client system. This complexity makes broad database connectivity unfeasible for most developers and users today.

Approaches to Database Connectivity

Several vendors have attempted to address the problem of database connectivity in a variety of ways. The primary approaches include using gateways, a common programming interface, and a common protocol.

Gateways

Application developers use one vendor's programming interface, SQL grammar, and DBMS protocol. A gateway causes a target DBMS to appear to the application as a copy of the selected DBMS. The gateway translates and forwards requests to the target DBMS and receives results from it. For example, applications that access SQL Server can also access DB2 data through the Micro Decisionware DB2 Gateway. This product allows a DB2 DBMS to appear to a Windows-based application as a SQL Server DBMS. Note that an application using this gateway would need a different gateway for each type of DBMS it needs to access, such as DEC Rdb, Informix, Ingres, and Oracle.

The gateway approach is limited by architectural differences among DBMSs, such as differences in catalogs and SQL implementations, and the need for one gateway for each target DBMS. Gateways remain a very valid approach to database connectivity, and are essential in certain environments, but are typically not a broad, long-term solution.

{ewc mvimg, mvimage,!707_03.bmp}

Common interface

A single programming interface is provided to the developer. It is possible to provide some standardization in a database application development environment or user interface even when the underlying interfaces are different for each DBMS. This is accomplished by creating a standard API, macro language, or set of user tools for accessing data and translating requests for, and results from, each target DBMS. A common interface is usually implemented by writing a driver for each target DBMS.

Common protocol

The DBMS protocol, SQL grammar, and networking protocols are common to all DBMSs, so the application can use the same protocol and SQL grammar to communicate with all DBMSs. Examples are remote data access (RDA) and distributed relational database architecture (DRDA). RDA is an emerging standard from SAG, but not available today. DRDA is IBM's alternative DBMS protocol. Common protocols can ultimately work very effectively in conjunction with a common interface.

There are several current vendor-specific approaches that address database connectivity. These current approaches have several limitations. Many companies expend resources solving the same problem. This results in diverse implementations and duplication of effort. The results are inconsistent interfaces for end users and developers, overlap in effort, and a compromise in functionality and connectivity options. Although some current implementations provide viable solutions, none have the critical mass to emerge as a de facto standard.

Common interfaces, protocols, and gateways may be combined. A common protocol and interface provides a standard API for developers as well as a single protocol for communication with all databases. A common gateway and interface provides a standard API for developers and allows the gateway to provide functionality, such as translation and connectivity to wide area networks, that would otherwise need to be implemented on each client station. Note that a common gateway or protocol still requires a common interface to hide complexities from developers.

The ODBC Solution

ODBC addresses the heterogeneous database connectivity problem using the common interface approach. Application developers use one API to access all data sources. ODBC is based on a CLI specification, which was developed by a consortium of over 40 companies (members of the SQL Access Group and others), and has broad support from application and database vendors. The result is a single API that provides all the functionality that application developers need, and an architecture that database developers require to ensure interoperability. This will result in a rich set of applications that use ODBC, and provide applications with much broader access to data than ever before.

How ODBC Works

ODBC defines an API. Each application uses the same code, as defined by the API specification, to talk to many types of data sources through DBMS-specific drivers. A Driver Manager sits between the applications and the drivers. In Windows, the Driver Manager and the drivers are implemented as dynamic-link libraries (DLLs).

{ewc mvimg, mvimage,!707_04.bmp}

The application calls ODBC functions to connect to a data source, send and receive data, and disconnect.

The Driver Manager provides information to an application such as a list of available data sources; loads drivers dynamically as they are needed; and provides argument and state transition checking.

The driver, developed separately from the application, sits between the application and the network. The driver processes ODBC function calls, manages all exchanges between an application and a specific DBMS, and may translate the standard SQL syntax into the native SQL of the target data source. All SQL translations are the responsibility of the driver developer.

Applications are not limited to communicating through one driver. A single application can make multiple connections, each through a different driver, or multiple connections to similar sources through a single driver.

To access a new DBMS, a user or an administrator installs a driver for the DBMS. The user does not need a different version of the application to access the new DBMS. This is a tremendous benefit for end users, as well as providing significant savings for IS organizations in support and development costs.

What ODBC Means to the End User

End users do not work directly with the ODBC API, but they benefit in several ways when they use applications written with ODBC. Users may:

® Select a data source from a list of data source names or supply the name of a data source in a consistent way across applications.

® Submit data access requests in industry-standard SQL grammar regardless of the target DBMS.

® Access different DBMSs by using familiar desktop applications. When a requirement arises to access data on a new platform, users will have a common level of functional capabilities while accessing the new data with familiar tools.

{ewc mvimg, mvimage,!707_05.bmp}

What ODBC Means to Application Developers

ODBC was designed to allow application developers to decide between using the least common denominator of functionality across DBMSs or exploiting the individual capabilities of specific DBMSs.

ODBC defines a standard SQL grammar and set of function calls that are based upon the SAG CLI specification, called the *core grammar* and *core functions*, respectively. If an application developer chooses only to use the core functionality, they need not write any additional code to check for specific capabilities of a driver.

With core functionality, an application can:

® Establish a connection with a data source, execute SQL statements, and retrieve results.

® Receive standard error messages.

® Provide a standard logon interface to the end user.

® Use a standard set of data types defined by ODBC.

® Use a standard SQL grammar defined by ODBC.

ODBC also defines an extended SQL grammar and set of extended functions to provide application developers with a standard way to exploit advanced capabilities of a DBMS. In addition to the above features, ODBC includes a set of extensions that provide enhanced performance and increased power through the following features:

® Data types such as date, time, timestamp, and binary.

® Scrollable cursors.

® A standard SQL grammar for scalar functions, outer joins, and procedures.

® Asynchronous execution.

® A standard way for application developers to find out what capabilities a driver and data source provide.

Finally, ODBC supports the use of DBMS-specific SQL grammar, allowing applications to exploit the capabilities of a particular DBMS.

What ODBC Means to Database Developers

One ODBC driver can be developed that provides access to the DBMS. Any ODBC application may then gain access to that DBMS. This provides a wider number and variety of tools that will work with the DBMS, resulting in larger market potential for vendors and a wider variety of tools for corporations to choose from.

Industry Commitment to ODBC

ODBC enjoys a great deal of industry momentum and acceptance as the dominant standard. Database vendors,

with the help of third-party developers, have created drivers for their products. Several major application vendors have now shipped products which are ODBC-enabled. ODBC's acceptance to date is due to a variety of reasons:

- It is an implementation of the SAG CLI specification, and is therefore vendor-neutral and open. This open systems approach solves a problem common to everyone in the software industry.
- As a portable API, it can be a common data access language for both the Windows and Macintosh environments, and possibly other operating systems in the future.
- By providing different conformance levels, ODBC allows developers to choose between a least common denominator approach (allowing common access to the broadest set of DBMSs), and being able to fully exploit advanced feature sets in the more robust DBMSs.

The following databases will be supported by one or more database drivers by the end of 1993:

ADABAS SQL Server	IBM DB2/6000	Quadbase
Btrieve	IBM SQL/400	Raima
CA-IDMS	IBM SQL/DS	R:BASE
CA-Datacom	Informix	Siemens/Nixdorf SESAM
CA-DB	Ingres	Siemens/Nixdorf UDSD
DAL	Integra SQL	SQL Server
dBASE	Interbase	SupraServer
DEC Rdb	Microsoft Access	Systems 2000
DEC RMS	Microsoft Excel	Tandem NonStop SQL
Focus	Microsoft FoxPro	Teradata
Formatted Text	Model 204	Text files
Gupta SQLBase	NetWareSQL	UNIFY
HP ALLBASE/SQL	Nomad Gateway	WATCOM SQL for Windows
HP Image/SQL	Oracle	White Cross 9000
IBI EDA/SQL	Paradox	XDB
IBM DB2	PICK	
IBM DB2/2	Progress	

The following application vendors have released ODBC-enabled products or have publicly endorsed ODBC technology as of July 1993:

Andyne	JYACC
Approach Software	Knowledgeware
Blue Sky Software	LABTECH
Blyth	Lotus Development
Brio	mdbs
Bull HN	Micro Design International
Canaan Analytics	Microsoft Corporation
Cincom Systems, Inc.	Natural Language
Computer Corporation of America	Neuron Data
Clear Access	PageAhead Software Corp.
Cognos	Parcplace Systems
Computer Associates	Pilot Technologies
Coromandel	Pioneer Software Systems Corp.

CSA	Powersoft
DataEase International	Progress Software
Dharma Systems	Revelation
EASEL	SPC
Fairfield Software	SPSS
FileNet	Sterling Software
Genus Software	SoftwareTechnologies
Great Plains Software	The Dodge Group
Guild Software	Trinzic
Gupta Technologies, Inc.	Visionware
Hewlett-Packard Company	Winclient
Icons International	Xdb

Future Plans for ODBC

The next release of the ODBC Software Development Kit, version 2.0, will be available in the first quarter of 1994, and will provide considerable enhancements based on input from software vendors and corporate developers. The new version will support the 32-port technology of Windows NT, scrollable cursors independent of driver capability, additional sample applications, and sophisticated debugging tools.

Apple has endorsed ODBC technology and will continue to enable developers to exploit the power of ODBC. Apple has announced their ODBC Software Developers Kit and will announce additional ODBC-compatible drivers later this year. ODBC is a completely portable API, and may be ported to other major operating system environments in the future.

The Current Status of ODBC

ODBC is available for software vendors and corporate developers. The ODBC SDK includes development tools, documentation, a dBASE test driver for developing and testing ODBC applications, and an application for testing ODBC drivers. ODBC will be included in future versions of the Windows operating system. Microsoft has shipped numerous Windows-based applications with ODBC technology, and these products will continue to provide data connectivity through ODBC in future releases.

ODBC SDK

The Microsoft ODBC SDK contains everything necessary for developing Windows-based ODBC applications and drivers. The ODBC SDK version 1.0 includes the following:

® ODBC Programmer’s Reference

® ODBC SDK Guide

® Sample source code

® Driver Manager DLL

® ODBC administrator (for configuring data sources)

® ODBC test program (for testing drivers)

® Visual Basic demonstration application

® dBASE test driver

® On-line Help

Apple has announced its ODBC Software Developers Kit, which enables Macintosh developers to build applications and drivers using ODBC. The kit consists of an installer disk, ODBC test application, a test DBMS driver, and other components to assist in the development of ODBC-compatible applications for the Macintosh. Please contact Apple for availability and pricing information.

Driver Catalog

The Microsoft *ODBC Driver Catalog* provides a quick reference for driver availability. This catalog contains key information on vendors that provide ODBC drivers, pricing, availability, and contacts.

Information Resources for ODBC

CompuServe provides the Windows Extension forum, which has an ODBC section. Please refer to this forum for updates on ODBC's status. The library in this section contains relevant files that may be downloaded. To access the forum from CompuServe, type GO WINEXT, and then select the ODBC section.

Technet, Microsoft's technical information network, is a community of support professionals, system integrators and solution builders, many of whom have experience implementing ODBC technology.

The Microsoft Developer Network publishes technical information for all developers who write applications for Microsoft operating systems or who use Microsoft development tools. ODBC-experienced developers contribute information to the network, which helps speed the acceptance and implementation of ODBC technology.

Summary

Providing data access to applications in today's heterogeneous database environment is very complex for software vendors as well as corporate developers. ODBC solves this data access problem for software vendors and corporations by providing a standard, open, and vendor-neutral API. ODBC allows corporations and software vendors to protect their investments in existing DBMSs, and protect developers' acquired DBMS skills. ODBC benefits users as more end-user applications connect to additional data sources, making the vast volumes of corporate data more readily available. ODBC is a portable API, which allows it to be a cross-platform tool. It is based upon the SQL Access Group (SAG) Call Level Interface (CLI) and provides a standard SQL language based upon ANSI standards. With ODBC, Microsoft provides many benefits to developers, end users, and the industry by creating a vendor-neutral, open, and powerful means of accessing data.

Before working with database functionality from Visual Basic, you should understand data access capabilities, as well as database terminology.

This section includes the following topics:

® Data Access Options in Visual Basic

® Understanding Database Fundamentals

The **Data** control implements data access by using the Microsoft Jet database engine. This technology provides access to many database formats, and enables you to create data-aware applications without writing any code.

To create a database application that uses the **Data** control, you use the following two steps:

1. Add a **Data** control to your form, and set properties to specify the database and table from which you will obtain the data.
2. Add data-aware controls to your form, and set properties to bind the controls to the **Data** control so the data can be displayed.

Using Data-Aware Controls

When you bind a data-aware control that you have placed on a form, the data from the database is automatically displayed in the bound control. If a user changes the data in a bound control, those changes are automatically updated in the database when the user moves to another record. A number of the intrinsic controls in Visual Basic are data-aware, including the check box, image, label, picture box, text box, list box, combo box, and OLE container controls.

The following illustration shows a sample form that contains a **Data** control and three bound controls.

{ewc mvimg, mvimage,!v02g020.bmp}

To see a demonstration of how to use the **Data** control to view and edit information from the Northwind database, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Setting Properties of the Data Control

The following steps explain how to connect a **Data** control to a database.

To connect a Data control to a database

1. Specify the database to which you want to gain access by setting the **DatabaseName** property to the name of a database.
2. To specify which records to retrieve, set the **RecordSource** property to the name of a table within the database, or to an SQL string.

Note To connect to a dBase, Paradox, or Btrieve database, set the **DatabaseName** property to the folder that contains the database files, and set the **Connect** property to the appropriate database type.

Binding Controls

After you set the properties for the **Data** control, you must bind the individual data-bound controls to the **Data** control, and then specify which field in the table each control will display.

To bind a data-bound control to a Data control

1. At design time, set the **DataSource** property of the bound controls to a **Data** control.
2. At design time or run time, specify which field you want displayed in the bound control by setting the **DataField** property.

   The **DataField** property can be set at design time or run time.

In a database application, users work with the **Data** control to move between records in the database. Users can click the buttons on the **Data** control to move forward or backward either one record at a time, or directly to the first or last record.

This topic explains how the **Data** control uses a **Recordset** object to display data requested by the user.

What Is a Recordset?

A recordset is the entire set of records to which a **Data** control refers. The recordset is stored in memory, overflowing to disk if necessary.

To manipulate the recordset, you use the **Recordset** property of the **Data** control. The **Recordset** contains one current record. The information from the current record is displayed in the bound controls. You can change the position of the current record by clicking the **Data** control or by writing code that uses methods of the **Recordset** object.

Determining Recordset Boundaries

If you use code to change the position of the current record, you must determine the beginning and end of the recordset by checking the **EOF** and **BOF** properties of the **Recordset** object. When you move to the **EOF** or **BOF** record, the action indicated by the value of the **BOFAction** or **EOFAction** property is executed.

For example, you can set **EOFAction** to add a new record automatically. If you set the **EOFAction** property to **EOF**, then no action will be taken when you move to the **EOF** record. If you move one record past the **BOF** or **EOF** records, you receive a run-time error.

The following illustration shows how the **BOF** and **EOF** properties determine the boundaries of the **Recordset** object.

{ewc mvimg, mvimage,!v02g025.bmp}

To use the **Recordset** object of a specific **Data** control, specify the **Recordset** property of the Data control, as shown in the following code:

```
Data1.Recordset.MoveNext 'Moves current record.
If Data1.Recordset.EOF Then
    Data1.Recordset.MoveLast
```

By Dale Rogerson
Briefly described in Books Available from Microsoft Press
Inside COM
Published by Microsoft Press

Chapter 1

Components

An application usually consists of a single monolithic binary file. Once the compiler generates the application, the application doesn't change until the next version is recompiled and shipped. Changes in operating systems, hardware, and customer desires must all wait for the entire application to be recompiled. The application is a rock in the river of change. The entire software industry rushes on into the future as the shipped application becomes older and more outdated.

With the current pace of change in the software industry, applications cannot afford to be static after they have shipped. Developers must find a way to breathe new life into applications that have already shipped. The solution is to break the monolithic application into separate pieces, or components. (See Figure 1-1.)

{ewc mvimg, mvimage,!illust.bmp}
Figure 1-1. Breaking a monolithic application (left) into components (right) makes it adaptable to change.

As technology advances, new components can replace the existing components that make up the application. (See Figure 1-2.) The application is no longer a static entity destined to be out-of-date before it ships. Instead the application evolves gracefully over time as new components replace older components. Entirely new applications can be built quickly from existing components.

Traditionally, an application has been divided into files, modules, or classes, which are compiled and linked to form the monolithic application. Building applications from components, a process called *component architecture,* is very different. A component is like a mini-application; it comes packaged as a binary bundle of code that is compiled, linked, and ready to use. The monolithic application is no more. In its place is a custom component that connects with other components at run time to form an application. Modifying or enhancing the application is a simple matter of replacing one of these constituent components with a new version.

{ewc mvimg, mvimage,!illust.bmp}
Figure 1-2. A new, improved component D has replaced the old component D.

Breaking the monolithic application into components requires a powerful hammer. The hammer we will use is called *COM*. COM, the *Component Object Model,* is a specification for a way of creating components and building applications from these components. COM was developed more than four years ago at Microsoft to make Microsoft applications more flexible, dynamic, and customizable. Almost all currently shipping Microsoft applications use COM. Microsoft's ActiveX technologies are built from COM components.

This book is about building custom COM components using C++. By following the examples in this book, you'll see how to build COM components that can be assembled to form applications that are likely not only to survive but to grow and to evolve with the passage of time.

Before we look at COM in more detail, let's examine some of the benefits of a component architecture and what is required to build an application out of components.

The topics included in this section are:

® Component Benefits
® Component Requirements
® COM
® Component Conclusions

Dale Rogerson is a veteran writer and a founding contributor to the Microsoft Developer Network on the subject of COM. He currently codes internal COM interfaces as part of Microsoft's Visual C++ development team.

Click here to connect to the Microsoft Visual Basic Home Page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This site provides information about Microsoft Visual Basic and links to numerous other pages for related information.

Click here to connect to the Visual Basic for Applications Home Page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This page describes, in a question and answer format (FAQ), Visual Basic for Applications (VBA), and provides tips and tricks for integrating VBA into your applications through various workshops provided by Microsoft (some of which are free). You can also discover what companies are licensing VBA and jump to their web sites from here. Information on how to license VBA is also provided.

Click here to connect to the Advanced Visual Basic Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This page is maintained by developers for developers. It has information and advanced (not beginner) topics on Visual Basic 4.0, Visual Basic Script, OLE, and more. The site also contains a page with the "Best of the Microsoft Knowledge Base." The site is a wealth of material, including an updated "What's New?" page, providing bits and pieces of interesting news relating to Visual Basic and related topics, articles on Visual Basic programming topics, hints and tricks, and links to other sites.

Click here to connect to the Ask the VB Pro Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This Web site features information on Visual Basic and VBScript. The site also includes information about new products (including news releases from vendors) relating to Visual Basic, book reviews, and a question and answer section. You'll also find a list of Visual Basic Internet Resources, and many other topics relating to Visual Basic.

Click here to connect to The BASIC Archives Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This site is devoted to the BASIC programming language and provides answers to commonly asked programming questions, the history of the BASIC language, and articles on solving specific programming problems.

Click here to connect to Chris & Tim's Rapid Application Development Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This web page is devoted to helping developers learn how to write dynamic link libraries (DLLs) and Visual Basic controls (VBXs). The site has many Visual Basic 3.0 16-bit controls and Visual Basic 4.0 32-bit controls that can de downloaded; visitors may also upload their own Visual Basic application programs to the site, so others may use them. For those new to Visual Basic, this site provides a section that guides you through creating your first application program in Visual Basic. You'll also find a Visual Basic mailing list for submitting questions and receiving answers to programming problems

Click here to connect to the Entisoft Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This site provides information relating to Entisoft's Visual Basic add-on products. It also provides links to other sites, including the most popular Web search engines, such as Yahoo.

Click here to connect to the Jens Balchen Jr. - Visual Basic Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This site provides a fantastic number of links to other Web sites and pages relating to Visual Basic. The site also provides useful information about Visual Basic 4.0, including tips and tricks, FAQ documents for Visual Basic, and a section devoted to Visual Basic resource material (books, electronic magazines, etc.). You can also download software from this site, including Microsoft Internet Explorer.

Click here to connect to the Mike Dixon's QAID Pages:
{ewc mvimg, mvimage,!intjump.bmp}

This huge site contains two pages, one devoted to Visual Basic, the other to Windows 95. The site provides information about Visual Basic in a question and answer format, provides tips and tricks, code samples, and product information.

Click here to connect to Paul Treffers Visual Basic Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This page, which is presented graphically as a Windows form, provides links to other Visual Basic sites and contains extensive information about Visual Basic applications and tools (commercial and shareware) that you can download. The Visual Basic Code Corner section of this page contains code examples you can use in your own applications.

Click here to connect to The Phil Weber Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This site lets you access all code samples from Phil Weber's articles published in the *Visual Basic Programmer's Journal*. In addition, this site provides a number of useful Visual Basic tip articles that explain how to accomplish a specific programming task.

Click here to connect to the TegoSoft Self-Study Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

From this site you can download a large number of ActiveX (OCX) controls. You can also download a sample of the Self-Study tutorials for Visual Basic 4.0, including information on how to order the commercial tutorials from TegoSoft. In addition, this site provides a list of several Visual Basic programming books, with a description of the content of each book.

Click here to connect to The Toolbox Visual Basic Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This site claims to have compiled all information and resources into one location. The information has been culled from many Internet sources and is updated on a monthly basis. This site has pages devoted to Visual Basic Script, Java, VRML development; a reference shelf; newsgroups and user groups; electronic magazines; and links to other Visual Basic-related Web sites.

Click here to connect to the Visual Basic Education & Certification Page:
{ewc mvimg, mvimage,!intjump.bmp}

This Web page is devoted to helping people learn Visual Basic and provides help for those studying for the Microsoft Visual Basic Certification Exam 70-65 (which is required to obtain the Microsoft Certified Solution Developer certification). The page is designed in the format of a study sheet for the exam.

See The Microsoft Certified Professional Program for more information on certification requirements and exams.

Click here to connect to the Visual Basic Programmer's Journal Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This Web page is provided as an addition to the *Visual Basic Programmer's Journal* magazine, which is devoted to Visual Basic programming.

Click here to connect to The Visual Basic Resource List Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This large site offers a search engine to find vendors who produce third-party Visual Basic controls. The site also provides links to other sites, categorized by subject matter, such as Books and Magazines.

Click here to connect to the Visual Basic Starting Point Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This site is a search engine that lets you find any Visual Basic related site or file. The site also features a Question and Answer section, and a Hall-of-Fame section for determining who's who in the Visual Basic world.

Click here to connect to the VB4UandMe Web Page:
{ewc mvimg, mvimage,!intjump.bmp}

This site contains an archive of Visual Basic source code files you can download. The site also has a page devoted to beginning Visual Basic developers, and a page with links to other Visual Basic-related sites. You can also fill out a submissions form to add a link through this page to your own Web pages.

Click here to connect to the VBxtras, Inc. Home Page:
{ewc mvimg, mvimage,!intjump.bmp}

This site contains many Visual Basic add-ons product reviews, and provides links to other Visual Basic Web sites, online magazines, and newsgroups.

Click here to connect to the Wild Web of Visual Basic Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This site is devoted exclusively to the novice Visual Basic developer. The site offers code samples, back issues of *Visual Basic Tips and Tricks* newsletters, and links to other Visual Basic sites.

Click here to connect to the Fawcette Technical Publications web site.
{ewc mvimg, mvimage,!intjump.bmp}

Fawcette Technical Publications is the leading supplier of information products for professional Windows developers including: Visual Basic Programmer's Journal, the leading Windows development magazine; Microsoft Interactive Developer, the premier source of information on developing interactive applications, The Development Exchange, the most comprehensive online resource available for Windows development information and tools; VBITS conferences; the monthly VBCD CD-ROM; Companion Products directories; and the Windows Components Forums on CompuServe.

Fawcette Technical Publications
209 Hamilton Avenue
Palo Alto, CA 94301
Toll Free: 800-848-5523
International: 415-833-7100
Fax: 415-853-0230

Microsoft Press, the book publishing division of Microsoft, offers a wide variety of training and reference materials for developers. Below are their latest Visual Basic-related releases.

® Active Visual Basic

Guy Eddon, Henry Eddon

Active Visual Basic introduces the features and capabilities of Visual Basic that give programmers the power to create Internet-enabled applications and interactive Web content. After a technical overview of the Internet and Internet-related capabilities of Visual Basic, the book covers the Internet Control Pack and the creation of ActiveX controls and documents. Advanced topics in the final section include overviews of developing Internet servers and accessing the Windows Internet API. Highly motivated developers, eager to keep up with the most exciting growth area for Visual Basic development, will want this book.

ISBN: 1-57231-512-1

Price: $39.99

® Hardcore Visual Basic, Second Edition

Bruce McKinney

The first edition has been a big hit with serious Visual Basic programmers. And in this second edition, they'll find more help for expanding what Visual Basic can do. They'll get clear direction on exploiting object-oriented code to achieve behavior that once seemed to be the sole province of C or C++. Like the first edition, this one tackles tough issue with smart coding, practical tools and analysis, and an unblinking style. As Visual Basic continues to evolve, it delivers more power (and complexity) to its audience—and this book is a valuable way for programmers to keep up with that evolution. *Hardcore Visual Basic, Second Edition*, is expected to continue the sales success of the previous edition.

ISBN: 1-57231-422-2

Price: $39.99

® Inside COM

Dale Rogerson

A foundational book for a core technology.

Because COM is the foundation of both OLE and ActiveX, professionals are rapidly grasping how important it is to future development work everywhere. This book shows developers just how critical—and accessible—COM can be, especially for experienced C++ programmers. In Inside COM, you'll find a concise and practical account of how COM supports the creation of interchangeable binary components. You'll see how to write programs to exploit COM architecture. You'll also get a CD-ROM containing sample source code, a complete sample application, documentation and development tools for COM, and more. All in all, Inside COM is a definitive title on an increasingly significant subject.

ISBN: 1-57231-349-8

Price: $34.99

® Microsoft Visual Basic Step by Step

Michael Halvorson

Microsoft Visual Basic Step by Step is the easiest and fastest way to learn the next version of Microsoft Visual Basic. And it's for anyone who wants a quick and easy way to program for Windows. Prepared by an author with numerous Visual Basic and Basic titles to his credit, the book includes approximately nine hours of instruction, along with plenty of helpful screen shots and illustrations. As Visual Basic is extended to cover burgeoning applications on the Internet, demand for instruction in the language will only increase. This book will give its readers the thorough grounding they need to enter and take advantage of the world of Visual Basic.

ISBN: 1-57231-435-4

Price: $34.99

Sockets provide a mechanism for network interprocess communication for TCP/IP and the Internet. You implement socket servers and clients by using the **Winsock** control.

Features of Sockets

A socket has the following three essential features.

® The interface to which the socket is bound, specified by an IP address.

® The port number or ID to which the socket will send data, or from which it will receive data.

® The type of socket (either stream or datagram).

The primary difference between stream and datagram types of sockets is their connection state. The stream or Transmission Control Protocol (TCP) is a connection-based state. It is similar to a telephone transaction in that the user must establish a connection before communication can occur.

The datagram or User Datagram Protocol is a connectionless state. It is similar to a radio broadcast in that an explicit, one-to-one connection is not made between the server and client.

Generally, a server listens on a well-known port for all installed network interfaces, and a client initiates communication from a specific interface by using any available port.

Uses for Sockets

Typically, sockets are used to create the following types of applications.

® Client applications that collect user information before sending it to a central server.

® Server applications that function as a central collection point for data from several users.

® Chat applications.

Adding the Winsock Control to a Project

If the **Winsock** control is not already on the Toolbox for your project, you can add it as follows.

u **To add the Winsock control to your project's Toolbox**

1. On the **Project** menu, click **Components**.
2. On the **Controls** tab, select **Microsoft Winsock Control 5.0**, and then click OK.

This topic introduces you to the methods, properties and events used to connect servers and clients through the **Winsock** control. It also explains how to create a peer-to-peer connection between two computers.

Coding a Winsock Server

To code a server to use the **Winsock** control, you use the following process.

1. Use the **Protocol** property to specify the protocol that the server will use when communicating with clients.
2. Use the **LocalPort** property to specify the port that the server will use when communicating with clients.
3. Use the **Listen** method to code a method that will listen for incoming connection requests from clients.
4. Use the ConnectionRequest event to code an event that accepts or rejects a request for a connection.

   For more information about this process, see Coding a Winsock Server.

Coding a Winsock Client

To code a client to use the **Winsock** control, you follow a process that closely parallel those described for coding the server.

1. Use the **RemoteHost** property to specify the name of the host with which the client will communicate.
2. Use the **RemotePort** property to specify the port on the host through which the client will communicate with the host.
3. Use the **Connect** method to initiate a connection between the client and the host.

   For more information about this process, see Coding a Winsock Client.

Coding Client and Server to Send and Retrieve Data

Once you have established communication between a client and a server, data can be passed between them. To enable a client and server to pass data, use the following process.

1. Use the **SendData** method in both the client and the server to pass data to the other computer.
2. Use the **GetData** method in the DataArrival event in both the client and the server to retrieve data from the other computer.

   For information about sending data between a client and a server, see Sending and Retrieving Data.

Coding a Peer-to-Peer Application

A simpler approach to writing a socket application is to create a peer-to-peer application, because a peer-to-peer application does not require an explicit connection.

To code the first peer, you use this process.

1. Use the **RemoteHost** property to specify the name of the host with which the first peer will be communicating.
2. Use the **RemotePort** property to specify the port on the host through which the first peer will communicate with the host.
3. Use the **Bind** method to connect the local port to the remote port by using the local Internet Protocol (IP) address.

To code the second peer, you use this process.

1. Use the **RemoteHost** property to specify the name of the host with which the second peer will communicate.
2. Use the **RemotePort** property to specify the port on the host through which the second peer will communicate with the host.
3. Use the **Bind** method to connect the local port to the remote port by using the local Internet Protocol (IP) address.

To see a demonstration of how to create a peer-to-peer chat application, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

When you create a server application for use with the **Winsock** control, you set it to listen on a designated port for messages from clients. When a client makes a connection request, the server can accept the request and complete the connection.

Setting the Protocol Property

To create a Winsock server, you must first define the type of server by setting the **Protocol** property either at design time or at run time. The default value for this property is **TCP**.

Setting the LocalPort Property and Invoking the Listen Method

You then set the **LocalPort** property to an integer, and invoke the **Listen** method. This can be done in the Form_Load event, as shown in the following code.

```
Private Form_Load()
    tcpServer.LocalPort = 1001
    tcpServer.Listen
End Sub
```

Invoking the ConnectionRequest Event

The final step is to invoke the ConnectionRequest event, and check the state of the server. If the server has a port open, you use the **Accept** method to accept the connection to the client. For example:

```
Private Sub tcpServer_ConnectionRequest _
    (Index as Integer, ByVal requestID as Long)
    If Index = 0 Then
        intMax = intMax +1
        Load tcpServer(intMax)
        tcpServer(intMax).LocalPort = 0
        tcpServer(intMax).Accept requestID
    End If
End Sub
```

Visual Basic implements FTP and HTTP functionality by using the **Internet Transfer** control. This control connects a client to any server that supports either of these protocols, and retrieves files by using either the **OpenURL** method or the **Execute** method.

Server functionality is generally provided by an application such as the Microsoft Internet Information Server or Personal Web Server.

If the **Internet Transfer** control is not already on the Toolbox for your project, you can add it by using the following steps.

To add the Internet Transfer control to your project's Toolbox

1. On the **Project** menu, click **Components**.
2. On the **Controls** tab, select **Microsoft Internet Transfer Control 5.0**, and then click OK.

This topic introduces you to the basic methods, properties, and events of the **Internet Transfer** control that you use to implement FTP and HTTP functionality in a client. It is assumed that you already have knowledge of the FTP and HTTP protocols.

{ewc mvimg, mvimage,!tip.bmp}

The OpenURL and Execute Methods

To transfer data from a server to a client application, you use either the **OpenURL**method or the **Execute** method. The following discussion will help you to determine which method is most appropriate in your situation.

The **OpenURL** method is a synchronous call that opens and returns the document specified by the user with either FTP or HTTP. The variant data returned can be retrieved as either string or binary data. The **OpenURL** method is much easier to code than the **Execute** method.

The **Execute** method is an asynchronous request to a remote server. The status of the asynchronous operation is generally provided in the StateChanged event of the **Internet Transfer** control. The **Execute** method provides the client with the ability to execute a wide range of file operations, such as downloading, uploading, deleting, and receiving directory listings. The **Execute** method supports HTTP commands such as GET, HEAD, POST, and PUT, and also supports FTP commands such as GET, PUT, SEND, and DELETE. Although **Execute** provides greater functionality than **OpenURL** it is more complex to code.

For a complete list of the commands that the **Execute** method supports, search for **Internet Transfer Control** in Visual Basic Books Online, and then click **Microsoft Internet Transfer Control**.

Coding a Client for Asynchronous Operation

To code a client for asynchronous functionality, you use the following process.

1. Use the **AccessType** property to set a reference to a valid server.
2. Use the **Execute** method to begin the transfer of data.
3. Use the StateChanged event to monitor the control's connection state by using the StateChanged event.
4. Use the **ResponseCode** and **ResponseInfo** properties to handle any errors that occur.
5. Use the **GetChunk** method to retrieve data from the control's buffer.
6. Use the **StillExecuting** property to enable asynchronous processing.
7. Use the **Cancel** method to stop a transfer when a user wants to stop a data transfer before it is completed.

For information about coding a client for asynchronous operation, see Coding for Asynchronous Operations.

Coding a Client for Synchronous Operation

To code a client for synchronous functionality, you use the following process.

1. Use the **AccessType** property to set a reference to a valid server.
2. Use the **OpenURL** method to transfer data.

For information about coding clients for synchronous operation, see Coding for Synchronous Operation.

To see a demonstration of how to code clients for FTP and HTTP functionality, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

This topic explains the steps you follow to enable asynchronous functionality in a client application.

Setting the AccessType Property

First, you set the **AccessType** property of the **Internet Transfer** control to specify how the server should be accessed, either directly or through the proxy server.

The following code shows you how to set the **AccessType** property by using the **icUseDefault** constant, which uses the default settings found in the registry to access the Internet.

```
Inet1.Accesstype = icUseDefault
```

Using the Execute Method

Next, initiate a request for data by using the **Execute** method, as shown in the following code.

```
Inet1.Execute url, Operation
```

The **Execute** method has four arguments: *url*, *operation*, *data*, and *requestHeaders*. FTP operations take only the *operation* argument, and an optional *url* argument.

For a complete list of the various file operations implemented from the **Execute** method, search for the **Execute Method** in Visual Basic Books Online, and then click **Execute Method (Internet Control)**.

The following code uses the **Execute** method to retrieve a directory listing from an FTP server.

```
Inet1.Execute "ftp://ftp.microsoft.com", "DIR"
```

Using the StateChanged Event

Next, code the **StateChanged** event to handle the download of data, and to provide a mechanism for error handling.

You determine when to retrieve data by testing for the **icResponseCompleted** state in the StateChanged event. You then use the **GetChunk** method to retrieve the data.

The following code shows how to use a **Select Case** statement to test for **icResponseCompleted**.

```
Private Sub Inet1_StateChanged(ByVal State as Integer)
    Select Case
    Case icResponseCompleted
        ' Copy the data using GetChunk.
    Case icError
        Inet1.Cancel
        ' Notify the user of the error and terminate
        ' the asynchronous operation.
    End Select
End Sub
```

Using the GetChunk Method

Next, move the data from the control's buffer to a final location by using the **GetChunk** method. To move the data, you can include code in the **icResponseCompleted** portion of the **Case** statement, as shown in the following code.

```
    case icResponseCompleted
        Dim s As String
        ' Get a chunk of data.
        s = Inet1.GetChunk(1024)
        Do While Len(s) > 0
            txtOutput = txtOutput & s
```

```
            s = Inet1.GetChunk(1024)
        Loop
```

Using the StillExecuting Property

Finally, use the **StillExecuting** property to monitor the state of the control. The control returns **True** when it is engaged in an operation such as retrieving a file from the Internet. When the control is busy, it does not respond to other requests.

The following code shows how to use the **DoEvents** function to pass control to the operating system so that other events in its queue can be processed.

```
Inet1.Execute ...
While Inet1.StillExecuting
    DoEvents
Wend
' Operation complete.
```

Canceling an Asynchronous Operation

One advantage of asynchronous operations is that a user can terminate an operation before completion. You can provide this functionality in a number of ways. For example, you can have users click a **Cancel** button on a modeless form, or press the ESC key. Regardless of the method, an application can use the **Cancel** method of the **Internet Transfer** control to terminate the operation.

The following code uses the Click event of a command button to enable a user to terminate an asychronous operation.

```
Private Sub cmdCancel_Click()
    If Inet1.StillExecuting Then
        INet1.Cancel
    End If
End Sub
```

Components can be designed and built to provide services from different locations, depending on the intended use of the component.

For example, a user interface component would probably be best located as close to an individual client as possible, while a component that supplies statistical calculations of remote data would probably be on a separate computer, with the data those calculations manipulate.

Components can be run in any one of three places: in the same address space as the client, in an address space separate from the client application, or on a remote computer. The following table defines the differences between these types of locations for components.

Type of server	Location
In-process	An in-process component is usually implemented as a dynamic-link library (DLL). It runs in the same process space as its client application. This enables the most efficient communication between client and component, because you need only call the function to obtain the required functionality.
Out-of-process	An out-of-process component is usually implemented as an executable file, and runs in its own process space. Communication between the client and component will be slower because parameters and return values must be marshalled across process boundaries. However, a single instance of an out-of-process component can service many clients, share global data, and insulate other client applications from problems that one client might encounter.
Remote	Remote components are also out-of-process components, but they are located on a separate computer. While communication time between a client and a remote server is much greater than on a local server, remote servers allow processing to be done on a more powerful computer. The component can also be located closer to the work it is doing. For example, a component can be located closer to a remote database with which it interacts.

As with the **Instancing** property, the Visual Basic client does not have any control over the location of the component. However, the location can greatly impact the performance of the client, and in some cases, the location may also impact how the client will interact with the component.

When you work with out-of-process and remote components, you should ensure that the client minimizes the number of calls to objects created from the component.

For example, if you make repeated calls to an object, passing data on which the object acts, knowledge of the component may reveal a way to pass data in bulk with as few calls as possible.

Once you have set a reference to a type library, you declare object variables to hold pointers for the objects you want to create for an Automation client.

You can declare an object variable either as generic or specific, depending on how you will use the variable, and how the object will affect the performance of your application.

Note The scoping rules for variable declarations apply, even though they are object variables.

Generic Object Variables

Generic object variables can hold pointers to any type of object. The most generic variable is of type is **variant**. A **variant** data type variable holds numeric and string data, and can also hold a pointer to any type of object.

It is advisable to use the **Object** data type, as shown in the following code.

```
Dim x As Object
```

When you use generic variables, the compiler uses late binding to connect with the object. The object or objects to be used will not be known until run time. Because late binding does not use the type library at design time, you do not need to set a reference to the component in Visual Basic.

Specific Object Variables

Specific object variables refer to a discrete object type and can only hold pointers to that type.

For example, the following code declares a variable **ex** that will only hold a pointer to a Microsoft Excel **Application** object.

```
Dim ex as Excel.Application
```

If you try to store a different object type in that variable, an error will result. The use of specific object variables requires Visual Basic to have explicit knowledge of the object through its type library. Therefore, you must set a reference to the component. The Visual Basic compiler uses early binding when using specific object variables, which can greatly enhance performance.

Another advantage of using specific object variables is that Visual Basic will detect an object variable when you write code for the client application. With information from the type library, Visual Basic will display data about the available methods and properties, as well as the syntax of each method or property call.

Personal Web Server is available from a number of different sources. It is included in both Windows 95 and the Microsoft Internet Explorer Starter Kit. Personal Web Server is also available on the Internet, free of charge.

To download Personal Web Server from Microsoft.com, click this icon.
{ewc mvimg, mvimage,!intjump.bmp}

The **WebBrowser** control adds browsing, document viewing, and data downloading capabilities to your clients. It enables users to browse sites on the World Wide Web, and folders on a local file system and network.

The **WebBrowser** control supports navigation by means of hyperlinks and Uniform Resource Locators (URLs). The control maintains a history list that enables users to browse forward and back through previously browsed sites, folders, and documents.

The **WebBrowser** control includes support for parsing and displaying HTML-encoded documents. It is also an ActiveX document container, and can host any ActiveX document.

Richly formatted documents, such as a Microsoft Excel spreadsheet or Microsoft Word document, can be opened and edited in-place within the **WebBrowser** control.

Uses for the WebBrowser Control

Some of the uses of the **WebBrowser** control are:

® Enabling users to navigate to and view HTML documents on the World Wide Web or on an intranet.

® Providing a single frame in which users can view and edit all types of ActiveX documents.

® Creating a customized Web application based on the **WebBrowser** control.

Adding the WebBrowser Control to a Project

If the **WebBrowser** control is not already on the Toolbox for your project, you can add it by using the following steps.

u **To add the WebBrowser control to your project's Toolbox**

1. On the **Project** menu, click **Components**.
2. On the **Controls** tab, select **Microsoft Internet Controls**, and then click OK.

This topic introduces you to the methods, properties, and events of the **WebBrowser** control that you can use to create a Web browser.

To code a client to enable Web browser capabilities, you use the following steps.

1. Use the **Navigate** method to specify a resource to view. The resource must be identified by a URL.
2. Display the name and URL of a resource in controls on the **WebBrowser** control by using the **LocationURL** and **LocationName** properties in the NavigateComplete event.
3. Use the **GoBack**, **GoForward**, **GoSearch**, and **GoHome** methods to code the ability to navigate sites included in the **History** list.
4. Use the **Busy** property to determine whether the **WebBrowser** control is in the process of navigating to a new location or downloading a file, and use the **Stop** method to enable canceling a navigation or download activity before it is finished.
5. Use the **Refresh** and **Refresh2** methods to code quick reloading of a page that has already been displayed.
6. If you enable downloading from a Web page, use the DownloadBegin, ProgressChange, and DownloadComplete events to code a series of events that display a status bar during downloads.

Most of the work in creating a Web browser involves enabling navigation from one Web page to the next. This topic describes the steps you need to follow to enable navigation.

Using the Navigate Method

First, you specify a home page for the client. Typically, you do this in the Form_Load event by using the **Navigate** method.

The following code shows how to use the **Navigate** method to find a URL.

```
Private Form_Load()
    WebBrowser.Navigate _
        URL:="http://yourdirectory/yourpage.htm"
End Sub
```

Using the NavigationComplete Event

Next, you display the URL and name of the site that has been navigated to. Typically, you do this in the NavigationComplete event.

The following code puts the URL in a combo box on the form, and puts the name of the page in the second panel of the progress bar.

```
Private Sub wb_NavigateComplete(ByVal URL As String)
  cboURL = WebBrowser1.LocationURL
  sbMain.Panels(2) = WebBrowser1.LocationName
End Sub
```

Implementing the Navigation Buttons

Next, you add code for the **Back**, **Forward**, **Stop**, and **Refresh** buttons. The process for doing this is the same for all four buttons. You place command buttons on the form or toolbar, and in the Click event for each button, you place a statement that invokes the appropriate method.

This code shows how to use the **GoBack** method in a Click event.

```
Private Sub cmdGoBack_Click()
    WebBrowser1.GoBack
End Sub
```

Stopping a Navigation

To enable stopping a navigation, you use both the **Busy** property and the **Stop** method.

The following code shows how to code a command button with an **If...Then** construct to determine whether the **WebBrowser** control is busy, and to stop it if it is.

```
Private Sub cmdStop_Click()
  If WebBrowser1.Busy Then
    WebBrowser1.Stop
    WebBrowser1.GoBack
  End If
End Sub
```

Refreshing the View of a Web Page

To enable reloading of a page that has already been displayed, you can use either the **Refresh** method or the **Refresh2** method.

The **Refresh** method reloads the page that the browser is currently displaying, retrieving the data from the

cache of the client computer. The **Refresh2** method takes a parameter that specifies the level of refresh to be performed. When you use **Refresh2**, the page can be refreshed from the cache, refreshed only if the page has expired, or be refreshed from the server where it originated.

This code uses the **Refresh** method to reload the page currently displayed by the browser.

```
Private cmdRefresh_Click()
    WbBrowser1.Refresh
End Sub
```

Displaying Download Progress

If you are implementing download capabilities within the browser, you use the DownloadBegin, ProgressChange, and DownLoadComplete events. These events inform the user about what is happening during the download.

First, you code the DownloadBegin event to initialize the progress bar, as shown in the following code.

```
Private Sub WebBrowser1_DownloadBegin()
  ' Initialize the ProgressBar.
  pbMain.Move sbMain.Panels(1).Left, 10, _
    sbMain.Panels(1).Width, sbMain.Height - 10
  pbMain.Value = 0
  pbMain.Min = 0
  pbMain.Max = 100
  pbMain.Visible = True
End Sub
```

Next, you code the ProgressChange event to calculate the amount of progress that the process has made toward completion, and then display that value. For example:

```
Private Sub wb_ProgressChange(ByVal Progress As Long, _
    ByVal ProgressMax As Long)
  ' Update the ProgressBar.
  If Progress <> -1 And ProgressMax <> 0 Then
    pbMain.Value = Progress * 100 / ProgressMax
  End If
End Sub
```

You use the DownloadComplete event to inform the user that the download is complete.

The following code hides the progress bar after the download is complete by setting the **Visible** property of the progress bar to **False**.

```
Private Sub WebBrowser1_DownloadComplete()
' Hide the ProgressBar.
  pbMain.Visible = False
End Sub
```

The COM specification defines the blueprint for creating components. While the specification dictates a component's responsibilities, it does not specify how those responsibilities are implemented. Instead, the COM specification defines:

® How an object is created from a component.

® How a client accesses features of the object.

® The object's responsibility for destroying itself when it is no longer in use.

All of these activities are handled by interfaces, which make using a server much easier than before.

Prior to the development of component technologies, servers often provided services to clients through a wide array of loosely organized functions. The documentation detailed the various functions that were exported by the server, and clients then called those functions as appropriate. Learning which functions to use was often time-consuming.

This caused three major problems.

® Clients had to be specially written to take advantage of a server.

® When a server changed, clients had to be modified to accommodate the changes.

® Servers were difficult to understand because there was no logical structure to the functions they offered.

Through the use of interfaces, COM addresses all of these problems.

To gain an understanding of COM, you should be familiar with the following terms and their definitions:

® Component

A unit of executable code that provides functionality. Components are provided by servers which are either .exe, .dll, or .ocx files. Servers can be made up of one or more than one components, and components provide the templates from which objects are created. The Component Object Model (COM) specifies how components are created and how client applications connect to components. COM also processes requests from client applications to create objects.

® Object

A combination of code and data that can be treated as a unit. An object has a lifetime; that is, it is created and it is destroyed.

® Automation

Part of the COM specification that defines a standard method for creating components and using objects. Automation objects are also known as programmable objects.

To the developer of client applications, an object takes input and provides output, but its internal workings are not available to the developer. Developers only need to understand how to use the functionality offered.

This functionality is offered through one or more published interfaces. These interfaces are the means by which client applications communicate with the component.

The **IDispatch** interface exposes Automation capabilities, so any client that can use **IDispatch** can use such a component. However, **IDispatch** can require two function calls (one to **GetIDsOfNames** and one to **Invoke**) whenever a property or method is invoked (see Introduction to Binding).

The **Invoke** method must also determine which method or property is being invoked, and then unpack the parameters accordingly. This is not the most efficient method of providing Automation. Instead, it is more efficient to use a dual interface.

A dual interface is a custom interface defined by the component and derived from **IDispatch**. A dual interface contains all of the functions contained in **IDispatch**, as well as custom functions for each method and property defined for the object. A dual interface allows COM-compliant client applications the most efficient access to the methods and properties of Automation objects.

The following illustration shows what a dual interface might look like. It contains all of the **IDispatch** functions, and any custom functions for the properties and methods that the interfaces support.

{ewc mvimg, mvimage,!v06g014.bmp}

When a less sophisticated client connects to an object that supports a dual interface, it uses the standard implementation of **IDispatch** to call **Method1** and **Method2** through the **Invoke** function.

A more advanced client, such as one written in Visual Basic 5.0, can call directly to the **Method1** and **Method2** functions in the interface. This is a much more efficient form of access, known as vtable binding, which is discussed in the next topic.

By Michael C. Amundsen
Inside Visual Basic for Windows
Published by The Cobb Group

The data-bound grid control that ships with Visual Basic 4.0 has several valuable features that let you create spreadsheet-like listings of your data tables. In this article, we'll show you how to give your users the option of sorting a grid by any field in the data table. You'll find that it's simple to add this sorting mechanism to your VB 4.0 projects.

Laying out the SortGrid form

You need only two controls on a VB 4.0 form in order to create a fully functional data grid: a data control and a data-bound grid control. To begin, start a new VB 4.0 project and add a data control to the default form. Set the data control's Align property to Bottom, to automatically force the control to the bottom of the form, then stretch it to the form's full length.

Next, add a data-bound grid (DBGrid) control to the form. Set its Align property to Top and pull its lower edge down to meet the data control. Then, set the control's RecordSource property to Data1.

Figure A shows how your form should look at this point. Now you're ready to add some code.

{ewc mvimg, mvimage,!illust.bmp}
Figure A. Create this SortGrid form to hold your sorted grid.

Coding the Form_Load event

First, you need to declare a few form-level variables for this program. You'll access these variables from more than one subroutine or function. Open the general declaration area and add this code:

```
'
' form level vars
'
Dim cDatabase As String
Dim cTable As String
Dim cSelect As String
Dim cSort As String
Dim cField As String
```

In this example, you'll add most of the code to the `Form_Load` event. (In a larger project, you might want to put some of this code in separate modules to make it easier to maintain.) This code will handle four basic tasks:

1. Set the values of the form-level variables
2. Open the database and data table
3. Set up the data-grid properties
4. Set the form properties

Add the code in **Listing A** to the `Form_Load` event of the form. When you do so, be sure that you initialize the `cDatabase` variable (shown in blue) with a value that's valid for your workstation.

Listing A. The Form_Load event

```
Private Sub Form_Load()
    '
    ' set form vars
    '
    cDatabase = "i:\vb4\biblio.mdb"
    cTable = "Publishers"
    cSort = " ORDER BY "
    cField = ""
    cSelect = "SELECT * FROM "
```

```
    '
    ' set up data control
    '
    Data1.DatabaseName = cDatabase
    Data1.RecordSource = cSelect + cTable
    Data1.Refresh
    '
    ' set grid vars
    '
    DBGrid1.ColumnHeaders = True
    DBGrid1.Caption = cTable + " Table"
    DBGrid1.Refresh
    '
    ' set form vars
    Me.Caption = "DBGrid Sort Demo"
    Me.Left = (Screen.Width - Me.Width) / 2
    Me.Top = (Screen.Height - Me.Height) / 2
    '
End Sub
```

Important note about adding field names at runtime

Here's an important consideration: *do not* load the field names into the DBGrid control at design time. If you do, you won't be able to remove the field names at runtime. Instead, you add the field names at runtime by setting the ColumnHeaders property to `True` and then invoking the DBGrid `Refresh` method.

Now, save and run the project. VB will load the data and initialize the data-bound grid, and your screen should look like the one shown in **Figure B**.

{ewc mvimg, mvimage,!illust.bmp}
Figure B. When you run the example project, you'll see this window.

Adding the sort option

Now that you have the grid up and running, you're ready to add the sorting option. All you need to do is add some code to the DBGrid control's `Head_Click` event. This event fires each time the user clicks on the DBGrid header. The event returns an index value that indicates the column on which the user clicked. You can use this column number to determine the table field, then use that field name to create an SQL statement to sort the data.

The code in **Listing B** performs these tasks. You'll note that we use VB 4.0's continuation character (_) in this code.

Listing B. The HeadClick event

```
Private Sub DBGrid1_HeadClick(ByVal ColIndex As _
    Integer)
  ' sort based on column clicked by user
  '
  Dim nType As Integer
  '
  nType = Data1.Recordset.Fields(ColIndex).Type
  If nType <> dbMemo And _
    nType <> dbLongBinary Then
      cField = Data1.Recordset. _
        Fields(ColIndex).Name
      '
      Data1.RecordSource = cSelect + cTable + _
        cSort + "[" + cField + "]"
      Data1.Refresh
```

```
  Else
      MsgBox "Can't sort on Binary or Memo _
        Fields", vbExclamation, "Sort Error"
  End If
  '
End Sub
```

Remember that before you attempt to sort on the selected field, you need to ensure the selected field is *not* a Binary or Memo field. Visual Basic doesn't allow sorting on these types of fields.

Next, you'll see that the code wraps the field name in brackets (`[]`), then adds it to the sort statement. These brackets are necessary because the Microsoft JET engine allows field names that contain spaces (for instance, "Company Name"). However, SQL syntax can't handle field names with spaces. To avoid a runtime error, you must place brackets around all field names that contain spaces.

Finally, the `Form_Load` event initialized the values for the variables `cSelect`, `cTable`, and `cSort`. You're now ready to save and run the project.

Once the grid appears, click on a column name to make the program sort the data. You can click on any column you wish, and VB 4.0 will re-sort the data based on your selection, as shown in **Figure C**.

{ewc mvimg, mvimage,!illust.bmp}
Figure C. We sorted our grid data on the Address field.

Now, scroll over the last field in the grid (Comments) and click to sort the list based on that field. When you do, you'll see an error message telling you VB can't sort on memo fields.

By Michael C. Amundsen
Inside Visual Basic for Windows
Published by The Cobb Group

One of the themes that grows broader with each new version of the Windows operating system is the concept of the document-centered interface. According to the Windows way of thinking, the ideal operating system should be smart enough to figure out what kind of document you're working with and to launch the appropriate program. You shouldn't have to worry about loading a program first, then selecting a document to work with—you should be able to just select the document and go.

Clearly, this capability is a useful tool, and one that users have become more accustomed to with each Windows upgrade. In this article, we'll show you how to incorporate document-centered features in all of your Visual Basic applications. When we're through, you'll have a simple library module you can drop into any VB project.

VB and the document-centered approach

The simplest way to add document-centered features to your VB projects is to let users select items from a list, and then have VB launch the associated program for the user. You might think you could use VB's `Shell` function, as follows:

```
RetVal = Shell("C:\WINDOWS\README.DOC", 1)
```

Unfortunately, the `Shell` function won't accept a document as the run parameter—you must supply an executable filename. You could get the user's filename, parse the tail command, look it up in the WIN.INI file, and then launch the application with the document as a parameter—but that approach is too much work. As you might guess, there's an easier way.

The ShellExecute API

Windows' `ShellExecute` API does just what we need, and even provides some added options for Windows 95 and Windows NT 4.0. The basic syntax of the `ShellExecute` function is

```
ShellExecute(hwnd,lpOperation, lpFile,lpParameters,lpDirectory,nShowCmd)
```

Table A describes each parameter of this very valuable API call.

Table A. ShellExecute API parameters

Parameter	Description
hwnd	Specifies a parent window
lpOperation	Points to a null-terminated string that specifies the operation to perform; valid operation strings are "`open`", "`print`", and "`explore`" (Win 95 only)
lpFile	Points to a null-terminated string that specifies an executable file or document to open or print; Win 95 only: `lpFile` can point to a null-terminated string specifying a folder to open or explore
lpParameters	If `lpFile` specifies an executable file, `lpParameters` points to a null-terminated string that specifies parameters to be passed to the application; if `lpFile` specifies a document file, `lpParameters` should be NULL
lpDirectory	Points to a null-terminated string that specifies the default directory
nShowCmd	If `lpFile` specifies an executable file, `nShowCmd` specifies how the application should appear when it opens; this parameter can be any of these values: SW_HIDE, SW_MAXIMIZE, SW_MINIMIZE, SW_RESTORE, SW_SHOW, SW_SHOWDEFAULT, SW_SHOWMAXIMIZED, SW_SHOWMINIMIZED, SW_SHOWMINNOACTIVE, SW_SHOWNA, SW_SHOWNOACTIVATE,

SW_SHOWNORMAL

The `ShellExecute` function returns an *instance handle* —an internal value to identify the running program or process. If an error occurs, the program will return a value between 0 and 32. **Table B** shows the possible errors. Now, let's see how you can use this API call in a Visual Basic program.

Table B. Errors returned by ShellExecute

Error	Description
0	Operating system out of memory or resources
ERROR_FILE_NOT_FOUND	Specified file not found
ERROR_PATH_NOT_FOUND	Specified path not found
ERROR_BAD_FORMAT	EXE file is invalid (non-Win32 EXE or error in EXE image)
SE_ERR_ACCESSDENIED	Win 95 only: Operating system denied access to the specified file
SE_ERR_ASSOCINCOMPLETE	Filename association incomplete or invalid
SE_ERR_DDEBUSY	DDE transaction couldn't be completed because other DDE transactions were being processed
SE_ERR_DDEFAIL	DDE transaction failed
SE_ERR_DDETIMEOUT	DDE transaction couldn't be completed because the request timed out
SE_ERR_DLLNOTFOUND	Win 95 only: Specified DLL not found
SE_ERR_FNF	Win 95 only: Specified file not found
SE_ERR_NOASSOC	No application associated with the given filename extension
SE_ERR_OOM	Win 95 only: Not enough memory to complete the operation
SE_ERR_PNF	Win 95 only: Specified path not found
SE_ERR_SHARE	Sharing violation

Building the ShellDoc library module

You can create the Shell Execute Demo for VB 4.0 in three main steps. First, you need to create a module that includes the API call, all the needed constants, and some support routines. Next, you can create a simple form that will let users select documents or folders to view. Finally, you'll add a bit of code to the form.

Start a new VB 4.0 project and add a new BAS module. Set its Name property to *modShellDoc*. Now, add to the module the code shown in **Listing A**. Note that this example includes code for both 16-bit and 32-bit versions of VB.

Listing A. Declaring ShellExecute

```
Option Explicit

' ShellExecute API declaration
'
#If Win16 Then
  Private Declare Function ShellExecute Lib "shell.dll" (ByVal hWnd As Integer, _
     ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As
String, _
     ByVal lpDirectory As String, ByVal nShowCmd As Integer) As Integer
#Else
  Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
```

```
     (ByVal hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, _
     ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As
Long) As Long
#End If
```

Be careful when you type this code into your project—typing errors here can result in a locked-up PC! If you want to save time, you can use the API Viewer application that ships with VB 4.0 and load this API call from the Win32api set.

Next, you need to add the constants used by the API call to control window focus and sizing. You can also find these constants in the API Viewer database; just copy them using the Viewer or enter them from **Listing B**.

Listing B. Adding window control constants

```
' ShellExecute constants
'
Public Const SW_HIDE = 0
Public Const SW_MAXIMIZE = 3
Public Const SW_MINIMIZE = 6
Public Const SW_NORMAL = 1
Public Const SW_SHOWDEFAULT = 10
Public Const SW_SHOWMAXIMIZED = 3
Public Const SW_SHOWMINIMIZED = 2
Public Const SW_SHOWMINNOACTIVE = 7
Public Const SW_SHOWNA = 8
Public Const SW_SHOWNOACTIVATE = 4
Public Const SW_SHOWNORMAL = 1
```

The final section of initialization code is the set of error constants the `ShellExecute` function uses. You can enter these from **Listing C** or find them in the API Viewer.

Listing C. Adding error constants

```
' ShellExecute error messages
'
Public Const ERROR_BAD_FORMAT = 11&
Public Const ERROR_FILE_NOT_FOUND = 2&
Public Const ERROR_PATH_NOT_FOUND = 3&
'
Public Const SE_ERR_ACCESSDENIED = 5
Public Const SE_ERR_ASSOCINCOMPLETE = 27
Public Const SE_ERR_DDEBUSY = 30
Public Const SE_ERR_DDEFAIL = 29
Public Const SE_ERR_DDETIMEOUT = 28
Public Const SE_ERR_FNF = 2
Public Const SE_ERR_NOASSOC = 31
Public Const SE_ERR_OOM = 8
Public Const SE_ERR_PNF = 3
Public Const SE_ERR_SHARE = 26
```

You'll need to add two helper functions to this module. The first, `ShellEx`, is an API wrapper function that calls the API. The wrapper function simplifies the API call by supplying default parameters. You can also add error-handling to your wrapper function, thereby keeping your top-level code clean and easy to read.

Insert a new subroutine called *ShellEx* into the module and enter the code shown in **Listing D**. The description at the top of the routine shows the required and optional parameters. It's always a good idea to add this type of documentation to routines that you plan to use in other projects, just to help you remember what the routine does.

Listing D. The ShellEx subroutine

```
Public Sub ShellEx(lhWnd As Long, cPath As String, cDoc As String, Optional
cAction As Variant, _
   Optional cParms As Variant, Optional nShowCmd As Variant)
' ========================================
' ShellExecute API wrapper function
'
' Required Inputs:
'   lhWnd      handle to parent window
'   cPath      device & folder of doc/object
'   cDoc       complete name of object (no path)
'
' Optional Inputs:
'   cAction    action to perform:
'                  "open" (default if missing)
'                  "print"
'                  "explore" (win95/WinNT4 only)
'   cParms     parameters for cDoc (if EXE file)
'   nShowCmd   controls focus/min-max, etc
'                  SW_NORMAL is default
'                  see Const Declares for list
' ==========================================
'
Dim lRtn As Long
Dim cMsg As String
'
' check optional parms
If IsMissing(cAction) Or Trim(cAction) = "" Then
  cAction = "open"
End If
'
If IsMissing(cParms) Then
  cParms = vbNullString
End If
'
If IsMissing(nShowCmd) Then
  nShowCmd = SW_NORMAL
End If
'
' execute API call
lRtn = ShellExecute(lhWnd, cAction, cPath & cDoc, cParms, cPath, nShowCmd)
'
' check return value
If lRtn <= 32 Then
  ShellErr (lRtn) ' show error
End If
'
End Sub
```

You need to add one more helper routine to the module—the error handler. **Listing E** shows the code that will produce the proper messages, based on the error value that `ShellExecute` returns.

Listing E. The ShellErr routine

```
Public Sub ShellErr(lRtn As Long)
'
' display ShellExecute error messages
'
Dim cMsg As String
```

```
'
Select Case lRtn
  Case 0 ' memory error
     cMsg = "Memory Error"
  Case ERROR_BAD_FORMAT ' 11&
     cMsg = "Bad Executable Format"
  Case ERROR_FILE_NOT_FOUND ' 2&
     cMsg = "File not found"
  Case ERROR_PATH_NOT_FOUND ' 3&
     cMsg = "Path not found"
  Case SE_ERR_ACCESSDENIED ' 5
     cMsg = "Access Denied"
  Case SE_ERR_ASSOCINCOMPLETE ' 27
     cMsg = "Association incomplete"
  Case SE_ERR_DDEBUSY ' 30
     cMsg = "DDE Busy error"
  Case SE_ERR_DDEFAIL ' 29
     cMsg = "DDE failed"
  Case SE_ERR_DDETIMEOUT ' 28
     cMsg = "DEE time out"
  Case SE_ERR_FNF ' 2
     cMsg = "File not found"
  Case SE_ERR_NOASSOC ' 31
     cMsg = "No association for this file"
  Case SE_ERR_OOM ' 8
     cMsg = "Out of Memory"
  Case SE_ERR_PNF ' 3
     cMsg = "Path not found"
  Case SE_ERR_SHARE ' 26
     cMsg = "Sharing violation"
  Case Else
     cMsg = "Unknown Error!
End Select
    '
MsgBox cMsg, vbCritical, "Shell Execute Error [" & CStr(lRtn) & "]"
'
End Sub
```

Save this module as SHELLDOC.BAS and the project as SHELLDOC.VBP. At this point, you're ready to create a sample form to test the API call.

Creating the ShellDoc form

Figure A shows the form you'll use to test the `ShellExecute` API call. Build this form with the controls listed in Table C. Be sure to add the option buttons as a control array within the Frame control. Save the form as SHELLDOC.FRM and save the project before continuing with the last step—adding the code to the form.

{ewc mvimg, mvimage,!illust.bmp}
Figure A. Lay out the ShellDoc form in this configuration.

Table C. ShellDoc form controls

Control	Property	Setting
cmdApply	Caption	"&Apply"
cmdExit	Caption	"E&xit"
Dir1		
Drive1		

File1		
frmShellDoc	BorderStyle	3 'Fixed Dialog
	Caption	"Shell Execute Demo"
	ForeColor	&H80000008&
	MaxButton	0 'False
	MinButton	0 'False
Frame1	Caption	"Action"
optAction	Caption	"Explore"
optAction	Caption	"Print"
optAction	Caption	"Open"
	Value	-1 'True

Coding the ShellDoc form

To begin, add the following code to the form:

```
Private Sub Dir1_Change()
  File1.Path = Dir1.Path
End Sub

Private Sub Drive1_Change()
  Dir1.Path = Drive1.Drive
End Sub
```

This code links the Drive, Directory, and FileList controls so they work together. Each time the user changes the current drive letter using the Drive control, the Directory control's Path property will update to show the new path. In addition, each time the user changes the path in the Directory control, the FileList's Path property will update, forcing it to display a new file list.

At this point, add the code shown in **Listing F** to the Apply button's Click event. This code will take the selected file or directory and execute the `ShellEx` subroutine.

Listing F. The cmdApply_Click event

```
Private Sub cmdApply_Click()
' start a program with a document
Dim cPath As String
Dim cDoc As String
Dim cAction As String
'
' determine action to perform
cAction = IIf(optAction(0), "open", cAction)
cAction = IIf(optAction(1), "print", cAction)
cAction = IIf(optAction(2), "explore", cAction)
' get folder or path
cPath = Dir1.Path & "\"
' get doc (if selected)
If File1.ListIndex = -1 Then
  cDoc = ""
Else
  cDoc = File1.List(File1.ListIndex)
End If
' make call
ShellEx Me.hWnd, cPath, cDoc, cAction
End Sub
```

The routine's immediate `If` functions (`IIF`É) determine which option button the user selected. Then, the routine stores the names of the selected folder and documents in local variables. Finally, the code passes all parameters to the `ShellEx` routine.

You also need to add the following line of code behind the Exit button:

```
Private Sub cmdExit_Click()
Unload Me
End Sub
```

This code lets you safely exit the project.

The last bit of code lets users double-click on a filename to launch the document. Simply add a line that fires the `cmdApply_Click` event, as follows:

```
Private Sub File1_DblClick()
cmdApply_Click        ' fire button event
End Sub
```

That's all the coding you need to do. Save this form and the project before running the project and testing the API call.

Running the Shell Execute Demo

After you've built the example project, you can start experiencing the advantages of document-centered programming. First, select a standard text file from one of the folders on your drive and click Apply. You'll see Windows load the NOTEPAD.EXE program and display your selected file. This technique will work with any file type you've registered with the operating system.

If you're running Windows 95 or the new NT shell, you can also open or explore folders. Opening a folder gives you a slightly different view than the `explore` action.

By James C. Crabb
Inside Visual Basic for Windows
Published by The Cobb Group

*Much of the code in this article was developed by **Benjamin Bourderon** of Villeurbanne, Rhšne Alpes, France.*

The Windows 95 taskbar features a *notification area* (sometimes called the *system tray*)—a location where utilities running in the background can display an icon, giving the user access to the application without tying up a slot on the taskbar. For instance, a printer icon appears in the notification area when you send a document to print.

In this article, we'll show you how to place an icon in the notification area and respond to mouse events. Of course, before you display an icon in the notification area, you should be sure it makes sense for your application to do so. If too many applications display such icons, the notification area will become overcrowded, defeating its intended purpose.

The API

We'll start by alerting you to the bad news: The technique we'll demonstrate requires a call to the `Shell_NotifyIcon` 32-bit API. You can't call this API from the 16-bit version of Visual Basic—`Shell_NotifyIcon` works only under the Windows 95 and Windows NT 3.51 operating systems.

The good news is that the `Shell_NotifyIcon` API will place an icon in the notification area, assign tool tip text, and manage mouse events. This API accepts two parameters. The first parameter is the notification message, which tells `Shell_NotifyIcon` what function to perform. There are three valid messages:

- `NIM_ADD` (&H0)—registers an icon with the system and places it in the system tray for the first time. You should call it only once for this purpose.
- `NIM_MODIFY` (&H1)—makes changes after the icon is registered.
- `NIM_DELETE` (&H2)—removes the icon from the system tray. This message is important because icons aren't automatically removed when applications terminate.

`Shell_NotifyIcon` 's second parameter is a structure containing the information that the API needs in order to process your request. **Table A** lists these elements and their purposes.

Table A. NOTIFYICONDATA structure

Element	Purpose
hWnd	Handle of the window that receives notification messages associated with an icon in the taskbar notification area.
uID	Application-defined identifier of the taskbar icon.
uFlags	Array of flags indicating which other structure members contain valid data. This member can be a combination of the following values: `NIF_ICON` (meaning the `hIcon` member is valid); `NIF_MESSAGE` (meaning the `uCallbackMessage` is valid); and `NIF_TIP` (meaning the `szTip` member is valid).
uCallbackMessage	Application-defined message identifier. The system uses this identifier for notification messages it sends to the window (identified by `hWnd`) whenever a mouse event occurs in the bounding rectangle of the icon.
hIcon	Handle of the taskbar icon to add, modify, or delete.
szTip	Tool tip text to display for the taskbar icon.

The project

Our sample project consists of one form and one module. The form contains two picture boxes and a menu. The picture boxes' picture properties contain the filenames identifying the icon images for the notification area. One of the picture boxes will receive messages when the cursor moves across the icon in the notification area. In this

example, we'll display a pop-up menu when the user right-clicks on the icon. Let's get started with our project.

Creating the menu

Begin by opening a new VB project. Set the form's visible property to False, then choose Menu Editor from the Tools menu. In the resulting dialog box, create a Shell_Menu menu by typing *Shell_Menu* in the Caption box and *mnuShell* in the Name box. Click Next, then type *On* in the Caption box and *mnuOn* in the Name box. Click the É button to make *On* a submenu item, then click Next.

In the same way, create the submenu items Off, -, and Close, with the names *mnuOff*, *mnuSep1*, and *mnuClose*, respectively. When you finish, the Menu Editor should look like **Figure A**. Click OK to return to your form.

{ewc mvimg, mvimage,!illust.bmp}
Figure A. Create a menu that has these settings.

Adding the picture boxes

Next, place two picture boxes on the form. Assign a different icon to the picture property of each picture box—we used the TRFFC09 and TRFF14 icon files from Visual Basic's \ICONS\TRAFFIC directory. Note that the images must be icons—if they aren't, the call to `Shell_NotifyIcon` will fail. **Figure B** shows how our form looks at this point.

{ewc mvimg, mvimage,!illust.bmp}
Figure B. After we add the icons, our form looks like this.

Declaring the function

Now, open a new module. In the General Declarations window, declare the `Shell_NotifyIcon` API, `NOTIFYICONDATA` structure, and the necessary constants, as shown in **Listing A**. Next, you'll place code in Form1 at the following locations: `Form_Load`, `Picture1_MouseMove`, `mnuClose_Click`, `mnuOn_Click`, and `mnuOff_Click`.

Listing A. Module1 declarations

```
Public Declare Function Shell_NotifyIcon _
           Lib "shell32.dll" _
           Alias "Shell_NotifyIconA" _
           (ByVal dwMessage As Long, _
           lpData As NOTIFYICONDATA) As Long

   Public Type NOTIFYICONDATA
        cbSize As Long
        hWnd As Long
        uID As Long
        uFlags As Long
        uCallbackMessage As Long
        hIcon As Long
        szTip As String * 64
   End Type

   Global t As NOTIFYICONDATA

   Global Const NIM_ADD = &H0
   Global Const NIM_MODIFY = &H1
   Global Const NIM_DELETE = &H2
   Global Const NIF_MESSAGE = &H1
   Global Const NIF_ICON = &H2
   Global Const NIF_TIP = &H4
   Global Const WM_MOUSEMOVE = &H200
```

Form_Load

The `Form_Load` event initializes the variables in the `NOTIFYICONDATA` structure and adds the icon to the notification area. **Listing B** contains the code that passes Picture1's handle to the `t.hWnd` parameter and makes Picture1 the recipient of notification messages from `Shell_NotifyIcon`. You next pass `Picture.Picture` to the `t.hIcon` parameter, thereby telling `Shell_NotifyIcon` to display Picture1's picture property. You pass the literal `"Shell_NotifyIcon Test"` to `t.szTip` as the tool tip text.

Listing B. The Form_Load event

```
Private Sub Form_Load()

    t.cbSize = Len(t)
    t.hWnd = Picture1.hWnd
    t.uID = 1&
    t.uFlags = NIF_MESSAGE Or _
                    NIF_ICON Or _
                    NIF_TIP

    t.uCallbackMessage = WM_MOUSEMOVE

    t.hIcon = Picture1.Picture
    t.szTip = "Shell_NotifyIcon Test"  & Chr$(0)
    Shell_NotifyIcon NIM_ADD, t

End Sub
```

The `Picture1_Mouse_Move` event receives notifications when the mouse pointer passes over the icon in the notification area. (We describe this event in the next section.) The `MouseMove` event fires because you pass `WM_MOUSEMOVE` to the `uCallbackMessage` parameter.

Picture1_MouseMove

The following code displays Form1's menu when you right-click on the icon:

```
Private Sub Picture1_MouseMove _
            (Button As Integer, _
             Shift As Integer,  _
             X As Single, Y As Single)

    If Hex(X) = "1E3C" Then
        Form1.PopupMenu Form1.mnuShell
    End If

End Sub
```

The `X` parameter of the `Picture1_MouseMove` event contains the notification message. **Table B** lists the valid hex values for the notification message.

Table B. Shell_NotifyIcon notification messages

Hex value	Meaning
'1E00'	Mouse is moving
'1E0F'	Left button down
'1E1E'	Left button up
'1E3C'	Right button down

‘1E4B’	Right button up
‘1E2D’	Left button double-click
‘1E5A’	Right button double-click

The `X` parameter of `Picture1_MouseMove` will be `Hex’1E3C’` when you click the right mouse button. The `PopupMenu` method displays mnuShell at the cursor position.

The menu procedures

From the menu, you can change the icon and its associated tool tip text, or remove the icon altogether. To add this functionality, place the following lines in the `mnuClose_Click` event:

```
Shell_NotifyIcon NIM_DELETE, t
End
```

The `NIM_DELETE` message removes the icon from the notification area. *This is important:* Without this message, entries in the notification area will remain when an application terminates.

The `Click` event for the Off menu item toggles the checked values of mnuOff and mnuOn and uses the `NIM_MODIFY` message to change the icon and tool tip text. The `t.hIcon` parameter receives the icon image stored in Picture2’s picture value, and the event passes a new text string to `t.szTip`. Place the following code in the `mnuOff_Click` event:

```
mnuOff.Checked = True
mnuOn.Checked = False
t.hIcon = Picture2.Picture
t.szTip = "Shell_NotifyIcon is Off" & _ Chr$(0)
Shell_NotifyIcon NIM_MODIFY, t
```

The `Click` event for the On menu item works like the Off menu item’s `Click` event, passing the icon image stored in Picture1’s picture value and another text string for the tool tip. The following code goes in the `mnuOn_Click` event:

```
mnuOn.Checked = True
mnuOff.Checked = False
t.hIcon = Picture1.Picture
t.szTip = " Shell_NotifyIcon is On " & _ Chr$(0)
Shell_NotifyIcon NIM_MODIFY, t
```

Try it out

Now that you’ve entered all the code, start your project. When you do, VB will place the icon from Picture1 in the taskbar notification area, as shown in **Figure C**.

{ewc mvimg, mvimage,!illust.bmp}
Figure C. When you run your project, VB displays an icon in the taskbar notification area.

Open the icon’s menu by right-clicking on the icon. To see your icon’s tool tip, simply move your mouse pointer over the icon, as shown in **Figure D**.

{ewc mvimg, mvimage,!illust.bmp}
Figure D. A tool tip appears when you move your mouse pointer over the icon.

Finally, choose Close from the icon’s menu to close the project. When you do, VB will remove the icon from the notification area.

Notes

By checking for the notification messages shown in **Table B**, you can detect mouse events other than a right click. Some applications, such as background printing, may not require the notification feature. Changing the

values in `t.uFlags` will eliminate the tool tip and notify features.

Conclusion

The multi-threading capability of Windows 95 makes it possible to perform tasks and run applications in the background. When those background applications are running, you have the perfect opportunity to place icons in the notification area. In this article, we used the `Shell_NotifyIcon` API to post icons in the notification area and to react to mouse events.

By Stefanie R. Kushner
Inside Visual Basic for Windows
Published by The Cobb Group

Have you ever tried to perform a task in Visual Basic and realized the language doesn't support that functionality? Have you figured out a way to do something in VB, only to have it run too slowly? Or, have you needed to create a reusable piece of compiled code in VB 3.0 and run it with several different executable files?

If you answered *yes* to any of those questions, the answer to your dilemma is spelled *D-L-L*. In this article, we'll discuss the fundamentals of what DLLs are and how you can use them to enhance your VB applications.

Dynamic link libraries

One of the great things about programming in VBA (and in the Windows environment, in general) is that you can take advantage of functions written in C, almost as if those functions were native to VBA. C functions reside in *Dynamic Link Libraries* (DLLs), which are simply collections of functions available at runtime. The only tricky part about using DLLs is that you have to declare them before you use them, so the application knows where they're located, what arguments they take, and what values they return. You then call these procedures as if they were general procedures you wrote.

DLLs provide several advantages. First, they add functionality not available in the VBA language. In addition, C-compiled DLLs typically run faster than VBA- interpreted code. You'll be able to share code between applications, and, when you maintain code in a DLL, you don't need to recompile the application using the DLL.

Windows API

Windows versions previous to 3.0 stored all dynamic link libraries as EXE files. This historical legacy is why the core libraries that make up Windows 3.1 itself reside in EXE files. These core function libraries reside in three main files: GDI.EXE, KRNL386.EXE, and USER.EXE.

In Windows 95 and NT, these files have a DLL extension. The 32-bit operating system files are GDI32.DLL, KERNEL32.DLL, and USER32.DLL.

Using the functions contained within these three files is sometimes referred to as programming the Windows API. *API* stands for Application Programming Interface, and simply means that you can take advantage of the functionality external to your core application via function calls to an external function library.

The Windows API is (obviously!) provided by Microsoft along with Windows. **Table A** defines the three DLL libraries. Vendors provide other DLLs, and a wide variety are available as shareware or freeware.

Table A. Windows API

Library	Contents
GDI	Functions related to the creation of line, text, and bitmap output on different output devices
User	Functions related to Windows messaging, dialog boxes, and any interaction with controls (including menus) on a form or window
Kernel	System-related functions—that is, functions that disclose the amount of available resources or let you work with INI files or the active window—and any functions relating to memory

16- vs. 32-bit DLLs

Windows 95 and Windows NT include both the 16- and the 32-bit Windows API, which means that you can run your 16-bit applications in a 32-bit operating system. The application—not the operating system—decides which DLLs you call.

For example, from VB 3.0 or 16-bit VB 4.0, you'll make 16-bit API calls when running under Windows 3.1 or Windows 95. From 32-bit VB 4.0, you'll make 32-bit API calls. A 16-bit application can't make 32-bit DLL calls, and a 32-bit application can't make 16-bit DLL calls. These restrictions apply whether you're making calls to the API or creating your own DLL.

Declaring the functions

You need to declare all DLL functions. Once you've properly declared a function, you can call it exactly as you would a built-in VB function. In the `Declare` statement, you tell VB

® the name of the function

® which disk file contains the function

® the number of arguments the function expects and their data types

® whether the function returns a value, and if so, which data type to look for

The location of the `Declare` statement determines the scope of the procedure. For example, a procedure declared at the form level can be called only from within the form. A procedure declared in a BAS file can be called from anywhere in the application.

The general syntax for the `Declare` statement is as follows:

```
Declare Function procedurename Lib "libname" [Alias aliasname] [(argument list)]
As datatype
```

The *procedurename* component is the name of the function (or sub) contained within the library. The *libname* component is the name of the library containing the procedure. Since it's a literal string, you must enclose it in quotes. Normally, you need to include the file extension here, but you can ignore it if you're using one of the core Windows API libraries (USER, KRNL386, GDI, USER32, KERNEL32, or GDI32).

The *[Alias aliasname]* component is optional; you can use it to declare to VB a procedure name different from the procedure name contained within the library. You might use an alias if the library procedure name contains characters not allowed in a VB procedure name. Here's an example:

```
Declare Function lOpen Lib "Kernel" Alias "_lopen" ...
```

The function name in the library starts with a character that's not allowed for a VBA function, so you must use `Alias` to rename it.

Function data types

It's important to note the data type of a function's returning variable, as well as the arguments being passed. The data type of the value returned from a DLL must match the variable that will receive that value.

In C, arguments are passed By Value as the default. An asterisk (`*`) indicates that an argument should be passed By Reference. **Table B** outlines the differences between C and VBA data types.

Table B. Differences between C and VBA data types

C data type	Visual Basic data type
`*char s`	ByVal *var* As String
char c	ByVal *var* As Byte
int I	16-bit: ByVal *var* As Integer
	32-bit: ByVal *var* As Long
*int I	16-bit: *var* as Integer
	32-bit: *var* as Long
long I	ByVal *var* as Long
*long I	*var* As Long
float f	ByVal *var* as Single
*float f	*var* as Single
double d	ByVal *var* as Double

*double d	*var* as Double
NULL	ByVal As String

C versus VBA strings

For the most part, passing an argument by value or by reference is the same in VB as it is in C—except when it comes to strings. When you pass a string to a DLL, the function expects a null-terminated string. However, VB strings aren't null-terminated. To terminate a VBA string with a null character, you must pass the string with the `ByVal` statement. Even though you use this statement, a pointer to the null-terminated string passes to the C function, thereby allowing the function to change the value of the argument being passed.

Padding strings

Another problem with passing strings to C functions is that VB strings are variable-length and are initialized as empty strings. C functions can't increase or decrease the size of these strings. If you pass an empty string to a function, or pass a string that's too small, the function simply writes beyond the end of the string—probably in some other application's memory—thereby forcing a GPF!

If you pass a string to a function in a DLL, and the function will write to that variable, the string must be large enough to handle the largest possible value. You can accomplish this safeguard in two ways: by passing a fixed-length string or by padding the string with spaces or null characters.

You create a fixed-length string by placing an asterisk (`*`) and the string length after the declaration. The following line creates a 255-byte string:

```
Dim MyString as String * 255
```

Since the string is fixed-length, you can't strip off any extra characters.

To pad a string with null characters or spaces, use either the `String` or `Space` function. The following example pads a string with 255 null characters:

```
Dim MyString as String
MyString = String(255, 0)
```

To pad the string with spaces, use code like this:

```
Dim MyString as String
MyString = Space(255)
```

When you pass a padded string, you should trim the string of all unnecessary characters after returning from the function. Some functions that write values to a string return a value indicating how many characters were written out. In this case, you can use the `Left` function to extract just those characters. If a function doesn't indicate how many characters it wrote, the `Instr` function will find the null termination character written by the function and extract up to that point.

String example

The following code passes a string to the function `GetWindowsDirectory`:

```
Dim sDir as String
Dim lLen as Long

sDir = String(255, 0)
lLen = GetWindowsDirectory(sDir, Len(sDir))

sDir = Left(sDir, lLen)
Print sDir
```

This string will return from the function with the name of the Windows directory.

Differences between 16- and 32-bit

As we mentioned earlier, the decision to call a 16- or 32-bit function depends on how you compiled the VB executable: as a 16- or 32-bit EXE. The type of DLL must match the type of EXE. You should be aware of a couple of differences when you port your 16-bit code to 32-bit code, including case-sensitivity and ANSI versus Unicode. Let's take a look at these differences.

Case-sensitivity

In VB 3.0, it doesn't matter what case you use when declaring or calling a DLL function from your code. For example, the following two declarations are exactly the same in VB 3.0:

```
Declare Function GetFreeSpace Lib "Kernel" (ByVal wFlags As Integer) As Long

Declare Function GETFREESPACE Lib "Kernel" (ByVal wFlags As Integer) As Long
```

On the other hand, when you make 32-bit calls, the name of the function is case-sensitive, and you must enter it correctly in the `Declare` statement. The following two declarations are *not* the same in VB 4.0:

```
Declare Sub MessageBeep Lib "User32" (ByVal N As Long)

Declare Sub MESSAGEBEEP Lib "User32" (ByVal N As Long)
```

The first `Declare` will work, but the second won't.

The easiest way to convert your non-case-sensitive DLL calls from VB 3.0 to case-sensitive function calls in VB 4.0 is to `Alias` them. The `Alias` keyword of a `Declare` statement follows the name of the library and holds the actual case-sensitive name of the function. What follows the `Declare` sub/function statements is the name you want to use to call the function from your code. The following line `Alias`es the `MessageBeep` function:

```
Declare Sub MESSAGEBEEP Lib "User32" Alias MessageBeep" (ByVal N As Long)
```

The Win32API.TXT file `Alias`es all function calls, eliminating the case-sensitivity problem.

ANSI versus Unicode

There are two interfaces in 32-bit operating systems: the American National Standards Institute (ANSI) character set and the Unicode character set. In the ANSI character set, a single byte represents each character. In the Unicode character set, two bytes represent each character, which allows foreign languages—some of which have more than 26 letters—to include all their characters.

Windows 3.x and Windows for Workgroups use the ANSI character set. Although VB 4.0 uses Unicode internally for storing strings, the ANSI character set is the default when you make DLL calls. Therefore, you'll always use the ANSI versions of DLLs in VB 4.0.

You can tell an ANSI function call from a Unicode function by checking if the name of the function is followed by an *A* or *W*. ANSI functions add an *A* to the end of the function, while Unicode versions add a *W*. (The *W* stands for *wide*, because of the increased width of the bytes for each character.)

For instance, the following code is the ANSI version of `ThisDLLFunction`:

```
Declare Function ThisDLLFunctionA Lib ThisDLL.DLL () As Long
```

The Unicode version of ThisDLLFunction looks like this:

```
Declare Function ThisDLLFunctionW Lib ThisDLL.DLL () As Long
```

Remember, you can't call out to a Unicode function, so you'd never have this `Declare` in your project.

Putting it all together

Let's take a look at a practical example of using DLLs. A very common use for the Windows API is to figure out if

an application is already running. The FindWindow function of the User32 (or User) library takes the name of the window to find and returns its handle. If the function returns 0, that means the function couldn't find the specified window. If the function returns the handle, you can pass it to the ShowWindow function to display the application.

The first thing you need to do is copy the declares for these two functions to a BAS module. The ShowWindow function also accepts an argument to specify how the window should appear: maximized, minimized, or restored. You can copy the constant and function declarations from the Text Viewer, or type them in as follows:

```
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As
String, ByVal lpWindowName As String) As Long

Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As
Long) As Long

Public Const SW_RESTORE = 9
```

Take a look at the declare for the FindWindow function. Notice that the function is Aliased and its real name is FindWindowA. This is the ANSI version of the function, and it's Aliased just for readability's sake.

You can also tell that the function takes two strings and returns a long value. The first string is the window's class name. If you don't know the class name, you can pass a null value. The second argument is the window's caption.

The ShowWindow function takes the handle of the window to display and a value indicating how to display it. It also returns a long value.

Suppose you want to check whether the calculator is running. You can do so using these lines of code:

```
Dim lResult As Long

lResult = FindWindow(vbNullString, "calculator")
If lResult > 0 Then
  lResult = ShowWindow(lResult, SW_RESTORE)
Else
  lResult = Shell("calc", vbNormalFocus)
End If
```

If the calculator is running, you'll use the ShowWindow function to restore it; if it isn't, you'll use VB's Shell function to run it.

By Michael C. Amundsen
Inside Visual Basic for Windows
Published by The Cobb Group

One of Visual Basic 4.0's new features gives you the ability to use compiler directives to control how your VB 4.0 code compiles. By supplying the required control flags at compile time, you can write sections of code that VB will include only in the final compilation. Unlike command-line parameters, which control code that executes during runtime, compiler directives let you control which lines of code you include in the executable file you distribute to end users. In this article, we'll present four examples of how you can use compiler directives (CDs) in your VB programs.

Using compiler directives

You might choose to use CDs when you find yourself in the following problematic programming situations:

® Building programs that work on both 16-bit and 32-bit Windows platforms

® Controlling the inclusion of debug code

® Initializing variables during testing

® Handling any customer-specific code modifications

Let's look at how you can apply VB compiler directives in each of these situations.

Building multi-platform code

Sometimes, code that works on a 32-bit platform won't work on 16-bit Windows. Using CDs, you can maintain both sets of code in a single VB project to keep maintenance and support efforts to a minimum.

For example, consider the use of Windows API declarations for VB 4.0. The 32-bit API calls have different LIB sources than the old 16-bit calls, and many have different function names. Without compiler directives, you'd be stuck writing two versions of the API call. Instead, you can use CDs to place both versions in your code, and then let the compiler decide which one to use.

For example, the code in **Listing A** establishes a common link to the Windows Help system. Notice the use of the construct `#If...#Else...#End If` (all compiler directives use the # prefix).

Listing A. Using compiler directives to enclose platform-specific code

```
#If Win16 Then
  Declare Function WinHelp Lib "User" _
     (ByVal hwnd As Integer, _
     ByVal lpHelpFile As String, _
     ByVal wCommand As Integer, ByVal dwData As Any) _
     As Integer

#Else

  Declare Function WinHelp Lib "user32" Alias _
     "WinHelpA" (ByVal hwnd As Long, _
     ByVal lpHelpFile As String, _
     ByVal wCommand As Long, _
     ByVal dwData As Any) As Long
#End If
```

`Win16` is a pre-defined compiler constant within VB 4.0; there's also a `Win32` constant. You can declare your own compiler constants using the `#Const` construct, as follows:

```
#Const TestMode = 1
#Const DebugOn = 1
```

You can then test these constants in your VB 4.0 code to control how your code is compiled and how it runs.

Controlling use of debug code

Now you can add debug-related code to your project. Once all the bugs are gone, simply change the directives at compile time to ignore the debug routines when you build your final executable.

Listing B shows how you can create an error routine that uses CDs. This routine will write a bug report for later inspection, as long as you set the proper compiler directive. Simply add a command button to a new VB 4.0 project and place this code behind the button.

Listing B. Using CDs to control code execution for debugging

```
Private Sub Command1_Click()
'
' show how to use compiler directives
' during error handling
'
On Error GoTo LocalErr
'
Error 53 ' force error
Exit Sub
'
LocalErr:
Dim cMsg As String
'
cMsg = Error$
#If cdDebugYes = 1 Then
  cMsg = cMsg & Chr(13) & _
     "<Debug File: debug.log>"
  Open "debug.log" For Output As 1
  Print #1, Error$; ", command1, "; _
     Format(Now(), "general date")
  Close #1
#End If
MsgBox cMsg, vbCritical, "Error Report"
'
End Sub
```

Before you can run this code, you need to create a compiler constant. To do so, add the following line to the general declarations area of your form:

```
#Const cdDebugYes = 1
```

Now run the project. Did you see DEBUG.LOG as part of the error message? No, you didn't—even though you created the compiler constant `cdDebugYes`, you must also turn it on before you run the program.

You do this by choosing OptionsÉ from VB 4.0's Tools menu and clicking the Advanced tab. Then set the Conditional Compilation Arguments string in the resulting window.

Notice that you can place multiple CDs on the same line; simply separate them with a colon (:). Add the following line to the Conditional Compilation Arguments box:

```
cdDebugYes = 1
```

Now, when you run your project and click the command button, you'll see an error message that tells you VB has created a DEBUG.LOG file.

Initializing variables during the testing phase

Many large locations have established test areas on their networks separate from the production environment. This arrangement can cause problems when you initialize variables for the test environment. Often you use a set of unique values in the test area (directories, file names, default log-in IDs, and so on) that you won't use once

the system is in production.

You can use CDs to establish and initialize these test variables. Once you get ready to build the production version of the system, you can switch off the CDs to ensure these default values don't appear when you release the final version of your software.

Add the code in **Listing C** to your project to establish the values of internal variables. You'll notice that you can bypass an entire set of routines using CDs.

Listing C. Using CDs to set variables and control code execution during testing

```
Private Sub Form_Load()
'
' using VB 4.0 compiler directives
'
#If Win16 Then
  cWinType = "Windows 16-bit!"
#Else
  cWinType = "Windows 32-bit!"
#End If
'
#If cdTesting = 1 Then
  cDbDir = "c:\test\"
#Else
  cDbDir = "c:\prod\"
#End If
'
#If cdTesting Then
  cUserName = "TestUser"
#Else
   ' Call UserLogIn Routine
  cUserName = "UserLogIn Must be Called"
#End If

' write compiler strings to screen
'
Label1.Caption = cWinType
Label2.Caption = cDbDir
Label3.Caption = cDbFile
label4.Caption = cUserName
'
End Sub
```

Next, add the code in **Listing D** to the general declarations area of your form to establish the new compiler constants. Finally, add four Label controls to the form so you can view the results. Set the conditional compilation arguments to various values and run the project to test your CDs.

Listing D. Using CDs to create constants and form-level variables

```
Option Explicit
' set some compiler directive constants
' Win16 and Win32 are pre-defined constants!
'
#Const cdDebugYes = 1
#Const cdTesting = 1
'
Dim cDbDir As String ' database directory
Dim cDbFile As String ' database file
Dim cWinType As String ' windows type
Dim cUserName As String ' default username
```

Handling customer-specific code requirements

If you have more than one customer using your software, or if one customer has multiple sites, you probably have code you must modify to meet specific needs. Usually you can meet these needs by creating INI file or Registry values to control how a code routine works. But sometimes the variations needed by the customer require you to dramatically alter code modules.

Listing E shows how you can use CDs to handle customer-specific options. In this example, you're serving three customers with a single system. You control the details of database location by setting the compiler directives at runtime. Add this code to the general declarations and the `Form_Load` event of your project. Modify the arguments line to control how the code is compiled, then run the example to see the results.

Listing E. Using CDs to control customer-specific code features

```
' Add this to the general declarations
#Const cdStockholm = 1
#Const cdNewYork = 2
#Const cdRedmond = 3

'(Add this to the Form_Load event)
'(just before you set the label values)
#If cdStockholm Then
  cDbFile = "stock.mdb"
#ElseIf cdNewYork Then
  cDbFile = "york.mdb"
#ElseIf cdRedmond Then
  cDbFile = "gates.mdb"
#Else
  cDbFile = "local.mdb"
#End If
```

Handling long filename input

You can also use CDs to control how VB handles user input—for example, long filenames under 32-bit platforms. If you ask a user for a filename, you need to run checks for embedded spaces and maximum length under 16-bit Windows. However, under 32-bit Windows, you have a different set of rules.

Listing F shows how you can write a single code routine that automatically knows what set of rules to apply when checking filenames. Add another command button to your project and then place this code in the `Command2_Click` event.

Listing F. Using CDs to check for long filenames

```
Private Sub Command2_Click()
' if 16-bit, check for valid filename
'
Dim cFileName As String
'
cFileName = InputBox("Enter file to load:")
cMsg = ""
'
' do this only if you're running 16-bit version
#If Win16 Then
  If InStr(cFileName, " ") <> 0 Then
     cMsg = "Invalid FileName:_
        no spaces allowed"
  End If
  If Len(cFileName) > 12 Then
     cMsg = cMsg & Chr(13) & "Invalid _
```

```
        FileName - too long"
  End If
#End If
'
If Len(cMsg) = 0 Then
  cMsg = "FileName [" & cFileName & "] is valid"
End If
'
MsgBox cMsg, vbInformation, _
  "FileName Validation"
'
End Sub
```

The code in **Listing F** first asks the user for a potential filename. Then, the routine checks to see if you're running the 16-bit or 32-bit version of Visual Basic. If you're running under 16 bits, the code checks the input for spaces and total length, then returns any needed error messages. Notice that you don't have to set any compiler argument string to make this code example work, since it uses the built-in `Win16` constant.

Visual Basic Control Creation Edition, version 5.0, the newest member of the Microsoft Visual Basic programming system family, is designed specifically to be the fastest and easiest way to create ActiveX Controls. ActiveX controls are language independent software components based on open standards, for use in Internet and Client/Server applications.

Until now creating active components (ActiveX controls, Java applets or plug-ins) has been something only the technically elite could accomplish. The Control Creation Edition has changed that overnight. In much the same way the Visual Basic programming system version 1.0 opened the doors of Windows-based development to millions of developers, Visual Basic version 5.0 has opened the doors for active component development. Now the ability to create ActiveX controls is something practically any developer can do.

Visual Basic Control Creation Edition joins the other Editions of the Visual Basic programming system, Standard, Professional and Enterprise, in the lineup of standalone Visual Basic-based development tools. The additional members of the Visual Basic family are Visual Basic Scripting Edition (VBScript) and Visual Basic, for Applications Edition (VBA). Both VBScript and VBA are designed to be integrated into hosting environments, providing state-of-the-art programmability to products such as the Microsoft Internet Explorer and the Microsoft Office as well as many third party products. Together, the product line of Visual Basic-based tools delivers a scaleable development platform in many host environments that is immediately accessible to the more than 3 million developers using Visual Basic today.

{ewc mvimg, mvimage,!illust.bmp}

Visual Basic Control Creation Edition differs from its standalone siblings in three main areas:

® Distribution mechanism—Control Creation Edition will not be available in retail outlets but will be available for downloading from the Internet.

® Price—Control Creation Edition is free, aside from download costs.

® Functionality—While Control Creation Edition is used to create ActiveX Controls for use in other environments it can not be used to create a standalone application.

Aside from those differences noted above, Control Creation Edition also shares many major features with its standalone Visual Basic 5.0 siblings:

® Development Environment—Control Creation Edition features the same intelligent development and debugging environment as the other Editions of Visual Basic 5.0 and Visual Basic for Applications Edition 5.0.

® Third Party Support—Control Creation Edition's capabilities are easily extended with any of the over 2,000 commercially available ActiveX Controls.

ActiveX controls created with the Control Creation edition are full fledged ActiveX controls that are backward compatible with ActiveX control host environments such as the Visual Basic programming system version 4.0, Visual C++ development system version 4.2, Visual FoxPro and Microsoft Access 95. The latest version of Visual Basic for Applications Edition (VBA), version 5.0 also adds support for ActiveX controls. This means that you can now start to use Visual Basic 5.0-based controls in any of the host environments for Visual Basic For Applications version 5.0 including Microsoft Office 97 and upcoming releases from any of the Visual Basic For Applications version 5.0 licensees including products from Visio, Adobe, SAP, AutoDesk and many others. For more information on Visual Basic For Applications 5.0 go to http://www.microsoft.com/vba.

The Control Creation Edition creates fast, compact controls based on the technology described in the latest release of the Microsoft ActiveX Control SDK. A typical control built with the Control Creation Edition is approximately 20K bytes, and as such performs equally well in applications targeting the Internet or Client/Server architectures. In either case, ActiveX controls built with the Control Creation Edition are downloaded on demand and interpreted by the Visual Basic virtual machine on the client in the same manner as a Java applet. The Visual Basic virtual machine's presence is required to run a Control Creation Edition built ActiveX control. If an end-user that doesn't have the virtual machine encounters a Web page with a Control Creation Edition ActiveX control present, the virtual machine will be automatically downloaded for them as a one-time compressed download of approximately 700K compressed. Subsequent ActiveX controls encountered only require the downloading of the control itself.

There are three general usage scenarios for creating ActiveX controls with the Control Creation Edition:

® Creating ActiveX Controls from scratch—Everything is included in the Control Creation edition to allow the creation of complete, standalone ActiveX controls from scratch. While this is possible it is expected that the following two usage scenarios will be more common.

® Subclassing and customizing an existing ActiveX Control—Developers can take advantage of the over 2,000 commercially available ActiveX controls as a starting point, subclass and customize the control and then compile it creating a custom version of that control. An example of this might be taking a third party Tabbed Dialog control, setting the orientation to "left", the style to "3 ring binder" and the "picture" to the company logo. This can then be compiled as a new custom ActiveX control with these new properties.

® Aggregating multiple ActiveX Controls into a control "assembly"—Developers can take advantage of the large market of commercially available ActiveX controls by aggregating multiple controls together in an ActiveX control project, customizing their look and behavior, and then compiling the group of controls together into a single control. The resultant control can then be inserted in a Web page or a client/server application and can be used to "wrap" or contain the entire user-interface elements of that application. It is anticipated that this will be a popular way for developers to share interfaces and code thus gaining valuable reuse capabilities.

Based on observing what is becoming general practice in the industry, licensing implications for the use of third-party ActiveX controls in Control Creation Edition built controls are consistent with those which currently exist with third-party ActiveX control usage inside an application. ActiveX controls customarily supply two licensable elements—a design time element and a runtime element. A developer who obtains rights to use a third-party control for inclusion in an application typically purchases the full design time rights to use that control in design time in any development environment. When they have finished an application they then have the right to distribute the application .EXE along with the runtime element of the control to their end-users. End-users get the benefits of that ActiveX control inside their application, but they cannot use the control in design mode should they have a development tool installed.

The situation is the same for the Internet scenario. One may create an ActiveX control with Control Creation Edition based on a purchased third-party ActiveX control and you can then distribute that control inside a Web page. An end-user that "runs" the control inside the Web page has full runtime rights to that control inside that page, but may not use the control in design-mode without first obtaining a license to use the control in design mode. Note that this generally describes the customary legal usage of ActiveX controls available today, but some ActiveX controls do have licensing policies that vary from the above—you should always consult the license agreement provided with ActiveX controls for exact usage rights and restrictions.

The next section of this reviewer’s guide is designed to familiarize the reader with the major new features of this release of Visual Basic Control Creation Edition version 5.0. Following the New Features section is a "Test Drive" section will then lead the reader through the process of building their first ActiveX Control. You should be able to build your first control in about 10 minutes. We are sure that once you have built your first ActiveX control you will fully realize the impact that Visual Basic Control Creation Edition 5.0 is going to have on the computing world. With over 100,000 downloads in the first several weeks of availability, Control Creation Edition is going to be one of the most significant products to be released in 1996.

The following topics are included in this section:

® Environment Enhancements

® Creating Your First ActiveX Control

® FAQs

Software developers and reviewers need information that answers questions like "What's new?" or "Why Should I upgrade?" *This section is a guide to the new and enhanced programming environment in Microsoft Visual Basic 5.0, Control Creation Edition.*

Microsoft Visual Basic 5.0, Control Creation Edition is not only the fastest and most efficient way to create ActiveX Controls, it is the most widely used programming environment for Windows. Today, there are over three million developers programming using Visual Basic.

Microsoft Visual Basic 5.0, Control Creation Edition improves on the development foundation first introduced in previous versions in three key areas:

- **Familiar, Easy To Use Control Creation Paradigm.** The popularity of Visual Basic stems from its ease of use. Interfaces are "drawn or painted", and sophisticated applications are assembled from pre-built, pre-tested components. Visual Basic 5.0, Control Creation Edition applies this same assembly metaphor to the construction of ActiveX components. ActiveX controls can be created from scratch or by assembling any of the thousands of commercially available ActiveX controls. If you can create a Visual Basic application, you can create an ActiveX control.
- **Improved programming capability.** A new version of Visual Basic for Applications (Version 5.0) now in Excel, Word, PowerPoint , Access, and Project can also be found at the heart of the Visual Basic 5.0, Control Creation Edition. The Integrated Development Environment (IDE) has been fully re-architected and standardized for greater consistency across all host applications. In addition, IntelliSense technology has now been brought to the developer, making writing error-free code in the Visual Basic 5.0, Control Creation Edition easier and faster than ever.
- **New and improved debugging capability.** Visual Basic 5.0, Control Creation Edition provides many new unique debugging facilities which make the debugging of ActiveX controls as straightforward as debugging applications. With its new multi-project development environment, Visual Basic 5.0, Control Creation Edition allows both ActiveX controls, and their host, to be designed, implemented, and debugged, all within a single development session.

The following topics are included in this section:

- Integrated Development Environment
- ActiveX Control Creation Tools
- Extensibility—ActiveX Controls
- Customizable Development Environment

Visual Basic for Applications (VBA) is the edition of the Microsoft Visual Basic programming system designed specifically to provide rich development capabilities inside an application environment. Applications that integrate VBA become full fledged development platforms immediately accessible to the more than 3 million developers using Visual Basic today. The first version of VBA appeared in 1993 inside Microsoft Excel, followed by Microsoft Project (1994), and Microsoft Access (1995).

Since the release of VBA in Microsoft Excel in 1993, Microsoft has received requests from hundreds of independent software vendors (ISV's) requesting to license VBA for use in their applications. Broadly licensing VBA has always been part of the vision for VBA, but the initial implementations were difficult to integrate. With the release of the third generation of VBA, version 5.0, this vision can now become a reality.

Visual Basic for Applications; version 5.0 not only delivers an ISV-ready release in terms of interfaces and robustness, but it also delivers on important, ISV-requested features. First, VBA 5.0 now provides a complete, integrated development environment (IDE) that includes many of the elements familiar to Visual Basic developers—a Project Window, a Properties Window, debugging tools, etc. Second, previous to this release VBA did not support much of the functionality that put the word "Visual" in Visual Basic; specifically, support for Forms and ActiveX Controls (formerly named OLE Controls). VBA 5.0 delivers rich, visual features with Microsoft Forms, a cross-platform forms package, and support for the thousands of available ActiveX controls. In addition to the new IDE and Microsoft Forms and ActiveX support, there are literally hundreds of other new features that advance VBA's productivity and ease-of-use to new levels, making VBA 5.0 the bar to measure other development environments against.

VBA 5.0 will be a core component of the next version of Microsoft Office application suite (VBA 5.0 is now in Microsoft Word and the Microsoft PowerPoint presentation graphics program, in addition to Microsoft Excel and Microsoft Access in Microsoft Office 97). Through the announced licensing policy, Microsoft is also making the exact same VBA in Microsoft Office broadly available for use inside non-Microsoft applications, providing the same ease of use and power of Visual Basic to many new host environments. One point that deserves emphasis is that this is not merely the exact same language, syntax etc. that is being licensed, but ***exactly*** the same development environment, everywhere it is implemented. Immediate benefits include:

Benefits to ISVs:

® ISV's can immediately gain the 3 million developers using the Visual Basic programming system as potential customers.

® Developers build solutions on top of ISV products leading to increased sales of ISV products.

® ISV's can leverage the large Visual Basic infrastructure already in place:

- Hundreds of training facilities
- Support centers
- Hundreds of books and magazines
- Hundreds of seminars
- Events
- Tradeshows
- Thousands of ActiveX Controls etc
- Web sites.

In addition, hosting VBA 5.0 allows ISVs to concentrate on their core competency, not language development. A large marketing campaign around VBA will have a direct positive impact on their bottom line, and the increased functionality will provide competitive advantage.

Benefits to Developers:

® Each VBA-hosted application will expose an object model expanding the ActiveX-based component set available for developers to leverage.

® Developer skills become more marketable because they can work across many platforms.

® Immediate code reuse advantage—same Visual Basic everywhere.

Perhaps most dramatically, VBA 5.0 enables developers to build new solutions that previously could not be built due to cost or functionality that now is available through the integration of different applications or from the core

functionality from different vendors.

Benefits to MIS Manager:

® Reduced dependency on developers for application specific solutions.

® Reduced backlog of end user application demands through code reuse, faster response.

® Developers can be easily moved across development platforms/projects.

Visual Basic for Applications 5.0 will also play a large role in helping MIS managers and their companies lower training costs by reducing the number of development environments/languages they need to train developers on.

Benefits to users of application-based solutions

® High performance applications through in-process ActiveX-based support.

® Power users can utilize common VBA scripting/recording capability.

® Solutions look and work like the applications they already know.

® Ability to have solutions that enable user- customization, such as print options or query creation.

Overall, users will benefit the most for improved solution quality and availability as the applications they use today incorporate richer functionality enabled by VBA, and a richer set of vertical applications appears.

Microsoft began the VBA 5.0 roll-out by running a pilot program in the first half of 1996 with 8 companies. The goal of the pilot program was to learn about and address the technical issues associated with integrating VBA into third-party applications, paving the way for widespread licensing. The companies were selected because they had been persistent requesters of the technology and they understood object models. Diversity in terms of company size and markets was intentionally sought out to insure the model would work as broadly as possible. Microsoft and the following pilot program companies have announced that they have signed letters of intent to license the technology:

Autodesk	Adobe	SAP	Rockwell	Sagent
D Eye	NetManage	Antares	Attachmate	Intellution
DataMirror	Digital Impact	Interpersonal Computing	TA Engineering	
OnPoint Technologies	Siemens	Dynapro	Iconics	Intergraph
OSI Software	StarBase Corp.	Systems Modeling	Vantive Corp.	EC Company
Rational	Tan Data	McAfee	Documentum	LogicWorks
Visio	MicroGrafx	NetManage	B&C Microsystems	HAHT Software

Microsoft also announced on June 6 that it would be licensing VBA 5.0 though 2 partners—Summit Software of Syracuse, New York and Mystic River Software of Cambridge, Massachusetts. Prior to the announcement Summit and Mystic River were the two leading vendors of VBA-compatible language technology with over 200 companies licensing their combined technologies. Since the announcement the response for the ISV community has been overwhelming with hundreds of vendors expressing interest in VBA 5.0 for their products. Implementations of VBA 5.0 integration will begin to appear at the end of 1996 and continue through 1997 and beyond. Based on ISV interest, developers should expect VBA 5.0 to bring many exciting new development platforms to the millions of Visual Basic developers over the next few years.

The following topic is included in this section:

® Visual Basic for Applications, Version 5.0 Features

The COM specification requires the following functionality of all objects.

® The object must be able to keep track of the number of connections made to it. When no longer in use, it must be able to destroy itself.

® The object must be capable of being queried by a client for additional interfaces that it may support.

In order to provide this functionality, all objects have a built-in interface named **IUnknown**. The **IUnknown** interface is supported by every COM object. Further, every other interface on the object must include the functionality provided by **IUnknown**.

Functions of the IUnknown Interface

The **IUnknown** interface has three functions. **AddRef**, **Release**, and **QueryInterface**. The functions **AddRef** and **Release** keep track of the creation and destruction of the object, and **QueryInterface** lets clients query for other interfaces provided by the object.

The **AddRef** function increments the usage count of the object when it assigns an interface pointer. The **Release** function decrements the usage count when a variable that points to the object goes out of scope.

AddRef is called when you use a **Set** statement to initialize an object variable. **Release** is called when you set an object variable to **Nothing**, or when the variable goes out of scope. When the usage count falls to zero (no client has a pointer to an object interface), the object destroys itself.

For more information about using the **Set** statement, see Creating Objects.

Querying Additional Interfaces

When a client contains an object variable that points to a valid COM interface, it can query any additional interfaces provided by the object with the **QueryInterface** method. Visual Basic also provides this capability automatically through the **Set** statement.

For more information about gaining access to additional functionality by querying an interface, see Getting to Additional Interfaces.

Visual Basic for Applications is a shared component. Specifically, it is a common development environment consisting of a forms package (which is a host of ActiveX components), a code editor, and a debugger. Visual Basic 5.0, Control Creation Edition and many other Microsoft products including Microsoft Excel, Access, Word, PowerPoint, and Project all share this common development environment. Collectively, these products represent hundreds of ActiveX components that can be easily tailored to solve specific business problems.

As a result of the recent Microsoft announcement to license the Visual Basic For Applications environment to third parties the programmers reach extends even farther. Programmers can now look forward to being able to program products from Adobe, Autodesk, Visio, SAP, and many, many more.

The Visual Basic for Applications environment includes the Visual Basic for Applications language engine, a powerful editor, an Object Browser, and debugging tools.

The Integrated Development Environment (IDE) of Visual Basic for Applications is a substantially improved programming environment. The new IDE provides:

® **New IntelliSense features** provide developers instant syntax reference and object model assistance to reduce programming time and assure error free code.

® **Multi-Project Support** enables multiple ActiveX controls and their test hosts to be loaded (and debugged in a single session).

® **Enhanced Editor** with syntax checking, color coded syntax, and support for code drag and drop within and across code windows.

® **Enhanced Project Window** for navigating through the project components and managing the programmers workspace.

® **Enhanced Properties Window** for setting and viewing object properties. Properties can be viewed by categories or alphabetically.

® **New Debugging tools** to help track program execution, monitor the status of global and local variables and eliminate bugs.

® **New Form Layout Window** for visually setting the startup position of forms as well as previewing form screen locations at varying monitor resolutions.

® **Enhanced Object Browser** for browsing and searching for properties and methods across object model libraries.

{ewc mvimg, mvimage,!illust.bmp}
Figure 1. VBA 5.0 Editor

Code Window

Drag and Drop has been implemented throughout the editing environment. Developers can drag and drop code and variables:

® between code windows

® into and out of the Watch window

® into and out of the Locals window

® across projects

Project Explorer

{ewc mvimg, mvimage,!illust.bmp}
Figure 2. Project Explorer

Visual Basic 5.0 now has the ability to load several projects at once. This is extremely useful for designing and debugging reusable ActiveX controls since both the control project and the host project can be simultaneously loaded. By placing a breakpoint in the source code of the control, the programmer can debug her application, line by line starting in the host application, stepping into the source code of the control, and back out to the host again.

The Project Explorer displays project components for all loaded projects, (i.e., ActiveX controls, forms, classes, modules, resource files, etc.) associated with each currently open project, in an outline view. Each project

appears as a new root in the outline control. This enables easy switching between different documents' projects for developers working on multiple projects simultaneously. A project exists for each open document and template (Figure 2).

For easy managing of a workspace that may have several different projects open, expanding or collapsing a folder shows or hides all open components (forms, code modules, etc.) within that project.

You can expect to find the following components listed components listed with a given project:

® Forms that belong to the project.

® UserControls that belong to a project (Note: a userControl is name of the empty "form" that the programmer turns into the final ActiveX control.)

® Code and class modules that belong to the project.

® Resource files that belong to the project.

Properties Window

{ewc mvimg, mvimage,!illust.bmp}
Figure 3. Properties Window

The Visual Basic environment has a Properties Window that displays properties of userControls, forms, modules, and classes.

There are two tabs: **Alphabetic** and **Categorized**. The **Alphabetic** tab view provides an alphabetical list of properties. In the **Categorized** tab view, properties are grouped by category, e.g., all properties related to color, or font, or position. These categories can be expanded or collapsed in the Properties window. ActiveX controls created in the Visual Basic 5.0, Control Creation Edition can specify the desired category for any custom properties it chooses to expose.

Like most of the operations in the Visual Basic For Applications editor, component type information is used extensively to the programmers advantage. Since ActiveX components are based on COM, they all contain type information that describes their interfaces. In other words, any development tool can easily determine all exposed objects, properties, methods, and even version numbers of ActiveX components. Whenever a property is selected in the property browser, descriptive functional information is displayed at the bottom of the window. (Figure 3)

Debugging Tools

{ewc mvimg, mvimage,!illust.bmp}
Figure 4. Debug Menu

VBA includes new and enhanced debugging tools to help the developer identify compile errors, program logic errors and run-time errors. The debugging tools in VBA include a Locals Window, Watch window, and the Immediate Window. The Locals window also includes a Call Stack Browser which shows the current variable and enables the developer to jump to procedure definitions and references.

Local variables window automatically displays all of the declared variables in the current procedure and their values.

Watch window enables monitoring the value of a particular variable or expression. Code execution may be interrupted when a watch expression's value changes or equals a specified value.

Immediate window that instantly evaluates any Visual Basic expression or statement, such as call to a Sub or Function.

Call Stack displays a list of currently active procedure calls during break mode.

To speed the debugging process, code can be dragged and dropped from the editor into the immediate and locals windows.

Object Browser

{ewc mvimg, mvimage,!illust.bmp}

Figure 5. Object Browser

The VBA environment has an improved ActiveX Object Browser (Figure 5). The browser differentiates between built-in properties, custom properties, methods, event handlers and user-defined procedures as well as indicates globally accessible members. The browser also shows function return types, parameter names and types, and user defined types and constants. Hyperlink jumps to referenced objects enable easy navigation of the object hierarchy. New to the Object Browser is the ability to search for objects and members across type libraries.

IntelliSense Features

Visual Basic for Applications, Version 5.0 brings Office *IntelliSense* technology to the developer, providing on-the-fly syntax and programming assistance and reference. The developer can choose to turn these automated features off and access them on demand through the VBA menu or with keystroke combinations. The following features are available while in the Code window, as well as from the Immediate Window:

® **Complete Word.** Completes the word that is being typing once enough letters are entered to make it distinct. Keystroke equivalent: Ctrl+Alt+A. For example, in the code window, type msg followed by Ctrl+Alt+A to complete the word MsgBox.

® **Quick Info.** When a procedure or method name is entered (followed by a space or an opening parenthesis), a tip automatically appears underneath the line of code writing. The tip gives syntax information about the procedure. Keystroke equivalent: Ctrl+I . This tip works for built-in language constructs as well as any user defined functions.

{ewc mvimg, mvimage,!illust.bmp}
Figure 6. Quick Info

® **List Properties/Methods.** Displays a popup listing the properties and methods available for the object that precedes the period. Keystroke equivalent: Ctrl+J. Note: This works for any built-in or, external, or user created ActiveX component.

{ewc mvimg, mvimage,!illust.bmp}
Figure 7. List Properties And Methods

® **List Constants.** Displays a popup listing the constants that are valid choices for the property typed and that precede the equals sign (=). Keystroke equivalent: Ctrl+Shift+J. Note: This works for any built-in or, external, or user created ActiveX component.

{ewc mvimg, mvimage,!illust.bmp}

Figure 8. List Constants

® **Data Tips.** When VBA is in break mode and the cursor is placed over a variable, the value of the variable is displayed in a tooltip-like window.

{ewc mvimg, mvimage,!illust.bmp}

Figure 9. Data Tips

Note expressions can be partially selected when in break mode for immediate evaluation by a datatip.

{ewc mvimg, mvimage,!illust.bmp}

Figure 10. Data tips for partially selected code fragments

® **Margin Indicators.** Developers can set a breakpoint, set the next statement, or set a bookmark by clicking in the margin of the code editor

{ewc mvimg, mvimage,!illust.bmp}

Figure 11. Margin Indicators

Block Comment and Uncomment

An Edit toolbar provides quick access to this feature which enables developers to select blocks of code and, with the click of a button, comment out the entire selected block of code.

Before the Visual Basic 5.0, Control Creation Edition, ActiveX components could only be created with the C/C++ language. The Visual Basic 5.0, Control Creation Edition represents a powerful new approach to creating ActiveX controls. Using the same visual metaphor for building controls as it does applications, the Visual Basic 5.0, Control Creation Edition allows controls to be created entirely from scratch, by modifying existing controls (subclassing), or by assembling multiple controls.

The ability to create new controls from existing controls gives Visual Basic programmers an astonishing head start when developing new, specialized components. Rather than starting from scratch, programmers can use Visual Basic 5.0, Control Creation Edition to customize any of the 2000 or more commercially available ActiveX controls.

ActiveX controls can be hosted in a wide variety of development tools. In Microsoft Office 97, ActiveX controls can be placed on Microsoft Forms or directly on Microsoft Office documents. Using the ActiveX Control Pad, Internet Studio, or Frontpage, ActiveX controls can also be placed on web pages. When ActiveX controls are placed on web pages, they behave like Java applets. Like Java applets, if the user surfing the web page does not have the ActiveX control already on their machine, it is automatically downloaded. Likewise, if the user has an outdated control, the newer control will get automatically downloaded.

ActiveX Control Features

When constructing an ActiveX control with the Visual Basic 5.0, Control Creation Edition there are many new features upon which the developer may draw. These new features are designed for ease of development, high performance and functionality.

The following list contains some of the new features that ActiveX control developers may take advantage of:

Owner drawn	The control itself handles all drawing operations which allows the developer full freedom with respect to what the control graphically displays
Subclassing (Controls Built From Existing Controls)	ActiveX controls can draw on the capabilities of another control but alter those capabilities as required
Composite Controls	Composite controls are created by combining existing ActiveX controls into a new control
Data awareness	ActiveX controls can be made "data aware", enabling them to easily link to a database
Container controls	Container controls allow developers to create controls that can house other controls. For example, a Tab control is a container control
Events	Events can be added by the controls creator to create custom events for the new control
Container specific controls	Controls can be created that determine what application is hosting them at run-time and act accordingly
Enumerations	Enumerations provide support for populating the Quick Info feature of Visual Basic
Default Properties	Default properties are used when a control is referred to without specifying a particular property
Property Pages	Property Pages are useful for design time configuration of control
Multiple OCX’s per OCX file	OCX files can contain multiple OCX controls
Design-time vs. Runtime behavior	This allows control authors to differentiate between design mode and run mode, allowing them to tailor behavior to both design time and run time
Digital signing	Controls can be digitally signed which helps

	protect from end users from viruses and other malicious mischief
Asynchronous Property Reading	Controls, especially Internet controls, can download their property settings from a URL (in the background) while the control remains responsive to the end user
Visible/Invisible at run-time	properties allow control authors to decide whether the control will be visible at run time. The Timer control is invisible for example.
Licensing	provides a mechanism for who can use controls at design time
Menus	controls can have standard menus. For example, a user interface form may be packaged as a control, complete with menus.
Object models	can be used to provide a sophisticated hierarchical object model within a control
Transparent backgrounds	lets the programmer create non-rectangular controls
Builder support	provides controls the ability to determine when they are in design mode and walk the developer through the initial setup of the control. For example, an ActiveX control may contain a Wizard that assists the programmer in setting the desired properties

ActiveX Control Wizards

Visual Basic 5.0, Control Creation Edition comes with two invaluable Wizards that greatly assist the programmer with the creation of ActiveX controls. The *ActiveX Control Interface Wizard* allows the programmer to easily define the properties, methods, and events for a user created ActiveX control.

{ewc mvimg, mvimage,!illust.bmp}
Figure 12. The ActiveX Control Interface Wizard

{ewc mvimg, mvimage,!illust.bmp}
Figure 13. The ActiveX Control Interface Wizard

The *Property Page Wizard* automatically creates custom property pages for a user defined control. By creating custom property pages, the author of an ActiveX control can create custom interfaces for the setting of properties.

{ewc mvimg, mvimage,!illust.bmp}
Figure 14. The Property Page Wizard

Below is an example of a custom property page for a *Calendar* control created with Visual Basic 5.0, Control Creation Edition. Regardless of the development tool hosting this control (Visual Basic 4.0, Visual Basic 5.0, Visual C++, Office 97, ActiveX Control Pad, etc.), right clicking on the control at design time will always display the custom property page. Custom property pages not only make ActiveX controls easier to reuse across different development tools, they also provide a consistent, intuitive interface for the control. Note that the custom property page can also contain a "preview" of how the control will look after the changes are applied. This is illustrated in Figure 15 below.

{ewc mvimg, mvimage,!illust.bmp}
Figure 15. An example of a custom property page

Setup Wizard

ActiveX controls, like Java applets, are designed to automatically download themselves to a users machine on demand. The process of compressing a control and any other files it may depend on, creating a CAB (cabinet)

file, and a working example of an HTML page is entirely handled by the Setup Wizard.

{ewc mvimg, mvimage,!illust.bmp}
Figure 16. The Setup Wizard

Visual Basic 5.0, along with all of the Microsoft Office ’97 applications support ActiveX Controls, formerly called OLE controls or custom controls. ActiveX controls are pre-built, reusable software components. Any of these controls can be used as is, off the shelf, or can be customized using the Visual Basic 5.0, Control Creation Edition.

Today, there are over 2000 commercially available ActiveX controls in almost every conceivable category including data access, 3D modeling, multimedia, imaging, manufacturing, real time data acquisition, charting, reporting, data encryption, data compression, mainframe connectivity, interface design, telephony, fax, voice, instrumentation, and many more.

Toolbox

{ewc mvimg, mvimage,!illust.bmp}
Figure 17. Toolbox

The toolbox displays all ActiveX controls that are registered for the current development session. By right clicking on the toolbox, developers can fully customize and organize their controls by whatever category they desire as well as add and remove controls.

- **Adding And Removing Controls.** By right clicking on the Toolbox and selecting the Components command, developers can easily add new controls to the Toolbox (see Figure 18 below).
- **Adding and Removing Tabs.** Developers can customize and organize their ActiveX controls by adding “tabs” to the Toolbox. A developer may choose to organize controls by project, by functionality, or any other way that makes sense (charting, Internet, etc.).

{ewc mvimg, mvimage,!illust.bmp}
Figure 18. Adding ActiveX Controls To The Development Environment

The Visual Basic For Applications environment now includes Command Bars, a new type of menu/toolbar system. This new system offers dockable/undockable and floating toolbars that are fully customizable. Developers can modify any built-in menu bar or toolbar and can create and modify custom toolbars, menu bars, and shortcut menus. Because toolbars and menus are included in Command Bars, developers use the same kind of controls on each one. For example, the docked toolbar shown in the following illustration contains three buttons.

{ewc mvimg, mvimage,!757_18a.bmp}

The menu shown in the following illustration contains the same three commands displayed as menu items.

{ewc mvimg, mvimage,!757_18b.bmp}

In Visual Basic For Applications , menu bars and toolbars can contain menus. The floating toolbar shown in the following illustration contains three buttons and a menu with the same three buttons displayed as menu items.

{ewc mvimg, mvimage,!757_18c.bmp}

The Visual Basic 5.0 Control Creation Edition makes creating ActiveX controls as easy as creating typical Visual Basic applications. If you have never seen just how easy it can be, you should definitely read on.

This document is designed to give you a fast track overview of the simple process involved in creating ActiveX controls with Visual Basic. If you are familiar with any of the previous versions of Visual Basic you are only about 10 minutes away from creating your first ActiveX control!

If you follow the steps below, when you reach the end you will have built what is commonly called a "spinner" control. A spinner control is a graphical ActiveX control that allows the user to increment or decrement a value using a mouse instead of using a keyboard. Here is a picture of one:

{ewc mvimg, mvimage,!764_01.bmp}

Figure 1.

Now that you have an idea of what you will be building, lets get started.

® Step 1: Create a Test Container

® Step 2: Add a Blank ActiveX Control Project

® Step 3: Draw the Visual Interface for the Control

® Step 4: Write Event Driven Code

® Step 5: Use and Test the Control

® Where To Go From Here?

In this exercise, you will use the **WebBrowser** control to create a browser that is capable of viewing files, HTML pages and ActiveX documents.

You begin by starting a new project, and then creating a toolbar for the browser. You will also create a status bar and add Web browser navigation capabilities. Finally, you will implement a progress bar within the status bar.

Creating the User Interface

In this first procedure, you will create the initial project and set up the user interface components, as shown in the following illustration.

{ewc mvimg, mvimage,!v11g045.bmp}

u Create the initial project

1. Create a new Standard EXE.
2. In the main form, add **Toolbar**, **Statusbar**, and **ProgressBar** controls. These controls are all located in the Microsoft Windows Common Controls 5.0 component.
3. For the **ProgressBar** control, set the **Visible** property to **False**.
4. Add a **WebBrowser** control to the main form. This control is located in the Microsoft Internet Controls component.
5. Save the project in the folder <*install folder*>\Lab11.

u Design the toolbar

1. Add a drop-down combo box to the toolbar that enables users to type in new URLs and store past URLs. Add an appropriate label for the combo box.
2. To the right of the combo box, add five command buttons to the toolbar. Create the buttons as a control array, providing each one with the same name and incrementing the **Index** property.
   {ewc mvimg, mvimage,!tip.bmp}

   Initialize the controls based on the information in the following table.

Name	Picture	Purpose
cmdStop	Trffc14.ico	Stop an asynchronous Web Browser operation.
cmdGoBack	Trffc04.ico	Go to the prior page as displayed in the Web Browser
cmdGoForward	Trffc02.ico	Go to the next page as displayed in the Web Browser
cmdHome	House.ico	Go to the home page, as specified in the registry.
cmdSearch	Binoculr.ico	Go to the Web search page, as specified in the Registry.

u Design the status bar

1. Set the **AutoSize** property of the first panel to **sbrContent**. This will create the size of the panel based on the size of the text it contains, but with the minimum width specified by the **MinimumWidth** property. This panel will be the location of the **ProgressBar** control when it is displayed.
2. Insert a second panel to display the title of the current page displayed in the Web Browser. Set its **AutoSize** property to **sbrSpring** so that its size is based on the width of the status bar.

Coding the Application

In the next procedure, you will add the underlying code for the application. You will also learn how to use a status bar panel to display a progress bar, and how to bring other application windows into the foreground.

Note The following instructions do not specify the addition of error handlers. However, many of the operations performed by this application may fail, such has navigating to an invalid URL. It is strongly suggested that you provide error handling throughout this application.

---

u Initialize the WebBrowser control

1. Implement the Start Page code.
   a. Declare a form-level string variable to hold the name of the Start Page.
   b. In the Form_Load event, use the **GetSetting** statement to extract the default Start Page from the registry. Use the parameter values listed in the following table.

Parameter	Value
AppName	IAwareApp
Section	Startup
Key	StartPage
Default	HomePage.HTML in the application's directory (see App.Path).

   c. Use the **Navigate** method of the **WebBrowser** control to display the Start Page.
   d. In the Form_Unload event, use the **SaveSetting** statement to save the name of the Start Page to the registry.
2. Size the **WebBrowser** control based on the size of the form.
   a. Add a handler for the Form_Resize event.
   b. Size the **WebBrowser** control so that it fills the display area of the form, between the **Toolbar** control and the **StatusBar** control.
   {ewc mvimg, mvimage,!tip.bmp}

u Add WebBrowser navigation

1. Implement the combo box for the URL.
   a. Add a handler for the combo box KeyPress event.
   b. If the KeyPress value is a carriage return (ASCII value 13) and the text in the Edit field of the combo box is not a null string, use the **Navigate** method of the **WebBrowser** control to display the URL specified, and use the **AddItem** method to add the URL to the combo box list.
   c. Add a handler for the combo box Click event.
   d. In the handler, navigate the **WebBrowser** control to the URL selected in the combo box.
2. Implement the navigation command buttons.
   a. In the Click event of the **Stop** button on the toolbar, stop the current operation and return to the prior URL if the Web Browser is busy (refer to the **Busy** property).
   b. In the Click event of the **Back** button, invoke the **GoBack** method of the Web Browser. Repeat this step for the **Forward**, **Home**, and **Search** buttons by using their associated methods.

u Implement the ProgressBar control

1. Make the **Statusbar** control the parent of the **ProgressBar** control. This will enable the **ProgressBar** control to be displayed on top of the **StatusBar** control, and to be located based on the **StatusBar** control coordinates.
   a. Add a standard module to the project.
   b. In the standard module, add a declaration statement for the **SetParent** API. Because this is a Win32 function, you can use the API Text Viewer that ships with Visual Basic to add the correct **Declare** statement.
   c. In the Form_Load event, set the parent of the **ProgressBar** control window to the **StatusBar** control window by using the **SetParent** API.
2. Display and update the **ProgressBar** control during a download operation.

a. Add a handler for the DownloadBegin event of the **WebBrowser** control.

b. Initialize the **ProgressBar** control to have a minimum value of 0, a maximum value of 100, and a starting value of 0.

c. Move the **ProgressBar** control to fill the first panel of the **StatusBar** control, and set its **Visible** property to **True**.

3. Hide the **ProgressBar** control when downloading is complete.

a. Add a handler for the DownloadComplete event of the **WebBrowser** control.

b. Set the **Visible** property of the **ProgressBar** control to **False**.

4. Update the **ProgressBar** control during the download operation.

a. Add a handler for the ProgressChange event of the **WebBrowser** control.

b. If the **Progress** parameter is not –1, and the **ProgressMax** parameter is not 0, set the **Value** property of the **ProgressBar** control to percentage that the operation is complete.
{ewc mvimg, mvimage,!tip.bmp}

u Use InternetExplorer to extend context-sensitive Help

1. Add a menu commands to the form, as listed in the following table.

File	Options	Help
Open	Set Start Page	MS on the Web >
(separator)		Home Page
Exit		Developer Only
		Visual Basic
		About

2. Enable Internet Explorer to display the **MS on the Web** sites.

a. Add a private form-level object variable of type **InternetExplorer**.

b. Add a handler for the **MS on the Web** menu commands.

c. In the handler, check to see if an instance of the **InternetExplorer** control is running. If not, create a new instance, as shown in the following code:

```
If ie is Nothing then
    Set ie = New InternetExplorer
End If
```

d. Use the **Navigate** method of **InternetExplorer** control to display an appropriate Web site based on the following table.

Site	URL
Home Page	http://www.microsoft.com
Developers Only	http://www.microsoft.com/devonly
Visual Basic	http://www.microsoft.com/VBasic

e. Set the **Visible** property of the InternetExplorer control to **True**.

3. Make the **InternetExplorer** control the topmost application.

a. In the standard module, add a declaration for the Win32 API **SetForegroundWindow** function.

b. After the statement that makes the **InternetExplorer** control visible, add a call to **SetForegroundWindow**.

u Finish coding the application

1. Implement the **Set Start Page** menu command to enable users to specify a different Start Page.
2. Implement the **Open** menu command that displays a dialog box to let users specify and navigate to a URL.
3. Implement the **Exit** menu command by unloading the form.

{ewc mvimg, mvimage,!tip.bmp}

In this exercise, you will build a reusable component that provides FTP downloading services for a client.

You will create a class module that interacts with clients, and a form that provides users with a visual indication of the download progress and the ability to cancel anytime before completion.

You will also use the **Internet Transfer** control to work with FTP, and generate events from forms.

The following illustration shows the architecture of the completed component.

{ewc mvimg, mvimage,!v11g040.bmp}

u Create project

1. Create a new ActiveX DLL project.
   a. Set the project name to FTPDownload.
   b. Set the class name to clsDownload.
   c. Set the instance of the class to MultiUse.
2. Add the form.
   a. Add a new form to the project.
   b. Set the name of the form to frmFTPCopy.
   c. Set the **Caption** property to FTP Download.

u Implement the frmFTPCopy form

1. Add the user interface.
   a. Add a label that displays the source and destination.
   b. Add a **Cancel** command button.
   c. Add an **Internet Transfer** control to the form (the location is not important).
2. Declare events for the form.
   a. Declare a CopyCancelled event.
   b. Declare a CopyCompleted event.
   c. Declare a CopyError event with two parameters, an integer for an error code and a string for an error description.
3. Implement a public entry point.
   a. Create a public function named **BeginCopy**. This function should take three parameters: the URL of the FTP service, and the file names of the source and the destination.
   b. Make the form visible.
   c. Use the **Execute** method of the **Internet Transfer** control to begin an FTP **Get** operation, specifying the source and destination.

      The following code shows the form of the statement:

      ```
      sOperation = "" & " Get " & Source & " " & Destination & ""
      INet1.Execute URL, sOperation
      ```

   d. Following the **Execute** call, enter a loop that calls **DoEvents** until the **StillExecuting** property of the **Internet Transfer** control returns a value of **False**.
4. Implement the StateChanged event handler.
   a. Add the StateChanged event handler
   b. If the state the state of the copy is **icResponseCompleted**, unload the form and raise the CopyCompleted event.
   c. If the state is **icError**, unload the form and raise the CopyError event. Pass the **ResponseCode** and **ResponseInfo** property values of the **Internet Transfer** control as the error code and error description parameters, respectively.
5. Implement the **Cancel** button.

a. Add a handler for the Click event of the **Cancel** command button.

b. Call the **Cancel** method for the **Internet Transfer** control.

c. Unload the form.

d. Raise the CopyCancelled event.

u Implement the clsDownload class module

1. Create an instance of the frmFTPCopy form.

   a. Declare an object variable of type frmFTPCopy that allows the **clsDownload** class to receive events. (Refer to the **WithEvents** declarator.)

   b. In the Initialize event handler for the class, create a new instance of the frmFTPCopy form.

   c. Declare the three public events CopyCompleted, CopyCancelled, and CopyError to match those of the frmFTPCopy form, using copy and paste.

   d. Implement a handler for each of the form events, and raise the associated class event back to the client, as shown in the following code:

```
Private Sub frmFTPCopy_CopyCompleted()
    RaiseEvent CopyCompleted
End Sub
```

2. Add the client interface.

   a. Add a public function named Copy, which takes three string parameters (URL, Source, and Destination) and returns a **Boolean** value that indicates success or failure.

   b. Validate that the URL begins with the prefix FTP://. If not, fix the URL appropriately.

   c. Call the BeginCopy function in the form, passing the appropriate parameters.

Build and Test the Component

In the next procedure, you will build the FTPDownload component, and update the client so that it can use the FTP service.

There are also two optional activities that you can do: You can add a function that validates the destination folder, and you can provide users with a progress indicator during the download.

u Build the FTPDownload.dll component

1. Create the DLL component by clicking the **Make DLL** command on the **File** menu.
2. Run the component so that it can be debugged. There is no visible representation of the component until it is called by the client.

u Update the client to use the FTPDownload service

1. Provide the download user interface.

   a. Load the client application created in the first exercise into a separate instance of Visual Basic.

   b. Add a reference to the FTPDownload component. Be sure that the reference is to the running instance of the component (the .vbp file), rather than to the DLL.

   c. Add a new form to the project named frmDownload.

   d. In the form, add labels and text boxes for users to provide the URL, and the file names for the source and destination.

   e. Add the **Copy** and **Cancel** command buttons.

2. Implement the client's FTPDownload capability.

   a. In the code of the new form, declare a form-level object variable of type **FTPDownload** that can receive the component's events.

   b. In the form's Load event, create an instance of the FTPDownload component.

   c. Implement a handler for the **Copy** command button's Click event that invokes the **Copy** method of the **FTPDownload** object.

d. Implement a handler for the **Cancel** command button's Click event that causes the form to be unloaded.

e. Add a subroutine, **EnableCopyButton**, that enables the **Copy** command button only if all three text boxes contain user-entered text.

f. Add a handler for the Change event of the URL text box that calls the **EnableCopyButton** subroutine. Repeat this step for the source and destination text boxes.

g. Add an event procedure for the component's CopyCompleted event that displays a message box informing users about the outcome of the FTP download operation. Repeat this step for the CopyCancelled and CopyError events.

h. On the **File** menu of the main form, add a **Transfer...** command.

i. Add a handler for the **Transfer** menu item that shows the FTPDownload form.

3. Build and test the FTPDownload capability.

Extending the Component (Optional)

This is the first of two optional activities that have you add functionality to your component. First, add a function that validates the destination folder. Second, add a progress indicator to the service.

u Add a function that validates the destination directory

Two attributes of the destination should be addressed. In this section, you will ensure that the destination directory exists and prevent copying over a destination file should it already exist.

1. Implement the destination validation function in the clsDownload class module.
   a. Add the private function **ValidDestinationPath** that takes a destination string as a parameter and returns **True**, if the destination is a valid path, or **False** if it is not valid.
   b. Extract the path of the destination string.
   c. Using the **Dir$** function, test whether or not the path is valid, and return the appropriate value.
2. Implement the **ReplaceFile** function. In this function, test for the existence of the destination file. If it already exists, display a message to users that asks whether or not they want to replace the file. If they answer Yes, rename the exiting file with a .bak extension.
3. In the **Copy** function, invoke the **ReplaceFile** and **ValidDestinationPath** functions before calling the **BeginCopy** function of the frmFTPCopy form.
4. Update the CopyCancelled and CopyError events to restore the .bak file to its original name, if one was created.

u Provide user feedback during download

One problem with the download service is that users get no feedback that the operation is still underway. In this section, you will implement the flying icon similar to that displayed when copying files with Windows Explorer.

1. Install the GlobalTimer component located in the folder\Labs\Lab11\FTP\GTimer.
2. Update the user interface.
   a. Scale the frmFTPCopy form of the component to a width and height of 100 units.
   b. Place the icon Files02b.ico in an image control at position (10, 50).
   c. Place the icon Openfold.ico in an image control at position (80, 20).
   d. Place the icon Drag1pg.ico in an image control anywhere on the form.
3. Implement the flying icon.
   a. Although the **Internet Transfer** control provides notification on the download process through the State_Changed event, it is not consistent enough to display the flying icon. For this reason, use the Timer component included with the *Mastering Microsoft Visual Basic 5* CD-ROM, and add a reference to the GlobalTimer component.
   b. Declare a form-level object variable of type **CTimer** that allows receipt of object events.
   c. Declare a form-level variable, FlyingImageX, of type **Single** that holds the current X position of the flying icon.
   d. Declare the form-level constants listed in the following table.

Name	Type	Value
interval	**Single**	5
FlyingImageStart	**Single**	20
FlyingImageEnd	**Single**	80

e. In the Form_ Load event, create an instance of the **Timer** object.

f. Add a private function, **FlyingImageY**, that does not take any parameters and returns the type **Single**.

g. In the handler, calculate the current Y position of the flying icon and increment the current value of the X position, as shown in the following code:

```
Private Function FlyingImageY() As Single
    FlyingImageY = 1 / 36 * (FlyingImageX - 50) * _
                   (FlyingImageX - 50) + 20
    FlyingImageX = FlyingImageX + interval
    If FlyingImageX > FlyingImageEnd Then
        FlyingImageX = FlyingImageStart
    End If
End Sub
```

h. Add a private **Sub** routine named FlyImage that calls the **FlyingImageY** function, and moves the flying image control based on the current values of the X and Y positions.

i. Add a handler for the TimerEvent event to the **Timer** control that calls FlyImage.

j. In the StateChanged event, if the state is icConnected, initialize the FlyingImageX variable to the FlyingImageStart constant, and call the flying image (FlyImage) once. Follow this with a call to start the timer with an interval of 200.

k. If the event is icResponseCompleted or icError, stop the timer.

l. In the Click event of the **Cancel** command button, stop the timer.

4. Recompile and test the component.

In this exercise, you will extend the FTPDownload component to provide users with the ability to view the files on a remote computer, and selecting a file for downloading.

You will create the component so that it lists only files in the main FTP folder of the server. Optionally, you can modify the component to provide the ability to navigate through the folder structure.

u Create the user interface

1. Add a second form to the component.
2. In the form, add a list box to display the list of available files, and the command buttons **OK** and **Cancel**.
3. Place a copy of the Internet Transfer Control anywhere on the form.

u Implement the frmFTPGetFileName form

1. Declare three events: FileGetCompleted, FileGetCancelled, and FileGetError.
2. Add the public entry subroutine, BeginGetFile, which takes a URL string as a parameter. In this routine, execute an FTP Directory operation by using the URL parameter.
3. Add a handler for the StateChanged event. If the event is icResponseCompleted, use the file names in buffered data (see **GetChunk**) to fill the list box. Names are separated by carriage return/linefeed pairs and folders are signified by a forward slash (/) in the right-most position of the name. If the event is icError, raise the error back to the form's client, and unload the form.
4. Add a handler for the **OK** command button. The handler should raise the appropriate event, and pass the selected file name back to the client, and unload the form.
5. Add a handler for the **Cancel** command button that cancels any executing FTP operations, raises the appropriate event, and unloads the form.
6. Add a handler to resize the list box as appropriate based on the size of the form.

u Update the class module

1. Declare a private object variable to hold a pointer to an instance of the form frmFTPGetFileName. Be sure that it allows you to receive events.
2. Declare three events to pass to the component's clients, one for each of the events to be received from the frmFTPGetFileName form.
3. Add a public subroutine, **GetFileName**, which receives the URL as a parameter from the client.
4. Create an instance of the frmFTPGetFileName form, if one does not exist. Ensure that the URL has the prefix FTP://, and invoke the BeginGetFile routine in the form.
4. Add a handler for each of the events raised by the frmFTPGetFileName form, and pass the associated class event to the client.

u Modify the client to use the revised component

The final step is to modify the client application to take advantage of the new component functionality.

1. Update the FTP Download form to include a command button labeled **Get File**.
2. Add code to enable the **Get File** command button, only if the URL text box contains user-entered text.
3. In the handler for the Click event of the **New** button, invoke the **GetFileName** method of the **FTPDownload** object.
4. Add handlers for the new component events. If the event is GetFileCompleted (or the equivalent), fill the **Source** text box with the selected file name.
5. Build and test the component.

Start the Control Creation Edition, highlight Standard EXE, and click Open as shown below. This creates the host application. This host will be used as the test container for the spinner control that is about to be created.

{ewc mvimg, mvimage,!illust.bmp}
Figure 2.

From the File Menu, select the Add Project Command.

{ewc mvimg, mvimage,!illust.bmp}
Figure 3.

In the Add Project dialog, highlight ActiveX Control and Click Open.

{ewc mvimg, mvimage,!illust.bmp}
Figure 4.

At this point there should be two projects open. As you can see in the following diagram, both projects look extremely similar. Also note that there is a new control that is now visible in the toolbox (highlighted in Figure 5 with the number 1). If you hover your mouse over this control in the Toolbox, the Tooltip should popup and display the current name of the control, "UserControl1". However, the icon in the Toolbox will be gray (disabled) at this point. It will become enabled a few steps from now!

Visual Basic 5.0 uses the same visual metaphor for building ActiveX Controls as it does applications. Using this metaphor you first "draw" the interface, set some properties, write some event driven code, and you are on your way.

{ewc mvimg, mvimage,!illust.bmp}
Figure 5.

Get your mouse ready, it is time to create the visual interface for the spinner control. The spinner control can be created using a powerful new feature of the Control Creation Edition—the ability to combine existing controls into new, more specialized controls. To create the spinner control, a standard textbox and a vertical scrollbar will be combined.

First, click on the textbox control in the Visual Basic toolbox. The textbox control is identified with the number 1 in the figure below. Using the mouse, draw a small textbox in the upper left hand corner of the Project2 window. Second, click on the vertical scrollbar control in the toolbox and draw it just to the right of the textbox control. The vertical scrollbar control is identified with the number 2 in the figure below. Finally, drag the control sizing handle to surround the newly drawn controls. The control sizing handle is identified with the number 3 in the figure below. Your ActiveX spinner control should now look similar to the diagram below.

{ewc mvimg, mvimage,!illust.bmp}
Figure 6.

At this point we now have a visual interface for a spinner control. The next step is to write some event driven code that will place the current value of the vertical scrollbar into the textbox. When the user clicks on the up or down arrow of the vertical scrollbar, the value displayed in the textbox needs to increment or decrement. To make this happen, some code needs to be written in the *Change* event of the vertical scrollbar. Double click on the vertical scrollbar that you just drew. This will display the code window. In the code window, type the following line of code:

```
text1.text = vscroll1.value
```

Hopefully you were surprised as you began to type the line of code. You just experienced a small glimpse of the new "intelligence" added to the Visual Basic 5.0, Control Creation Edition development environment. As soon as you typed the "dot", Visual Basic 5.0 displayed a list of all allowable properties for the textbox. All ActiveX components contain this type of information and Visual Basic makes it automatically available when needed.

After the code is entered, close the code window by clicking on the close box (#1 in Figure 7 below). Finally, close the spinner control form by clicking on its close box (#2 in Figure 7 below).

Figure 7.

At this point, if all has gone well and you closed the spinner form, the spinner control no longer appears gray in the toolbox and is ready to be used/tested (#1 in Figure 8 below). Your environment should now look similar to the figure below.

{ewc mvimg, mvimage,!illust.bmp}
Figure 8.

To test the newly created control, click on it in the toolbox and draw it on Form1 as shown below. Press F5 to run the application. As you click the up and down arrow in the spinner control, the value in the textbox changes, just as we coded it.

{ewc mvimg, mvimage,!illust.bmp}
Figure 9.

Congratulations, you have just created your first ActiveX control and you only wrote a single line of code!

Your next steps at this point should be to review the documentation located on http://www.microsoft.com/vbasic/controls and learn more about the great new features that can easily be built into your ActiveX controls. Some of these include custom property pages, making your controls data aware, and placing your controls on Web pages as an alternative to writing Java applets.

Once your ActiveX control is created you can immediately put it to use in all Microsoft ActiveX control hosts including:

Office ’97

Visual Basic 4.0

Visual C++

Front Page 2.0

If you would like to place your ActiveX Controls on web pages then you should use the ActiveX Control Pad. The ActiveX Control Pad ships as a part of the Visual Basic 5.0, Control Creation Edition and can also be freely downloaded from the following location:

{ewc mvimg, mvimage,!intjump.bmp}

Additionally, ActiveX controls can be hosted by all applications that have licensed the Visual Basic For Applications environment. As of the creation of this document, these include:

{ewcmvimg,mvimage,!intjump.bmp} Adobe Systems

{ewcmvimg,mvimage,!intj Autodesk

ump.bmp}

{ewc mvimg,mvimage,!intjump.bmp}

B&C Microsystems Inc.

{ewc mvimg,mvimage,!intjump.bmp}

HAHT Software, Inc.

{ewc mvimg,mvimage,!intjump.bmp}

Micrografx

{ewc mvimg,mvima

ge.!intjumpb mp}

{ewcmvimg,mvimage.!intjumpb mp}

NetManage

{ewcmvimg,mvimage.!intjumpb mp}

Onpoint Technologies, Inc.

{ewcmvimg,mvimage.!intjumpb mp}

OSI Software, Inc.

{ewcmvim

g.
m
vi
m
a
g
e.
!
in
tj
u
m
p.
b
m
p}

{e
w
c
m
vi
m
g.
m
vi
m
a
g
e.
!
in
tj
u
m
p.
b
m
p}
Rockwell Software

{e
w
c
m
vi
m
g.
m
vi
m
a
g
e.
!
in
tj
u
m
p.
b
m
p}
Sagent Technology, Inc.

{e
SAP

wcmvimg,mvimage,!intjumpb mp}

{ewcmvimg,mvimage,!intjumpb mp}

StarBase Corporation

{ewcmvimg,mvimage,!intjumpb mp}

Systems Modeling, Inc.

{ewcmvimg,mvimage,!intjum

p.
b
m
p}

{e
w
c_
m
vi
m
g,
m
vi
m
a
g
e,
!.
in
tj
u
m
p.
b
m
p}

Tandata Corp.

{e
w
c_
m
vi
m
g,
m
vi
m
a
g
e,
!.
in
tj
u
m
p.
b
m
p}

The EC Company

{e
w
c_
m
vi
m
g,
m
vi
m
a
g
e,
!.
in
tj
u
m
p.
b
m
p}

Vantive

{e
w
c_
m
vi
m
g,
m
vi
m
a
g
e,

!intjump.bmp}

{ewc mvimg.mvimage.!intjump.bmp}

Visio Corporation

For the most up to date list, see the following web location:

{ewc mvimg, mvimage,!intjump.bmp}

If you would like to use your ActiveX controls on Web pages you can use the ActiveX Control Pad to script the interaction between ActiveX controls and VBScript. The ActiveX Control Pad ships as a part of the Visual Basic 5.0, Control Creation Edition and FrontPage 97.

What is the Microsoft Visual Basic programming system version 5.0, Control Creation Edition?

The Visual Basic 5.0 Control Creation Edition is the newest member of the Visual Basic programming system, the world's most popular rapid application development tool. The Visual Basic 5.0 Control Creation Edition allows developers to build ActiveX controls quickly and easily, opening up Web development to the over 3 million Visual Basic developers in the industry today. The Control Creation Edition will join the Standard, Professional, and Enterprise Edition when version 5.0 becomes available.

What is an ActiveX Control?

ActiveX technologies make it easy to create, integrate and reuse software components, or "controls," over the Internet or intranets. An ActiveX Control is a software component that can be integrated into Web pages, Microsoft Office, Microsoft Access and Visual Basic (i.e., any host that supports ActiveX Controls).

With ActiveX, developers can create components in any programming language, integrate them with any scripting language, and run those components from any type of application, including Web browsers and many of the world's most popular business applications. Assembling Web sites from a wide variety of existing software components speeds time to market, allows Web site producers to build more engaging and effective sites, and results in a more intriguing and productive experience for Web surfers.

What is ActiveX?

ActiveX is a set of technologies that integrate software components in a networked environment, regardless of the language in which they were created. This integration of components enables content and software developers to easily create interactive applications and Web sites. As a leading commercial object model, ActiveX has been widely adopted by corporate MIS and ISV communities and is used by millions of application and content developers today.

What features will Visual Basic, Control Creation Edition offer?

Visual Basic 5.0, Control Creation Edition allows developers to build ActiveX Controls quickly and easily. ActiveX Controls can be created using any of the following scenarios:

® Data aware controls

® Custom property pages

® Multiple controls per OCX file

® Container controls

® Irregularly shaped controls

® Properties that download asynchronously

® Self describing controls with type libraries

Where and how can these ActiveX Controls be used?

Controls created with Visual Basic 5.0, Control Creation Edition can be included in any application that supports ActiveX Controls, including Microsoft Internet Explorer, Visual Basic 4.0, the Visual C++ development system, the Visual FoxPro database system, Microsoft Office 97, Delphi, Netscape Navigator, via plug-ins and more, as well as the application of any vendor that has licensed Visual Basic 5.0, Applications Edition.

How does the Control Creation Edition relate to the other versions of Visual Basic 5.0?

Visual Basic 5.0, Control Creation Edition includes the new intelligent code editor, forms engine and interactive debugger of Visual Basic 5.0, but it does not include some of the functionality found in the retail editions of Visual Basic, such as the Jet database engine, report writing, integration with the Visual SourceSafe version control system and other features not required for control creation. As a result, the version available for downloading from the Web is a lean 5.5 MB. While the Control Creation Edition will provide support for building

ActiveX Controls, it cannot be used to develop standalone applications.

Visual Basic 5.0, Control Creation Edition will be available as a standalone product as well as integrated into the Standard, Professional and Enterprise editions of Visual Basic 5.0 when they are released.

How do controls used within a custom ActiveX control get downloaded to the web users computer?

The Visual Basic 5.0 Setup Wizard analyzes all dependencies and creates everything you need for web applications. It creates a compressed file containing all files needed for your control, an informational file that the browser uses to install and register the components, and a working HTM page that contains everything your HTM page needs for downloading, including the codebase reference.

How does the release of the Control Creation Edition affect Visual Basic for Applications?

All licenses of Visual Basic for Applications will become ActiveX Control hosts. Visual Basic for Applications 5.0 is an ActiveX Control container and the language engine at the heart of Visual Basic 5.0, Control Creation Edition, Office 97 and a growing number of third-party licensees. For more information on Visual Basic for Applications, please visit http://www.microsoft.com/vba/.

How does Visual Basic 5.0, Control Creation Edition relate to Visual C++ and the Visual J++ development tool and "Internet Studio"?

Visual Basic 5.0, Control Creation Edition, Visual J++ and Visual C++ can all create ActiveX components. However, Visual Basic 5.0, Control Creation Edition is the only tool of the three that applies RAD techniques to creating controls. "Internet Studio" is a member of the visual tools family, and makes it easy to assemble and integrate components developed in the other visual tools within an overall Web solution. Visual J++, Visual C++ and Visual Basic are programming languages that are *component producers*, while "Internet Studio" is a c*omponent consumer*.

How large are controls produced by Visual Basic, Control Creation Edition?

The average control will be about 20k bytes, but actual size varies according to complexity.

Do ActiveX controls created with Visual Basic 5.0, Control Creation Edition share the same limitations as Java Applets?

No. Unlike Java applets, ActiveX controls created in Visual Basic 5.0, Control Creation Edition can take full advantage of Windows 95 and NT operating systems. ActiveX controls produced with Visual Basic 5.0, Control Creation Edition are pre-built for digital signatures, giving the Web surfer the option to accept or deny the downloading of ActiveX components.

What is the Visual Basic Virtual Machine and how does it get to the web users computer?

Visual Basic controls are interpreted by the virtual machine on the web surfers computer in much the same manner as a Java applet. The virtual machine (MSVBVM5.DLL) also provides high performance services common to all Visual Basic authored ActiveX controls. The virtual machine is automatically downloaded and installed 1) the first time a person navigates to a web page containing any Visual Basic authored control and 2) when any control built using a newer version of the virtual machine is accessed

How many ActiveX Controls are currently available? How will Visual Basic 5.0, Control Creation Edition affect this market?

More than 2,000 ActiveX components are available today from hundreds of developers for use both over the Internet and in line-of-business applications. The number of ActiveX Control ISVs has grown by over 400 percent since the beginning of the year. With the release of Visual Basic 5.0, Control Creation Edition and its

ease of use, we expect to see a significant increase in the number and types of controls available, providing substantial benefits to developers and customers.

Giga Information Group estimates that the ActiveX component market will generate $240 million in revenues this year and projects that this market will grow to more than $1 billion by 2000.

The following topics are included in this section:

® Integrated Development Environment
® ActiveX Controls
® Microsoft Forms

When you code a client application for use with the **Winsock** control, you specify the name for a remote host and the port through which communication will take place. You then use the **Connect** method to connect with the server.

Specifying the Remote Host and the Remote Port

To code a client application, you first specify the name of the remote host and the port it is listening on in the Form_Load event of the client.

When you reference the remote host, you can use either the Internet Protocol address (121.111.1.1) for the host, or the friendly name. The following code uses the friendly name.

```
Private Sub Form_Load()
    Winsock1.RemoteHost = "RemoteComputerName"
    Winsock1.RemotePort = 1001
End Sub
```

Initiating the Connection

Once you have specified the remote host and remote port, you can initiate a connection with the server by using the Connect event in the Click event of a **Command** button, as shown in the following code.

```
Private Sub cmdConnect_Click()
    Winsock1.Connect
End Sub
```

The integrated development environment of Visual Basic for Applications is an entirely new environment for developers for Microsoft Office. The most significant change is that the new Visual Basic Editor exists outside the host application window. This enables developers to write code in Visual Basic for Applications and simultaneously view the results of their programming in the host Office application. Although the IDE exists outside the application window, it runs in the same memory space as its host, thereby benefiting from tight integration for event handling, as well as enhanced performance. The IDE also provides the following:

- **Enhanced editor** with syntax checking, color-coded syntax, block comment/uncomment, and support for code drag-and-drop.
- **New Project Explorer** for navigating the project components.
- **Enhanced Properties window** for setting and viewing object properties.
- **New debugging tools** to help track program execution and find bugs.
- **Enhanced Object Browser** for browsing and searching for properties and methods across object model libraries.
- **New IntelliSense features** for instant syntax reference and object model assistance to speed programming time.
- **Improved code security** with improved password protection and encryption to lock both documents and projects (developers for Microsoft Access can remove source code from their projects for added protection).
- **Conditional compilation** for incorporating debugging code that will not execute in the final version of the application.

{ewc mvimg, mvimage,!illust.bmp}
Figure 1. Visual Basic Editor

Code Window

The IDE of Visual Basic for Applications includes a code window for the document, for any document sections that support code behind them (for example, Microsoft Excel sheets), and for each code and class module and form in the project.

Drag-and-drop functionality has been implemented throughout the editing environment. Developers can drag and drop code and variables between code windows, into the Watches window, into the Locals window and across projects.

Project Explorer

{ewc mvimg, mvimage,!illust.bmp}
Figure 2. Project Explorer

The Project Explorer displays the project components, (specifically forms, modules and references) associated with each open project, in an outline view. Each project appears as a new root in the outline control, enabling easy switching among different documents' projects for developers working on multiple projects simultaneously. A project exists for each open document and template (see *Fig. 2*).

The following project components are shown in the tree view:

- The document that owns the project (Word document, Microsoft Excel workbook or PowerPoint presentation).
- Sections internal to the document that can have code behind them (for example, worksheets in Microsoft Excel).
- Forms that belong to the project.
- Code and class modules that belong to the project.
- References to other Microsoft Office documents (references are permitted to documents within the same application only).

Properties Window

{ewc mvimg, mvimage,!illust.bmp}
Figure 3. Properties window

The Visual Basic Editor has a Properties window that displays the properties of Microsoft Office documents, forms and controls. The Properties window can also be shown in the host application's window. For example, Word uses the Properties window to display properties of ActiveX Controls that are embedded in a Word document.

The Properties window has two tabs: Alphabetic and Categorized. The Alphabetic tab view provides an alphabetical list of properties. In the Categorized tab view, properties are grouped by category (for example, all properties related to color, font or position). These categories can be expanded or collapsed in the Properties window.

Debugging Tools

{ewc mvimg, mvimage,!illust.bmp}
Figure 4. Debug menu

Visual Basic for Applications includes new and enhanced debugging tools to help the developer identify compile errors, program logic errors and run-time errors. The debugging tools in Visual Basic for Applications include a Locals window, Watches window and an Immediate window. The Locals window also includes a Call Stack Browser that shows the current variable and enables the developer to jump to procedure definitions and references.

® **Locals variables window** automatically displays all the declared variables in the current procedure and their values.

® **Watches window** enables monitoring the value of a particular variable or expression. Code execution may be interrupted when a watch expression's value changes or equals a specified value.

® **Immediate window** instantly evaluates any expression or statement in Visual Basic, such as call to a Sub of Function.

® **Call Stack** displays a list of active procedure calls during break mode.

To speed the debugging process, code can be dragged and dropped from the editor into the Immediate and Locals windows.

Object Browser

{ewc mvimg, mvimage,!illust.bmp}
Figure 5. Object Browser

The Visual Basic Editor has an improved Object Browser (see Fig. 5). The browser differentiates between built-in properties, custom properties, methods, event handlers and user-defined procedures, as well as indicates globally accessible members. The browser also shows function return types, parameter names and types, and user-defined types and constants. Hyperlink jumps to referenced objects enable easy navigation of the object hierarchy. New to the Object Browser is the ability to search for objects and members across type libraries.

IntelliSense Features

Visual Basic for Applications, Version 5.0, brings IntelliSense technology to the developer, providing on-the-fly syntax and programming assistance and reference. The developer can choose to turn these automated features off and access them on demand through the Visual Basic for Applications menu or with keystroke combinations. The following features are available while in the code window, as well as from the Immediate window:

® **Complete Word.** Completes the word that is being typed, once enough letters are entered to make it distinct. (Keystroke equivalent: Ctrl+Alt+A)

® **Quick Info.** When a procedure or method name is entered (followed by a space or an opening parenthesis), a tip automatically appears under the line of code writing. The tip gives syntax information about the procedure. (Keystroke equivalent: Ctrl+I)

{ewc mvimg, mvimage,!illust.bmp}
Figure 6. Quick Info

® **List Properties/Methods.** Displays a pop-up menu listing the properties and methods available for the object that precedes the period. (Keystroke equivalent: Ctrl+J)

{ewc mvimg, mvimage,!illust.bmp}

Figure 7. List Properties/Methods

® **List Constants.** Displays a pop-up menu listing the constants that are valid choices for the property typed and that precede the equal sign (=). (Keystroke equivalent: Ctrl+Shift+J)

{ewc mvimg, mvimage,!illust.bmp}
Figure 8. List Constants

® **DataTips pop-up information.** When Visual Basic for Applications is in break mode and the cursor is placed over a variable, the value of the variable is displayed in a tooltips-like window.

{ewc mvimg, mvimage,!illust.bmp}
Figure 9. DataTips

® **Margin Indicators.** Developers can set a breakpoint, set the next statement, or set a bookmark by clicking in the margin of the code editor.

{ewc mvimg, mvimage,!illust.bmp}
Figure 10. Margin Indicators

Project Import and Export

Projects and their components (forms, class modules and procedures) can be exported and imported for code storage and sharing. Exported project components are stored as standard text.

Block Comment and Uncomment

An Edit toolbar provides quick access to the Block Comment and Uncomment feature, which enables developers to select blocks of code and, with the click of a button, comment out the entire selected block of code.

Project Security

Developers need the ability to distribute their solutions in a secure and protected form to prevent modification to their code base. Projects associated with a Microsoft Office document can be protected without restricting access to the rest of the document. This protection is achieved by setting a protection property on the document in the property sheet. Developers using Microsoft Access can also remove the source code from databases that they distribute to others, thereby reducing the size of the database and protecting their intellectual property. For more information, see “Remove Source Code” in the Microsoft Access 97 section, later in this document.

Conditional Compilation

Developers can set compilation flags within their code to control the resulting behavior of the application. For example, developers may wish to include debugging code or custom messages during the development phase that they would not want end users to see. Conditional compilation enables developers to create multiple compilations of their application easily by setting flags in their code.

Class Modules

Visual Basic for Applications now includes support for class modules. A class module can be thought of as a template for a user-defined object. Within a class module, developers can add public procedures that define custom methods and properties for the object. When an instance of the class is created, developers can apply these custom methods and properties as they would for any object. Modules can be shared across projects through drag-and-drop or export and import functionality or from within a Visual Basic-based automation server.

{ewc mvimg, mvimage,!illust.bmp}
Figure 11. ActiveX Controls menu

All of the Microsoft Office applications support ActiveX Controls, formerly called OLE Controls or custom controls. ActiveX Controls are pre-built, reusable software components that enable developers to add rich interactive capabilities easily to their Microsoft Office solutions. In Microsoft Office 97, ActiveX Controls can be placed on Microsoft Forms or directly on Microsoft Office documents. More than 1,000 ActiveX controls are available today from third-party vendors.

{ewc mvimg, mvimage,!illust.bmp}
Figure 12. Microsoft Forms

Microsoft Excel, Word and PowerPoint share a powerful new Forms Designer, Microsoft Forms, which enables the developer to create custom dialog boxes. Because these applications use the same Forms Designer, developers learn how to create custom dialog boxes only one way for all three applications, and custom dialog boxes designed in one application can be shared with another.

Forms support the architecture for the next generation of ActiveX Controls and provide the following benefits:

® Superior performance (for example, time to save, load, redraw) when compared to today's windowed controls.

® Advanced features for forms design, including alignment and layout tools, drag and drop of controls and in-place editing of captions.

® Ability to place the same controls on Microsoft Office documents.

® A broad set of controls, including all the standard interface controls and container controls such as frame and multipage controls.

Toolbox

{ewc mvimg, mvimage,!illust.bmp}
Figure 13. Toolbox

The toolbox can be customized to display all registered controls in the system. Developers can fully customize the Microsoft Forms toolbox in the following ways:

® **Adding and removing pages.** Developers can customize and organize ActiveX Controls by adding pages to the toolbox.

® **Exporting and importing pages.** Developers can customize toolbox sharing and distribution of toolboxes and toolbox pages.

® **Control templates.** Developers can create control templates by dragging a control (or a number of "grouped" controls) from the form back to the toolbox. This action creates a template of that control, which can then be reused just like any other control. Templates carry the property settings of the parent control.

® **Customizing controls.** Developers can customize the icon representing the control or control templates created. The ToolTip presented for the control is also entirely customizable.

A second toolbox is used to place ActiveX Controls directly on Microsoft Office documents. Developers can maintain separate toolboxes for managing controls for forms and controls for documents.

To successfully complete installation of Personal Web Server and several of the labs for this chapter, you will need to make sure that three services are running.

To start the services for the labs in this chapter

1. Open the Control Panel, double-click the **Personal Web Server** icon, and make sure that the Personal Web Server, FTP, and HTTP services are running.
2. Click the **Startup** tab, and make sure that Personal Web Server is started. If it is not, click the **Start** button.
3. Click the **Services** tab, and make sure that the status of the FTP service is **Running**. If it is not, select **FTP** from the **Services** list, and then click the **Start** button.
4. On the **Services** tab, make sure that the status of the HTTP service is **Running**. If it is not, select **HTTP** from the **Services** list, and then click the **Start** button.

You can create an instance of Internet Explorer and use it as an ActiveX component. Internet Explorer has an object model that exposes methods and properties for use by other applications.

An **InternetExplorer** object is implemented as an in-process (DLL-based) Component Object Model (COM) class.

Properties of the InternetExplorer Object

Internet Explorer supports the same properties and methods as the **WebBrowser** control, along with several that the **WebBrowser** control does not support. Almost all of these additional properties pertain to the user interface.

This table describes the additional properties supported by the **InternetExplorer** object.

Property	Description
FullName	Specifies the fully qualified path of the executable file that contains the Internet Explorer application.
FullScreen	Indicates whether Internet Explorer is in full-screen or normal window mode.
MenuBar	Determines whether the Internet Explorer menu bar is visible or hidden.
Name	Returns the name **Microsoft Internet Explorer**.
StatusBar	Determines whether the Internet Explorer status bar is visible or hidden.
StatusText	Sets or returns the text for the status bar.
Toolbar	Determines whether the Internet Explorer toolbar is visible or hidden.

The **InternetExplorer** object also supports the **Quit** method.

For information about the basic methods, properties, and events that are available from the **InternetExplorer** object, see Basic Operations of the WebBrowser and Coding a WebBrowser.

Adding a Reference to the InternetExplorer Control to a Project

To use Internet Explorer as an ActiveX component, you add a reference to the Microsoft Internet Explorer Controls component.

u To add a reference to Internet Controls component in Visual Basic

1. On the **Program** menu, click **References**.
2. Select **Microsoft Internet Explorer Controls**, and then click OK.

You use the **InternetExplorer** object just as use any other object in Visual Basic.

The following code creates an object variable and an instance of Internet Explorer:

```
Dim browser as InternetExplorer
Set browser = CreateObject("InternetExplorer.Application")
```

You can now set or return properties, or call methods of the **InternetExplorer** object.

Like other ActiveX components, Internet Explorer is invisible when it starts. To make it visible, set the **Visible** property to **True**.

This code navigates to the Web site www.Microsoft.com.

```
browser.Visible = True
browser.Navigate "http://www.Microsoft.com"
```

Clients get information about Automation objects from a server by inspecting the server component's type library. Also known as an object library, a type library provides the following information.

® Definitions of interfaces supported by the object.

® Descriptions of the properties, methods, and events provided by the object.

® The return types and parameters types of the object's methods and events.

® The Dispatch IDs these methods and properties use.

® The name of the Help file and Help topics to be displayed about the object and its methods and properties.

As with servers, type libraries are registered with the operating system. The keys for type libraries are located in HKEY_CLASSES_ROOT, in the TypeLib folder.

Type libraries can be stored as part of the .exe. or .dll file that they describe, or they can be stored as separate .olb or .tlb files. When you use Visual Basic to create a component, its type library is built into the executable file.

An Object Browser is a tool that lets you view the methods and properties of an Automation object.

The following illustration shows the methods and properties that are available from a fictitious object named **Project1**.

{ewc mvimg, mvimage,!v06g005.bmp}

The **Project/Library** list box shows that the Object Browser is viewing the **Project1** type library. The **Classes** pane shows that there are two objects available from **Project1**: **Class1** and **Form1**. The **Members of** pane shows that **Class1** contains two properties (**gsUser** and **User**) and two methods (**ShowDate** and **SquareIt**).

To see a demonstration of how the Object Browser works, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

Note For a more information about the features of objects, use the Object Browser OLE2VW32, located in the folder Visual Basic\Tools.

Together, COM and Automation make programming easier because they use interfaces to structure access to objects and group methods and properties. But, this is only one of the ways that Automation organizes your view of an object. When a server is designed the objects that it offers can be structured into an object model.

An object model defines the relationship among objects available from a server. An object model organizes the objects in the server into a hierarchy that represents which objects contain other objects. This structure makes the server easier to use.

The following illustration shows a simplified view of the objects that are available from Microsoft Excel.

{ewc mvimg, mvimage,!v06g025.bmp}

When creating client applications, it is important to understand the relationships between constituent objects (objects that get created automatically with the creation of another object) and how you use with them. While some of the objects are publicly creatable (with **New** and **CreateObject** in Visual Basic), some objects are created by invoking a method in another object. For example, the **Application** object in Microsoft Excel is publicly creatable, while its contained **Range** object is not.

For information about how to create an object model, see Creating an Object Model in Chapter 7: Creating ActiveX Code Components.

Command buttons added through the property page for the toolbar are always added starting at the left edge of the toolbar. This will interfere with the combo box you have just added. For this reason, add the command buttons from the command toolbox.

With the Visual Basic 5.0 Control Creation Edition, developers now have the ability to create ActiveX controls as easily and quickly as creating form based applications with previous versions of Visual Basic. The benefits of creating these re-usable, industry standard components are numerous, including:

® Construction of applications based on re-usable components.

® Ability to use these components in applications and tools that support ActiveX, including Visual Basic, Internet Explorer, Microsoft Office, FoxPro, Borland Delphi, and so on.

® Ability to create rich Internet- and intranet-based applications by integrating Visual Basic-authored ActiveX controls with HTML pages.

Creating ActiveX controls for use in HTML pages opens up a new world of opportunities. The combination of matching the up the rich content capabilities of HTML and application-specific functionality of Visual Basic provides the ability to create solutions not possible before. Additionally, these solutions are more accessible to end users, who can simply navigate to them over the Internet or intranet using their Internet browser.

Note If you plan to distribute your control to other developers who may use the control at design time (in development), be sure to use the traditional options in Setup Wizard, rather than creating an Internet component download.

The following topics are included in this section:

® Steps to Preparing Your Control for the Web

® Preparing Software for Internet/intranet Download with Setup Wizard

® Deployment Steps in Depth

® Testing Your Internet Component Download

® Advanced Customization Through the .inf File

® Appendix: Check List of Resources

® Appendix: Check List for Creating a Professional Internet Component Download

We have already discussed one of the advantages of component architectures: evolving an application over time. You'll discover benefits of building applications out of components in addition to this ability to make convenient and flexible upgrades of existing applications. These benefits include application customization, component libraries, and distributed components.

Application Customization

Users often want to customize their applications, just as home owners often want to customize their homes. End users like to make an application work the way *they* work. Corporate programmers building custom solutions with off-the-shelf applications demand adaptable applications. Component architectures lend themselves to customization because each component can be replaced with a different component that better meets the needs of the user.

Suppose we have components based on the editors *vi* and Emacs. In Figure 1-3, User 1 has configured the application to use *vi,* while User 2 prefers Emacs. Applications can be easily customized by adding new components or changing existing components.

{ewc mvimg, mvimage,!illust.bmp}
Figure 1-3. Building applications from components allows for greater customization. User 1 prefers to use the *vi* editor, while User 2 is a fan of Emacs.

Component Libraries

One of the great promises of component architectures is rapid application development. The fulfilled promise would have you choose components from a component library and snap them together like Lego building blocks to form applications. (See Figure 1-4.)

{ewc mvimg, mvimage,!illust.bmp}
Figure 1-4. Components can be assembled into libraries from which applications can be rapidly developed.

Snapping applications together from standard parts has been an unrealized dream of software engineers for a long time. However, the process has already started with the development of ActiveX controls, previously called OLE controls. Visual Basic, C, C++, and Java programmers can all take advantage of ActiveX controls to speed up the development of their applications and web pages. All applications will, of course, still need some custom components, but the majority of an application can be built using standard components.

Distributed Components

With increasing bandwidth and importance of networks, the need for applications composed of parts spread all over a network is only going to increase. Component architecture helps simplify the process of developing such distributed applications. Client/server applications have already taken the first step toward a component architecture by splitting into two parts: the client part and the server part.

Making a distributed application out of an existing application is easier if the existing application is built of components. First, the application has already been divided into functional parts that can be located remotely. Second, since components are replaceable, you can substitute for one component a component that has the sole purpose of communicating with a remotely located component. For example, in Figure 1-5, Component C and Component D have been located on different remote machines on the network. On the local machine, they have been replaced by two new components, Remoting C and Remoting D. These new components forward requests from the other components, across the network to Component C and Component D. The application on the local machine doesn't care that the real components are remotely located. Similarly, the remote components themselves don't care that they are remotely located. With the proper remoting component, the application can be completely ignorant of where the actual component is located.

{ewc mvimg, mvimage,!illust.bmp}
Figure 1-5. Components located across a network on a remote system.

Now that we've seen some benefits of using components, let's look at what is required to build components. Then we'll take a look at the role COM plays in building components.

The advantages of using components result directly from their ability to dynamically plug into and unplug from an application. In order to achieve this capability, components must meet two requirements. First, components must link dynamically. Second, components must hide (or *encapsulate*) the details of how they were implemented. Trying to determine which requirement is more important is a chicken and egg problem. Each requirement is dependent on the other. I tend to think of dynamic linking as the crucial requirement for a component, and information hiding as a necessary condition for dynamic linking. Let's examine these two requirements in detail.

Dynamic Linking

Our ultimate goal is to have an end user replace components in our application while the application is running. While we will not always provide the user this amount of control, we would like the ability to support it. Support for changing components at run time requires the ability to dynamically link components together.

The best way to understand the importance of dynamic linking is to picture an application built from components that can't link at run time. If you wanted to change one of the components in the system, you would be required to statically relink or recompile the program and then redistribute it. After all, you can't expect your end users to relink the application for you. Even if they knew how to link your application, they probably wouldn't have a linker, much less the correct linker. An application made of components that must be statically relinked every time a component changes is equivalent to a monolithic application.

Encapsulation

Let's see why dynamic linking requires encapsulation. To form applications, components are connected to one another. If you want to replace a component with a new component, you must disconnect the old component from the system and then connect the new one. The new component must connect in the same manner as the old component, or you'll have to rewrite, recompile, or relink these components. It doesn't matter whether the components and the application support dynamic linking. If you change the way a component connects to other components, you break the system of components and force at least a recompile, if not a rewrite.

To understand how this leads to encapsulation, we need to define some terms. A program or a component that uses another component is called the *client*. A client is connected to a component through an *interface*. If the component changes without changing the interface, the client doesn't have to change. Similarly, if the client changes without changing the interface, the component doesn't have to change. However, if changing either the component or the client changes the interface, the other side of the interface must also change.

Therefore, to take advantage of dynamic linking, components and clients must strive not to change their interfaces. They must be encapsulated. Details of how the client and the component are implemented must not be reflected in the interface. The more the interface is isolated from implementation details, the less likely the interface will change as a result of changing the client or the component. If the interface doesn't change, changing a component will have little effect on the rest of the application.

Isolating the client from the component's implementation puts some important constraints on the component. The following is a list of these constraints:

1. The component must hide the computer language used for its implementation. Any client should be able to use any component regardless of the computer language in which the client or the component is written. Exposing the implementation language creates new dependencies between the client and the component.
2. Components must be shipped in a binary form. If components are to hide their implementation language, they must be shipped already compiled, linked, and ready to use.
3. Components must be upgradable without breaking existing users. New versions of a component should work with both old and new clients.
4. Components must be transparently relocatable on a network. A component and the program that uses it should be able to run in the same process, in different processes, or on different machines. The client should be able to treat a remote component the same way it treats a local component. If remote components were treated differently from local components, the client would have to be recompiled whenever a local component was moved elsewhere on the network.

Let's discuss a couple of these points in detail.

Language Independence

Many people don't believe in requiring components to be language independent as specified in the first

constraint above. For the sake of argument, let's say we have an application that can be customized only with components written in Objective C. Not many people are going to write components for us because most people are using C++. After a while, we realize no one is writing components for our application, so we come up with a way to write components in C++. This results in more components written for our application. However, a new language, EspressoBeans, becomes popular and everyone flocks to it, leaving their C++ compilers to gather dust. To stay in the ball game, we come out with yet another way to write components: one using EspressoBeans. Now we have three completely different ways to write components for our application. But at this moment we go out of business. It seems that in our market segment more users were programming in Visual Basic than in any other language. Our competitor allowed its customers to write components in any language, including Visual Basic, and is still in business.

In a language-independent architecture, components can be written by everyone. They don't become obsolete as programming languages evolve. The architecture can flourish in the marketplace.

Versions

A user might have two client applications using the same component. Suppose one application was designed to use a new version of a component while the other was designed to use the old version. Installing the new version of the component shouldn't break the application that uses the old component. In Figure 1-6, the old application is using a new *vi* component, just like the new application.

{ewc mvimg, mvimage,!illust.bmp}
Figure 1-6. New components must not break old components but should enhance other new components.

However, backward compatibility shouldn't restrict the growth of a component. It should be possible to radically change the behavior of a component for new applications while still supporting the old applications.

Next let's see how COM meets these requirements for building components.

COM is a specification. It specifies how to build components that can be dynamically interchanged. COM provides the standard that components and clients follow to ensure that they can operate together. Standards are as important to component architectures as they are to any system with interchangeable parts. If there weren't a standard for VHS video tapes, you'd be lucky to find a tape that would work in your VCR. The thread sizes of garden hoses and outdoor faucets are governed by standards. PCMCIA cards and their card slots have a set of standards that they must follow. The signal that a television or a radio receives is specified in a standard. Standards are especially important when different parts are developed by different people in different organizations working in different countries. Without standards, nothing would work together. Even at Microsoft we have programming standards that we follow. (At least we follow them most of the time.)

The COM Specification is a document that sets the standard for our component architecture. The components we develop in this book follow the standard. The COM Specification is included on this book's companion CD-ROM. However, you are probably wondering, "What are COM components, exactly?"

COM Components Are...

COM components consist of executable code distributed either as Win32 dynamic link libraries (DLLs) or as executables (EXEs). Components written to COM standard meet all our requirements for a component architecture.

COM components link dynamically. COM uses DLLs to link components dynamically. But as we've seen, dynamic linking by itself doesn't guarantee a component architecture. The components must be encapsulated.

COM components can be encapsulated easily because they satisfy our constraints:

® COM components are fully language independent. They can be developed using almost any procedural language from Ada to C to Java to Modula-3 to Oberon to Pascal. Any language can be modified to use COM components, including Smalltalk and Visual Basic. In fact, there are ways to write COM components that can be used by macro languages.

® COM components can be shipped in binary form.

® COM components can be upgraded without breaking old clients. As we'll see in Chapter 3, COM provides a standard way to implement different versions of a component.

® COM components can be transparently relocated on a network. A component on a remote system is treated the same as a component on the local system.

COM components announce their existence in a standard way. Using COM's publication scheme, clients can dynamically find the components they need to use.

COM components are a great way to provide object-oriented APIs or services to other applications. COM components are also great for building language-independent component libraries from which applications can be rapidly built.

While COM components are many things, there are many things that COM is not but with which COM is confused.

COM Is Not...

COM is not a computer language. COM does not compete with computer languages. Discussions about whether C++ is better than COM or vice versa don't make any sense because COM and C++ have different purposes. COM tells us how to write components. We are free to choose the language in which we write the components. In this book, we will exclusively use C++ to develop COM components.

COM also does not compete with or replace DLLs. COM uses DLLs to provide components with the ability to dynamically link. However, COM is, in my opinion, the best way to take advantage of the ability of DLLs to dynamically link. Any problem that can be solved using a DLL can be solved better using a COM component. I won't use a DLL except to support COM components. That's how effectively COM uses DLLs.

COM is not primarily an API or a set of functions like the Win32 API. COM does not provide services such as *MoveWindow* or the like. (COM does provide some component management services, described below.) Instead, COM is primarily a way to write components that can provide services in the form of object-oriented APIs. COM is also not a C++ class library like the Microsoft Foundation Classes (MFC). COM provides a way to develop language-independent component libraries, but COM does not provide any implementation.

The COM Library

There is more to COM than just the specification—COM does entail some implementation. COM has an API, the COM Library, which provides component management services that are useful for all clients and components. Most of the functions in the API are not too difficult to implement yourself when you're developing COM-style components on a non-Windows system. The COM Library was written to guarantee that the most important operations are done in the same way for all components. The COM Library also saves developers time implementing their own components and clients. Most of the code in the COM Library is support for distributed or networked components. The Distributed COM (DCOM) implementation on Windows systems provides the code needed to communicate with components over a network. Not only does this save you from writing the networking code, it saves you from having to know how to write it.

The Way of COM

My favorite aspect of COM is that it is a way of writing programs. You can use the COM programming style on any operating system and with (as I have said) any language. You don't need any of the COM code implemented on Windows systems to write COM-like components. In this book, the samples in the first eight chapters can be easily modified to not use any Windows code. COM embodies the concepts of component programming. Like structured programming or object-oriented programming, COM is a method for organizing software. The concepts of good software design are embedded in the COM specification.

COM Exceeds the Need

COM meets all of the component architecture requirements that we have discussed. COM uses DLLs to provide components that can be interchanged at run time. COM insures that these components can take full advantage of dynamic linking by:

® Providing a standard for components to follow

® Allowing for multiple versions of components in an almost transparent way

® Enabling similar components to be treated in the same way

® Defining a language-independent architecture

® Supporting transparent links to remote components

COM enforces strict separation of the client and the component. The power of COM lies in this extreme isolation. Those who are interested can now take a look at the history of COM.

A Brief History of the COM Universe

COM was developed to make applications more customizable and flexible. The original goal was to support a concept known as *object linking and embedding.* The idea was to have a document-centric view of the world where, for example, you could edit your spreadsheet from your word processor. Microsoft's version of object linking and embedding is called OLE. The first version of OLE used something called *dynamic data exchange* (DDE) to communicate between the client and a component. There was no COM in OLE 1. DDE was built on top of the message-passing architecture of Windows. About the politest thing I can say about DDE is that it does work—sort of. DDE is slow. It is also hard to write DDE code that works correctly. In addition, DDE isn't very robust or flexible. Needless to say, something better had to be found.

The solution was COM. COM is smaller, faster, and more robust and flexible than DDE. For its second version, OLE was rewritten using COM instead of DDE. COM is the new foundation on which OLE is written. However, OLE was the first COM system developed. As such, OLE isn't an example of the best that COM can do. OLE has a reputation for being big, slow, and hard to code. The way OLE is implemented causes this trouble; it's not the result of using COM.

Still, remember OLE was attempting to do something that had not been done before. With OLE, you can place a picture from one vendor's drawing program into another vendor's word processor and edit it without leaving the word processor. OLE was trying to do this in a nonrestrictive way. Instead of forcing the client and the component to make only a limited connection, OLE specifies a rich connection between client and component. The component can do just about anything it wants, and the client has to be prepared. This makes OLE very difficult to program.

New products based on COM are forthcoming from Microsoft, and some of them are amazing. Other developers are undoubtedly writing COM components as well. The future of COM is bright!

Technology in the software industry progresses much too rapidly to have your application sitting on the shelf for two years waiting for you to upgrade it. The solution is to break the application into separate little applications or components. These components are then assembled at run time to form applications. The components can be upgraded independently of each other, allowing the application to evolve over time.

COM provides a standard way to write components. Components following the COM standard can be combined to form applications. It doesn't matter who wrote the component or how it was implemented. A COM component can be used with other components. The key to interchanging components is encapsulation. COM provides encapsulation by emphasizing the connection or interface between the component and the client. In the next chapter, we'll see why interfaces are so important for components. We'll also see how to implement a COM interface.

There are four steps to take in preparing your control component for use on Web pages:

® Package the component for Internet Component Download.

® Digitally sign the software to be distributed.

® Ensure that the component is safe for scripting and initialization, and mark it as such.

® Arrange to provide for licensing of components that require it.

Internet Component Download

On the Web, the browser is responsible for copying all of the needed files to users' hard disks—and for not copying those they already have. Depending on the browser's security settings, they may not get your control at all.

When you use Setup Wizard to create Internet Component Download, you create a cabinet (.cab) file for your ActiveX control. This .cab file contains all of the information necessary to download, install, and register the components required to run your control on an HTML page. The benefits of this architecture include:

® File compression for faster download.

® A single file for your .ocx and .inf which describes other files required.

® Dependency files, such as Msvbvm50.cab, will only be downloaded as necessary—thus minimizing download time.

® Easier updating when new versions of your component are created.

® Automatic installation performed when the control component is downloaded.

For More Information The Setup Wizard step by step later in this document illustrates these concepts.

Digital Signing

Code received via the Internet lacks shrink-wrapped packaging to vouch for its reliability, and users are understandably skeptical when they're asked to download it. A *digital signature* provides an opportunity for you to reassure them by creating a path from them to you, should your software harm their system. (Note that digital signing puts your name to your code, but does not ensure that it's hazard-free.)

When you develop software for distribution over the Internet, you work with a third party known as a *certificate authority* (CA) to obtain a *digital certificate,* which will give users information about you. The CA provides and renews your certificate, authenticates your identity, and handles legal and liability issues for broken security. In addition, the CA typically provides the tools you need to digitally sign your components. Your digital certificate is included with all code you digitally sign and distribute over the Internet.

Important The default setting of Internet Explorer doesn’t allow software not digitally signed to be downloaded to the end user's machine. It is very important that you obtain a digital signature for software components you intend to distribute them over the Internet.

ActiveX SDK

The ActiveX SDK, available from Microsoft, includes tools to begin digitally signing your components, as well as test certificates you can use when testing and debugging downloading scenarios. For more information and to download the ActiveX SDK, use your Internet access to visit www.microsoft.com/intdev/sdk/.

For More Information Details on digital signing are available later in this document. The latest on digital signing can be found at www.microsoft.com/intdev/security/misf8-f.htm within the Site Builder Workshop on the Microsoft Web site.

Safety

Unless you design your component so that it's guaranteed to interact safely with script and data passed to it during initialization, a malicious script or data could have harmful results on users' computers—and users would come looking for you when that happens. By default, Internet Explorer will display a warning, and will not download a component that has not been marked safe for scripting and initializing.

Safe for Scripting

On an HTML page, your component's functionality is accessed through scripting, such as when events are handled through VBScript. For ActiveX controls, scripting is the only way to fully utilize the control's features in a browser. So while your control may be a trusted component from a reputable source (you), a malicious script may be able to use its methods to delete files on the user's machine, install macro viruses, and worse. A component is safe for scripting when it can't be scripted to harm the user's computer.

Safe for Initialization

Another potential security hazard is initializing your control’s state using untrusted data. On an HTML page, your control's initial state is set from the PARAM attributes that accompany the OBJECT tag in the HTML embedding the control. A component is safe for initializing when its properties can't be passed data that in some way harms the user's computer.

For More Information Details on control safety are available later in this document. Also, guidelines for making your controls safe for scripting and initialization are available in "Designing Controls for Use With HTML" in the document, "Building ActiveX Controls," available with the Control Creation Edition.

Licensing

If you wish to make a business of creating ActiveX content for the Internet, you can create controls that require a license for use in an HTML page. A Visual Basic application supplies the run-time license to a control automatically but this process does not happen with a HTML page. The license manager supplies the license to the control from the .lpk file.

Note It's important not to confuse this sort of *run-time* license—in which your control is being used as part of an HTML solution—with the *design-time* license developers receive when they download and run a setup program containing your control, so that they can use the control in their own applications.

For More Information Details on the licensing model and the .lpk file is available later in this document, and from the ActiveX SDK.

Before packaging your control components with the Visual Basic Setup Wizard, it's important to understand the issues involved in deploying controls on the Web. The following topics offers details on the remaining three steps in preparing your component:

® Digital Signing Explained

® Marking Controls as Safe for Scripting and Safe for Initialization

® Licensing of Software on the Internet

Internet Explorer's default security setting requires that any ActiveX component (portable executable, .inf, or .cab) must have a digital signature before that component can be downloaded to the client's machine.

A digital signature identifies the legal entity that created the software (who may be held legally responsible). In other words, it is a claim to the user made under a digital signature for authenticity.

There are four types of files available in Visual Basic 5.0 that may have a digital signature:

® .exe files

® .cab files

® .dll files

® .ocx files

Digitally signing controls and components is required for smooth, responsible ActiveX control distribution. Adding a digital signature establishes accountability between your name and certain types of files.

Digital signatures themselves must be purchased from a certificate authority—a company that validates your identity and issues a certificate for you. If a legal action is brought against you as a digital signature holder, the certificate authority becomes an identity witness.

If you cannot obtain a signature for your own use, you may need to arrange to have your control signed by a firm that has a signature. Typically, a third party firm providing a signature would receive your complete source code, review the code, and then sign the component file after recompiling it from the source code.

Note There are different types of signatures, called classes. Some classes of signatures may not be available in some parts of the world for various reasons.

Authenticode

The Authenticode technology used in the ActiveX SDK is derived from public-key signature algorithms. The technology uses an algorithm that requires one key to encrypt the data and a second different but complimentary key to decrypt the data. These two keys are called a key pair. For technical details on how this is done, search the Internet for RSA or public-key signatures.

Full encryption or decryption of a large file is very time consuming, so an alternative approach is done with ActiveX controls. In this approach, a number called a hash is calculated from all of the bytes in the file (similar to a CRC, or Cyclical Redundancy Check). This hash is encrypted using the private key of the key pair and inserted into the file. When the file is downloaded or installed, a hash is calculated from the file and compared to the inserted hash. If the numbers match each other, then the file is deemed to be verified.

This process works for verifying the contents but does not identify the source of the software. The certificate authority verifies the identity of the source and issues a certificate that contains the source's name encrypted with the private key of the certificate authority. Internet Explorer has access to the public key so it can determine your identity and the name of the authority supplying the public key.

The following figure illustrates the nesting of one set of data and public keys inside another set. To find out the name of the publisher you must progressively apply the public key obtained from one level to unlock the next layer. For someone to create a bogus certificate, they must obtain the private key on the next higher level. It is required that this information be protected by physical security schemes.

{ewc mvimg, mvimage,!illust.bmp}

The goal of Authenticode is to establish clear accountability as a deterrent to the distribution of harmful or irresponsible code.

The utility needed to apply a digital signature to your software is not included in Visual Basic 5.0. The Authenticode software utility and documentation are in the ActiveX SDK, available for download from Microsoft's Internet site. The digital signature must be obtained from an issuing authority such as GTE, VeriSign, Inc. (See the ActiveX SDK for more information on obtaining and using signatures.)

For More Information For more information and to download the ActiveX SDK, use your Internet access to visit www.microsoft.com/intdev/sdk/.

The Holder of the Digital Signature

Keeping your digital signature safe is very important. Some firms (including Microsoft) do not keep their signature file on site. The signature is kept with the certificate authority and files are sent there for signing. If you decide to keep your signature on site, then you must ensure that it is strictly controlled. Any file signed with the signature is warranted by your firm—regardless of whether the signature was authorized or not.

The default security setting of Internet Explorer requires that before any ActiveX component can be instantiated on an HTML page (with specific values or used with a script), it must be marked safe for scripting and safe for initialization. This basically states that that the developer certifies that the control cannot be made to do “harmful” things (such as reformatting your hard drive) via an HTML script or during its initialization.

Safe for Scripting Responsibilities

Marking a control as safe for scripting means that it is safe when automated by any HTML writer or developer. The developer who marks their software as safe for scripting states a warranty: “No matter what the code in VBScript or Java Script is, this control cannot be made to do anything wrong.” The software does not allow any harm to the user. In particular, there should be no possibility of corrupting the user’s PC or obtaining unauthorized information from the PC (for example, no security leaks have been introduced to the system). A control that is safe for scripting is usually safe for initialization, unless there are properties that cannot be set except at initialization.

A control that permits any of the following actions to occur as a result of scripting may not be safe for scripting:

® Create a file with a name specified by scripting.

® Insert information into the registry or .ini files.

® Retrieve information from the registry or .ini files.

® Read a file from the hard drive with a name specified by scripting.

Obtaining any information automatically about the user via an ActiveX control and exposing it to the script writer means the control is not safe for scripting. Such innocent activities may be criminal acts in some countries.

There is a fine line between a safe or unsafe action. For example, a control that always writes information to its own registry entry may be safe. A control that allows you to name the registry entry is unsafe. A control that creates a temporary file with a name it creates without using any initialization or scripting value may be safe. A control that allows the name of the temporary file to be assigned at initialization or by scripting is unsafe.

Determining whether a control is safe is not a trivial exercise. To understand what safety means in terms of what not to implement for your control, you might begin by noting the API calls and commands not implemented in VBScript (see http://www.microsoft.com/vbscript/).

Prior to marketing a control as safe for scripting, it is a good practice to create documentation recording the justification. Remember that you are making an assertion under a digital signature—you should take the same type of care due any legal contract. One part of your documentation might be two tables listing the following elements:

® All the exposed methods, events, and properties of the ActiveX control.

® All the files opened, API calls used, and the information obtained or set.

If there are any dependencies or data transfer between the elements of these two tables, then the control is probably not safe for scripting. A second part of the documentation may be a review by a seasoned external expert developer who understands both the source code and VBScript.

Safe for Initialization Responsibilities

Marking a control as safe for initialization means that it is safe when initialized by either a well meaning but incompetent HTML writer or a seasoned developer. Safe for Initialization is a weaker safety standard than safe for scripting. The control makes no claim about the safety of its methods or of run-time properties. There are no claims about what information the control makes available to the VBScript writer.

A control marked safe for initialization guarantees to do nothing bad no matter what data it is initialized with. The definition of “bad” is the same as in safe for scripting. The critical issue is what may happen as a result on initialization with particular values. A control that automatically sends information at initialization to an HTTP server may be safe, as long as the server address cannot be changed. You may create files and any other activities needed for the control as long as the file properties (name, file size) or activity’s nature does not change because of assigned values.

A safe for initialization control does not write or modify any registry entries, .ini files, or data files as a *result of initialization* parameters.

What the Safety Flags Do Not Mean

In the two prior sections there seem to be long lists of activities that a control could not do. Actually, these are lists of things that the control cannot do as a result of *initialization* or as a result of *scripting*. A control can create, update or delete a file, modify an .ini file, or update a registry entry and still be marked as safe for scripting and safe for initialization. These activities must occur as a result of using the control *regardless of how scripting or initialization is done*.

Safe for scripting or safe for initialization does not necessarily mean that the control is safe for use. A control could reformat your hard-drive after 10 uses, since this action does not occur as a result of scripting or initialization, it may be marked as safe. Of course, the person who writes such a control is liable for the usual penalties reserved for virus and Trojan horse writers.

The following examples illustrate these two ideas:

- A control that is licensed until December 31,1999 or 10 usages on a specific PC may modify a registry entry on each use, and on December 31, 1999, spawn a batch file to erase itself. This behavior is not the result of either scripting or initialization values.
- A file that creates a temporary file of pointers to memory locations and other working space elements. When the size of a temporary file is significantly dependent on the initialization parameters (for example 10 times the size of a .GIF file pointed to), this behavior may render it as an unsafe control.

The key phrase to understand is "as a result of". In the second preceding example, creating a temporary file that corresponds to the file size of a resource passed to the control, may seem a gray area to many developers. If the developer adds code to prevent the temporary file from ever exceeding 10 percent of the available free space on a drive then the control *could* be marked safe.

If, on the other hand, there are no such checks, an HTML writer could consume every free byte available on the drive; the control would be unsafe. In the latter case, the user's PC may become unusable without a reboot.

Responsibility for this does not belong to the end user, who has only one or two megabytes of free space; nor does it belong to the HTML writer, who assigned a three megabyte image file to the control. The responsibility belongs to the developer, who failed to include adequate safeguards.

The bottom line to ensure the safety of your controls is to always have your code independently reviewed by a seasoned developer who understands the issues well.

For More Information Guidelines for making your controls safe for scripting and initialization are available in "Designing Controls for Use With HTML" in the document, "Building ActiveX Controls," available with the Control Creation Edition.

Some controls can require a license for use in an HTML page. A Visual Basic application supplies the run-time license to a control automatically, but this process does not happen with an HTML page. The license manager supplies the license to the control from the .lpk file.

For More Information Detailed information on the licensing model and the .lpk file is available in the ActiveX SDK.

Licensing ActiveX controls for use on the Internet is a complex subject. This section explains Microsoft's current licensing scheme, how to implement it with Visual Basic 5.0, and how HTML writers will use it on their pages. A general discussion of some licensing issues is also included. An expression in a license such as "used on one PC" has a totally different meaning on the Internet when the "one PC" may be an HTTP server that is hosting 200 different IP addresses, with 30,000 pages, created by 300 HTML writers. The software may be in-use (running in memory) on thousands of anonymous PCs and never in use on the PC that it was licensed for. These issues should be reviewed before you start licensing your ActiveX controls for use on the Internet.

Microsoft's Licensing scheme

Visual Basic 5.0 implements a licensing scheme using the **IClassFactory2** mechanism that is implemented by most existing licensed controls. The ActiveX Software Development Kit includes a section entitled "Justification of the Proposed Scheme" that reviews some of the strengths and weakness of this scheme.

Referring back to the HTML file written by the wizard we see the following lines for the License Manager.

```
<!--If any of the controls on this page require licensing, you must create a
license package file.
    Run LPK_TOOL.EXE in the tools directory to create the required LPK file.

<OBJECT CLASSID="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
    <PARAM NAME="LPKPath" VALUE="LPKfilename.LPK">
</OBJECT>
-->
```

The basics of License Manager can be summarized as follows:

® Each HTML page using a licensed control requires a license file (.lpk).

® Only the first encountered .lpk file is used on any HTML page.

® The .lpk file must contain runtime licenses for all of the licensed controls on the HTML page.

® The .lpk file must be on the same server as the HTML page.

® The .lpk file contains a plain text copyright notice to dissuade casual copying of .lpk files.

The HTML writer uses the License Package Authoring Tool to create appropriate .lpk files for their pages. This tool presents the HTML writer with a list of installed controls that they can embed in an .lpk file.

{ewc mvimg, mvimage,!illust.bmp}

The .lpk file appears similar to what is shown below:

```
LPK License Package
////////////////////////////////////////////////////
// WARNING: The information in this file is
// protected by copyright law and international
// treaty provisions. Unauthorized reproduction or
// distribution of this file, or any portion of it,
// may result in severe criminal and civil penalties,
// and will be prosecuted to the maximum extent
// possible under the law. Further, you may not reverse
// engineer, decompile, or disassemble the file.
////////////////////////////////////////////////////
{3d25aba1-caec-11cf-b34a-00aa00a28331}
```

AQWWF/QT0BG9ewCg0QKOmo=
BQAAAA=yhtrFpw/zxGAdURFU1QAACkAAAB
DAG8AcAB5AHIAaQBnAGgAdAAgACgAYwApACAAMQA5ADkANQAsACAAMQA5ADkANgAg
AE0AYQBjAHIAbwBtAGUAZABpAGEALAAgAEkAbgBjAC4A=

Alternative License Models

At least one firm is offering alternative licensing services. This alternative licensing service uses a simple effective license mechanism for the Internet. The control requires a key that varies with time. In the online environment, a call is made to an HTTP server that then examines the URL of the HTML page making the call (and the user's IP location if needed for runtime licenses) and returns a license key if appropriate. This solution does not work well in an intranet environment isolated from the Internet.

There are other licensing mechanisms possible. For example, if run-time licenses are required, then Basic authentication, NT authentication, or other authentication may be required before the license file may be obtained.

Testing your download file is more complex than testing a conventional setup program because the software installs only if it's missing from the PC or if an older version exists.

Two aspects of component download should be tested:

® Downloading, or copying the files.

® Verifying safety levels.

Also, when there are files listed in the Confirm Dependencies dialog box when you run Setup Wizard, you should test the download on Window 95, Windows NT 3.51, and Windows NT 4.0 to detect any operating system specific problems.

Checking the Installation of Download Components

The simplest level of checking your download is to remove the software registration from the registry. This may be done by running Regsvr32, as in the following example:

```
regsvr32 /u controldemo.ocx
```

Thereafter, loading the HTML page generated by the wizard should cause the software to be downloaded and re-installed.

Note Remember to select Binary Compatibility in your Visual Basic Project settings to prevent multiple different CLSIDs for your software.

If you have left the `DestDir=` line in the .inf file blank (the default), you should see the file appear in your Occache directory. You may also find other copies of your component in subdirectories (called conflict directories), such as Occache\Conflict.2 and Occache\Conflict.5. The date/time of each file will be different, indicating different builds of the control (and different CLSIDs).

If, on the other hand, you specify the System directory in the `DestDir=` line in the .inf file, only the latest version will saved. Care must be taken to ensure that the CLSID remains the same between the builds.

If you want to do heavy duty testing to ensure that all of the support files also install and register, you could un-register everything in your system directory and your Occache directories. This may be done by the following command lines in each directory:

```
for %f in (*.ocx *.dll) do regsvr32 /u /s %f
```

When you are finished testing, then the following restores the entries:

```
for %f in (*.ocx *.dll) do regsvr32 /s %f
```

Important When making such dramatic changes to the contents of your System directory, it's a good idea keep your setup disks handy.

Checking the Safety Levels

You can check safety levels by creating additional HTML pages for testing, as suggested in the following table:

Type of HTML Page	Characterized By
No initialization or scripting	No PARAM values or other variables set by the control.
Initialization with no scripting	PARAM values assigned.
Scripting with no initialization	PARAM values are set by VBScript only.
Scripting and initialization	Initial PARAM values are set and then modified by VBScript.

You can customize the installation process by modifying the .inf file. The modified .inf file can be included in a manually built .cab file (using the .ddf project file), or it can be directly referenced by the Codebase attribute of the Object tag.

Note An .inf file is not normally used because it cannot have a digital signature. If an .inf file is used, the .ocx file should have a digital signature.

Typical modifications to an .inf file include:

® Adding a license agreement.

® Adding a Read-Me.

® Adding additional documentation.

Note As an ActiveX Control developer, you can modify your Internet Component Download, but you should be aware that doing so may place a potential liability on yourself or your firm if the modifications are done incorrectly. Some changes indicate that you guarantee, assure, or warranty that the changes are correct and truthful. Attempts to avoid these liabilities by citing “as is” or “suitability” clauses in a license agreement may be ruled invalid by many courts.

Here is an example of an .inf file:

```
;INF file for ControlDemo.ocx
;DestDir can be 10 for Windows directory, 11 for Windows\System(32) directory, or
left blank for the Occache directory.

[version]
signature=$CHICAGO$

[Add.Code]
CONTROLDEMO.OCX=CONTROLDEMO.OCX
MSVBVM50.DLL=MSVBVM50.DLL

[CONTROLDEMO.OCX]
file-win32-x86=thiscab
RegisterServer=yes
clsid={F651BF93-239B-11D0-8908-00A0C90395F4}
DestDir=
FileVersion=1,0,0,1

[MSVBVM50.DLL]
hook=MSVBVM50.cab_Installer
FileVersion=5,0,34,21

[MSVBVM50.cab_Installer]
file-win32-x86=http://activex.microsoft.com/controls/vb5/MSVBVM50.cab
InfFile=MSVBVMb5.inf
```

Note Only the original developer may legally mark a control as safe. There are *no* circumstances where you should ever mark someone else’s control as safe. Changing the safety of a control may not only be a copyright infringement but may result in criminal charges.

Hand Building .cab Files

If you have modified the .inf file generated by Setup Wizard, you will need to rebuild the .cab file. To rebuild the .cab file you will need Microsoft’s Diamond Cabinet Builder, Diantz.exe. This file is usually located in \VB\

SetupKit\kitfil32.

The simplest way is to use the existing .ddf file with the following command:

```
DIANTZ /F CONTROLDEMO.DDF
```

If you modify the .inf file in the Support directory and digitally sign the .ocx in the Support directory, be sure to change the locations in the .ddf. You will also need to verify that any files added in the .inf file have also been added to the .ddf file so they will be included in the .cab (and thus available for installation).

Here is an example of a .ddf file. This was created by the Setup Wizard in the step by step procedure in "Preparing Software for Internet/intranet Download with Setup Wizard":

```
.OPTION EXPLICIT
.Set Cabinet=on
.Set Compress=on
.Set MaxDiskSize=CDROM
.Set ReservePerCabinetSize=6144
.Set DiskDirectoryTemplate=
.Set CompressionType=MSZIP
.Set CompressionLevel=7
.Set CompressionMemory=21
.Set CabinetNameTemplate="ControlDemo.CAB"
"C:\Website\Cabfiles\ControlDemo.INF"
"C:\ControlDemo\ControlDemo.ocx"
```

The following entries are very significant and should not be changed:

Entry	Description
MaxDiskSize=CDROM	This allows the .cab file to be as large as needed.
ReservePerCabinetSize=6144	This reserves space for a digital signature.

The Diantz.exe generates two additional informational files, Setup.inf and Setup.rpt.

If the .cab file is not created, try the following command to get a verbose log of the build process.

```
DIANTZ /V3 /F CONTROLDEMO.DDF
```

Please refer to the Microsoft Developer's Network for information on modifying .inf files.

Recommended Readings

For detailed information about .inf files, see the documentation included with Microsoft's Visual C product and on MSDN. The following overview article is available on the Microsoft Developers Network Library.

® Teri Schiele, Windows 95 Application Setup Guidelines for Independent Software Vendors, MSDN Library

To obtain further information about this resource, click here to go to the Microsoft Developer Network: {ewc mvimg, mvimage,!intjump.bmp}

The following resources may be needed to create and test Internet Component Download. Most of the resources are part of the Visual Basic 5.0 package.

Resource	Purpose	Source
SetupWiz.exe	Application Setup Wizard	Visual Basic 5.0
Lpk_Tool.exe	License Pack Tool	Visual Basic 5.0 ActiveX SDK
Diantz.exe	Diamond Cabinet Builder	Visual Basic 5.0
SignCode.exe	Authenticode Signing Software	ActiveX SDK
Authenticode Certificate	Identifies the developer	Verisign Inc., GTE, and so on
Internet Explorer 3.0 or later	Testing of HTML download and installation	http://www.microsoft.com
RegSvr32.exe	Register and un-register software for testing	Visual Basic 5.0

The following check list shows the steps that you might follow to create a fully signed *safe* Internet Component Download Component.

Step	Comments
Design the software specifying the intended software safety level.	Have the design of the control reviewed by a VBScript developer.
Create the software and test.	Features may be added by the developer that inadvertently make the control unsafe.
Create a document showing the object is safe.	Most developers are unaware of what their code actually does (they know what it is supposed to do well). Powerful features often create safety problems accidentally.
Independent review.	
Set the appropriate Safety Flags and recreate the .cab.	The .inf file is changed with the addition of safe for scripting and safe for initialization entries
All digital signature to files created by you (.ocx, .dll, exe).	Never sign a file that originates from someone else. A digital signature is a statement of *ownership,* not a statement of distribution source or origin.
Create your final .cab file by using Diantz.exe	Do not recreate any file that has been digitally signed or the signature will be lost.
Digitally sign your .cab file.	Both the .cab and the appropriate contents of the .cab are signed.
Test this final .cab with each platform it may run on.	Safety levels must always be checked carefully.

This section outlines the basic steps for creating a downloadable Windows setup file as a cabinet (.cab) file for a Visual Basic 5.0-created ActiveX control using the Visual Basic 5.0 Application Setup Wizard.

The following Setup Wizard step by step illustrates the process of packaging components for Internet Component Download. In this example, the ActiveX control project is the same project you can create with the step by step procedure in "Creating an ActiveX Control," available with the Control Creation Edition. If you've created your own ActiveX control project, you can easily substitute it and other details for those given here.

Note Before using the Setup Wizard to package your own control component for Internet Component Download, you should become familiar with several important concepts—including component safety and digital signing, covered in detail later in this document.

Using Setup Wizard to Prepare a Control for Internet Component Download

1. Click the **Start Menu**, point to **Programs**, point to **Visual Basic 5 Control Creation Edition**, then click **Application Setup Wizard**.
2. If the **Introduction** dialog box appears, click **Next**.
3. On the **Select Project and Options** dialog box, click in the box near the top and enter the location of your Visual Basic project.

   You can also click **Browse** to find the project.
4. Click **Create Internet Download Setup**.

   {ewc mvimg, mvimage,!illust.bmp}

   Before proceeding with the wizard it's a good idea to click **What's New** to check the Microsoft Web site for the latest information. The Internet is a rapidly changing environment that requires developers to keep up to date with current technology and developments.

Important If you are creating a setup program to distribute your control to other developers who may use the control at design time, be sure to click **Create a Setup Program** and use the traditional options in Setup Wizard, rather than creating an Internet component download. A setup program for Internet component download (providing for *run time* use) is configured very differently from a traditional setup program (which provides for *design time* use).

---

5. Click **Next** to open the **Internet Distribution Location** dialog box.

   {ewc mvimg, mvimage,!illust.bmp}

   Using this dialog box, you can specify where to put the .cab, .htm, and support files the Setup Wizard will create. The .cab file is the software installation file. If you look at your Windows 95 Installation CD-ROM, for example, you will see that Windows 95 software images are stored in .cab files. Placing all of your .cab files in one directory can simplify administration and tracking of the components.
6. Under **Destination**, select the drive to which Setup Wizard will copy the files, then select a path.
7. Click **Next** to open the **Internet Package** dialog box.
8. {ewc mvimg, mvimage,!illust.bmp}

   Using this dialog box, you can choose whether to use Microsoft's Web site as a source for common files, or to supply them from your own server. Unless you are in an intranet environment with no access to the Microsoft Web site, you should accept the default setting. This ensures that your users will get the latest versions of Microsoft-supplied DLLs, and that customized versions of files (say, for a particular operating system) will be sent. The correct version will be sent based on information about its operating system that Internet Explorer reports to the HTTP site.

   Common support files available on the Microsoft Web site and the Visual Basic 5.0 Control Creation Edition include:

   - Msvbvm5b.cab—Visual Basic 5.0 VM, required for all Visual Basic 5.0 built applications.
   - CmCtlb32.cab—Visual Basic 5.0 Common Controls, including TreeView, ListView, TabStrip, and so on.

- Cmdlgb32.cab—Visual Basic 5.0 Common Dialog Control.

Each of these files have been digitally signed by Microsoft and may be freely downloaded.

In an intranet environment you should use a single intranet URL to reduce file replication and allow better administration. Ideally, all developers on your intranet will use the same URL.

Note These .cab files will not be downloaded if the correct version already exists on the end user's machine.

---

9. Click **Safety** to open the **Safety** dialog box.

   {ewc mvimg, mvimage,!illust.bmp}

   Using this dialog box, you can select **Safe for scripting** and **Safe for initialization** for each control listed under **Components**.

Important Safety is an important issue that will be discussed later in this document. Be sure you have a thorough understanding of safety issues before using these settings with controls you deploy on the Web.

---

10. Click **OK** to close the **Safety** dialog box, then click **Next** to open the **ActiveX Components** dialog box.

    {ewc mvimg, mvimage,!illust.bmp}

    Using this dialog box, you can specify ActiveX components to include with your control.

11. Click **Next**.

    If your control uses ActiveX components, the **Confirm Dependencies** dialog box will open so that you can verify or change your selections. Since ControlDemo.ocx has no dependencies, the **File Summary** dialog box opens. This dialog box shows what additional files the application uses.

    {ewc mvimg, mvimage,!illust.bmp}

    At this point, it is good to do a bit of double-checking before proceeding to the next step. It can be very difficult to take back or alter things once they are posted to the Internet, so caution is needed. Redistributed files must not be altered in any way, for both legal and practical reasons.

Important The list of files displayed here doesn't necessarily mean all of these files will be put into your .cab file—it's simply a list of all files required to run your control. For example, Msvbvm50.dll will not be included in the .cab file. If you are unsure about the files displayed, use those Setup Wizard lists by default.

---

12. Click **Next** to open the **Finished!** dialog box.

    {ewc mvimg, mvimage,!illust.bmp}

    Here you can save the template for later use.

13. Click **Finish**.

Files Created by the Wizard

Viewing the target .cab directory shows the files created by the wizard. There are two sets of files: distribution files and support files (for any customization such as adding license information). The distribution files are located in the Cabfiles directory in this example, as shown in the following illustration. The support or intermediary files are not deleted. You may need to rebuild the .cab files with additional files or you may wish to apply a digital signature to the components in the .cab file.

{ewc mvimg, mvimage,!illust.bmp}

The 15KB ActiveX control is now a smaller 12KB .cab file. The Support directory was created by the wizard. The Support directory contains the input files for the .cab and a Diamond Directives (.ddf) file. (Additional information about .cab files is available at www.microsoft.com/workshop/java/CAB-f.htm.)

The contents of the Support directory are shown in the following illustration:

{ewc mvimg, mvimage,!illust.bmp}

All together, the wizard creates five file types, described in the following table:

Extension	Description
.cab	Windows setup file or "cabinet" file that contains the .ocx file, the .inf file and other dependent files. You can digitally sign this file to prevent tampering.
.htm	HyperText Markup File. This file illustrates how to insert the ActiveX control into an HTML page with both Codebase attribute for automatic downloading, and with license files.
.ddf	Diamond Directives File. This is the project file for creating the .cab files.
.inf	Setup disk information file. This file includes information on how the control should be installed. It permits customization of installation.
.ocx	ActiveX control component. You can digitally sign this file to prevent tampering.

HTML Object Tag

The following fragment from an .htm file created in the preceding example illustrates what Setup Wizard produces. ActiveX controls are placed in an HTML page by using the Object tag as defined by the World Wide Web Consortium (W3C). The actual download is controlled by the Codebase attribute. (You can obtain more information about the Object tag and its use at http://www.w3c.org/.)

```
<HTML>
<!--If any of the controls on this page require licensing, you must create a
license package file.
    Run LPK_TOOL.EXE in the tools directory to create the required LPK file.

<OBJECT CLASSID="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
    <PARAM NAME="LPKPath" VALUE="LPKfilename.LPK">
</OBJECT>
-->

<OBJECT
    classid="clsid:F651BF93-239B-11D0-8908-00A0C90395F4"
    id=ShapeLabel
    codebase="ControlDemo.CAB#version=1,0,0,1">
</OBJECT>
</HTML>
```

Examine the second Object tag in the fragment. This tag contains a reference to the WmfView control. It has a Class Identifier (CLSID) of D68CFE2A-139A-11D0-BD7B-00A0D1028E9A. Each different ActiveX control you create will have a different CLSID. The CLSID is used to create an instance of the control on the HTML page (similar to the process of placing a control on a Visual Basic form).

Using this information, Internet Explorer checks the registry to see if the control exists. When it does not exist, or when the version number of the control is less than the version specified in the Codebase attribute of the Object tag, Internet Explorer downloads and installs the file specified in the Codebase attribute.

The important parts of the Object tag are summarized below:

Tag Attributes	Description
Classid	Class Identifier contains the CLSID for the ActiveX control.
Id	Name of the control. Used in scripting, this is equivalent to the Name property of a control on a Visual Basic form.
Codebase	Minimum version number of the control required and the location of an installation point.
Width	The width of the control in pixels.

Height	The height of the control in pixels.

For More Information For more information on Internet component download and modifying the download files created by the Setup Wizard see "Advanced Steps of Preparing Software for Internet/Intranet Download" later in this document.

Percent complete is calculated as **Progress** times 100, divided by **MaxProgress**.

Do not use the **End** statement in the event procedure of the **Exit** menu item because the Unload event will not occur and the reference to the startup page will not be saved to the registry.

In this exercise, you will create an application that uses the **Winsock** control to conduct a peer-to-peer chat session with another application on the same or remote computer.

Create the Project

1. Create a new Standard EXE project.
2. Add the Winsock component to the project.
3. Add the **Winsock** control to the form.
4. Modify the form to resemble the following illustration.

   {ewc mvimg, mvimage,!v11g035.bmp}
5. Save the project as MyChat in the folder \Lab11.

Implement binding

In the event handler for the binding Click event:

1. Set the **RemoteHost** property of the **Winsock** control.
2. Set the **RemotePort** property of the **Winsock** control.
3. Use the **Bind** method to set the local port.

Implement data sending

1. Add a handler for the **Send** button's Click event.
2. Use the **SendData** method of the **Winsock** control to send data from the appropriate text box.
3. Clear the text in the text box.

Process received data

1. Add a handler for the **Winsock** control's Data_Arrival event.
2. If data exists, receive it from the control's buffer and add it to the beginning of the **Received Data** text control. Store only the last 2KB of the existing received data.

Enable the bind button

1. Set the **Enabled** property of the **Bind** button to **False**.
2. Add a private procedure that enables the **Bind** button only if the controls on the remote computer, remote port, and local port contain data.
3. To call the added procedure, add an event handler for the Change event of the remote computer, remote port, and local port.

Test the application

1. Compile the application.
2. Run the application in the Visual Basic environment.
3. Set the Remote Computer name to the network name of your computer.
4. Set the Remote Port to 1000.
5. Set the Local Port to 1001.
6. Bind the application.
7. Run a copy of the compiled application on the same computer.
8. Set the Remote Computer name to the network name of your computer.
9. Set the Remote Port to 1001.
10. Set the Local Port to 1000.
11. Bind the application.
12. Send text from each port, and validate that it is properly received.

Click here to connect to the Microsoft Site Builder page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

The Site Builder Workshop is the place to get the latest and greatest information about Internet technologies. Check out the What's New page for a list of changes and additions. The site includes information about:

® Authoring/editing

® Designing/creating

® Programming

® Site administration

® Planning/production

® Web gallery

A DLL can notify or give information to a client by calling a routine provided by the client. This client routine is known as a callback function. A single call to the Windows API can invoke this callback function multiple times.

Enumeration procedures, which are those that list a set of items, are a common example of DLLs that use callback functions to provide information to the client.

Callbacks in Visual Basic

Visual Basic enables you to create callback procedures that provide notification to a client. For example, you can create a callback procedure to notify a client each time a server performs an operation. You then use a DLL to call the procedure.

The following illustration shows how callback procedures work in Visual Basic.

{ewc mvimg, mvimage,!v05g005.bmp}

Passing a Callback Procedure to a DLL

Visual Basic uses the **AddressOf** operator to pass a callback procedure to a DLL. The callback procedure must be located in a standard module, and it must have the correct syntax.

Finding the exact syntax for a callback procedure can be difficult, because it is generally found only in the documentation. Using the correct syntax is important, because Visual Basic does not provide syntax checking to notify you of an error

---

Important Using an incorrect syntax in a callback procedure will most likely cause a fatal error to occur in your application during run time.

When specifying callbacks from Visual Basic, be sure to follow the following two requirements.

® The callback function must be located in a standard module.

® You must use the **AddressOf** operator when passing the callback procedure to a DLL.

The rest of this topic shows how to create and use a Visual Basic callback procedure with a DLL.

Declaring the Function

The Windows API exports the function **EnumChildWindows**, which enumerates all of the child windows of a parent window, and contains the following declaration.

```
Declare Function EnumChildWindows Lib "user32" Alias "EnumChildWindows" (ByVal
hWndParent As Long, ByVal lpEnumFunc As Long, ByVal lParam As Long) As Boolean
```

The first parameter, **hWndParent**, requires the handle of the parent window. The second parameter, **lpEnumFunc**, requires the address of the callback procedure. The third parameter is any 32-bit value that the client wants to pass. This value will be passed subsequently to the callback function.

Writing the Callback

The following code shows how a callback routine is written in Visual Basic.

```
Public Function EnumProc(ByVal hWndChild as Long , _
    ByVal lParam As Long) As Boolean
   ' Do some processing.

End Function
```

Calling the DLL

Call the enumeration procedure, and then specify the callback procedure by using the **AddressOf** operator, as shown in the following code.

```
EnumChildWindows form1.hWnd, AddressOf EnumProc, 0
```

To see a demonstration of how to use callback functions, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

It is not necessary to size the form in the Form_Load event because the Form_Resize event occurs just before the form is initially displayed.

In this exercise, you will use the **AddressOf** operator to establish a callback function to receive timer notifications.

Create the initial project

1. Create a new .exe project and save it in the <*Install Directory*>\Labs\Lab05.
2. Add the component for the Microsoft Windows Common Controls.
3. Place a **ProgressBar** control on the form.
4. Place a command button on the form.
5. Add a module to the project and save the project.

Add code to start the timer

1. Add declarations to the module for the **SetTimer** and **KillTimer** Windows API functions.
   You can use the API Viewer that ships with Visual Basic to make this task easier.
2. Create a **Sub** procedure named **StartTimer** and add it to the form.
3. Declare a form-level integer variable to hold the return value of the timer ID.
4. Call the **SetTimer** function with the parameter settings as shown in the following table.

Parameter	Value	Comment
HWnd	0	Not used when specifying a callback routine.
idEvent	0	Not used when specifying a callback routine.
Interval	200	The interval, in milliseconds, at which the timer while fire.
Callback	Addressof TimerProc	The address of the timer callback procedure; the procedure with the name TimerProc.

5. Set the value of the **ProgressBar** to zero.
6. Set the caption on the command button to **Stop**.

Add code to end and reset the timer

1. Create a **Sub** procedure named **EndTimer** and add it to the form.
2. If the timer ID is not 0, call the **KillTimer** function with the appropriate parameter values.
3. Set the timer ID variable to 0.
4. Set the caption on the command button to **Start**.
5. Add a Form_Unload event handler that calls **EndTimer**.

Code the controls on the form

1. In the Click event of the Command button, test for the following conditions:
   a. If the timer ID variable is 0, call the **StartTimer** procedure.
   b. If the timer ID variable is not 0, call the **EndTimer** procedure.
2. Create a public **Sub** procedure named **UpdateProgressBar**.
3. Calculate the next **ProgressBar** value.
   This is the current value plus some increment; an increment of five works well here.
4. Test for the following conditions:
   a. If the next value is 100 or greater, set the **ProgressBar** value to 100 and call **EndTimer**.
   b. If the next value is less then 100, set the **ProgressBar** value to the calculated value.
   {ewc mvimg, mvimage,!code.bmp}

Add the timer callback function

1. In the module, add the callback function with the following syntax:

```
Public Sub TimerProc(ByVal hwnd As Long, ByVal msg As Long, _
  ByVal idEvent As Long, ByVal curTime As Long)
```

2. In the callback function, call the form's **UpdateProgressBar** function.
3. Save and test the application.

   The **Start** button activates the timer and invokes the callback procedure. The timer continues to fire at intervals of 200 milliseconds (you can see this as the progress bar values update) until the callback is explicitly terminated by a call to the **KillTimer** function.

```
Private Sub Command1_Click()
    On Error GoTo HandleError
    Set cc = New Lab.CreditCard
    cc.ExpireDate = "1/1/99"
    cc.PurchaseAmount = 50
    cc.CardNumber = 1234
    MsgBox "Approval = " & cc.Approve
    Exit Sub

HandleError:
    Select Case Err.Number
    Case vbObjectError + 1000
        MsgBox "object error"
    Case Else
        MsgBox "Unknown error: " & Err.Number & " " & Err.Description
    End Select
End Sub
```

Client Event Code

```
Dim WithEvents cc As Lab.CreditCard

Private Sub cc_Status(ByVal StatusText As String)
    Label2.Caption = StatusText
End Sub
```

CreditCard Class Event Code

```
Public Event Status(ByVal StatusText As String)

Public Function Approve() As Boolean
    Dim sngEndTime As Single

    RaiseEvent Status("Dialing bank...")
    'simulate delay dialing bank
    sngEndTime = Timer + 2
    Do While Timer < sngEndTime
        DoEvents    ' Yield to other processes.
    Loop

    RaiseEvent Status("Processing card...")
    'simulate delay processing card
    sngEndTime = Timer + 2
    Do While Timer < sngEndTime
        DoEvents    ' Yield to other processes.
    Loop

    'dummy logic for approving credit card
    If msngAmount < 1000 And ExpireDate > Now() Then
        Approve = True
    Else
        Approve = False
    End If
End Function
```

There are several scenarios for testing a Personal Web Server. You can test a Personal Web Server when it is:

® Connected to the Internet.

® Connected to an intranet.

® In a stand-alone configuration.

The first two scenarios are discussed in the Personal Web Server documentation. For information about testing a server, see "Getting Started with Microsoft Personal Web Server" in the documentation.

Configuring Personal Web Server on a Stand-alone Computer

A special case of setting up and testing Personal Web Server is running Personal Web Server on a stand-alone computer. This enables you to test the labs for this chapter, even if you are not connected to either the Internet or an intranet.

To follow this procedure, you must have a network adapter, network drivers, and TCP/IP installed on the computer you want to run in stand-alone mode.

Important Do not follow this procedure if your computer is connected to a network. Otherwise, you may cause problems for your own computer and other computers on the network.

Make sure to install Personal Web Server before starting this procedure.

---

u **To configure your network settings to run Personal Web Server on a stand-alone computer**

Before starting this procedure, you should close all open applications. For the setting changes to take effect, restart your computer.

Be sure to record your current computer settings in steps 3 through 5, so that you can restore them after you have finished testing Personal Web Server.

1. Open the Control Panel and double-click the **Network** icon.
2. In the **Network** dialog box, click the **Configuration** tab. Select the TCP/IP component for the network adapter installed on your computer, and then click the **Properties** button.

   To see an illustration of the **Network** dialog box settings for Step 2, click this icon.
   {ewc mvimg, mvimage,!illust.bmp}
3. After completing Step 2, the **TCP/IP Properties** dialog box is displayed. To assign a specific IP address to your computer, click the **IP Address** tab.
   a. Record the current computer settings so you can return your computer to its original state.
   b. Select the option **Specify an IP address**.
   c. Enter a specific IP address (for example, 123.45.6.78) in the area provided.
   d. Leave the **Subnet Mask** information blank; a value will be assigned for you.

   To see an illustration of the **TCP/IP Properties** dialog box settings for Step 3, click this icon.
   {ewc mvimg, mvimage,!illust.bmp}
4. To make your stand-alone computer a WINS Server, click the **WINS Configuration** tab.
   a. Record the current computer settings so you can return your computer to its original state.
   c. Select the option **Enable WINS Resolution**.
   d. Type the same address you entered in Step 3.c (123.45.6.78) in both the **Primary** and **Secondary WINS Server** address fields.
   e. Leave the **Scope ID** field blank.
   f. Click OK to accept the setting changes.

   To see an illustration of the **TCP/IP Properties** dialog box settings for Step 4, click this icon.
   {ewc mvimg, mvimage,!illust.bmp}
5. If you want to be able to use computer names instead of IP addresses when testing applications on your

stand-alone computer, you must also change the DNS configuration settings. To change these settings, click the **DNS Configuration** tab.

a. Record the current computer settings so you can return your computer to its original state.

b. Select the option **Enable DNS**.

c. Type the name of your computer in the area provided for the host name.

d. Leave all other fields blank.

To see an illustration of the **TCP/IP Properties** dialog box settings for Step 5, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

6. Click OK to close the **Network** dialog box, and click **Yes** when you are prompted to restart your computer.

You can use DLLs to extend the functionality of your Visual Basic applications. DLLs can provide you with many performance advantages that are not available with Visual Basic.

This section introduces DLLs, and explains the basic steps involved in declaring and calling DLL procedures. It also describes how you can use the API Viewer in Visual Basic to copy declaration statements from the Windows API into your application.

This section includes the following topics:

® Introduction to DLLs

® Windows DLLs

® Using DLLs with Visual Basic

® Using the API Viewer

To use a DLL in your Visual Basic application, you first declare and then call the procedure that contains the declaration. If you are using the Windows API, the API Viewer enables you to retrieve the appropriate declaration for the procedure.

This section discusses how to declare a DLL procedure in Visual Basic, focusing on the elements of the **Declare** statement. This section also describes:

® Mapping C language elements to their Visual Basic equivalents.

® Enabling DLL procedures to accept more than one data type.

® Understanding how Visual Basic handles ANSI and Unicode translation of string arguments.

This section includes the following topics

® Using the Declare Statement

® Converting C Declarations to Visual Basic Data Types

® Declaring an Argument as Any

® Unicode vs. ANSI DLLs

In DLLs, the data type of the argument determines the data type returned by the procedure. However, in Visual Basic you can define a procedure so that its argument can accept any data type.

For example, suppose you want your Visual Basic application to return postal codes for addresses in the United States and in other countries. You can define a DLL procedure that accepts either a numeric value or a string value for its argument.

If your application is used in the United States, the **Integer** data type can return the numeric value of the zip code. If the application is used in a country where zip codes are not used, the **String** data type can return a string value instead.

Flexible Argument Types

To declare an argument that can accept more than one data type, you can use the **Any** data type so Visual Basic will not perform type checking. Visual Basic will then enable you to pass any type of variable for that argument.
{ewc mvimg, mvimage,!tip.bmp}

Important You must make sure that a DLL can accept the data type of an argument. If you pass an incorrect data type, the DLL may cause a general protection fault.

After declaring a DLL procedure, you then call the procedure in Visual Basic. When calling a DLL procedure, there are several issues to consider about how arguments are passed. These include:

® Passing a null value in a DLL procedure.

® Choosing whether to pass arguments by their value or by a reference to that value.

® Ensuring that Visual Basic string arguments are large enough to accommodate C strings.

This section covers some of the differences in the way that Visual Basic and Windows API function procedures treat values passed to arguments.

This section includes the following topics:

® Passing Null Values

® Passing Arguments by Value or by Reference

® Passing Strings

This section elaborates on the use of DLLs, providing information that goes beyond the basic ways of declaring and calling DLL procedures.

It includes a description of the DLL procedures in the Windows API that are often used to extend Visual Basic applications. It also lists some commonly used Windows DLL procedures, and includes an example of one of these procedures. Becoming familiar with these DLL procedures will be useful as you create your own Visual Basic applications.

This section includes the following topics:

® Providing Callback Procedures

® Creating a Wrapper Procedure

® Useful DLLs

Visual Basic provides an add-in called the Class Builder that automates the process of building classes. With the Class Builder, you can visually define classes and their interfaces, as shown in this illustration.

The following illustration shows the **Class Builder** dialog box.

{ewc mvimg, mvimage,!v07g005.bmp}

u To add and use the Class Builder

1. On the **Add-Ins** menu, click **Add-In Manager**.
2. In the **Available Add-Ins** list, select **Visual Basic Class Builder Utility**, and then click OK.
3. On the **Add-Ins** menu, click **Class Builder Utility**.
4. Add a new class to the project, and then add methods, properties, and events to the class.

Component Code

```
Dim localSalary As Single

Public Property Let Salary(ByVal newSalary As Single)
    If newSalary < 0 Then
        Err.Raise 12345 + vbObjectError, , "Salary must be positive"
    Else
        localSalary = newSalary
    End If
End Property

Public Property Get Salary() As Single
    Salary = localSalary
End Property
```

Client Code

```
Private Sub Command1_Click()
    Dim x As Company.Employee
    On Error GoTo HandleError
    Set x = New Company.Employee
    x.Salary = -50
    MsgBox x.Salary
    Exit Sub
HandleError:
    If Err.Number = 12345 + vbObjectError Then
        MsgBox "Salary must be positive."
    End If
End Sub
```

Windows provides DLLs that you can call from Visual Basic. These DLLs create the Windows application programming interface (Windows API). You can purchase or write your own DLLs.

Windows API Libraries

The 32-bit versions of the Windows operating system (Windows 95 and Windows NT) consist of three primary DLLs: User32, GDI32, and Kernel32. These DLLs provide most of the Windows API functionality.

® User32

User32.dll handles the tasks related to managing windows, menus, controls, and dialog boxes. For example, you can call the **FlashWindow** procedure in User32.dll to cause a window to flash.

® GDI32

GDI32.dll is responsible for graphics output.

® Kernel32

Kernel32.dll handles operating system tasks. For example, you can call the **GetWindowsDirectory** procedure in Kernel32.dll to obtain the current path of the Windows folder.

In addition to these three DLLs, Windows 95 and Windows NT contain many other DLLs that provide extended functionality to Visual Basic.

Visual Basic passes arguments to procedures by reference, using the 32-bit address of the data value, or by value, using the actual value of the argument variable.

By default, Visual Basic passes arguments by reference. However, many DLL procedures expect arguments to be passed by value. If you pass an argument incorrectly (for example, if you pass a value to a procedure that expects an address), it could cause a general protection fault.

To see an animation of the difference between passing arguments by value and by reference, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

Passing Arguments by Value

When you pass an argument by value, a copy of the original data value is passed to the procedure. If the procedure changes the value, the original variable will not be affected.

To pass an argument by value, place the **ByVal** keyword before the argument declaration in the **Declare** statement, as shown in the following code.

```
Declare Function FlashWindow Lib "User32" _
    (ByVal hwnd as Long, _
    ByVal bInvert As Long) As Long
```

Note You can specify **ByVal** in the declaration or in the procedure. When used with strings, **ByVal** has a different purpose.

Passing Arguments by Reference

When you pass an argument by reference, the memory address of the original variable is passed to the procedure. If the procedure changes the value of this variable, the original value will also change.

You can specify **ByVal** or **ByRef** in the procedure declaration. If you do not specify either keyword, Visual Basic automatically passes the arguments by reference. Although you do not need to include **ByRef** in the declaration, your code will be more readable if you explicitly use either **ByRef** or **ByVal**.

```
Public Property Get Active() As Boolean
    Active = Timer1.Enabled
End Property

Public Property Let Active(ByVal New_Active As Boolean)
    Timer1.Enabled = New_Active
    PropertyChanged "Active"
End Property

Public Property Get Font() As Font
    Set Font = txtStockPrice.Font
End Property

Public Property Let Font(ByVal New_Font As Font)
    Dim objCtl As Control
    'loop through all controls
    For Each objCtl In Controls
        'if control is a label or textbox
        If (TypeOf objCtl Is Label) Or _
                (TypeOf objCtl Is TextBox) Then
            Set objCtl.Font = New_Font
        End If
    Next
    PropertyChanged "Font"
End Property

Public Sub Refresh()
    Timer1_Timer
End Sub
```

After the connection between the client and server has been established, the data can be passed.

Sending Data

To send data to a remote computer, you use the **SendData** method of the **Winsock** control. You can send either string or binary data.

The following code sends a string to a remote computer.

```
Private Sub cmdSend_Click()
    ' Send the data contained in a text box.
    Winsock1.SendData txtSendData.Text
End Sub
```

To send binary data, replace the parameter to the **SendData** method with a bit array.

Retrieving Data

To retrieve data from a remote computer, you use the **GetData** method of the **Winsock** control. In the event handler, you use the **GetData** method to retrieve the data that has been cached by the control.

The syntax for the **GetData** method is:

object.**GetData** *data*, [*type*,] [*maxLen*]

The *data* is of type **Variant**, but by providing the *type* parameter, you can specify the format in which you want to retrieve data. The *maxLen* parameter specifies the maximum amount of data that can be held by the *data* variable.

The following code uses the DataArrival event to retrieve data.

```
Private Sub Winsock1_DataArrival (ByVal bytesTotal As Long)
    Dim strData As String
    Winsock1.GetData strData, vbString
    ' Add the retreived data to a textbox.
    txtDisplay = txtDisplay & strData
End Sub
```

The DataArrival event also indicates the number of bytes that will be retrieved.

The best way to select all of the controls is to use the **Pointer** button and create a selection rectangle with the mouse.

Coding a synchronous file download consists of setting the access type and invoking the **OpenURL** method.

The **OpenURL** method uses the following syntax:

object.**OpenURL** *strURL*[, *datatype*]

The *strURL* parameter is any valid FTP or HTTP URL. If a file name is specified, the file will be downloaded. If a directory name is specified, a directory listing will be downloaded.

The *datatype* parameter specifies the type of data to return, either string (**icString**) or binary (**icByteArray**).

The following code uses the **OpenURL** method to download a directory listing of the Microsoft FTP Internet server to a text box.

```
Dim s As String
s = Inet1.OpenURL("ftp://ftp.microsoft.com", icString)
txtListing.Text = s
```

For information about setting the **AccessType** property, see Coding for Asynchronous Operation.

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Active", Timer1.Enabled, True
    PropBag.WriteProperty "Font", txtStockPrice.Font
End Sub

Public Property Let Active(ByVal New_Active As Boolean)
    Timer1.Enabled = New_Active
    PropertyChanged "Active"
End Property

Public Property Let Font(ByVal New_Font As Font)
    Dim objCtl As Control
    'loop through all controls
    For Each objCtl In Controls
        'if control is a label or textbox
        If (TypeOf objCtl Is Label) Or _
                (TypeOf objCtl Is TextBox) Then
            Set objCtl.Font = New_Font
        End If
    Next
    PropertyChanged "Font"
End Property

Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Active = PropBag.ReadProperty("Active", True)
    Font = PropBag.ReadProperty("Font")
End Sub
```

```
Public Event TickerKepPress(KeyAscii As Integer)

Private Sub txtStockTicker_KeyPress(KeyAscii As Integer)
    RaiseEvent TickerKepPress(KeyAscii)
End Sub
```

```
<HTML>

<!--If any of the controls on this page require licensing, you must
    create a license package file.  Run LPK_TOOL.EXE to create the
    required LPK file.  LPK_TOOL.EXE can be found on the ActiveX SDK,
    http://www.microsoft.com/intdev/sdk/sdk.htm.  If you have the Visual
    Basic 5.0 CD, it can also be found in the \Tools\LPK_TOOL directory.

    The following is an example of the Object tag:

<OBJECT CLASSID="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
    <PARAM NAME="LPKPath" VALUE="LPKfilename.LPK">
</OBJECT>
-->

<OBJECT ID="StockPrice" WIDTH=226 HEIGHT=76
    classid="clsid:6A7A58F2-3DFC-11D0-A520-0080C776418A"
    codebase="Stock.CAB#version=1,0,0,0">
    <PARAM NAME="Active" VALUE="0">
</OBJECT>

</HTML>
```

```
<HTML>

<SCRIPT LANGUAGE="VBScript">
<!--
Sub cmdRefresh_OnClick()
    StockPrice.Refresh
End Sub

Sub StockPrice_PriceUpdated()
    txtLastUpdate.Value = "Last Updated: " & Now
end sub
-->
</SCRIPT>

<!--If any of the controls on this page require licensing, you must
    create a license package file.  Run LPK_TOOL.EXE to create the
    required LPK file.  LPK_TOOL.EXE can be found on the ActiveX SDK,
    http://www.microsoft.com/intdev/sdk/sdk.htm.  If you have the Visual
    Basic 5.0 CD, it can also be found in the \Tools\LPK_TOOL directory.

    The following is an example of the Object tag:

<OBJECT CLASSID="clsid:5220cb21-c88d-11cf-b347-00aa00a28331">
    <PARAM NAME="LPKPath" VALUE="LPKfilename.LPK">
</OBJECT>
-->


<OBJECT ID="StockPrice" WIDTH=226 HEIGHT=76
    classid="clsid:6A7A58F2-3DFC-11D0-A520-0080C776418A"
    codebase="Stock.CAB#version=1,0,0,0">
    <PARAM NAME="Active" VALUE="0">
</OBJECT>

<INPUT TYPE=button VALUE="Refresh" NAME="cmdRefresh">
<INPUT TYPE=txt NAME="txtLastUpdate" SIZE="40">

</HTML>
```

Optimizing an application means making sure that it is as efficient as possible. Visual Basic includes a number of tools that you can use to determine how efficient your code is.

For example, you can optimize your application so that your users can save their current preferences and working environment settings. In subsequent sessions, users will not have to reconfigure their computers at startup.

This section describes how to use the Code Profiler add-in to optimize your application. It also discusses a number of coding techniques that you can use to enhance the performance of your application.

This section includes the following topics:

® The Visual Basic Code Profiler

® Saving Application Settings

® Tuning an Application

® Logging Events

Microsoft Visual Basic offers you two tools that can significantly increase your productivity. You can use Visual SourceSafe to manage changes to source code file.

You can also use resource files as repositories of data (such as strings) that you know will be changed when you localize your application.

This section includes the following topics:

® Using Visual SourceSafe

® Using Resource Files

You optimize your application throughout the development process. A poorly designed application will run slowly, no matter what you do to try to improve performance.

Optimization is a tradeoff. If you improve one area of performance, you can cause poorer performance in another area. For example, if you keep forms loaded but hidden, they will be displayed quickly, but they will consume memory while they are loaded.

If your user's computer does not have enough memory, your application may run more slowly. Determine the factors that are most important to your user, and develop your application accordingly.

To improve the speed of your application, consider the following techniques.

Optimizing Actual Speed

These suggestions describe how to optimize the speed of your application.

® Some data types are faster at calculation speed than others. Use the simplest data type possible.

® Accessing a variable is faster than accessing a property. If you are accessing a property in a loop, set the property to a variable first, and then use the variable in the loop.

® Consider using a DLL to improve performance.

Optimizing Apparent Speed

Here are some techniques to make your application appear faster to the user.

® Use progress indicators to inform the user of the status of a task.

® If you use a timer to run a long process in the background, the user can continue to work while the process runs. You may want to enable a command that indicates when the process is complete.

Optimizing Display Speed

To make display operations run faster, use these techniques.

® Use the form's **AutoRedraw** property appropriately.

If your application generates complex graphics that change frequently, you will get better performance if you set **AutoRedraw** to **False**, and handle repainting graphics yourself.

® Use an **Image** control instead of a picture box.

An **Image** control requires less overhead than a picture box. If you only need to display a picture and respond to a Click event, use an **Image** control.

® Use a picture box to simulate a group of controls.

Consider using a picture to simulate a complex set of controls, rather than using many individual controls.

® If a form is used frequently, keep the form hidden rather than unloaded. A hidden form displays quicker than an unloaded form.

Reducing Memory Size

To reclaim memory from an application, consider the following suggestions.

® When you no longer need the data in a long string, set the string to " " to reclaim the memory used by the string. When you no longer need the data in a dynamic array, use **Erase** or **ReDim Preserve** to discard unneeded data, and reclaim the memory used by the array. **Erase** eliminates an array; **ReDim Preserve** can be used to make the array smaller.

® When you no longer need the picture displayed in a picture object, use the **LoadPicture** function with an empty string parameter to clear the object.

The ability to write key information to an event log is new in Microsoft Visual Basic version 5.0.

Event-logging methods and properties are part of the **App** object, which is a global object accessed with the **App** keyword. The **App** object specifies information about many aspects of an application, including the title and version information.

The **StartLogging** and **LogEvent** methods, and the **LogMode** property, are all part of the **App** object that is used to control event logging.

You generally begin with the **StartLogging** method to set the log mode and log target of an operation.

You use the following syntax for the **StartLogging** method.

App.StartLogging *logTarget*, *logMode*

The settings for *logMode* are listed in the following table.

logMode	**Description**
vbLogAuto	Specifies where messages will be logged to. If running Windows 95, messages will be logged to the location specified by the **LogFile** property. If running Windows NT, messages will be logged to the NT application event log.
vbLogOff	Turns off logging.
vbLogToFile	Forces logging to a file.
vbLogToNT	Forces logging to the NT event log.
vbLogOverwrite	Specifies that the log file should be restarted each time the application starts. This value can be manipulated programmatically with any of the other *LogMode* constants.
vbLogThreadID	Indicates that the current thread ID will be added to the beginning of the message. This value can be manipulated programmatically with any of the other *logMode* constants.

You use the **LogEvent** method to write an event to the application's log target. On Windows NT platforms, the method writes to the NT event log. On Windows 95 platforms, the method writes to the file specified in the **StartLogging** method. If no file is specified, events are written to the file VBEvents.log.

You use the following syntax for the **LogEvent** method.

App.LogEvent (*logBuffer*, *eventType*)

The *logBuffer* argument is the text to be written to the log. The *eventType* argument specifies the type or severity of data being written: error, warning, or information data.

The **LogMode** property is read-only and returns a value that determines how logging (through the **LogEvent** method) will be carried out.

You use this syntax for the **LogMode** property.

mode = **App.LogMode**

The **LogPath** property is read-only, and returns the path to the current log file. You use this syntax for the **LogPath** property.

logFile = **App.LogPath**

The following code shows how to log events in Visual Basic.

```
Select Case Index
  Case 0    ' Start Logging.
    Dim logMode As Long

    ' Compute the logMode value.
```

```
    logMode = cboLogMode.ListIndex + _
      &H10 * chkLogOverwrite.Value + _
      &H10 * chkLogThreadID.Value
    ' Set the logging parameters.
    App.StartLogging txtLogTarget, logMode
    cmdAction(1).Enabled = True
    cmdAction(1).Default = True
  Case 1    ' Log an Event.
    App.LogEvent txtLogText, _
      cboEventType.ItemData(cboEventType.ListIndex)
End Select
```

To see a demonstration of how to create an event log, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

To display the source code for an HTML page in Internet Explorer, click **Source** on the **View** menu. The script code is contained in the HTML <SCRIPT> tag.

The LANGUAGE parameter tells the browser which interpreter to use when running the code. For Visual Basic Scripting Edition, the value of the LANGUAGE parameter is VBScript.

The following VBScript code will run when a user clicks the Hello button on a Web page.

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnHello_OnClick()
      MsgBox "Hello, world!"
End Sub
-->
</SCRIPT>
```

The VBScript code is enclosed in HTML comment tags (**<!**-- and **-->**). This prevents browsers that do not recognize the <SCRIPT> tag from using the code.

In this exercise, you will create a client application that uses the functionality of a version of the Credit Card component created in later chapters. This component has been created for you in the folder <*Install Folder*>\Labs\Lab06. This exercise adds an interface to that component.

This version of the Credit Card component exposes two interfaces that provide the following properties and methods:

1. The default interface validates the purchase of an item with these properties and methods.

Name	Type
CardNumber	Property
ExpireDate	Property
PurchaseAmount	Property
Approve	Method

2. The **IManage** interface, which uses the following property, is used to change the credit limit of a client.

Name	Type
CreditLimit	Property

This interface would require some level of security to make use of its functionality.

Register the component

1. Locate the file cc.dll, which contains the credit card component.
2. Use RegSvr32.exe to register the component with your system.

Create a project

1. Start a new standard project in Visual Basic.
2. Modify the default form to look like the following:

   {ewc mvimg, mvimage,!v06g055.bmp}
3. Save the form as frmClient in the Lab06 folder.
4. Save the project as CCClient in the Lab06 folder.

Use the default Automation interface

1. Add a reference to the CreditCard component.
2. Dimension a form-level variable named **cc** as type Lab.CreditCard, as shown in the following code:

```
Public WithEvents cc as Lab.CreditCard
```

   Make sure it can receive events.
3. In the Form_Load handler, create a new instance of the CreditCard component, as shown in this code:

```
Set cc = New Lab.CreditCard
```

4. In the Click event procedure for the **Approve** command button, add the following code:

```
cc.ExpireDate = "1/1/99"
cc.PurchaseAmount = 50
cc.CardNumber = 1234
MsgBox cc.Approve
```

5. Press F8 to step through the code.

   When you set the **PurchaseAmount** property and call the **Approve** method, you step into the CreditCard project.
6. Try changing the value of **PurchaseAmount** to 2000. Does the **Approve** method return a value of **False**?

u Use the IManage interface

1. Add a second form, frmManage, that resembles the following:

   {ewc mvimg, mvimage,!v06g060.bmp}

2. Add a form-level variable of type **IManage**., as shown in this code.

```
Private mng As IManage
```

3. In the Form_Load handler, use the CreditCard object variable in frmClient to get a pointer to the **IManage** interface.

```
On Error Resume Next
Set mng = frmClient.cc
If mng Is Nothing Then
    MsgBox "The IManage interface is not supported on this object."
    Unload Me
    Exit Sub
End If
lblOldLimit = mng.CreditLimit
txtNewLimit = ""
```

4. In the Click event handler of the OK command button, use the **CreditLimit** property of the **IManage** interface to reset the credit limit based on the value provided by the user in the New Limit text box and then unload the form. This code is shown as follows:

```
If IsNumeric(txtNewLimit) Then
    mng.CreditLimit = txtNewLimit
    Unload Me
    Exit Sub
Else
    txtNewLimit.SetFocus
End If
```

5. In the Click event handler for the **Cancel** command button, unload the form.
6. In the GetFocus event for the **New Limit** text box, add the code to highlight the existing text.

```
Private Sub txtNewLimit_GotFocus()
    txtNewLimit.SelStart = 0
    txtNewLimit.SelLength = Len(txtNewLimit)
End Sub
```

7. Switch to frmClient, and add a handler for the **Manage** command button Click event that shows the frmManage form as a modal window.
8. Save and test the application.
9. Validate the **IManage** interface by setting a breakpoint on frmClient when showing the frmManage form and stepping through the code.

u Unregister the CreditCard component

Unregister this version of the CreditCard component to remove the references stored in the registry. This does not modify the DLL itself in any way.

1. Run RegSvr32.exe against the component cc.dll once again, but add the **/u** parameter, as shown in this code:

```
Regsvr32 /u cc.dll
```

Set the **RowSource** property to datEmployees.

Set the **ListField** property to LastName.

Set the **DataSource** property to datPrimaryRS (the data control created by the Data Form Wizard).

Set the **DataField** property to EmployeeID.

Set the **BoundColumn** property to EmployeeID.

A Web page can be either static or active. Static and active Web pages differ in the level of user interaction they provide.

Static Web pages do not contain any interactive elements other than hyperlinks. Active Web pages use ActiveX controls and script code to support interactive elements, such as forms, dialog boxes, and controls, and a custom user interface.

To see an animation of the difference between a static and an active Web page, click this icon.
{ewc mvimg, mvimage,!anim.bmp}

To review a sample application that uses resource files, open Atm.vbp in the Visual Basic folder \Samples\Resource.

This section introduces you to some of the fundamental concepts of Visual Basic development. It also describes the editions of Visual Basic that are available to you.

Finally, the section provides a brief summary of Help resources for Visual Basic.

This section includes the following topics:

® Understanding Event-Driven Programming

® Creating a Simple Visual Basic Application

® Editions of Visual Basic

® Getting Assistance

Visual Basic is derived from the Basic language, which is a structured programming language. However, Visual Basic uses an event-driven programming model.

This topic presents the event-driven programming model, and contrasts it with a procedural programming model.

Procedural Applications

In traditional or procedural applications, the application controls which portions of code run, and the sequence in which they run. Application execution starts with the first line of code, and follows a predefined path through the application, calling procedures as needed.

Event-Driven Applications

In an event-driven application, execution does not follow a predetermined path. Instead, it runs different code sections in response to events. Events can be triggered by the user's actions, by messages from the system or other applications, or from the application. The sequence of events determines the sequence in which the code runs. Therefore, the path through the application's code differs each time the program runs.

An essential part of event-driven programming is to write code that responds to the possible events that may occur in an application. Visual Basic makes it easy to implement an event-driven programming model.

The following illustration shows some actions that generate events to which you can respond by writing code. These events can occur in any order.

{ewc MVIMG, MVIMAGE,!v01g015.bmp}

You use the following two basic steps to create an application in Visual Basic.

1. Create the user interface of the application.
2. Write code that responds to actions taken in the user interface.

To see a demonstration of how to create a simple application in Visual Basic, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

A new picture and string are loaded.

The advantage of using multiple workspaces is that each workspace defines a session. Each session can establish a security level, and can manage transactions independent of other sessions. If you do not want multiple security levels or transaction scopes, you do not need to reference the **DBEngine** or **Workspace** objects. Visual Basic will use the default object, **DBEngine.Workspaces(0)**.

If your network supports it, you can use a network path for the database name, for example: \\myserver\myshare\mydb.mdb.

If you declare the function in a module, you can use the Object Browser to paste the call for the procedure into your application.

You can use the **Alias** clause to declare a DLL twice and provide data types in each declare, thereby avoiding the **As Any** syntax. Using explicit data types ensures that if you pass an incorrect data type, you receive a compile-time error, rather than a run-time error.

If you have the Professional Edition or Enterprise Edition of Visual Basic, you can compile your code either in standard Visual Basic pseudocode (p-code) format or in native code format. Native code compilation provides several options for optimizing and debugging that are not available with p-code.

P-code is an intermediate step between the high-level instructions of a Visual Basic application and the low-level native code that your computer's processor executes. At run time, Visual Basic translates each p-code statement to native code. By compiling directly to native code format, you eliminate the intermediate p-code step.

To compile a project to native code

1. In the Project window, select the project you want to compile.
2. On the **Project** menu, click **Project Properties**.
3. In the **Project Properties** dialog box, click the **Compile** tab.
4. Select the compile options you want to use, and then click OK.

   You can also click the **Advanced Optimizations** button for additional native code options.

The following illustration shows the native code options.

{ewc MVIMG, MVIMAGE,!v01g030.bmp}

For information about native code options, click the **Help** button in either the **Project Properties** or **Advanced Optimizations** dialog box.

```
Declare Function SetWindowPos Lib "User32" _
            (ByVal hWnd As Long, _
            ByVal hWndInsertAfter As Long, _
            ByVal X As Long, _
            ByVal Y As Long, _
            ByVal cx As Long, _
            ByVal cy As Long, _
            ByVal wFlags As Long) As Long

Const HWND_TOPMOST = -1
Const HWND_NOTTOPMOST = -2
Const SWP_NOSIZE = 1
Const SWP_NOMOVE = 2
Dim flags As Integer

Sub cmdSetTopMost ()
    flags = SWP_NOSIZE Or SWP_NOMOVE
    lResult = SetWindowPos frmDLLDemo.hWnd, _
                HWND_TOPMOST, 0,0,0,0, flags
End Sub

Sub cmdResetTomost ()
    flags = SWP_NOSIZE Or SWP_NOMOVE
    lResult = SetWindowPos frmDLLDemo.hWnd, _
                HWND_NOTOPMOST, 0,0,0,0, flags
End Sub
```

Place a hidden **Data** control on the form and set the **DataSource** property of the **DBGrid** control to the **Data** control. Then, you can set the **Recordset** property of the **Data** control equal to the recordset returned by the saved query.

Click here to connect to the Microsoft Visual Basic Product Documentation page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

Microsoft Visual Basic version 4.0 is the programming system for Windows that grows with your needs and experience. Find out how to create everything from simple programs to advanced, enterprise-wide client/server applications, and take advantage of the latest three-tier capabilities:

® Demonstrations

® Features and evaluation information

® How to buy

Click here to connect to the Microsoft Visual Basic Technical Information page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This section contains technical papers written by Microsoft and third-party developers. These papers provide in-depth explanations of Visual Basic programming techniques and strategies.

Click here to connect to the Microsoft Visual Basic Technical Support page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This site provides pointers to the following pages:

® Support Options

® Frequently Asked Questions

® Knowledge Base

® Feature Articles

® Troubleshooter

® Download Options

® Newsgroups

Click here to connect with the Microsoft Visual Basic Events page on the Microsoft Web page:
{ewc mvimg, mvimage,!intjump.bmp}

Look here for information about the conferences and trade shows that feature Visual Basic and related technologies.

Click here to connect to the Microsoft Visual Basic Training page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This site includes information on the Microsoft Online Institute (MOLI), the Mastering Microsoft Visual Basic datasheet, and details on how to become a Microsoft Certified Solution Developer (MCSD).

Check out the Microsoft Technical Training and Education section for more information.

Click here to connect to the Microsoft Visual Basic Feature Area page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

This page highlights specific technologies and concepts important to Visual Basic developers, and provides a listing of "Features-to-date."

Click here to connect to the Microsoft Visual Basic Tip of the Week page on the Microsoft Web page:
{ewc mvimg, mvimage,!intjump.bmp}

This page lists all the "Tip of the Week" tips that have appeared on the Visual Basic home page.

The Visual Basic Tip of the Week is provided by The Cobb Group, publishers of *Inside Visual Basic*, a monthly publication for Visual Basic users. Free trial subscriptions are available. Find out how to receive the Tip of the Week in e-mail!

Click here to connect to the Microsoft Visual Basic Free Downloads page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

Look here for the latest software downloads including (but not restricted to) beta releases, product updates, help files, utilities, and much more.

Click here to connect to the Microsoft Visual Basic Case Studies page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

Visit this Web site to find how Microsoft products and technologies play a key role in the development of real-world business solutions.

Click here to connect to the Microsoft Visual Basic What's New page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

An easy way to find out what's been added, changed, or updated to the Microsoft Visual Basic site.

Click here to connect to the Microsoft Visual Basic Community Resource page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

Visit this site for a compilation of Visual Basic links, training resources, recommended books and more.

After you have declared a DLL procedure, you can use the Object Browser to paste the correct syntax for the call to that procedure, complete with named arguments. Press F2 to display the Object Browser.

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii <> 8 Then 'Allow BACKSPACE through
        'Only digits are valid characters.
        If Chr(KeyAscii) < "0" Or Chr(KeyAscii) > "9" Then
            KeyAscii = 0'Set character to null if out of range
            Beep
        End If
    End If
End Sub
```

```
Private Sub cmdCreateXLObject_Click()
    set xl = CreateObject("Excel.Application")
    x1.Workbooks.Open App.Path & "\Earn.xls"
    xl.Visible = True
End Sub
```

```
Private Sub cmdEstimateEarnings_Click()
    xl.Worksheets("Earnings").Activate
    xl.Worksheets("Earnings").Range("Growth").Value = txtGrowth.Text
    xl.Worksheets("Earnings").Range("Inflation").Value = _
        txtInflation.Text
End Sub
```

```
Private Sub cmdChartEarnings_Click()
    If xlChart Is Nothing Then
        xl.Worksheets("Earnings").Range("net_profit").Select
        set xlChart = xl.Charts.Add()
        xlChart.Type = xl3DColumn
    Else
        xlChart.Activate
    End If
End Sub
```

The error should display as "3260: Couldn't update; currently locked by user 'name' on machine 'name'."

The error should display as "3197: The Microsoft Jet database engine stopped the process because you and another user are attempting to change the same data at the same time."

Click here to connect to the Microsoft Visual Basic Links page on the Microsoft Web site:
{ewc mvimg, mvimage,!intjump.bmp}

What are people saying about Microsoft Visual Basic? How are other developers using it to create exciting applications and tools? Where are the cool Visual Basic Web pages? Find the answers to these questions here.

In general, it is more efficient to use a stored query than to create an SQL statement at run time.

The scope and visibility of a variable determines where and when a variable is recognized. Where the variable is declared (in a procedure, form, or module) and how the variable is declared (as **Public** or **Private**) determines the scope of the variable.

Procedure-Level Variables

A variable declared in a procedure is recognized only within the procedure in which the variable is declared. The following code dimensions a variable as an integer within a procedure.

```
Dim iTest As Integer
```

Form-Level Variables

Variables declared within the General Declarations section of a form can be **Private** or **Public**.

Private variables are available to all procedures within the form, but are not visible to any procedures outside the form.

Public variables are available to an entire application as a property of the form. When accessing a **Public** form-level variable from outside the form, you must specify the form name. For example:

```
Public fTotal As Integer 'Declared in form.
frm.fTotal = 4 'Used outside form.
```

Standard Module-Level Variables

Variables declared within the General Declarations section of a module can be **Private** or **Public**.

Private variables are available only to the module.

Public variables are available to the entire application. You can access a module-level public variable by specifying only the variable name. For example:

```
'Declare variable in module.
Public giTest As Integer

'Use variable outside of module.
giTest = 5

'You can explicitly state module if
'the same variable name is used in several modules.
Module1.giTest = 5
```

```
Private Sub Form_Load()
    Me.KeyPreview = True
    cmdOK.Enabled = False
End Sub

Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
    'Enable OK button only if all text boxes have entries
    Dim curControl As Control
    For Each curControl In Controls
    If TypeOf curControl Is TextBox Then
        If curControl.Text = "" Then    'text box is empty
            cmdOK.Enabled = False   'disable OK button
            Exit Sub
        End If
        'Could check other types of controls if needed
    End If
    Next
    'all TextBoxes have information in them
    cmdOK.Enabled = True
End Sub
```

```
Private Sub cmdOK_Click()
    If Not (IsNumeric(txtDeptNo.Text)) Then
        MsgBox "Invalid dept. number."
        txtDeptNo.SetFocus
    ElseIf Not (IsNumeric(txtProjNo.Text)) Then
        MsgBox "Invalid project number."
        txtProjNo.SetFocus
    Else
        MsgBox "Data is ok."
        Unload Me
    End If
End Sub
```

When you build a component and provide classes for use by clients, the compiler in Visual Basic automatically generates a type library. Client developers can use the Object Browser to see what functionality the component offers, and can connect to your component by using its type library and the **References** dialog box.

Advantages of Setting a Reference to a Type Library

Setting a reference to a type library for a component offers three advantages to client developers:

® Early binding

Setting a reference to a type library during design time enables early binding. Early binding enables client developers to find out about the methods and properties available from a component at compile time, rather than at run time.

When early binding is used, calls to a component are very efficient because they can use **vtable** binding.

® Syntax checking

The syntax of calls to a component is checked during development.

® Access to Help

Client developers have access to any context-sensitive Help that you have provided.

For information about the implications of using early binding and **vtable** binding, see Introduction to Binding in Chapter 6: Creating ActiveX Clients.

Having evolved from OLE document objects, ActiveX documents have significantly enhanced the capabilities of visual editing. Visual editing is the process of editing an embedded object within the window of its container by using tools from its server. Instead of activating just one embedded object, an ActiveX document can represent an entire document within an application.

An example of this enhanced capability is a comparison between a Word object embedded in another application such as Microsoft Excel, and an ActiveX document object created with Word that opens in Internet Explorer. With the ActiveX document, you have the ability to view and print headers and footers. With the embedded Word object, you do not have this capability. This is because ActiveX documents implement additional interfaces that extend the functionality of embedded objects, and give more server functionality to the client application. In this case, Word acts as the server application to provide an ActiveX document to the client application (Internet Explorer).

To see an illustration that contrasts an embedded object with an ActiveX document, click this icon.
{ewc mvimg, mvimage,!illust.bmp}

When you create an ActiveX document as either an in-process (.dll) or out-of-process (.exe) component, the component functions as an Automation server. An Automation server enables other ActiveX container application (such as Internet Explorer) to host and activate the document.

When you compile or run a project, Visual Basic creates a document (.vbd) file. The .vbd file is an OLE structured storage, which means that data in the file can be accessed and manipulated through standard OLE interfaces. For information on interfaces, see Using Interfaces in Chapter 6: Creating ActiveX Clients.

In a browser such as Internet Explorer, users must navigate to the .vbd file to open the ActiveX document.

The following illustration shows files created when you compile an ActiveX EXE project.

{ewc mvimg, mvimage,!v10g010.bmp}

Compiling an ActiveX DLL project would create the Project1.dll file.

Before you can use an ActiveX document, you must place it within a container. Connecting an ActiveX document to its container is known as siting, which plays a key role in determining ActiveX document event behavior. Siting enables an ActiveX document to implement its functionality.

A document container possesses a collection of sites that it uses to assign ActiveX documents a site in the container at run time. A site refers to the window in the container in which a document is placed.

When a document is sited, the container becomes the client for the document and establishes a connection between the container, the site, and the document, according to an internal implementation. The **Parent** property of the ActiveX document then becomes available and returns a reference to the container.

After siting occurs, the properties of the container and the Visual Basic **Hyperlink** object become available to the ActiveX document. The properties and methods of the **Hyperlink** object of an ActiveX document can request a hyperlink-aware container, such as Internet Explorer, to jump to a given URL.

To determine the container of an ActiveX document, you use the **TypeName** statement with the **Parent** property of the **UserDocument** object, as shown in the following code:

```
Dim strContainer As String
    strContainer = TypeName(UserDocument.Parent)
```

In Internet Explorer, the string value returned by the **Parent** property is IWebBrowserApp.

The Show event occurs when an ActiveX document is sited on the container, so you can use the event to determine the container being used. When a document is sited, the container properties become available.

The following code checks to see whether IWebBrowserApp is the string value returned by the **Parent** property of the **UserDocument** object, and displays a message to the user:

```
Private Sub UserDocument_Show()
    Dim strContainer As String
    strContainer = TypeName(UserDocument.Parent)
    If strContainer = "IWebBrowserApp" Then
        MsgBox "Confirmation: this document  " & _
        "works with Internet Explorer " & _
        "3.0 or later."
    Else
        MsgBox "Sorry, please open this & _
        "document with Internet Explorer & _
        "3.0 or later."
    End If
End Sub
```

Note As with any component created with Visual Basic 5.0, ActiveX documents can send events to a client, providing the component with the ability to make asynchronous calls to the client. ActiveX documents can also use events and callbacks to call a client asynchronously. For information about server notification, see Receiving Notifications from Servers in Chapter 6: Creating ActiveX Clients. For information about implementing events in servers, see Using Events in Chapter 7: Creating ActiveX Code Components.

The Data control has the **Recordset** property.

You can use the **Alias** clause to declare a DLL twice, and provide explicit data types in each declaration to avoid the syntax. Using explicit data types ensures that if you pass an incorrect data type, you receive a compile-time error, rather than a run-time error.

1. Which characteristic of ActiveX controls on a Web page contributes to security?

{ewc mvimg, mvimage,!answer.bmp}
A. ActiveX controls execute on the Web server rather than the user's computer.

{ewc mvimg, mvimage,!answer.bmp}
B. ActiveX controls must have a continuous connection to the Web server for execution.

{ewc mvimg, mvimage,!answer.bmp}
C. Licensing makes it impossible for users to access source code downloaded from the Web server.

{ewc mvimg, mvimage,!answer.bmp}
D. Only the compiled ActiveX control is downloaded from the Web server and users never see the source code.

2. What is the purpose of an <OBJECT> tag on an HTML page?

{ewc mvimg, mvimage,!answer.bmp}
A. The <OBJECT> tag combines an ActiveX control and associated files into a .cab file.

{ewc mvimg, mvimage,!answer.bmp}

B. The <OBJECT> tag holds the source code for an ActiveX control on an HTML page.

{ewc mvimg, mvimage,!answer.bmp}

C. The <OBJECT> tag specifies an ActiveX control to be downloaded by the browser.

{ewc mvimg, mvimage,!answer.bmp}

D. The <OBJECT> tag is used to interpret the source code for an ActiveX control when an instance of the control is requested by the browser.

3. What is the purpose of the ID parameter in an HTML <OBJECT> tag?

{ewc mvimg, mvimage,!answer.bmp}

A. Specifies the object name.

{ewc mvimg, mvimage,!answer.bmp}

B. Specifies the object's unique class identifier stored in the system registry.

{ewc

C. Specifies a URL that points to a file containing an implementation of an object.

{mvimg,mvimage,!answer.bmp}

{ewc mvimg,mvimage,!answer.bmp}

D. Specifies a URL that points to the name of the object.

4. How do you create a license package file containing the licensing information for all controls on a Web page?

{ewc mvimg,mvimage,!answer.bmp}

A. With the Visual Basic Application Setup Wizard.

{ewc mvimg,mvimage,!answer.bmp}

B. With a utility from the LPK_TOOL directory on the Visual Basic CD ROM.

{ewc mvimg,mvimage,!answer.bmp}

C. With the Visual Basic License Manager.

{ewc mvimg,mvimage,!answer.bmp}

D. With the CODEBASE parameter of an HTML page's <OBJECT> tag.

{ewc mvi

mg,
mvi
ma
ge,!
ans
wer
.bm
p}

5. What must you do to test a licensed control on your development computer?

{ew
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

A. Add the license entry to your system registry.

{ew
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

B. Remove the license entry from your system registry.

{ew
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

C. Remove the control's .cab file from your development computer.

{ew
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

D. Remove the page's .lpk file from your development computer.

6. How do you prevent browsers that don't understand VBScript from displaying the VBScript code?

{ew
c

A. Embed the VBScript code within HTML comment tags.

{mvimg.mvimage.!answer.bmp}

B. Embed the VBScript code within the HTML <SCRIPT> tag.

{ewc mvimg.mvimage.!answer.bmp}

C. Set the LANGUAGE parameter in the <SCRIPT> tag to “VBScript”.

{ewc mvimg.mvimage.!answer.bmp}

D. Use the HTML <PARAM> tag to set the **Show** property of VBScript to **False**.

{ewc mvimg.mvimage.!answer.bmp}

7. Examine the VBScript code sample shown below:

```
<SCRIPT>
<!--
Sub BtnHello_OnClick()
      MsgBox "Hello, world!"
End Sub
-->
</SCRIPT>
```

What correction would you make to this HTML page to make sure the code runs correctly?

{ewc mvimg.mvi

A. Change the </SCRIPT> tag to <SCRIPT>.

{ewc mvimg, mvimage,!answer.bmp}

B. Remove the comment tags (<!-- and -->).

{ewc mvimg, mvimage,!answer.bmp}

C. Change the fourth line to <PARAM NAME = MsgBox VALUE = “Hello, World”.

{ewc mvimg, mvimage,!answer.bmp}

D. Add the LANGUAGE parameter to the <SCRIPT> tag.

{ewc mvimg, mvimage,!answer.bmp}

8. You have written the following code to set the initial properties of a control on an HTML page:

```
<OBJECT classid="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2" id=lblActiveLbl>
<PARAM NAME="Length" VALUE="40" NAME="Width" VALUE="20">
</OBJECT>
```

What do you need to change?

{ewc mvimg, mvimage,!answer.bmp}

A. You should use the <SCRIPT> tag rather than the <OBJECT> tag.

{ew

B. You should use a separate <OBJECT> tag for the clsid and the id

{ewc mvimg, mvimage,!answer.bmp}

parameters.

{ewc mvimg, mvimage,!answer.bmp}

C. You should use a separate <PARAM> tag for each property to be set.

{ewc mvimg, mvimage,!answer.bmp}

D. You should remove the quotation marks from the property names and values.

9. How can you apply a digital signature to you project's .cab file?

{ewc mvimg, mvimage,!answer.bmp}

A. Use the **Safety** dialog box in the Visual Basic Application Setup Wizard.

{ewc mvimg, mvimage,!answer.bmp}

B. Use the SIGNCODE.EXE utility from the ActiveX SDK.

{ewc mvimg, mvimage,!answer.bmp}

C. Use LPK_TOOL.EXE from the Visual Basic CD-ROM.

{ewc mvi

mg,
mvi
ma
ge,!
ans
wer
.bm
p}

{ew
c
mvi
mg,
mvi
ma
ge,!
ans
wer
.bm
p}

D. Use the AUTHENTICODE utility obtained from your certificate authority.

A client and an out-of-process server exist in separate address spaces; communication is more efficient when the client and server share the same address space.

For more information, see Client and Server Communication.

Proxy code is inserted on the client to trap server calls and enable communication across process boundaries, but this is not as efficient as communication with an in-process server.

For more information, see Client and Server Communication.

When the server is in-process, communication is more efficient because the client and server share the same address space.

For more information, see Client and Server Communication.

Stub code is inserted when a COM-compliant server is compiled to enable communication with local and remote clients.

For more information, see Client and Server Communication.

```
Declare Function SetTimer Lib "user32" (ByVal hwnd As Long, ByVal _
    nIDEvent As Long, ByVal uElapse As Long, _
    ByVal lpTimerFunc As Long) As Long
Declare Function KillTimer Lib "user32" (ByVal hwnd As Long, ByVal _
    nIDEvent As Long) As Long

Private idTimer As Integer
Const cIncrement As Integer = 5

Private Sub cmdTimer_Click()
  ' The timer command button simply toggles between starting
  ' and ending the timer.
  If idTimer > 0 Then
    EndTimer
  Else
    StartTimer
  End If
End Sub

Public Sub UpdateProgressBar()
  ' This public function is called by the timer callback routine
  ' to increment the precentdone bar. If the value is equal to
  ' or exceeds 100, we terminate the timer.

  Dim PercentDone As Integer

  PercentDone = ProgressBar1.Value + cIncrement
  If PercentDone >= 100 Then
    ProgressBar1.Value = 100
    EndTimer
  Else
    ProgressBar1.Value = PercentDone
  End If
End Sub

Private Sub StartTimer()
  ' Start the timer. Note the use of the AddressOf operator
  ' to pass the address of the timer procedure to Windows. When
  ' the timer fires, the callback procedure will be called. The
  ' timer will continue to fire at 200 millisecond intervals
  ' until it is explicitly terminated with a call to KillTimer.
  ' Because we are using a callback, the first two parameters of
  ' SetTimer, the HWnd and event ID, are not used.
  idTimer = SetTimer(0, 0, 200, AddressOf TimerProc)
  ProgressBar1.Value = 0
  cmdTimer.Caption = "&Stop"
End Sub

Private Sub EndTimer()
  ' Kill the timer and reset to start again
  KillTimer 0, idTimer
  idTimer = 0
  cmdTimer.Caption = "&Start"
End Sub
```

You can jump directly to the lab exercises by clicking on the titles below.

Lab 1: Visual Basic Review

- Exercise 1: Creating Forms
- Exercise 2: Adding Code
- Exercise 3: Creating an Error-Handling Routine
- Exercise 4: Restricting Input
- Exercise 5: Creating an Executable File

You can view Labs from within the Chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 2: Using the Data Control

- Exercise 1: Creating a Data Entry Form
- Exercise 2: Using the DBCombo Control
- Exercise 3: Using the Validate Event

DAO version 3.5 introduces a new client/server connection technology named Open Database Connectivity Direct (ODBCDirect). This technology establishes a connection directly to an ODBC data source, without loading the Microsoft Jet database engine.

This section introduces the features of ODBCDirect. Topics include:

® Defining an ODBCDirect Workspace

® Connecting to a Remote Data Source

® Retrieving Remote Data

You can view labs from within the Chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 3: Using Data Access Objects

- Exercise 1: Creating and Navigating a Recordset
- Exercise 2: Adding and Editing Records
- Exercise 3: Finding Records
- Exercise 4: Using Queries
- Exercise 5: (Optional) Disabling Buttons During Edit
- Exercise 6: (Optional) Using a Parameter Query

The technology of ODBCDirect enables you to access ODBC data sources directly by using data access objects (DAO). You can use the features of this client/server connection to access a database quickly, without loading the Microsoft Jet database engine.

The first step in using ODBCDirect is to define the type of workspace you will use. You can do this in one of two ways: by setting the default workspace type, or by defining a specific workspace type as ODBCDirect.

Setting the Default Workspace

To set the default workspace type as ODBCDirect, you use the **DefaultType** property of the **DBEngine** object. This workspace type will then be used whenever a **Workspace** object is created.

To define a specific workspace type as ODBCDirect, you set the **DefaultType** property to **dbUseODBC**. This setting prevents the Microsoft Jet database engine from being loaded into memory, if you have not already created a Microsoft Jet database workspace.

The following code sets the default workspace type to ODBCDirect.

```
DBEngine.DefaultType = dbUseODBC
```

Creating a Specific Workspace

You can override the default workspace setting for a specific workspace by using the *type* argument of the **CreateWorkspace** method.

The following code creates an ODBCDirect workspace.

```
Dim wspODBC As Workspace
Set wspODBC = DBEngine.CreateWorkspace _
    ("NewODBCWorkspace", "Admin", "", dbUseODBC)
```

Connecting to a remote data source is similar to opening a Microsoft Jet database. First you create a workspace (such as an ODBCDirect workspace), and then you open the data source.

To open a remote data source, you create a **Connection** object, just as you would create a **Database** object to open a Microsoft Jet database. Once a connection has been created, you can retrieve data from the remote data source.

Connection Declaration

The **Connection** object variable refers to your connection. Use the **Set** statement to open the connection.

The following code opens a connection in the default workspace.

```
Dim conNewConnection As Connection
Set conNewConnection = OpenConnection _
        ("New", dbDriverPrompt, false, _
        "ODBC;DATABASE=pubs;UID=sa;PWD;DSN=SQLServer")
```

OpenConnection Method

The following table lists the arguments to the **OpenConnection** method, and describes their use.

Argument	Description
name	Any string, if you specify a registered ODBC data source name (DSN). If a valid DSN is not included in the **OpenConnection** declaration, the name must refer to a valid ODBC DSN, which will also be the **Name** property.
options	Sets various options for the connection.
readonly	**True** or **False**. If **True**, no modifications are allowed. The default value is **False**.
connect	An ODBC connection string.

The *options* argument determines if and when to prompt the user to establish the connection, and whether or not to open the connection asynchronously.

The following table lists the values for the *options* argument and describes their use.

Constant	Result
dbDriverNoPrompt	The ODBC Driver Manager uses the connection string provided in the *connect* argument.
dbDriverPrompt	The ODBC Driver Manager displays the ODBC **Data Sources** dialog box, which displays any relevant information supplied in the *connect* argument. The connection string is made up of the DSN that the user selects by means of the dialog boxes, or, if the user doesn't specify a DSN, the default DSN will be used.
dbDriverComplete	If the *connect* argument includes all of the necessary information to complete a connection, the ODBC Driver Manager uses the string in the *connect* argument. Otherwise, it behaves as it does when you specify the default constant **dbDriverPrompt**. This is the default value.
dbDriverCompleteRequired	This option behaves like the **dbDriverComplete** constant, except the ODBC driver disables the prompts for any information not required to complete the connection.

dbRunAsync	Execute the method asynchronously. This constant can be used with any of the other *options* constants.

Retrieving remote data by using ODBCDirect is very similar to retrieving data from a Microsoft Jet database. Once you establish a connection to an ODBC-compliant database with ODBCDirect, you create a **Recordset** object by using the **OpenRecordset** method, just as you do when you retrieve records from a Microsoft Jet database.

Recordset Declaration

To see a code sample that creates an ODBCDirect workspace with a connection to the workspace, and then retrieves a recordset, click this icon.
{ewc mvimg, mvimage,!code.bmp}

The recordset retrieved from a remote data source can be controlled in the same manner as a recordset retrieved by using DAO with the Microsoft Jet database engine.

For more information, see Moving Through a Recordset in Chapter 3: Using Data Access Objects.

OpenRecordset Method with ODBCDirect

In addition to the options available with the **OpenRecordset** method for a **Database** object, the **OpenRecordset** method for a **Connection** object enables you to retrieve records from a database.

The following arguments of the **OpenRecordset** method are available when using an ODBCDirect workspace.

® *type*

- **dbOpenDynamic**

  Opens a dynamic-type recordset that you can use to add, change, or delete records from an underlying database table or tables. A dynamic cursor can contain fields from one or more tables in a database.

® *options*

- **dbRunAsync**

  Runs an asynchronous query.

- **dbExecDirect**

  Runs a query by skipping **SQLPrepare** and calling **SQLExecDirect** directly. Use this option only when you're not opening a recordset based on a parameter query.

® *lockedits*

- **dbOptimisticValue**

  Uses optimistic concurrency based on row values.

- **dbOptimisticBatch**

  Enables batch optimistic updating.

Note The default ODBCDirect workspace is read-only. To enable users to read and write records in an ODBCDirect workspace, you must specify the lock type **dbOptimistic**. For information about lock types, see the **OpenRecordset method** in Visual Basic Help.

To see a demonstration of how to use ODBCDirect to retrieve data from an external database, click this icon.
{ewc mvimg, mvimage,!democlip.bmp}

You can view labs from within the chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 4: Advanced Database Development

- Exercise 1: Handling Referential Integrity Violations
- Exercise 2: Multi-User Issues
- Exercise 3: (Optional) Using ODBC Direct

You can jump directly to the lab exercises by clicking on the titles below.

Lab 5: Using Dynamic-Link Libraries

- Exercise 1: Locating the Windows Folder
- Exercise 2: Creating a Topmost Window
- Exercise 3: Using Callbacks

You can jump directly to the lab exercises by clicking on the titles below.

Lab 6: Creating ActiveX Clients

- Exercise 1: Controlling Microsoft Excel
- Exercise 2: Creating a Client Application

You can view labs from within the chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 7: Creating ActiveX Code Components

- Exercise 1: Creating a Code Component
- Exercise 2: Debugging and Error Handling
- Exercise 3: Adding Component Information and Help
- Exercise 4: Defining and Using Events
- Exercise 5: Compiling and Registering the Component
- Exercise 6: (Optional) Creating an Asynchronous Method

1. What is the most significant difference between ActiveX controls and code components?

{ewc mvimg, mvimage,!answer.bmp}
A. ActiveX controls are version-independent.

{ewc mvimg, mvimage,!answer.bmp}
B. ActiveX controls expose functionality that can be used by other applications through Automation.

{ewc mvimg, mvimage,!answer.bmp}
C. ActiveX controls must exist within some type of container, usually a form or an application.

{ewc mvimg, mvimage,!answer.bmp}
D. ActiveX controls generate events through user interaction with a visual interface.

2. What should you do to enable other developers to place controls on instances of your ActiveX control?

{ewc mvimg, mvimage,!answer.bm
A. Set the **ControlContainer** property of the **UserControl** object to **True**.

p}

{ewc mvimg, mvimage,!answer.bmp}

B. Set the **ContainerControls** property of the **UserControl** object to **True**.

{ewc mvimg, mvimage,!answer.bmp}

C. Set the **ContainedControls** property of the **UserControl** object to **True**.

{ewc mvimg, mvimage,!answer.bmp}

D. Set the **Frame** property of the **UserControl** object to **True**.

3. When you create a Toolbox icon for your ActiveX control, why should you specify a 16 by 15 pixel bitmap?

{ewc mvimg, mvimage,!answer.bmp}

A. Visual Basic requires that you specify a 16 by 15 pixel bitmap.

{ewc mvimg, mvimage,!answer.bmp}

B. A bitmap of greater resolution uses too many resources and affects performance.

{ewc mvimg.mvimage.!answer.bmp}

C. Bitmaps of different resolutions will be scaled to 16 by 15 pixels, and may be distorted.

{ewc mvimg.mvimage.!answer.bmp}

D. Bitmaps with resolutions higher than 16 by 15 pixels will not fit on the Toolbox correctly and may cover other icons.

4. Which action will cause a design time instance of an ActiveX control to be created?

{ewc mvimg.mvimage.!answer.bmp}

A. Running an application containing the control at design time.

{ewc mvimg.mvimage.!answer.bmp}

B. Closing a form containing the control.

{ewc mvimg.mvimage.!answer.bmp}

C. Ending an application containing the control at design time.

{ewc

D. Running a compiled application containing the control.

{mvimg,mvimage,!answer.bmp}

5. If you run an application containing an ActiveX control at design time so that the design-time instance of the control is destroyed and a run-time instance created, which event does not occur?

{ewc mvimg,mvimage,!answer.bmp} A. WriteProperties

{ewc mvimg,mvimage,!answer.bmp} B. Terminate

{ewc mvimg,mvimage,!answer.bmp} C. Initialize

{ewc mvimg,mvimage,!answer.bmp} D. InitProperties

{ewc mvimg,mvimage,!answer.bmp} E. ReadProperties

{ewc mvi

{mg.mvimage.!answer.bmp}

F. Resize, Paint

{ewc mvimg.mvimage.!answer.bmp}

6. What is the function of the code shown below?

```
Private Sub ctlInstruction_Click()
        Dim Msg
        Msg = "Complete the displayed form."
        MsgBox(Msg)
End Sub
```

{ewc mvimg.mvimage.!answer.bmp}

A. Changes the **Msg** property of a **MsgBox** object whenever a user clicks a UserControl.

{ewc mvimg.mvimage.!answer.bmp}

B. Raises a Click event whenever a user clicks a UserControl.

{ewc mvimg.mvimage.!answer.bmp}

C. Declares a Click event on a UserControl.

{ewc mvimg.mvimage.!answer.bm

p}

{ewc mvimg, mvimage,!answer.bmp}

D. Passes the argument “Complete the displayed form.” of a previously declared Click event to a client.

7. How can you notify Visual Basic that properties on a properties page have changed and enable the Apply button on the Property Pages dialog box?

{ewc mvimg, mvimage,!answer.bmp}

A. Use the **Count** property of the **SelectedControls** collection to determine if the returned value is greater than one.

{ewc mvimg, mvimage,!answer.bmp}

B. Set the **PropertyPage** object’s **Changed** property to **True**.

{ewc mvimg, mvimage,!answer.bmp}

C. Use the SelectionChanged event to notify Visual Basic that properties have changed and use the ApplyChanges event to enable the **Apply** button.

{ewc mvimg, mvimage,!answer.bmp}

D. Use the **WriteProperties** method with the **PropertyBag** object to notify Visual Basic that properties have changed and enable the **Apply** button by setting the **PropertyPage** object’s **Changed** property to Enabled.

8. How can you write any changed property values back to currently selected controls?

{ewc mvimg, mvimage,!answer.bmp}

A. Use the ApplyChanges event.

{ewc mvimg, mvimage,!answer.bmp}

B. Use the **Connect Property Pages** dialog box.

{ewc mvimg, mvimage,!answer.bmp}

C. Set the **PropertyPage** object's **Changed** property to **True**.

{ewc mvimg, mvimage,!answer.bmp}

D. Use the SelectionChanged event.

9. What is the result if you compile your ActiveX control with the Require License Key box on the General tab of the Project Properties pages unchecked?

{ewc mvimg, mvimage,!answer.bm

A. A user will be unable to create a run-time instance of the control.

p}

{ewc mvimg, mvimage,!answer.bmp}

B. A user will be unable to create a design-time instance of the control.

{ewc mvimg, mvimage,!answer.bmp}

C. A license key will be written to the user's registry, allowing a design-time instance of the control to be created.

{ewc mvimg, mvimage,!answer.bmp}

D. No license key will be written to the user's registry.

You can view labs from within the chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 8: Creating ActiveX Controls

- Exercise 1: Creating a Control
- Exercise 2: Adding Properties and Methods
- Exercise 3: Saving and Loading Properties
- Exercise 4: Raising an Event
- Exercise 5: Creating a Property Page
- Exercise 6: (Optional) Creating a Data-Bound Control

You can view labs from within the chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 9: Using ActiveX Components on a Web Page

- Exercise 1: Creating a Setup Package
- Exercise 2: Scripting a Control
- Exercise 3: (Optional) Adding Licensing

You can view labs from within the chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 10: Creating and Using ActiveX Documents

- Exercise 1: Converting Forms to ActiveX Documents
- Exercise 2: Adding Properties to ActiveX Documents
- Exercise 3: Extending User Interface Functionality
- Exercise 4: (Optional) Creating a Document Container

You can view labs from within the Chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 11: Creating Internet-Aware Applications

- Exercise 1: Building the Browser Application
- Exercise 2: Building the FTP Download Component
- Exercise 3: Adding Browser Capabilities to the Component
- Exercise 4: Coding a Chat Session Application

You can view labs from within the chapter or jump directly to the lab exercises by clicking on the titles below.

Lab 12: Application Setup and Optimization

- Exercise 1: Using Resource Files
- Exercise 2: Using the Registry
- Exercise 3: Using the Setup Wizard
- Exercise 4: (Optional) Using Visual Basic Code Profiler

```
Sub OpenODBCDirectRecordset()
    Dim wspODBC As Workspace
    Dim conODBC As Connection
    Dim rstODBC As Recordset

    Set wspODBC = CreateWorkspace _
        ("", "sa", "", dbUseODBC)
    Set conODBC = wspODBC.OpenConnection _
        ("NewConncetion", dbDriverNoPrompt, False, _
        "ODBC;DATABASE=pubs;UID=sa;PWD=;DSN=SQLServer")
    Set rstODBC = conODBC.OpenRecordset _
        ("SELECT * FROM authors WHERE au_lname = 'Green'", _
            dbOpenSnapshot, 0, dbReadOnly)
    rstODBC.MoveLast
    MsgBox "This remote recordset contains " & _
        rstODBC.RecordCount & " records."
    rstODBC.Close
    conODBC.Close
    wspODBC.Close
End Sub
```

You can view self checks from within the Chapter or jump directly to the self-check questions section of Chapter 1 by clicking on the title below.

Chapter 1: Visual Basic Review, Self-Check Questions

You can view the Self Check questions from within the Chapter or jump directly to the Self Check questions in Chapter 2 by clicking on the title below.

Chapter 2: Using the Data Control, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 3 by clicking on the title below.

Chapter 3: Using Data Access Objects, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 4 by clicking on the title below.

Chapter 4: Advanced Database Development, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 5 by clicking on the title below.

Chapter 5: Using Dynamic-Link Libraries, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 6 by clicking on the title below.

Chapter 6: Creating ActiveX Clients, Self-Check Questions

You can view the Self Check questions from with the chapter or jump directly to the Self Check questions for Chapter 7 by clicking on the title below.

Chapter 7: Creating ActiveX Code Components, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 8 by clicking on the title below.

Chapter 8: Creating ActiveX Controls, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 9 by clicking on the title below.

Chapter 9: Using ActiveX Components on a Web Page, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 10 by clicking on the title below.

Chapter 10: Creating and Using Active X Documents, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 11 by clicking on the title below.

Chapter 11: Creating Internet-Aware Applications, Self-Check Questions

You can view Self Check questions from within the chapter or jump directly to the Self Check questions in Chapter 12 by clicking on the title below.

Chapter 12: Application Setup and Optimization, Self-Check Questions

```
SELECT au_lname FROM Authors ORDER BY au_lname
```

Asynchronous code components designed for unattended execution cannot contain forms. You can use the Windows API functions **SetTimer** and **KillTimer** in these components to provide timer functionality without using the **Timer** control on a form.

This is correct when you position the current record one before the first record. When there are no records in a Recordset, both BOF and EOF are True.

For more information, see Working with Recordsets

This is correct when you position the current record one after the last record. When there are no records in a Recordset, both BOF and EOF are True.

For more information, see Working with Recordsets

This is correct when you position the current record on the first through the last record. When there are no records in a Recordset, both BOF and EOF are True.

For more information, see Working with Recordsets

When there are no records in a Recordset, both BOF and EOF are True.

For more information, see Working with Recordsets

The Find method is used to locate a record in a dynaset-type Recordset. The Seek method should be executed to locate a record in a table-type Recordset.

For more information, see Searching a Dynaset or Snapshot.

The Find method is used to locate a record in a snapshot-type Recordset. The Seek method should be executed to locate a record in a table-type Recordset.

For more information, see Searching a Dynaset or Snapshot.

The Seek method should be executed to locate a record in a table-type Recordset.

For more information, see Searching a Table.

A select query can be used to return record.

For more information, see Using Queries

An action query is used to update, insert or delete a record; a select query returns records.

For more information, see Using Queries

An action query is used to update, insert or delete a record; a select query returns records.

For more information, see Using Queries

An action query is used to update, insert or delete a record; a select query returns records.

For more information, see Using Queries

Referential integrity does not determine when a run-time error occurs if a validation rule is violated. The value of the ValidateOnSet property determines if a rule violation generates an error when the incorrect data is placed on a form or when the user attempts to write the incorrect data to the database.

For more information, see Defining Validation Rules

The ValidationRule property defines the validation rule used by the application to accept or reject data. The value of the ValidateOnSet property determines if a rule violation generates an error when the incorrect data is placed on a form or when the user attempts to write the incorrect data to the database.

For more information, see Defining Validation Rules

The value of the ValidateOnSet property determines if a rule violation generates an error when the incorrect data is placed on a form or when the user attempts to write the incorrect data to the database.

For more information, see Defining Validation Rules

The ValidationText property contains the text displayed when a rule is violated. The value of the ValidateOnSet property determines if a rule violation generates an error when the incorrect data is placed on a form or when the user attempts to write the incorrect data to the database.

For more information, see Defining Validation Rules

The Update statement improves performance by buffering the operations and writing to disk only after the transaction is committed.

For more information, see Transactions

The Rollback method must be invoked if the transaction fails. The Update statement improves performance by buffering the operations and writing to disk only after the transaction is committed.

For more information, see Transactions

The Update statement improves performance by buffering the operations and writing to disk only after the transaction is committed.

For more information, see Transactions

The Update statement improves performance by buffering the operations and writing to disk only after the transaction is committed.

For more information, see Transactions

The dbDenyRead argument can be used only with table-type Recordsets.

For more information, see Opening a Table for Exclusive Use

The dbDenyRead argument can be used only with table-type Recordsets.

For more information, see Opening a Table for Exclusive Use

The dbDenyRead argument can be used only with table-type Recordsets.

For more information, see Opening a Table for Exclusive Use

The dbDenyRead argument can be used only with table-type Recordsets.

For more information, see Opening a Table for Exclusive Use

The value of the LockEdits property determines if optimistic or pessimistic locking is in effect. LockEdits is True by default, and pessimistic locking is in effect.

For more information, see Microsoft Jet Database Engine Locking

This is the period of time a page is locked when pessimistic locking is in effect. The value of the LockEdits property determines if optimistic or pessimistic locking is in effect. LockEdits is True by default, and pessimistic locking is in effect.

For more information, see Microsoft Jet Database Engine Locking

The value of the LockEdits property determines if optimistic or pessimistic locking is in effect. LockEdits is True by default, and pessimistic locking is in effect.

For more information, see Microsoft Jet Database Engine Locking

This is the time a page is locked when optimistic locking is in effect. The value of the LockEdits property determines if optimistic or pessimistic locking is in effect. LockEdits is True by default, and pessimistic locking is in effect.

For more information, see Microsoft Jet Database Engine Locking

```
Sub OpenTable()

    Dim wspCurrent As Workspace
    Dim dbCurrent As Database
    Dim rstProducts As Recordset

    Set wspCurrent = CreateWorkspace("New Workspace", "Admin", "", dbUseJet)
    Set dbCurrent = wspCurrent.OpenDatabase("C:\Program Files\DevStudio\VB\
Nwind.mdb")
    Set rstProducts = dbCurrent.OpenRecordset("Products", dbOpenTable)

    rstProducts.MoveLast
    MsgBox "There are " & rstProducts.RecordCount & " records."

    rstProducts.Close
    dbCurrent.Close
    wspCurrent.Close

End Sub
```

```
Private Sub txtCelsius_KeyUp(KeyCode As Integer, Shift As Integer)
    On Error GoTo handleErr
    txtFahrenheit.Text = (txtCelsius.Text * 9 / 5) + 32
    Exit Sub
handleErr:
    If Err.Number = 13 Then 'type mismatch
        txtFahrenheit.Text = "Can't convert"
    Else
        Err.Raise Err.Number
    End If
End Sub
```

They are version-independent only if compiled as source code. The most significant difference is the fact that controls have visual elements and generate events through user interaction.

For more information, see Introduction to Controls.

This is a description of code components. The most significant difference is the fact that controls have visual elements and generate events through user interaction.

For more information, see Introduction to Controls.

This is true, but the most significant difference between controls and code components is the fact that controls have visual elements and generate events based on user actions.

For more information, see Introduction to Controls.

The most significant difference is the fact that controls have visual elements and can generate events as a result of user actions.

For more information, see Introduction to Controls.

An ActiveX control can contain someone else's controls if the ControlContainer property of the UserControl object is set to True.

For more information, see Creating a Container Control.

ContainerControls is the collection used by ActiveX controls to perform operations on any contained controls. The ControlContainer property of the UserControl object must be set to True to allow others to place controls on your ActiveX control.

For more information, see Creating a Container Control.

The read-only ContainedControls property returns the collection of controls added to your ActiveX control at run-time. The ControlContainer property of the UserControl object must be set to True to allow others to place controls on your ActiveX control.

For more information, see Creating a Container Control.

A Frame is not a property, but a Visual Basic control that can hold other controls. The ControlContainer property of the UserControl object must be set to True to allow others to place controls on your ActiveX control.

For more information, see Creating a Container Control.

You can specify a bitmap with a different resolution, but Visual Basic will scale the bitmap to 16 by 15 pixels and probably distort the image.

For more information, see Specifying a Toolbox Bitmap.

Resource utilization is the same, since Visual Basic will scale the bitmap you specify to 16 by 15 pixels. The problem is that the image is usually distorted when this occurs.

For more information, see Specifying a Toolbox Bitmap.

You should specify a 16 by 15 pixel bitmap to make sure the icon has the appearance you desire.

For more information, see Specifying a Toolbox Bitmap.

Visual Basic will scale the bitmap you specify to 16 by 15 pixels. The problem is that the image is usually distorted as a result of the scaling.

For more information, see Specifying a Toolbox Bitmap.

This destroys the design time instance of the control and creates a run-time instance. A design time instance of a control is created when a control is placed on a form, and when you end an application containing a control at design time.

For more information, see Control Instancing and Events.

The design time instance of an ActiveX control is destroyed when you close the form containing the control. Ending an application containing the control at design time and placing a control on a form creates a design time instance of an ActiveX control.

For more information, see Control Instancing and Events.

A design time instance of a control is created when you end an application containing the control at design time, and when you place a control on a form.

For more information, see Control Instancing and Events.

Running a compiled application containing an ActiveX control creates a run-time instance of the control. A design time instance is created when the control is placed on a form, and when you end an application containing the control at design time.

For more information, see Control Instancing and Events.

The WriteProperties event occurs when you run an application containing a control at design time, and when you close a form containing the control. The InitProperties event does not occur when you run an application containing an ActiveX control at design time.

For more information, see Control Instancing and Events.

The Terminate event occurs when you run an application containing a control at design time, when you end an application containing the control at design time, and when you close a form containing the control. The InitProperties event does not occur when you run an application containing an ActiveX control at design time.

For more information, see Control Instancing and Events.

The Initialize event occurs when you run an application containing the control at design time, when you place a control on a form, when you end an application containing the control at design time, and when you run a compiled application containing an ActiveX control. The InitProperties event does not occur when you run an application containing an ActiveX control at design time.

For more information, see Control Instancing and Events.

The InitProperties event occurs when you place an ActiveX control on a form, but not when you run an application containing an ActiveX control at design time.

For more information, see Control Instancing and Events.

The ReadProperties event occurs when you run an application containing an ActiveX control at design time, when you end an application containing the control at design time, and when you run a compiled application containing the control. The InitProperties event does not occur when you run an application containing an ActiveX control at design time.

For more information, see Control Instancing and Events.

The Resize and Paint events occur when you run an application containing an ActiveX control at design time, when you place the control on a form, when you end an application containing the control at design time, and when you run a compiled application containing the control. The InitProperties event does not occur when you run an application containing an ActiveX control at design time.

For more information, see Control Instancing and Events.

This code displays the message box (raises the click event) when a user clicks on a UserControl.

For more information, see Raising Control Events.

This code raises a previously declared Click event (displays the message box) on a User Control.

For more information, see Raising Control Events.

This code raises a previously declared Click event (displays a message box) whenever a user clicks on a UserControl.

For more information, see Raising Control Events.

This code raises a previously declared Click event (displays a message box) whenever a user clicks on a UserControl.

For more information, see Raising Control Events.

The Count property of the SelectedControls collection is used to determine whether multiple controls are selected. To enable the Apply button and notify Visual Basic that properties have changed, you must set the PropertyPage object's Changed property to True.

For more information, see Coding Property Page Behavior.

Setting the PropertyPage object's Changed property to True will enable the Apply button on the Property Pages dialog box and notify Visual Basic that properties have changed.

For more information, see Coding Property Page Behavior.

The SelectionChanged event occurs when a property page is opened and when the list of selected controls changes. The ApplyChanges event is used to write any changed properties back to the selected controls. Set the PropertyPage object's Changed property to True to notify Visual Basic that properties have changed and enable the Apply button.

For more information, see Coding Property Page Behavior.

The WriteProperties method is used with the PropertyBag object to save the state of an object. You notify Visual Basic that properties have changed and enable the Apply button by setting the PropertyPage object's Changed property to True.

For more information, see Coding Property Page Behavior, Exposing Properties, Methods, and Events.

The ApplyChanges event is used to write any changed property values back to the currently selected controls.

For more information, see Coding Property Page Behavior.

The Connect Property Pages dialog box is used to assign one or more property pages to a control. The ApplyChanges event is used to write any changed properties back to selected controls.

For more information, see Creating the Property Page Interface.

This will notify Visual Basic that properties have changed and enable the Apply button on the Property Pages dialog box. The ApplyChanges event is used to write any changed properties back to the selected controls.

For more information, see Coding Property Page Behavior.

The SelectionChanged event is used to set the values of controls that display property values for editing. Use the ApplyChanges event to write any changed properties back to selected controls.

For more information, see Coding Property Page Behavior.

If the control is unlicensed, the user can create a run-time or a design time instance of your control. To protect your control, you should check the Require License Key box.

For more information, see Licensing Controls.

If the control is unlicensed, a user can create a design time or a run-time instance of your control. To protect your control, you should check the Require License Key box.

For more information, see Licensing Controls.

No license key will be written to the user's registry, but none will be required. A user will be able to create a run-time or a design time instance of your control. You should check the Require License Key box to protect your control.

For more information, see Licensing Controls.

This is correct, but if the control is not licensed, the user will be able to create a run-time or a design time instance of your control.

For more information, see Licensing Controls.

FTP and HTTP servers listen for messages from clients. Typically these servers assign port 21 as the location for incoming data via FTP, and use another port for outgoing data. Port 80 is used for both the incoming and outgoing data for HTTP.

The life of a variable refers to how long a variable is available. Where the variable is declared (in a procedure or module), and how the variable is declared (as **Static** or not **Static**), determines the life of the variable.

Procedure-Level Variables

A procedure-level variable lasts as long as the procedure does. When the procedure is called again, the variable is reinitialized.

If you do not want the variable to be reinitialized, declare the variable as **Static**. Variables declared as **Static** are still only available to the procedure; however, the variables retain their value the entire time the application is running.

In the following code, iRetryCount retains its value for each invocation of myProc. The variable iValue is reset to 0 each time myProc is invoked.

```
Sub myProc()
    Static iRetryCount As Integer
    Dim iValue As Integer
    iRetryCount = iRetryCount + 1
End Sub
```

Form-Level Variables

Variables declared at the form level are maintained until the form is set to **Nothing**. Form-level variables retain their value when the form is unloaded.

When you set a form variable to **Nothing**, the variables are cleared and the Terminate event of the form is executed. The following code shows how to set Form2 to **Nothing**:

```
Set Form2 = Nothing
```

Standard Module-Level Variables

Variables declared at the module level are available during the entire time an application is running.

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlip.bmp} Creating a Simple Application

{ewc MVIMG, MVIMAGE,! democlip.bmp} Getting Assistance

{ewc MVIMG, MVIMAGE,! democlip.bmp} Adding Controls and Setting Properties

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using the Debugging Tools

{ewc MVIMG, MVIMAGE,! democlip.bmp} Lab 1: Visual Basic Review

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using Windows API from Visual Basic

{ewc MVIMG, MVIMAGE,! anim.bmp} Passing Parameters by Value or by Reference

{ewc MVIMG, MVIMAGE,! democlip.bmp} Passing Strings

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using Callback Functions

{ewc MVIMG, MVIMAGE,! democlip.bmp} Lab 5: Using Dynamic-Link Libraries

You can view multimedia from within the chapter or jump directly to an animation, demonstration, or point of view by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! expposv.bmp} Introduction to Database Development

{ewc MVIMG, MVIMAGE,! anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using the Data Control

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using the Data Form Wizard

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using the DBCombo Control

{ewc MVIMG, MVIMAGE,! democlip.bmp} Lab 2: Using the Data Control

You can view multimedia from within the chapter or jump directly to an animation, demonstration, or point of view by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! expp ov.bmp} Introduction to ActiveX Component Development

{ewc MVIMG, MVIMAGE,! anim. bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlip. bmp} Introducing the Registry

{ewc MVIMG, MVIMAGE,! democlip. bmp} How the Object Browser Works

{ewc MVIMG, MVIMAGE,! anim. bmp} How Late Binding Works

{ewc MVIMG, MVIMAGE,! anim. bmp} How dispID Binding Works

{ewc MVIMG, MVIMAGE,! anim. bmp} How vTable Binding Works

{ewc MVIMG, MVIMAG

E.!
anim.
bmp}

{ewc MVIMG, MVIMAGE.!anim.bmp}

How Objects Are Created

{ewc MVIMG, MVIMAGE.!democlip.bmp}

Controlling Microsoft Excel

{ewc MVIMG, MVIMAGE.!democlip.bmp}

Lab 6: Creating ActiveX Clients

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,!anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,!democlip.bmp} Building a Simple ActiveX Control

{ewc MVIMG, MVIMAGE,!democlip.bmp} Saving and Retrieving Properties

{ewc MVIMG, MVIMAGE,!democlip.bmp} Using the ActiveX Control Interface Wizard

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating and Using a Property Page

{ewc MVIMG, MVIMAGE,!democlip.bmp} Building a Simple Bound Control

{ewc MVIMG, Lab 8: Creating ActiveX Controls

MVI
MAG
E.!
dem
oclip.
bmp}

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,!anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,!anim.bmp} Static vs. Active Web Pages

{ewc MVIMG, MVIMAGE,!anim.bmp} Downloading an ActiveX Control

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating an Internet Setup

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating a Licensing Package File

{ewc MVIMG, MVIMAGE,!democlip.bmp} Removing a Licensing Key from the Registry

{ewc MVIMG, MVIMAG Internet Explorer Security Options

E.!
dem
oclip.
bmp}

{ewc MVI MG, MVI MAG E.! dem oclip. bmp}

Lab 9: Using ActiveX Components in Internet Explorer

DLLs are not linked to applications at compile time. DLLs can provide functionality not available in Visual Basic.

For more information, see Introduction to DLLs.

DLLs can perform tasks that are difficult or impossible to code in Visual Basic.

For more information, see Introduction to DLLs.

In general, DLLs execute faster than Visual Basic code. DLLs can provide functionality not available in Visual Basic.

For more information, see Introduction to DLLs.

DLLs are not bound to any application; this modularity is one advantage of using DLLs. DLLs can provide functionality not available in Visual Basic.

For more information, see Introduction to DLLs.

An int in a 32-bit C compiler is equivalent to a Visual Basic Long, but an int in a 16-bit C compiler is equivalent to a Visual Basic Integer.

For more information, see Converting C Declarations to Visual Basic Data Types.

An int in a 32-bit C compiler is equivalent to a Visual Basic Long, but an int in a 16-bit C compiler is equivalent to a Visual Basic Integer.

For more information, see Converting C Declarations to Visual Basic Data Types.

A Visual Basic Byte data type is equivalent to a char in C. An int in a 16-bit C compiler is equivalent to a Visual Basic Integer.

For more information, see Converting C Declarations to Visual Basic Data Types.

A pointer to a string in C can be expressed as a String or Variant in Visual Basic. An int in a 16-bit C compiler is equivalent to a Visual Basic Integer.

For more information, see Converting C Declarations to Visual Basic Data Types.

The frame window should be the parent of the copied controls. If the frame is not selected when you paste the controls, the form will be the parent, rather than the frame.

You can use the **Any** keyword to pass any data type to a DLL as an argument with no type checking by Visual Basic, but this does not necessarily mean the DLL can accept the argument.

For more information, see Declaring an Argument as Any.

The Any keyword allows you to declare an argument that will accept any data type, and Visual Basic will pass the argument to a DLL with no type checking.

For more information, see Declaring an Argument as Any.

The Any keyword allows you to declare an argument that will accept any data type, and Visual Basic will pass the argument to a DLL with no type checking.

For more information, see Declaring an Argument as Any.

The Any keyword allows you to declare an argument that will accept any data type, and Visual Basic will pass the argument to a DLL with no type checking.

For more information, see Declaring an Argument as Any.

You should declare the argument with ByVal as String and pass vbNullString to the DLL.

For more information, see Passing Null Values.

You should declare the argument with ByVal as String and pass vbNullString to the DLL.

For more information, see Passing Null Values.

You should declare the argument with ByVal as String and pass vbNullString to the DLL.

For more information, see Passing Null Values.

You should declare the argument with ByVal as String and pass vbNullString to the DLL.

For more information, see Passing Null Values.

The ANSI version of SetWindowText, SetWindowTextA, will be called.

For more information, see Unicode vs. ANSI DLLs.

The ANSI version of SetWindowText, SetWindowTextA, will be called.

For more information, see Unicode vs. ANSI DLLs.

The ANSI version of SetWindowText, SetWindowTextA, will be called.

For more information, see Unicode vs. ANSI DLLs.

The ANSI version of SetWindowText, SetWindowTextA, will be called.

For more information, see Unicode vs. ANSI DLLs.

Basic business communications in Japanese requires about 6,000 distinct characters, and Unicode must be used.

For more information, see Unicode vs. ANSI DLLs.

Basic business communications in Japanese requires about 6,000 distinct characters, and Unicode must be used.

For more information, see Unicode vs. ANSI DLLs.

Basic business communications in Japanese requires about 6,000 distinct characters, and Unicode must be used.

For more information, see Unicode vs. ANSI DLLs.

Basic business communications in Japanese requires about 6,000 distinct characters, and Unicode must be used.

For more information, see Unicode vs. ANSI DLLs.

The variable containing an argument passed ByVal cannot be changed by the procedure. If the argument is passed ByRef, however, the procedure can change the original variable.

For more information, see Passing Arguments by Value or by Reference.

A copy of an argument passed ByVal is passed to a procedure. The address of the variable containing an argument ByRef is passed to a procedure.

For more information, see Passing Arguments by Value or by Reference.

When passing a string to a DLL that expects a C string, you should use the ByVal keyword when declaring the argument. If a procedure changes an argument passed ByRef, the original variable will be changed.

For more information, see Passing Arguments by Value or by Reference.

If a procedure changes an argument passed ByRef, the value of the original variable will be changed. When an argument is passed ByRef, the address of the variable containing the argument is actually passed to the procedure.

For more information, see Passing Arguments by Value or by Reference.

To specify a callback from Visual Basic, the callback procedure must be located in a standard module, and you must use the AddressOf operator with the name of the procedure to pass the procedure's address to the DLL.

For more information, see Providing Callback Procedures.

To specify a callback from Visual Basic, the callback procedure must be located in a standard module, and you must use the AddressOf operator with the name of the procedure to pass the procedure's address to the DLL.

For more information, see Providing Callback Procedures.

To specify a callback from Visual Basic, the callback function must be located in a standard module, and you must use the AddressOf operator with the name of the callback procedure to pass the procedure's address to the DLL.

For more information, see Providing Callback Procedures.

To specify a callback from Visual Basic, the callback function must be located in a standard module, and you must use the AddressOf operator with the name of the callback procedure to pass the procedure's address to the DLL.

For more information, see Providing Callback Procedures.

This DLL sets the show state and the normal, minimized, and maximized positions of a Window. SetWindowPos can be used to create a form that remains on top of other forms even when it does not have the focus.

For more information, see Useful DLLs.

This DLL activates a window. SetWindowPos can be used to create a form that remains on top of other forms even when it does not have the focus.

For more information, see Useful DLLs.

SetWindowPos can be used to create a form that remains on top of other forms even when it does not have the focus.

For more information, see Useful DLLs.

This DLL determines when a shell function has finished loading a program. SetWindowPos can be used to create a form that remains on top of other forms even when it does not have the focus.

For more information, see Useful DLLs.

This DLL activates a window. SetWindowPos can be used to create a form that remains on top of other forms even when it does not have the focus.

For more information, see Useful DLLs.

The SetTimer function requires the Interval and Callback parameters when specifying a callback routine, but the HWnd and idEvent parameters are not used.

For more information, see Providing Callback Procedures.

The SetTimer function requires the Interval and Callback parameters when specifying a callback routine, but the HWnd and idEvent parameters are not used.

For more information, see Providing Callback Procedures.

The SetTimer function requires the Interval and Callback parameters when specifying a callback routine, but the HWnd and idEvent parameters are not used.

For more information, see Providing Callback Procedures.

The SetTimer function requires the Interval and Callback parameters when specifying a callback routine, but the HWnd and idEvent parameters are not used.

For more information, see Providing Callback Procedures.

ActiveX controls execute on the user's computer, but they are compiled and users can't see the source code.

For more information, see Advantages of Using Controls on a Web Page.

ActiveX controls execute without talking to the server. Security is enhanced because the control is compiled and users can’t see the source code.

For more information, see Advantages of Using Controls on a Web Page.

No source code is downloaded from the web server. Security is enhanced because users never see the source code.

For more information, see Advantages of Using Controls on a Web Page.

This characteristic makes your control more secure.

For more information, see Advantages of Using Controls on a Web Page.

The Visual Basic Application Setup Wizard is used to package an ActiveX control and associated files into a .cab file. The <OBJECT> tag is used to specify an ActiveX control to be downloaded.

For more information, see Steps for Downloading a Control.

The <OBJECT> tag is used to specify a compiled ActiveX control to be downloaded.

For more information, see Steps for Downloading a Control.

The <OBJECT> tag is used to provide information to the browser for downloading and installing an ActiveX control.

For more information, see Steps for Downloading a Control.

Information in the <OBJECT> tag is used by the browser to download and install a compiled ActiveX control.

For more information, see Steps for Downloading a Control.

The ID parameter specifies the object name that can be used to refer to the object in VBScript.

For more information, see Adding a Control to an HTML Page.

The CLASSID is the unique class identifier stored in the system registry. The ID parameter is used to specify the object name.

For more information, see Adding a Control to an HTML Page.

The CODEBASE parameter specifies a URL that points to a file containing an implementation of an object. The ID parameter specifies the object name.

For more information, see Adding a Control to an HTML Page.

The CODEBASE parameter specifies a URL that points to the CAB file for an ActiveX control. The ID parameter specifies the object name.

For more information, see Adding a Control to an HTML Page.

The Visual Basic Application Setup Wizard is used to create an ActiveX control's CAB file. A license package file is created with the LPK_TOOL.EXE utility.

For more information, see, Including Licensing with Your Controls.

A license package file is created using the LPK_TOOL.EXE utility from the LPK_TOOL directory on the Visual Basic CD ROM.

For more information, see, Including Licensing with Your Controls.

The License Manager object in the HTML code of a web page is used to specify the license package file after it is created using the LPK_TOOL.EXE utility.

For more information, see Including Licensing with Your Controls.

The CODEBASE parameter of an <OBJECT> tag is used to specify a URL that points to an ActiveX control's CAB file. The LPK_TOOL.EXE utility is used to create the license package file for a page.

For more information, see Adding a Control to an HTML Page.

The license will already be in your development machine's system registry. You must remove the registry entry to make sure the control will run on user's machines.

For more information, see Including Licensing with Your Controls.

You must remove the license entry from your system registry to make sure the page's .lpk file will enable your control.

For more information, see Including Licensing with Your Controls.

The CAB file must be present. You must remove the license entry from your system registry to make sure the page's .lpk file will enable your control.

For more information, see Including Licensing with Your Controls.

The .lpk file must remain on your machine to make sure it will enable your control, but you must remove the license entry in your system registry.

For more information, see Including Licensing with Your Controls.

The will prevent browsers that do not recognize the <SCRIPT> tag from displaying the VBScript code.

For more information, see Adding Code to an HTML Page.

VBScript code is contained within the HTML <SCRIPT> tag, but this does not prevent browsers that do not recognize the <SCRIPT> tag from displaying the VBScript code. VBScript code is embedded within HTML comment tags to prevent browsers that do not recognize the <SCRIPT> tag from displaying the VBScript code.

For more information, see Adding Code to an HTML Page.

This should be done, but will not prevent browsers that do not recognize the <SCRIPT> tag from displaying the VBScript code. VBScript code is embedded within HTML comment tags so browsers that do not recognize the <SCRIPT> tag will not display the code.

For more information, see Adding Code to an HTML Page.

VBScript is a language, not an object. The <PARAM> tag is used to set initial properties when an object is created. VBScript code is embedded within HTML comment tags so browsers that do not recognize the <SCRIPT> tag will not display the code.

For more information, see Setting Initial Control Properties.

</SCRIPT> is correct. The first line of code should be changed to identify the VBScript language.

For more information, see Adding Code to an HTML Page.

The comment tags are used to make sure browsers that can't recognize the <SCRIPT> tag don't display the VBScript code. The first line of this code should be changed to identify the VBScript language.

For more information, see Adding Code to an HTML Page.

The <PARAM> tag is used to set initial properties when a control is created. The first line of this code should be modified to identify the VBScript language.

For more information, see Setting Initial Control Properties.

The <SCRIPT> parameter must tell the browser which scripting language interpreter to use when running the code.

For more information, see Adding Code to an HTML Page.

The <OBJECT> tag is correct. A separate <PARAM> tag must be used for each property to be set.

For more information, see Setting Initial Control Properties.

The <BJECT> tag is correct, but you must use a separate <PARAM> tag for each property to be set.

For more information, see Setting Initial Control Properties.

After identifying the object with the <OBJECT> tag, you must use a separate <PARAM> tag for each property to be set.

For more information, see Setting Initial Control Properties.

The quotation marks are correct, but you just use a separate <PARAM> tag for each property to be set.

For more information, see Setting Initial Control Properties.

You can mark a control as safe for initialization and safe for scripting with the Setup Wizard, but you must use SIGNCODE.EXE to apply your digital signature to a CAB file.

For more information, see Code Safety.

The SIGNCODE.EXE utility allow you to apply your digital signature to a CAB file.

For more information, see Signing a Control.

This utility creates license package files. You must use SIGNCODE.EXE to apply your digital signature to a CAB file.

For more information, see Including Licensing with Your Controls.

Authenticode (TM) is the technology derived from public-key signature algorithms that makes code signing possible, but you must use SIGNCODE.EXE to apply your digital signature to a CAB file.

For more information, see Signing a Control.

The change is automatically written to the database.

For more information, see Accessing Data with the Data Control.

The change is automatically written to the database.

For more information, see Accessing Data with the Data Control.

The change is automatically written to the database.

For more information, see Accessing Data with the Data Control.

The change is automatically written to the database.

For more information, see Accessing Data with the Data Control.

The DatabaseName property is set to the name of the directory containing the database files.

For more information, see Accessing Data with the Data Control.

The RecordSource property is set to the name of a specific table within a database or to an SQL string. The DatabaseName property is set to the name of the directory containing the database files.

For more information, see Accessing Data with the Data Control.

The DataSource property of bound controls is set to a data control at design time. The DatabaseName property is set to the name of the directory containing the database files.

For more information, see Accessing Data with the Data Control.

The DataField property specifies which field you want to display in a bound control. The DatabaseName property is set to the name of the directory containing the database files.

For more information, see Accessing Data with the Data Control.

The primary key for each table is a field or group of fields unique for each row. Rows in a Jet database are known as records.

For more information, see Understanding Database Fundamentals.

Fields are the columns in a Jet database, each containing a single piece of data for each record. Rows in a Jet database are known as records.

For more information, see Understanding Database Fundamentals.

Indexes are sorted lists used to speed up database searches. Rows in a Jet database are known as records.

For more information, see Understanding Database Fundamentals.

Rows in a Jet database are known as records.

For more information, see Understanding Database Fundamentals.

Each row in a table is usually unique, but to contain a single piece of information, each record would consist of a single field.

For more information, see Understanding Database Fundamentals.

If a table has a single record, the record might contain many fields, or pieces of information. To contain a single piece of information, each record would consist of a single field.

For more information, see Understanding Database Fundamentals.

A filed contains a single piece of information. To contain a single piece of information, each record would consist of a single field, or piece of information.

For more information, see Understanding Database Fundamentals.

A field contains a single piece of information. To contain a single piece of information, each record would consist of a single field.

For more information, see Understanding Database Fundamentals.

An index is a sorted list that makes access to a table faster.

For more information, see Understanding Database Fundamentals.

An index is a sorted list that makes access to a table faster.

For more information, see Understanding Database Fundamentals.

An index is a sorted list that makes access to a table faster.

For more information, see Understanding Database Fundamentals.

An index is a sorted list that makes access to a table faster.

For more information, see Understanding Database Fundamentals.

The deleted record will be the current record until you move to a different record.

For more information, see Using Recordset Properties and Methods.

The deleted record will be the current record until you move to a different record.

For more information, see Using Recordset Properties and Methods.

The deleted record will be the current record until you move to a different record.

For more information, see Using Recordset Properties and Methods.

The deleted record will be the current record until you move to a different record.

For more information, see Using Recordset Properties and Methods.

The Reposition event occurs when Visual Basic repositions the current record pointer and when a database is first opened.

For more information, see Using Data Control Events.

The Validate event occurs just before Visual Basic writes changes from bound controls to a database and repositions the current record pointer. The Reposition event occurs when Visual Basic repositions the current record pointer and when a database is first opened.

For more information, see Using Data Control Events.

Connect is a data control property that specifies the type of database to open. The Reposition event occurs when Visual Basic repositions the current record pointer and when a database is first opened.

For more information, see Using Data Control Events.

BOF is a data control property. When True, the BOF property determines what action the data control takes. The Reposition event occurs when Visual Basic repositions the current record pointer and when a database is first opened.

For more information, see Using Data Control Events.

The ListField property specifies the name of the field containing the data to be displayed in a data-bound list box.

For more information, see Using the DBCombo Control.

The DataField property contains the name of the field to be updated when a user adds or modifies a record. The ListField property specifies the name of the field containing the data to be displayed in a data-bound list box.

For more information, see Using the DBCombo Control.

The RowSource property specifies the name of a data control. The ListField property specifies the name of the field containing the data to be displayed in a data-bound list box.

For more information, see Using the DBCombo Control.

The RecordSource property specifies the name of the table containing the field to be displayed in a data-bound list box. The ListField property specifies the name of the field containing the data to be displayed in a data-bound list box.

For more information, see Using the DBCombo Control.

Invoke is an **IDispatch function**.

For more information, see The IDispatch Interface.

Release is a function of the **IUnknown** interface used to decrement an object's reference count.

For more information, see Using Interfaces.

IDispatch is a standard COM interface designed to expose component methods and properties to clients.

For more information, see The IDispatch Interface.

Every COM object supports the **IUnknown** interface.

For more information, see Using Interfaces.

GUIDs are created when a component is compiled.

For more information, see Client and Server Communication.

GUIDs are created when a component is compiled.

For more information, see Client and Server Communication.

GUIDs are created when a component is compiled.

For more information, see Client and Server Communication.

GUIDs are created when a component is compiled.

For more information, see Client and Server Communication.

Stub code is added to a COM-compliant server to enable communication with clients.

For more information, see Client and Server Communication.

GUIDs are generated at compile time to uniquely identify interfaces but do not aid in the communication process.

For more information, see Client and Server Communication.

Proxy code is involved in marshaling parameters and return values back and forth across the process boundary, but proxy code resides on the client.

For more information, see Client and Server Communication.

AddRef is a function of the **IUnknown** interface that increments the object's reference count.

For more information, see Using Interfaces.

This is a feature of an out-of-process component.

For more information, see Determining Where Server Components Run.

This is a feature of an out-of-process component.

For more information, see Determining Where Server Components Run.

In-process components run in the same address space as clients.

For more information, see Determining Where Server Components Run.

This is a feature of an out-of-process components.

For more information, see Determining Where Server Components Run.

The **IDispatch** interface allows you to modify the list of properties and methods supported by an object without changing the interface.

For more information, see Implementing Automation.

IUnknown defines three functions all interfaces must provide; **AddRef**, **Release** and **QueryInterface**.

For more information, see Using Interfaces.

Invoke is an **IDispatch** function.

For more information, see The IDispatch Interface.

AddRef is a function of the **IUnknown** interface that increments the object's usage count.

For more information, see Using Interfaces.

You cannot access an object with the Object Browser until you set a reference to its type library.

For more information, see Setting References.

You must reference a component before invoking the Object Browser.

For more information, see Setting References.

You can declare an object variable as data type Object without setting a reference to the component's type library, but the compiler will use late binding and you cannot access information with the Object Browser.

For more information, see Setting References.

You can declare an object variable as data type Variant without setting a reference to the component's type library, but the compiler will use late binding and you cannot access information with the Object Browser.

For more information, see Setting References.

Late binding results in the client making two calls to an object at run time to access a property or method, and results in slower execution than dispID or vTable binding.

For more information, see Introduction to Binding.

dispID binding is faster than Late binding, but vTable binding results in the fastest execution.

For more information, see Introduction to Binding.

vTable binding results in the fastest execution.

For more information, see Introduction to Binding.

Early binding allows for the compiler to optimize automation code. However, it is specifically the vTable binding form of early binding that provides the fastest execution.

For more information, see Introduction to Binding.

Declaring an object variable as data type Object will result in late binding and syntax will not be checked at design time.

For more information, see Introduction to Binding.

Declaring an object variable as data type Variant will result in late binding and syntax will not be checked at design time.

For more information, see Introduction to Binding.

The server's **Instancing** property is not relevant; an explicit object type allows the Visual Basic compiler to check syntax at design time.

For more information, see Characteristics of Server Components.

An explicit object type allows the Visual Basic compiler to check syntax at design time. This also requires that a valid reference to the component's type library be set.

For more information, see Introduction to Binding.

The **Parent** property refers to the object one level higher in the object model; in this case, the **Application** object.

For more information, see The Microsoft Excel Object Model.

The **Parent** property refers to the object one level higher in the object model; in this case, the **Application** object.

For more information, see The Microsoft Excel Object Model.

The **Parent** property refers to the object one level higher in the object model; in this case, the **Application** object.

For more information, see The Microsoft Excel Object Model.

The **Parent** property refers to the object one level higher in the object model; in this case, the **Application** object.

For more information, see The Microsoft Excel Object Model.

The usage count is incremented once for each **Set** statement.

For more information, see Creating Objects.

The usage count is incremented once for each **Set** statement.

For more information, see Creating Objects.

The usage count is incremented once for each **Set** statement.

For more information, see Creating Objects.

The usage count is incremented once for each **Set** statement.

For more information, see Creating Objects.

The term handshake implies two-way communication, whereas an event server broadcasts to an unknown audience.

For more information, see Events vs. Callbacks.

The term targeted implies that the recipient of the communication is known, whereas an event server broadcasts to an unknown audience.

For more information, see Events vs. Callbacks.

Conceptually, an event server broadcasts to an unknown audience.

For more information, see Events vs. Callbacks.

A run-time error will occur if Excel is not active and **GetObject** is used without a first argument.

For more information, see Creating an Instance of Microsoft Excel.

The compiler does not know if Excel will be active when this code runs or not; a run-time error will occur if Excel is not active and **GetObject** is used without a first argument.

For more information, see Creating an Instance of Microsoft Excel.

If a filename is included in the first argument, an instance will be returned with the **Visible** property set to False; a run-time error will occur if Excel is not active and **GetObject** is used without a first argument.

For more information, see Creating an Instance of Microsoft Excel.

If a filename is included in the first argument, an instance will be returned with the **Visible** property set to False; a run-time error will occur if Excel is not active and **GetObject** is used without a first argument.

For more information, see Creating an Instance of Microsoft Excel.

This will compile the project defined in the ACCPROJ make file to an executable named ACC.EXE.

For more information, see Compiling an EXE File.

You must use the /make switch, as shown in option a, to let the compiler know where to find your project files.

For more information, see Compiling an EXE File.

You must use the /make switch, as shown in option a, to let the compiler know where to find your project files.

For more information, see Compiling an EXE File.

You must use the /make switch, as shown in option a, to let the compiler know where to find your project files, and the make file should be listed first.

For more information, see Compiling an EXE File.

If you do not specify a property, you set the value of a control. The value of a text box is the text property.

For more information, see Setting Properties.

If you do not specify a property, you set the default property of a control. The default property of a text box is the text property.

For more information, see Setting Properties.

If you do not specify a property, you set the value of a control. The value of a text box is the text property.

For more information, see Setting Properties.

If you do not specify a property, you set the value of a control. The value of a text box is the text property.

For more information, see Setting Properties.

The primary difference between subs and functions is that functions can return values, but subs do not.

For more information, see Creating Procedures.

You can pass arguments to both subs and functions, but only functions can return a value.

For more information, see Creating Procedures.

You can pass arguments to both subs and functions, but only functions can return a value.

For more information, see Creating Procedures.

The primary difference between subs and functions is that functions can return values, but subs do not.

For more information, see Creating Procedures.

A private procedure in a form can be called only by other procedures located in the same form.

For more information, see Scope of Code.

A private procedure in a form can be called only by other procedures located in the same form.

For more information, see Scope of Code.

A private procedure in a form can be called only by other procedures located in the same form.

For more information, see Scope of Code.

A private procedure in a form can be called only by other procedures located in the same form.

For more information, see Scope of Code.

A variable declared in a procedure can only be used in that procedure.

For more information, see Scope of Variables.

A variable declared in a procedure can only be used in that procedure.

For more information, see Scope of Variables.

A variable declared in a procedure can only be used in that procedure.

For more information, see Scope of Variables.

A variable declared in a procedure can only be used in that procedure.

For more information, see Scope of Variables.

Form-level variables retain their value when a form is unloaded, but the variable will be cleared if the form is set to Nothing. To make a variable available the entire time your application is running, you should declare it in a standard module.

For more information, see Life of Variables.

Form-level variables retain their value when a form is unloaded, but the variable will be cleared if the form is set to Nothing. To make a variable available the entire time your application is running, you should declare it in a standard module.

For more information, see Life of Variables.

To make a variable available the entire time your application is running, you should declare it in a standard module.

For more information, see Life of Variables.

A non-static variable will be reinitialized if the procedure is reloaded. To make a variable available the entire time your application is running, you should declare it in a standard module.

For more information, see Life of Variables.

Break expressions halt program execution when certain conditions are met. Watch expressions are used to monitor a particular variable or expression in your code. The values of watch expressions are updated at each breakpoint.

For more information, see Tools for Debugging.

Watch expressions are used to monitor a particular variable or expression in your code. The values of watch expressions are updated at each breakpoint.

For more information, see Tools for Debugging.

Step options are used to run code one statement or procedure at a time. Watch expressions are used to monitor a particular variable or expression in your code. The values of watch expressions are updated at each breakpoint.

For more information, see Tools for Debugging.

Call Stack is used to view all active procedure calls and trace a series of nested procedures. Watch expressions are used to monitor a particular variable or expression in your code. The values of watch expressions are updated at each breakpoint.

For more information, see Tools for Debugging.

Break expressions halt program execution when certain conditions are met. The Call Stack is used to view all active procedure calls and trace the execution of a series of nested procedures.

For more information, see Tools for Debugging.

Watch expressions are used to monitor a particular variable or expression in your code. The Call Stack is used to view all active procedure calls and trace the execution of a series of nested procedures.

For more information, see Tools for Debugging.

Step options are used to run code one statement or procedure at a time. The Call Stack is used to view all active procedure calls and trace the execution of a series of nested procedures.

For more information, see Tools for Debugging.

The Call Stack is used to view all active procedure calls and trace the execution of a series of nested procedures.

For more information, see Tools for Debugging.

Select this template to build an ActiveX control. An in-process code component requires the ActiveX DLL template.

For more information, see Choosing the Type of Code Component.

Select this template to build an in-process code component.

For more information, see Choosing the Type of Code Component.

Select this template to build an out-of-process code component. An in-process code component requires the ActiveX DLL template.

For more information, see Choosing the Type of Code Component.

Select this template to build a standalone application. An in-process code component requires the ActiveX DLL template.

For more information, see Choosing the Type of Code Component.

This generates new type library information each time a component is compiled, but retains the type library identifier. You must set the StartMode property of the App object to Standalone(0) to keep the server running.

For more information, see Setting Project Properties.

With this value for the StartMode property, the server will end as soon as it is started in the development environment. The value of this property must be set to Standalone(0) to keep the server running.

For more information, see Setting Project Properties.

The code in the Sub Main procedure executes when the code component starts. Setting the StartMode property of the App object to Standalone(0) will keep the project running.

For more information, see Setting Project Properties.

This will keep your project running so you can start a client application.

For more information, see Setting Project Properties.

This results in permanent registration. A code component is temporarily registered when you run it from design mode.

For more information, see Registering a Component.

This results in permanent registration. A code component is temporarily registered when you run it from design mode.

For more information, see Registering a Component.

A code component is temporarily registered when you run it from design mode.

For more information, see Registering a Component.

This results in permanent registration. A code component is temporarily registered when you run it from design mode.

For more information, see Registering a Component.

The Property Let procedure always contains at least one argument...the value for the property. A read-only property can be created by defining only a Property Get procedure.

For more information, see Using Property Procedures to Create Properties.

This creates a property that returns a standard data type. For a read-only property, you must omit the Property Set or Property Let procedure.

For more information, see Using Property Procedures to Create Properties.

This will create a read-only property.

For more information, see Using Property Procedures to Create Properties.

A Property Get procedure must be defined to return the value of the property. A read-only property can be created by defining only a Property Get procedure.

For more information, see Using Property Procedures to Create Properties.

This creates a property that returns a standard data type. To create an Object data type property, define a Property Set procedure instead of a Property Let procedure.

For more information, see Using Property Procedures to Create Properties.

You must define a Property Get procedure to return the property value. To create an Object data type property, define a Property Get procedure and a Property Let procedure.

For more information, see Using Property Procedures to Create Properties.

This defines a read-only property. To create an Object data type property, define a Property Get procedure and a Property Let procedure.

For more information, see Using Property Procedures to Create Properties.

This will create a property that is an Object data type.

For more information, see Using Property Procedures to Create Properties.

If a property is created using a Public variable, code cannot be executed when the property is set or retrieved. The default property of an object can be set without explicitly referring to the object.

For more information, see Creating Properties.

The default property of an object can be set without explicitly referring to the object.

For more information, see Creating Properties.

Property procedures allow code to be executed when the property is set or retrieved. The default property of an object can be set without explicitly referring to the object.

For more information, see Creating Properties.

This is true if a property is created using a property procedure rather than a Public variable. The default property of an object can be set without explicitly referring to the object.

For more information, see Creating Properties.

When the client changes a parameter passed ByVal, the server will *not* see the change.

For more information, see Canceling a Procedure Within an Event.

When the client changes a parameter passed ByVal, the *server* will not see the change.

For more information, see Canceling a Procedure Within an Event.

When the client changes a parameter passed ByRef, the server *will* see the change.

For more information, see Canceling a Procedure Within an Event.

When the client changes a parameter passed ByRef, the server will see the change.

For more information, see Canceling a Procedure Within an Event.

The client needs to read, but not change, the pd (percent done) argument. The cancel argument must be passed ByRef so the component can see changes made to the value of the cancel parameter by the client.

For more information, see Canceling a Procedure Within an Event.

The cancel argument must be passed ByRef so the component can see changes made to the value of the cancel parameter by the client.

For more information, see Canceling a Procedure Within an Event.

If bCancel is set to True, the task will be canceled by the InProgress event. The cancel argument must be passed ByRef so the component can see changes made to the value of the cancel parameter by the client.

For more information, see Canceling a Procedure Within an Event.

The pd argument should be passed as a Long, but the cancel argument must be passed ByRef so the component can see changes made to the value of the cancel parameter by the client.

For more information, see Canceling a Procedure Within an Event.

When used as a standalone application, Word can be considered a document container, but in this case Word is a document server.

For more information, see Common ActiveX Document Containers.

Word can be considered a document container when used as a standalone application, but in this case Word functions as a server.

For more information, see Common ActiveX Document Containers.

When used as a standalone application, Word can be considered a container for its document objects, but in this case Word functions as a server.

For more information, see Common ActiveX Document Containers.

In this case, Word is a document server, lending its functionality to the document object started from Internet Explorer 3.0.

For more information, see Common ActiveX Document Containers.

Visual Basic cannot store graphical elements in .DOB text files, so these elements are stored in .DOX files.

For more information, see Working with ActiveX Document Projects.

The source code and property values of a User Document are stored in a .DOB file, but graphical elements are stored in .DOX files.

For more information, see Working with ActiveX Document Projects.

A .DLL results when you compile an ActiveX document as an in-process component.

For more information, see Working with ActiveX Document Projects.

A Visual Basic Document file results when you compile an ActiveX document, but Visual Basic stores graphical elements of a User Document in a .DOX file.

For more information, see Working with ActiveX Document Projects.

Files with the .DOB extension are used by Visual Basic to store the plain text elements of a User Document.

For more information, see Working with ActiveX Document Projects.

MYPROJ.DLL would be produced if you compiled the project as an in-process component.

For more information, see Working with ActiveX Document Projects.

In addition to the .EXE file, MYPROJ.VBD is produced when the project is compiled.

For more information, see Working with ActiveX Document Projects.

Files with the .DOX extension are used by Visual Basic to store the graphical elements of a User Document.

For more information, see Working with ActiveX Document Projects.

The NavigateTo method must be used.

For more information, see Navigating Between Documents.

GetObject is not a valid method of the Hyperlink object. The NavigateTo method must be used.

For more information, see Navigating Between Documents.

GetDocument is not a valid method of the Hyperlink object. The NavigateTo method must be used.

For more information, see Navigating Between Documents.

MoveTo is not a valid method of the Hyperlink object. The NavigateTo method must be used.

For more information, see Navigating Between Documents.

The Microsoft Binder does not support hyperlinking, but the registry will locate and start an application that does.

For more information, see Navigating Between Documents.

The Microsoft Binder does not support hyperlinking.

For more information, see Navigating Between Documents.

The Microsoft Binder does not support hyperlinking, so the registry locates and starts an alternative application to handle the request.

For more information, see Navigating Between Documents.

The Microsoft Binder does not support hyperlinking, but the request is not ignored. The registry locates and starts an alternative application to handle the request.

For more information, see Navigating Between Documents.

It is not necessary to read the current property value before changing it.

For more information, see Persisting Properties.

The container object must first be notified that a property value has changed using the PropertyChanged method of the UserDocument object.

For more information, see Persisting Properties.

Correct. The WriteProperties event occurs when the PropertyChanged method sets a flag to notify the container that a property value has changed.

For more information, see Persisting Properties.

The *container* must be notified of the change to trigger the WriteProperties event. PropertyChanged is a method of the UserDocument object.

For more information, see Persisting Properties.

The Initialize event occurs before the ActiveX document is sited, and the Parent property is not available.

For more information, see User Document Event Behavior.

The Show event occurs after the ActiveX document is sited, and the Parent property is then available.

For more information, see User Document Event Behavior.

The ReadProperties event occurs only after a property has been saved using the PropertyBag object.

For more information, see User Document Event Behavior.

The Hide event occurs when a user navigates off a document in a browser and just before the Terminate event.

For more information, see User Document Event Behavior.

SETUP.EXE copies the bootstrap files and executes SETUP132.EXE. SETUP132.EXE creates the startup icons for a Windows 95 application.

For more information, see Sample Setup Files.

SETUP1.EXE is the main setup program for 16-bit applications. SETUP132.EXE creates the startup icons for a Windows 95 application.

For more information, see Sample Setup Files.

SETUP132.EXE creates the startup icons for a Windows 95 application.

For more information, see Sample Setup Files.

SETUP.LST contains the list of files required by the application an information on default directories an disk space requirements. SETUP132.EXE creates the startup icons for a Windows 95 application.

For more information, see Sample Setup Files.

An image control requires less overhead than a picture box. You should set AutoRedraw to False and add code to refresh your graphics at reasonable intervals.

For more information, see Tuning Your Application.

This will cause your display to be redrawn every time data changes. To improve performance, you should set AutoRedraw to False and add code to refresh your status graphic at reasonable intervals.

For more information, see Tuning Your Application.

A hidden graphic will display faster than one that must be loaded. You should set AutoRedraw to False and add code to refresh your status graphic at reasonable intervals.

For more information, see Tuning Your Application.

In this scenario, your status might undergo continuous refreshing if AutoRedraw is set to True. You should set AutoRedraw to False and add code to refresh the graphic at reasonable intervals.

For more information, see Tuning Your Application.

You can reclaim memory from strings by setting the strings to “”.

For more information, see Tuning Your Application.

Erase eliminates an array from memory. You can reclaim memory from strings by setting the strings to “”.

For more information, see Tuning Your Application.

You can reclaim memory from strings by setting the strings to “”.

For more information, see Tuning Your Application.

You can reclaim memory by setting the strings to “”.

For more information, see Tuning Your Application.

This adds files to SourceSafe and copies files to the SourceSafe project. When you select SccAddin.SourceCodeControlAddin, the SourceSafe menu is added to the Visual Basic Add-Ins menu, and the Get, Check Out, and Check In commands are enabled.

For more information, see Using Visual SourceSafe.

When you select SccAddin.SourceCodeControlAddin, the SourceSafe menu is added to the Visual Basic Add-Ins menu, and the Get, Check Out, and Check In commands are enabled.

For more information, see Using Visual SourceSafe.

When you select SccAddin.SourceCodeControlAddin, the SourceSafe menu is added to the Visual Basic Add-Ins menu, and the Get, Check Out, and Check In commands are enabled.

For more information, see Using Visual SourceSafe.

A bitmap is indicated if the second argument of the LoadResPicture function is 0. A cursor is being loaded if the second argument is 2.

For more information, see Using Resource Files.

A cursor is being loaded if the second argument is 2.

For more information, see Using Resource Files.

An icon is indicated if the second argument of the LoadResPicture function is 1. A cursor is being loaded if the second argument is 2.

For more information, see Using Resource Files.

A cursor is being loaded if the second argument is 2.

For more information, see Using Resource Files.

The DataArrival event is used to declare a variable to contain incoming data and to invoke the GetData method. This code should run when the ConnectionRequest event occurs.

For more information, see Coding a Winsock Server.

The code shown should run when the ConnectionRequest event occurs. The Accept method is used to accept the connection to the client.

For more information, see Coding a Winsock Server.

Listen is a method often used in the Form_Load event. The code shown should run when the ConnectionRequest event occurs.

For more information, see Coding a Winsock Server.

The Form_Load event is often used to set the LocalPort property and invoke the Listen method. The code shown should run when the ConnectionRequest event occurs.

For more information, see Coding a Winsock Server.

The GoSearch method is used to navigate sites in the history list. The Navigate method is typically used in the Form_Load event to specify a home page.

For more information, see Basic Operations of the WebBrowser Control.

The Navigate method is typically used in the Form_Load event to specify a home page.

For more information, see Basic Operations of the WebBrowser Control.

The GoHome method is used to navigate to the home page in the history list. The Navigate method is typically used in the Form_Load event to specify a home page.

For more information, see Basic Operations of the WebBrowser Control.

LocationURL is a property rather than a method, and is used when the NavigationComplete event occurs to display the URL of a resource. The Navigate method is typically used in the Form_Load event to specify a home page.

For more information, see Basic Operations of the WebBrowser Control.

The Navigate method is typically used to specify a client's home page in the Form_Load event. The Refresh2 method can be used to reload a displayed page from the server.

For more information, see Using the WebBrowser Control.

The Refresh method reloads a displayed page from the client machine's cache. The Refresh2 method can be used to reload a displayed page from the server.

For more information, see Using the WebBrowser Control.

With the appropriate parameter, the Refresh2 method can be used to reload a currently displayed page from the server where it originated.

For more information, see Using the WebBrowser Control.

The GoBack method is used to reload a previously displayed page from the history file. The Refresh2 method can be used to reload a currently displayed page from the server where it originated.

For more information, see Basic Operations of the WebBrowser.

The LocationURL property is used to display the URL of a resource. You must set the browser.Visible property to True to make the InternetExplorer visible.

For more information, see Basic Operations of the WebBrowser.

The GoHome method is used to display a browser's home page from the client's history file. You must set the browser.Visible property to True to make the InternetExplorer visible.

For more information, see Basic Operations of the WebBrowser.

Like other ActiveX components, InternetExplorer is invisible when it starts, and you must set the Visible property to True.

For more information, see Using the InternetExplorer Object.

The Navigate method is used to specify a home page, but you must set the Visible property to True to see an instance of InternetExplorer.

For more information, see Using the WebBrowser Control.

```
Private Sub optNonTopmost_Click()
  SetWindowPos frmtopmost.hwnd, _
    HWND_NOTOPMOST, 0, 0, 0, 0, FLAGS
End Sub

Private Sub optTopmost_Click()
  SetWindowPos frmtopmost.hwnd, _
    HWND_TOPMOST, 0, 0, 0, 0, FLAGS
End Sub
```

The GetChunk method is used when coding an FTP client to move data from the control's buffer to a final location. The OpenURL method is used to open a connection with an HTTP host.

For more information, see Coding for Asynchronous Operation.

The Execute method is used to begin the transfer of data when coding a client for FTP functionality. The OpenURL method is used to open a connection with an HTTP host.

For more information, see Coding for Asynchronous Operation.

The Accept method is used when coding a Winsock server to accept a connection to a client. The OpenURL method is used to open a connection with an HTTP host.

For more information, see Coding a Winsock Server.

The OpenURL method is used to open a connection with an HTTP host.

For more information, see Basic Operations of the Internet Transfer Control.

The GetChunk method is used to retrieve data from a controls buffer. DoEvents passes control to the operating system so other events in its queue can be processed while the control is busy.

For more information, see Basic Operations of the Internet Transfer Control.

The Execute method is used to initiate a request to a remote server. DoEvents passes control to the operating system so other events in its queue can be processed while the control is busy.

For more information, see Coding for Asynchronous Operation.

The StillExecuting property returns True when the control is engaged in as operation such as retrieving a file from the Internet, and the DoEvents function passes control to the operating system so other event in its queue can be processed.

For more information, see Coding for Asynchronous Operation.

In a 32-bit version of Visual Basic, use the GetSetting function to read a setting from the Registry or an application specific .INI file. Use the Windows function GetProfileString to read a string in WIN.INI.

For more information, see Saving Application Settings.

The GetPrivateProfileString function reads an entry from an .INI file other than WIN.INI. Use the Windows function GetProfileString to read a string in WIN.INI.

For more information, see Saving Application Settings.

Use the Windows function GetProfileString to read a string in WIN.INI.

For more information, see Saving Application Settings.

Use this function to read a string in WIN.INI.

For more information, see Saving Application Settings.

You can view code samples from within Chapter 1 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp}	KeyPress Event
{ewc MVIMG, MVIMAGE,!code.bmp}	Enabling the OK Button
{ewc MVIMG, MVIMAGE,!code.bmp}	Validating all Fields on a Form
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code

You can view code samples from within Chapter 3 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp} Determining Recordset Type

{ewc MVIMG, MVIMAGE,!code.bmp} Printing Data from a Recordset

{ewc MVIMG, MVIMAGE,!code.bmp} Using the IsNull Function

{ewc MVIMG, MVIMAGE,!code.bmp} Checking the EOF Property

{ewc MVIMG, MVIMAGE,!code.bmp} Checking the Current EditMode

{ewc MVIMG, MVIMAGE,!code.bmp} Searching for a Specific Record

{ewc MVIMG, MVIMAGE,!code.bmp} Using Various SQL Statements

{ewc MVIMG, MVIMAGE,!code.bmp} SQL Insert Statement

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

You can view code samples from within Chapter 4 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp} Using Rule Properties

{ewc MVIMG, MVIMAGE,!code.bmp} Messages for Rule Violations

{ewc MVIMG, MVIMAGE,!code.bmp} Handling Referential Integrity Violations

{ewc MVIMG, MVIMAGE,!code.bmp} Handling Locking Errors

{ewc MVIMG, MVIMAGE,!code.bmp} Handling Errors on the Update Statement

{ewc MVIMG, MVIMAGE,!code.bmp} Handling Errors When Accessing Deleted Records

{ewc MVIMG, MVIMAGE,!code.bmp} Opening a Table Directly

{ewc MVIMG, MVIMAGE,!code.bmp} Using File I/O to Read a File

{ewc MVIMG, MVIMAGE,!code.bmp} Creating and Using an ODBCDirect Workspace

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

You can view code samples from within Chapter 5 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp} Creating a Topmost Window

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

You can view code samples from within Chapter 6 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp}	Creating Variables
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code

You can view code samples from within Chapter 7 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp}	Creating an Instance of a Class
{ewc MVIMG, MVIMAGE,!code.bmp}	Component Error Handling Code Sample
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code

You can view code samples from within Chapter 8 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

You can view code samples from within Chapter 9 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

{ewc MVIMG, MVIMAGE,!code.bmp} Lab Solution Code

You can view code samples from within Chapter 12 or you can view sample code from here by clicking an icon or title below:

{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code
{ewc MVIMG, MVIMAGE,!code.bmp}	Lab Solution Code

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,!anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,!anim.bmp} Working with the Recordset Object

{ewc MVIMG, MVIMAGE,!democlip.bmp} Retrieving a Recordset

{ewc MVIMG, MVIMAGE,!democlip.bmp} Adding and Editing Records

{ewc MVIMG, MVIMAGE,!democlip.bmp} Lab 3: Using Data Access Objects

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlip.bmp} Referential Integrity

{ewc MVIMG, MVIMAGE,! democlip.bmp} Handling Locking Errors

{ewc MVIMG, MVIMAGE,! democlip.bmp} Retrieving External Data with ODBCDirect

{ewc MVIMG, MVIMAGE,! democlip.bmp} Lab 4: Advanced Database Development

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,!anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,!anim.bmp} Class Instancing

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating and Using Classes

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating and Using a Code Component

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating and Using Events

{ewc MVIMG, MVIMAGE,!democlip.bmp} Making Asynchronous Calls with Events

{ewc MVIMG, MVI Lab 7: Creating ActiveX Code Components

MAG
E.!
dem
oclip.
bmp}

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlip.bmp} Using the ActiveX Document Migration Wizard

{ewc MVIMG, MVIMAGE,! democlip.bmp} Lab 10: Creating and Using ActiveX Documents

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,! animbmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,! democlipbmp} Using the Winsock Control

{ewc MVIMG, MVIMAGE,! democlipbmp} FTP and HTTP with the WinINet Control

{ewc MVIMG, MVIMAGE,! democlipbmp} Lab 11: Creating an Internet-Aware Application

You can view multimedia from within the chapter or jump directly to an animation or demonstration by clicking an icon or topic title below:

{ewc MVIMG, MVIMAGE,!anim.bmp} Chapter Introduction

{ewc MVIMG, MVIMAGE,!democlip.bmp} The Setup Wizard

{ewc MVIMG, MVIMAGE,!democlip.bmp} Using the Code Profiler

{ewc MVIMG, MVIMAGE,!democlip.bmp} Creating an Event Log

{ewc MVIMG, MVIMAGE,!democlip.bmp} Lab 12: Application Setup and Optimzation

This section describes various Microsoft and third-party developer resources that may be of use to you when designing and building applications with Visual Basic.

® Microsoft Programs

Details how to become a Microsoft Certified Professional and Microsoft Solution Provider, and tells how to participate in Microsoft logo programs.

® Microsoft Technical Support

Points you to the *Mastering Microsoft Visual Basic 5* Help file for information about online, CD-ROM-, Fax-, and phone-based support for Microsoft products.

® Microsoft Technical Training and Education

Describes the training options available through Microsoft—face-to-face, self-paced, and classes offered through the Microsoft Online Institute.

® Lists and References

Provides a listing of user groups and mailing lists that might be of interest to Visual Basic Developers.

® Publishers

Provides information about Microsoft Press and third-party publishers who create books and periodicals that include Visual Basic development topics.

® World Wide Web Sites

Provides pointers to many Microsoft Web sites that might be of interest to Visual Basic programmers.

Microsoft Visual Basic 5.0 contains a large selection of sample applications. They demonstrate everything from simple programming operations for beginners to more advanced topics that will be of interest to the experienced Visual Basic programmer. All of these sample programs can be found in the Visual Basic 5.0 Samples folder under Visual Basic.

Mastering Microsoft Visual Basic 5 also contains a number of sample applications that are contained in the SampleApps folder on the CD. The following is a short description of these applications.

CancelAsynch

This application demonstrates the creation and use of asynchronous tasks in a code component. Using the GlobalTimer component, the component regularly uses events to notify its client of the state of the task and to give the client the ability to cancel the operation before completion.

Event Notification

This application demonstrates a component which sends asynchronous events to a client. In this case, the component uses the Timer control to generate events.

In practice, if your component already contains a form, the control can be placed and used with no additional overhead. However, if no form is required by the component, using a Timer control requires adding a form to act as the control's parent. This will cause unwanted overhead. To avoid this use the Window's timers directly or indirectly through a code component as demonstrated by the GlobalTimer component.

GlobalTimer

This ActiveX DLL is a code component that uses Window's timers to allow clients access to timer functionality without the overhead of the Timer control. This sample demonstrates using the Window's timers, callback functions, and a Collection class.

ObjectSharing

This application demonstrates one method for multiple clients to share a single object. It also illustrates creating an abstract class to define an interface and implementing it in the object class.

In this case, a multi-use server is created and it instantiates one instance of a SomeObject component (a constituent component). As new clients connect, the server hands out references to the same object. When a client changes the Text property, the object notifies all clients of the change by raising a TextChanged event. On the client side, as one client changes properties, all clients see the change.

To test this application, you must register the server or run the server in the development environment. Then run two or more instances of the client. When setting the text in one client, you should observe the changing text in the others.

The course contains a number of code samples.

When you see this icon {ewc mvimg, mvimage,!code.bmp} in the course contents, click it to view the sample code.

This course includes a number of self-check questions at the end of each chapter. You can use these multiple-choice questions to test your understanding of the information that has been covered in the course.

To see whether you answered a question correctly, click this icon.
{ewc MVIMG, MVIMAGE,!answer.bmp}

Each answer also contains a jump to the associated chapter, so you can easily review the content.

You can either review the self-check questions at the end of each chapter, or you can jump to them directly by clicking the following chapter titles.

Chapter 1: Visual Basic Review

Chapter 2: Using the Data Control

Chapter 3: Using Data Access Objects

Chapter 4: Advanced Database Development

Chapter 5: Using Dynamic-Link Libraries

Chapter 6: Creating ActiveX Clients

Chapter 7: Creating ActiveX Code Components

Chapter 8: Creating ActiveX Controls

Chapter 9: Using ActiveX Components on a Web Page

Chapter 10: Creating and Using ActiveX Documents

Chapter 11: Creating Internet-Aware Applications

Chapter 12: Application Setup and Optimization