

Check or Add an Object Library Reference

If you use the objects in other applications as part of your Visual Basic application, you may want to establish a reference to the [object libraries](#) of those

applications. Before you can do that, you must first be sure that the application provides an object library.

To see if an application provides an object library

From the **Tools** menu, choose **References** to display the **References** dialog box.

The **References** dialog box shows all object libraries registered with the operating system. Scroll through the list for the application whose object library you want to reference. If the application isn't listed, you can use the **Browse** button to search for object libraries (*.olb and *.tlb) or [executable files](#) (*.exe and *.dll on Windows). References whose check boxes are checked are used by your [project](#); those that aren't checked are not used, but can be added.

To add a object library reference to your project

Select the object library reference in the **Available References** box in the **References** dialog box and click **OK**.

Your Visual Basic project now has a reference to the application's object library. If you open the [Object Browser](#) (press F2) and select the application's library, it displays the objects provided by the selected object library, as well as each object's [methods](#) and [properties](#). In the **Object Browser**, you can select a [class](#) in the **Classes** box and select a method or property in the **Members** box. Use copy and paste to add the syntax to your code.

Continue Code Execution

When you run your code, execution may stop if:

An untrapped [run-time error](#) occurs.

A trapped run-time error occurs, and **Break on All Errors** is selected on the **General** tab of the **Options** dialog box (**Tools** menu).

A previously set [breakpoint](#) is encountered.

A **Stop** statement in your code is encountered, switching the mode to [break mode](#).


An **End** statement in your code is encountered, switching the mode to [design time](#).

You halt execution manually at a given point.


A [watch expression](#), which you set to break when the value has changed or break when the value is true, is encountered.

To halt execution manually

To switch to break mode, choose **Break** (CTRL+BREAK) from the **Run** menu, or use the toolbar shortcut: .

To switch to design time, choose **Reset <projectname>** from the **Run** menu, or use the toolbar shortcut: .

To continue execution when your application has halted

On the **Run** menu, click **Continue** (F5), or use the toolbar shortcut: .

- Or -

On the **Debug** menu, click **Step Into** (F8), **Step Over** (SHIFT+F8), **Step Out** (CTRL+SHIFT+F8), or **Run To Cursor** (CTRL+F8)(.

To continue execution when your application has halted because of a handled error

Press ALT+F8 to step through the error-handler.

- Or -

Press ALT+F5 to resume execution by running through the error-handler.

Sometimes you may want to copy a useful example from Visual Basic Help. While many of the examples require more code to work correctly, some examples are useful to see how a particular [procedure](#) or control flow technique behaves.

To copy example code from Help to your application

Use Help to display the topic for the language element whose example you want to use.

Click the Example link in the non-scrolling region near the top of the page. The code example is displayed.

Right-click and select the part of the code you want to copy to your application. On the Macintosh, choose **Copy** from the **Edit** menu; a window appears in which you can select the code to copy.

Choose **Copy** from the shortcut menu. On the Macintosh, press the **Copy** button..

Move the focus back to the **Code** window and position the mouse pointer where you want the code example to be inserted.

Right-click again and choose **Paste** to insert the code example into the **Code** window.

Tip You can also press CTRL+C to copy a selected example in a Help window. Press CTRL+V to paste the example into the **Code** window.

Create a Procedure

Code within a [module](#) is organized into [procedures](#). A procedure tells the application how to perform a specific task. Use procedures to divide complex code tasks into more manageable units.

To create a procedure by writing code

Open the module for which you want to write the procedure.

You can create a **Sub**, **Function**, or **Property** procedure.

Type **Sub**, **Function**, or **Property**.

Press F1 to get Help with syntax, if necessary.

Type code for the procedure.

Visual Basic concludes the procedure with the appropriate **End Sub**, **End Function**, or **End Property** statement.

To create a procedure using the Insert Procedure dialog box

Open the module for which you want to write the procedure.

On the **Insert** menu, click **Procedure**.

Type the name for the procedure in the **Name** box of the **Insert Procedure** dialog box.

Select the type of procedure you want to create: Sub, Function, or Property.

Set the procedure's scope to either Public or Private.

You can select the **All Local Variables as Statics** to add the **Static** [keyword](#) to the procedure definition.

Click **OK**.

Enter a Declaration in Code

Declarations are nonexecutable code statements that name external [procedures](#), [constants](#), or [variables](#) and define their attributes (such as [data type](#)). You write declarations for [form](#), [standard](#), or [class](#) modules. To enter [module-level](#) declarations, go to the Declarations section of a module. To enter global declarations, go to the Declarations section of a module and use the **Public** statement for constants and variables. You can also use the **Dim**, **Static**, and **Private** [keywords](#) to make declarations.

You can also enter [procedure-level](#) declarations. For whatever code level and technique you use to declare a variable or constant, specific [scoping](#) rules may apply.

To open the Declarations section of a module

In the **Project** window, select the form, standard, or class module you want

to open and click the **View Code** button.

- Or -

Right-click and choose **View Code** from the context menu.

In the [Object box](#), select **(General)**.

The [Procedure box](#) automatically displays **(Declarations)**.

Enter one or more declarations.

Execute a Specific Statement

While execution of your code is halted, you can control the execution sequence of [statements](#) within a [procedure](#). You can resume execution at a statement you choose without executing any intervening code.

To set the next statement to be executed

In the **Code** window, position the insertion point anywhere within the statement.

On the **Debug** menu, click **Set Next Statement** (CTRL+F9).

- or on Windows -

Position the mouse pointer in the [margin indicator](#) next to the current execution point.

Drag the yellow arrow in the margin indicator to the statement you want to

execute next.

Note You can only skip to statements within the same procedure.

Used in combination with **Step Into**, executing specific statements with the **Set Next Statement** command enables you to step through procedures one statement at a time, and to closely examine your code. It's also helpful for correcting or avoiding [run-time error](#) conditions.

Find a Procedure

To find a procedure in the Code window

To view an existing general procedure, select **(General)** in the [Object box](#) in the **Code** window, and then select the procedure in the [Procedure box](#).

To view an event procedure, select the appropriate object in the **Object** box in the **Code** window, and then select the event in the **Procedure** box.

Note To visually separate [procedures](#) in the **Code** window, you can select the **Procedure Separator** check box on the **Editor** tab of the **Options** dialog box (**Tools** menu). You can switch between Procedure view and Full Module view using the buttons in the lower-left corner of the **Code** window.

To find a procedure in another module

On the **View** menu, click **Object Browser**, or press F2.

Select the project in the **Project/Library** box.

Select the module in the **Classes** list.

Double-click the procedure name in the **Members of** list.

The selected procedure is displayed in the **Code** window.

You can use the following keyboard shortcuts:

Press	To
CTRL+DOWN ARROW	Display the next procedure.
CTRL+UP ARROW	Display the previous procedure.
PAGE DOWN	Page down through the procedures in your code.
PAGE UP	Page up through the procedures in your code.
F2	Display the Object Browser .

Find a Variable Definition

To view the definition of a variable

In the **Code** window, select the [variable](#) whose definition you want to see.

From the **View** menu, choose **Definition** (SHIFT+F2).

To return the mouse pointer to its previous position

On the **View** menu, click **Last Position** (CTRL+SHIFT+F2).

To replace text in a module

On the **Edit** menu, click **Replace**.

The **Replace** dialog box appears.

In the **Find What** box, type the text you want to search for.

In the **Replace With** box, type the replacement text.

Select a **Search** option to specify where to look for the text.

Select a direction from the **Direction** list to specify the direction of the search.

To set limits on the search, select:

- **Find Whole Word Only** to search for the complete word by itself, and not as part of another word.
- **Match Case** for an exact match.
- **Use Pattern Matching** to use wildcard characters.

Choose **Find Next** if you want to confirm the change before replacing the text; choose **Replace** to replace the highlighted occurrence of the found text and automatically perform a **Find Next**; or choose **Replace All** to change all occurrences of the search text automatically.

Restart Execution

You can restart execution from [break mode](#). Restarting returns the code to a newly initialized state, resetting all [variables](#) and removing any suspended [procedures](#) from memory.

To restart execution

On the **Run** menu, click **Reset <projectname>**, or use the toolbar shortcut: ..

On the **Run** menu, click **Run Sub/UserForm (F5)**, or use the toolbar shortcut: .

To search for specific text in a module

On the **Edit** menu, click **Find**.

The **Find** dialog box appears.

In the **Find What** box, type the text you want to search for (if you didn't select the text before displaying the dialog box).

Select a **Search** option to specify where to look for the text.

Select a direction from **Direction** list to specify the direction of the search.

To set limits on the search, select:

- **Find Whole Word Only** to search for the complete word by itself and not as part of another word.
- **Match Case** to find an exact match.
- **Use Pattern Matching** to use wildcards or ranges.

Choose **Find Next**.

Set a Reference to a Type Library

Automation (formerly OLE Automation) enables you to use objects from other applications in Visual Basic code. An application that provides its objects for use by other applications also provides information about those objects in a [type library](#). To achieve the best possible performance when using another application's objects, you should set a reference to that application's type library.

To set a reference to an application's type library

Click **References** on the **Tools** menu.

Select the check boxes for the applications with type libraries you want to reference.

If you are writing code that manipulates objects in another application, you should set a reference to that application's type library for best possible access to those objects. You don't have to set a reference to use another application's objects, but doing so provides several advantages for your application.

Your code will run faster if you set a reference to another application's type library before you work with its objects. If you set a reference, you can declare an [object variable](#) representing an object in the other application as its most specific type. For example, if you are writing code to work with Microsoft Excel objects, you can declare an object variable of type **Excel.Application** if you created a reference to the Microsoft Excel type library. The following code is the fastest way to create a variable to represent the Microsoft Excel **Application** object.

```
Dim appXL As Excel.Application
```

If you haven't set a reference to the Microsoft Excel type library, you must declare the [variable](#) as a generic variable of type [Object](#). The following code runs more slowly.

```
Dim appXL As Object
```

If you set a reference to an application's type library, all of its objects and their [methods](#) and [properties](#) are listed in the **Object Browser**. This makes it easy to determine what properties and methods are available to each object.


For Microsoft applications that can also serve as Automation servers, you can set references to their type libraries from another application, and control their objects from that application.

Set and Clear a Breakpoint

You set a [breakpoint](#) to suspend execution at a specific statement in a [procedure](#); for example, where you suspect problems may exist. You clear breakpoints when you no longer need them to stop execution.

To set a breakpoint

Position the insertion point anywhere in a line of the [procedure](#) where you want execution to halt.

On the **Debug** menu, click **Toggle Breakpoint** (F9), click next to the statement in the **Margin Indicator Bar** (if visible), or use the toolbar shortcut: .

The breakpoint is added and the line is set to the breakpoint color defined on the **Editor Format** tab in the **Options** dialog box.

If you set a breakpoint on a line that contains several statements separated by colons (:), the break always occurs at the first statement on the line.

To clear a breakpoint

Position the insertion point anywhere on a line of the procedure containing the breakpoint.

From the **Debug** menu, choose **Toggle Breakpoint** (F9), or click next to the statement in the **Margin Indicator Bar** (if visible.)

The breakpoint is cleared and highlighting is removed.

To clear all breakpoints in the application

From the **Debug** menu, choose **Clear All Breakpoints** (CTRL+SHIFT+F9).

Note Breakpoints set in code are not saved when you save your code.

Set Project Properties

To set Project Properties

From the **Tools** menu, choose **<projectname> Properties**

Use the **General** tab in the **<projectname> Properties** dialog box to specify the following:

- Name of your project.
- Description of your [project](#).
- Name of the Help file associated with your project
- Context ID for the specific Help topic to be called when the user clicks the **Help** button while the application's [object library](#) is selected in the **Object Browser**.

Use the **Protection** tab in the **<projectname> Properties** dialog box to lock the project from viewing by others and specify a password for access to project properties. After you set protection, you must save and close your project for the protection to take effect.

Import a Text File into Code

You can import a text file containing code into the current [module](#) and use it in your code.

To import a text file into a module

Open the module into which you want to insert text and position the entry point at the place where you want the text inserted.

On the **Insert** menu, click **File**.

The **Insert File** dialog box appears.

Use the **Insert File** dialog box to select the file to import.

Click **OK**.

Set Visual Basic Environment Options

You can set the behavior and look of the Visual Basic development environment through the **Options** dialog box. Use the:

Editor tab to specify Code window and Project window settings.

Editor Format tab to specify the appearance of your code.

General tab to specify form, error handling, and compile settings for your project.

Docking tab to specify whether a window is attached or "anchored" to one edge of other dockable or application windows.

To set Environment options

On the **Tools** menu of the Visual Basic editor, click **Options**. Each option is described in the following tables.

Editor

Option	Description
Auto Syntax Check	Visual Basic automatically verifies correct syntax after you enter a line of code.
Require Variable Declaration	Explicit variable declarations are required in modules .
Auto Indent	After tabbing the first line of code, all subsequent lines start at that tab location.
Tab Width	The tab width, which can range from 1 – 32 spaces. (Default is 4 spaces.)
Default to Full Module View	Procedures for new modules are displayed in the Code window as a single, scrollable list or one procedure at a time.
Procedure Separator	Display separator bars at the end of each procedure in the Code window.
Auto List Members	At the insertion point, Visual Basic displays information that logically completes a

Auto Quick Info

statement.

Information about functions and their [arguments](#) is displayed as you type.

Auto Data Tips

Automatically display the value of any [variable](#) on which you place the mouse pointer. Available only in [break mode](#).

Drag-Drop in Text Editing

Code elements can be dragged from the **Code** window into the **Immediate** or **Watch** windows.

Editor Format

Option	Description
Foreground, Background, and Indicator	The color of different categories of text listed in the Code Colors list.
Font	The font used for displaying code.
Size	The size of the font used for code.
Margin Indicator Bar	Display the Margin Indicator Bar .

General

Option	Description
Show Grid	Display a grid on a form.
Grid Units	Lists the unit of measurement for units in the grid.
Width	The width of the grid cells on a form.
Height	The height of the grid cells on a form.
Align Controls to Grid	Automatically position the outer edge of controls on the closest grid lines.
Show ToolTips	Display ToolTips for toolbar buttons.
Collapse Proj. Hides Windows	Automatically close the project, UserForm , object, or module windows when a project is

Notify Before State Loss

collapsed in the **Project Explorer**.

Display a message that a requested action will cause all module-level variables to be reset for a running project.

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 produced in a class module causes the project to enter break mode at the line of code which produced the error.

Break on Unhandled Errors

Any other unhandled error causes the project to enter break mode.

Compile On Demand

A project is fully compiled before it starts, or code is compiled as needed.

Background Compile

Use idle time during run time to finish compiling the project in the background. (Available only if **Compile On Demand** is set.)

Docking

Option	Description
The check box for the appropriate window	A window can be anchored to an adjacent dockable window or the Visual Basic Editor window.

Split the Code Window

You can split the **Code** window horizontally into two panes to view different code segments of a [module](#) at the same time. Each pane scrolls separately, both horizontally and vertically. The [Procedure](#) and [Object box](#) options refer to the pane that has the focus. Code changes are immediately reflected in both panes.

To split the Code window into panes

Drag the split bar at the top of the vertical scroll bar down from the upper-right corner of the **Code** window.

To remove a split from the Code window

Double-click the split bar or drag it to the top or bottom of the **Code** window.

Start Code Execution

One way to test your code is to run it and work with it as a user would.

To start code execution

Choose **Run Sub/UserForm** (F5) from the **Run** menu.

If your application doesn't run, it may be because:

A [syntax error](#) or some other error exists in your code.

A [logic error](#) exists in your code, which may result in a [run-time error](#).

To get Help, click the **Help** button or press F1 while the error message is displayed. Consider the suggestions provided to correct the error before you run your code again.

Stop Code Execution

As you run your code, it may stop executing for one of the following reasons:

An untrapped [run-time error](#) occurs.

A trapped run-time error occurs, and **Break on All Errors** is selected on the **General** tab in the **Options** dialog box.

A [breakpoint](#) is encountered.


A **Stop** statement is encountered in your code, switching the mode to [break mode](#).


An **End** statement is encountered in your code, switching the mode to [design time](#).

You halt execution manually at a given point.

A [watch expression](#) that you set to break if its value changes or becomes true is encountered.

To halt execution manually

To switch to break mode, from the **Run** menu, choose **Break** (CTRL+BREAK), or use the toolbar shortcut: .

To switch to design time, from the **Run** menu, choose **Reset <projectname>**, or use the toolbar shortcut: .

To continue execution when your application has halted

From the **Debug** menu, choose **Step Into** (F8), **Step Over** (SHIFT+F8), **Step Out** (CTRL+SHIFT+F8), or **Run To Cursor** (CTRL+F8).

Trace Code Execution

You trace code execution because it may not always be obvious which statement is executed first. Use these techniques to trace the execution of code:

Step Into: Traces through each line of code and steps into [procedures](#). This allows you to view the effect of each statement on [variables](#).

Step Over: Executes each procedure as if it were a single statement. Use this instead of **Step Into** to step across procedure calls rather than into the called procedure.

Step Out: Executes all remaining code in a procedure as if it were a single statement, and exits to the next statement in the procedure that caused the procedure to be called initially.

Run To Cursor: Allows you to select a statement in your code where you want execution to stop. This allows you to "step over" sections of code, for example, large loops.

To trace execution from the current statement

From the **Debug** menu, choose **Step Into** (F8), **Step Over** (SHIFT+F8), **Step Out** (CTRL+SHIFT+F8), or **Run To Cursor** (CTRL+F8).

To trace execution from the beginning of the program

From the **Debug** menu, choose **Step Into** (F8), **Step Over** (SHIFT+F8), **Step Out** (CTRL+SHIFT+F8), or **Run To Cursor** (CTRL+F8).

Turn Syntax Checking On or Off

Visual Basic includes a syntax-checking feature that:

Checks each statement as you enter it for [syntax errors](#), such as a misspelled [keyword](#) or missing separator, and alerts you if there are errors.

Translates the code to an internal form if the syntax is correct, which speeds the transition to run time.

This feature is turned on when you first start, but you can turn it off if you prefer to write code without being alerted to errors as they occur.

To enable syntax checking

On the **Tools** menu, click **Options**.

Select the **Editor** tab.

Select the **Auto Syntax Check** check box.

Click **OK**.

To disable syntax checking

Click **Options** on the **Tools** menu.

Select the **Editor** tab.

Clear the **Auto Syntax Check** check box.

Click **OK**.

Use the Immediate Window

The **Immediate** window displays information resulting from debugging statements in your code or from commands typed directly into the window.

To display the Immediate window

From the **View** menu, choose **Immediate** window (CTRL+G)

To execute code in the Immediate window

Type a line of code in the **Immediate** window.

Press ENTER to execute the statement.

Use the **Immediate** window to:

Test problematic or newly written code.

Query or change the value of a [variable](#) while running an application. While execution is halted, assign the variable a new value as you would in code.

Query or change a property value while running an application.

Call [procedures](#) as you would in code.

View debugging output while the program is running.

Note **Immediate** window statements are executed in a context — that is, as if they are entered in a specific [module](#).

If you need help on syntax for functions, statements, [properties](#), or [methods](#) while working in the **Immediate** window, select the [keyword](#), the property name, or the method name, and press F1.

Use the Object Browser

The **Object Browser** allows you to browse through all available objects in your [project](#) and see their [properties](#), [methods](#) and events. In addition, you can see the

[procedures](#) and [constants](#) that are available from [object libraries](#) in your project. You can easily display online Help as you browse. You can use the **Object Browser** to find and use objects you create, as well as objects from other applications.

You can get help for the **Object Browser** by searching for **Object Browser** in Help.

To navigate the Object Browser

Activate a [module](#).

From the **View** menu, choose **Object Browser** (F2), or use the toolbar shortcut: .

Select the name of the project or library you want to view in the **Project/Library** list.

Use the **Class** list to select the [class](#); use the **Member** list to select specific members of your class or project.

View information about the class or member you selected in the **Details** section at the bottom of the window.

Use the **Help** button to display the Help topic for the class or member you selected.

Use the Project Explorer

The **Project Explorer** displays a hierarchical list of the [projects](#) and all of the items contained and referenced by each project.

To navigate the Project Explorer

From the **View** menu, choose **Project Explorer** (CTRL+R), or use the toolbar shortcut: .

Select an item from the collapsible tree that includes all projects and their components.

Click the **View Code** button to open the **Code** window to begin entering code.

- or -

Click the **View Object** button to open a window containing the specified object.

- or -

Click the **Toggle Folders** button to toggle between folder view and folder contents view.

Use the Properties Window

The **Properties** window lists the [design-time properties](#) for selected objects and their current settings. You can change these properties at design time. When you select multiple controls, the **Properties** window contains a list of the properties common to all the selected controls.

To navigate the Properties window

From the **View** menu of the Visual Basic Editor, choose **Properties** window (F4).

Select the object whose properties you want to display. You can either use the mouse to select the object or use the **Project Explorer** to choose from a list.

Click the **Alphabetic** tab to display properties in alphabetic order, or click the **Categorized** tab to display object properties by category.

To change a property's value

Select the property in the left column.

Change the property's value in the right column.

Note You can enter a property's value in the right column. For those properties that have a predefined set of values, click the value and then select one from the values displayed in the list box.