Overview: The LotusScript IDE

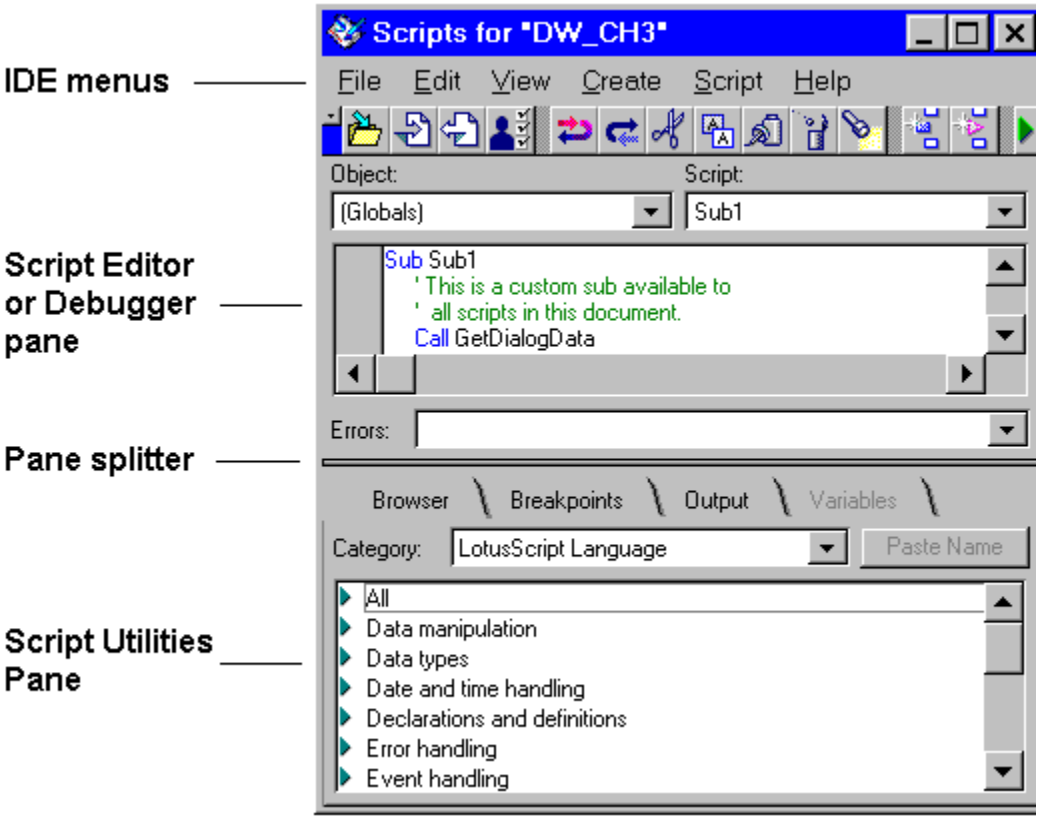
The LotusScript Integrated Development Environment (IDE) provides a usable and powerful set of tools for creating and debugging scripts in Lotus products. If you have worked in development environments such as IBM VisualAge™ or Microsoft Visual Basic®, you will be comfortable working in the LotusScript IDE.

The IDE has several features that differentiate it from other development environments that you may have used:

- The IDE uses one window containing menus, SmartIcons, an editor, a debugger, and other utilities to do all its work.
- The IDE is closely integrated with documents in your Lotus product. For each document that is active in your product, you can display an IDE window that, in turn, displays the scripts in that document. For example, if you are running 1-2-3 and have retrieved three 1-2-3 workbooks, you can display an IDE window for each of those documents. To run the IDE, you must first run a Lotus product and then open a document in that product.
- The IDE saves your scripts in product documents. If you develop scripts in the IDE for a Word Pro document named INVEST.LWP, all scripts are stored in INVEST.LWP when you save that document. If you want to save your scripts apart from product documents, you can also export them as .LSS text files or compiled .LSO files.
- The IDE uses context-sensitive menus and SmartIcons to focus your programming tasks. If you are entering or editing a script, the IDE displays a Script menu with menu commands specific to the Script Editor. If you are debugging a script, use the Debugger menu for commands specific to that tool.

Parts of the IDE Window

The IDE provides several tools in one window.



- IDE menus provide access to all IDE commands and preferences.
- The Script Editor or the Script Debugger are displayed in one pane. The Script Editor lets you write scripts and check their syntax. It also lets you set, clear, disable, and enable breakpoints that you use to debug your scripts. The Script Debugger lets you set, clear, disable, and enable breakpoints and step through scripts to locate the source of problems that may occur while script is executing.
- The Pane Splitter lets you resize, display, or hide panes in the IDE window.
- The Script Utilities pane has four panels, each containing a tool:
  - The Browser panel lists LotusScript keywords; constants, variables, procedures, and classes defined by your product; and type libraries and classes for OLE Automation objects. You can copy items from the Browser

panel and paste them into the Script Editor.

- The Breakpoints panel lists breakpoints that you set in your scripts in the order that you set them, and lets you navigate to them, clear them, disable them, or enable them.
- The Output panel displays output generated by any LotusScript Print commands that you include in your scripts.
- During debugging, the Variables panel displays information about variables for the current script and lets you change their values.

Navigating between these parts of the IDE window is easy. To activate a pane or panel, click it or use View menu commands.

Adding Comments to a Script

You can add comment lines to any script in your document. LotusScript does not execute any text in a comment. LotusScript uses the following conventions for comment lines:

- ' (quote) designates the beginning of a comment on one line. Although the ' (quote) need not begin the line because it can follow a statement, all text after the ' (quote) is part of the comment.

```
' A comment can be alone on a line.
Print "Rabble: The king is dead."

Print "Rabble: The king is dead."  'Comments can follow a statement.
```

- Rem designates the beginning of a comment on one line. Although Rem need not begin the line because it can follow a statement, all text after Rem is part of the comment.

```
Rem A comment can be alone on a line.
Print "Rabble: The king is dead."
```

- The %Rem and %End Rem directives designate the beginning and end of one or more lines of comment. %Rem and %End Rem must be the first text on a line although they may be preceded on the line by spaces or tabs. Each must be followed by one or more spaces, tabs, or newline characters before any more text appears. %Rem blocks cannot be nested.

```
%Rem
============================================================
World Press: According to sources, the reports of the King's
death have been greatly exaggerated.
An informed source close to the King reported:
"The old fellow was hale and hearty when I saw him last
year. Long live the king."
============================================================
%End Rem
```

Changing Variable Values in the IDE

You can change a variable's value when you are debugging a script.

Changing a variable value during debugging

To change a variable value during debugging, do the following:

1. In the Calls drop-down box in the Script Debugger, select the procedure that contains the value definition.
2. In the Variables panel, select the variable whose value you want to change.
   If the variable value can be changed, the value box is enabled.
3. In the value box, enter a new value for the variable.
   If a variable contains more than 1024 characters, you can view and edit only the first 1024 characters in the box.
4. Press ENTER or click the check mark next to the box to change the value of the variable.
   Tip Press ESC or click the check mark next to the box to cancel editing the value of the variable.

You can enter data in the following formats:

- Strings in " (double quotes)
- Numbers in a standard format for scripts
- Variant variables accept the constants True, False, and Null

Checking a Single Line of Script in the Script Editor

The IDE automatically checks single lines of script for syntax errors when you move the insertion point off that line. If the IDE finds an error, it changes the color of the line and reports the error in the Errors drop-down box. The Script Editor does not report errors for the following when it checks a single line:

- Calling procedures that are not defined
- Parameter errors
- Type mismatches

Checking Scripts for an Object in the Script Editor

Choose Script - Check Scripts for *object* or press F2 to check all statements in (Globals) and in all scripts for the object for which you are writing scripts.

Checking Scripts for a Document in the Script Editor

Checking all scripts for a document

Choose Script - Check All Scripts for *document* or press SHIFT+F2 to check all statements in all scripts for the current document.

Tip Use Script - Check Scripts for *object* or Script - Check All Scripts for *document* to update your current script after you have modified a script file that you reference with the %Include directive. Otherwise the IDE continues to use the previous version of the included file. Also use Script - Check All Scripts for *document* to update your scripts after you have changed an LSO file referenced in a Use statement.

Checking Scripts in the Script Editor

The IDE checks the scripts that you write in several ways.

Checking single lines of script

The IDE automatically checks single lines of script for syntax errors when you move the insertion point off that line. If the IDE finds an error, it changes the color of the line and reports the error in the Errors drop-down box. The Script Editor does not report errors for the following when it checks a single line:

- Calling procedures that are not defined
- Parameter errors
- Type mismatches

Checking an individual sub

Choose Script - Run Current Sub or press F5 to check all statements in the current sub and, if there are no errors, run it.

Checking all scripts for an object

Choose Script - Check Scripts for *object* or press F2 to check all statements in (Globals) and all scripts for the current object.

Checking all scripts for a document

Choose Script - Check All Scripts for *document* or press SHIFT+F2 to check all statements in all scripts for the current document.

Tip Use Script - Check Scripts for *object* or Script - Check All Scripts for *document* to update your current script after you have modified a script file that you reference with the %Include directive. Otherwise, the IDE continues to use the previous version of the included file. Also use Script - Check All Scripts for *document* to update your scripts if you have changed an LSO file referenced in a Use statement.

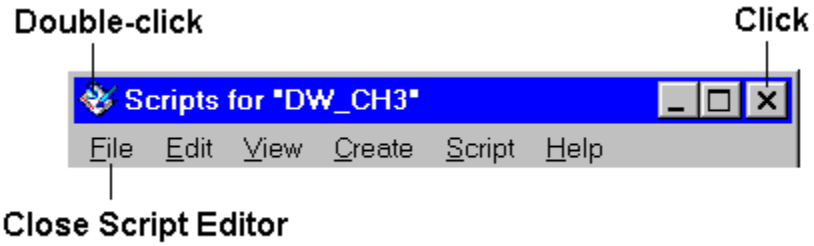
Viewing error messages

The IDE reports all errors in the Errors drop-down box. The IDE also displays a message box reporting the number of errors if you enable the feature in the File - Script Preferences dialog box.

Closing the IDE Window

There are several ways to close the IDE window:

- Double-click the program icon box in the top left corner of the IDE window.
- Click the Close button in the top right corner of the IDE window.
- Choose File - Close Script Editor in the IDE window.
- Close the product document associated with the IDE window.



Note When you close or deactivate the IDE, it completes the following automatically: %Rem blocks, multiline strings, sub, function, and property blocks, and user-defined type and class blocks. The IDE reformats line indentations and saves all scripts.

If you try to close the IDE while the Script Debugger containing an active breakpoint is running, it displays a message prompting you to continue debugging or to stop execution of the current script.

Overview: Closing the Script Debugger

You can close the Script Debugger for a document provided the document doesn't contain the currently executing script. If you try to close a Script Debugger when it contains the currently-executing script, a message informs you that closing the Script Debugger will stop script execution and prompts you to continue. If you click OK, script execution is stopped, and the Script Debugger is closed.

Closing a Debugger for the executing script affects other open Debuggers as follows:

- Debuggers that replaced Script Editors (when the Script Debugger was opened for the executing script) revert to being Editors.
- Debuggers that were opened automatically during script execution are closed.

You can close a Debugger in the following ways:

- Double-click the program icon box in the top left corner of the IDE window.
- Click the Close button in the top right corner of the IDE window.
- Choose File - Close Script Editor in the IDE window.
- Close the product document associated with the IDE window.

Checking Scripts in the Script Editor

You can check the syntax of your scripts in several ways.

Checking single lines of script

The IDE automatically checks single lines of script for syntax errors when you move the insertion point off that line. If the IDE finds an error, it changes the color of the line and reports the error in the Errors drop-down box. The Script Editor does not report errors for the following when it checks a single line:

- Calling procedures that are not defined
- Parameter errors
- Type mismatches

Checking an individual sub

Choose Script - Run Current Sub or press F5 to test all statements in the current sub and, if there are no errors, run it.

Checking all scripts for an object

Choose Script - Check Scripts for *object* or press F2 to check all statements in (Globals) and in all scripts for the object for which you are writing scripts.

Checking all scripts for a document

Choose Script - Check All Scripts for *document* or press SHIFT+F2 to check all statements in all scripts for the current document.

Tip Use Script - Check Scripts for *object* or Script - Check All Scripts for *document* to update your current script after you have modified a script file that you reference with the %Include directive. Otherwise, the IDE continues to use the previous version of the included file. Also use Script - Check All Scripts for *document* to update your scripts if you have changed an LSO file referenced in a Use statement.

Viewing error messages

The IDE reports all errors in the Errors drop-down box. The IDE also displays a message box reporting the number of errors if you enable the feature in the File - Script Preferences dialog box.

Continuing and Stopping Script Execution in the Script Debugger

Continuing script execution from a breakpoint

Choose Debug - Continue Execution to have script execution continue to the next breakpoint or to the end of the script.

Stopping script execution

Choose Debug - Stop Execution to have script execution terminate at the current statement. The Script Editor is displayed in place of the Script Debugger.

Creating Functions in the Script Editor

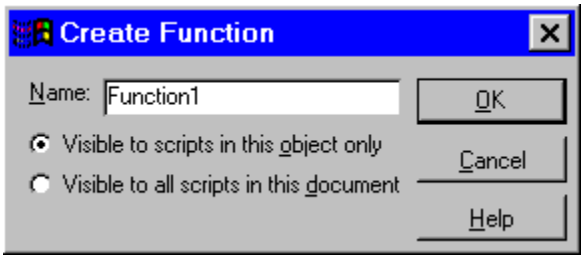
A function is a named script that performs a specific task and returns a value (unlike a sub, which does not return a value). Functions begin with the Function keyword followed by the name of the function and its parameters. Functions end with the line End Function.

```
Function MiniMult (x As Integer, y As Integer) As Integer
   MiniMult = x * y
End Function
```

Creating a function

You can use the Create menu to add new functions to your document. To create a function, do the following:

1. Navigate to the object, including (Globals), for which you want to create the function.
2. Choose Create - Function to display the following dialog box:



3. Enter a new name for the function or accept the default name that the Script Editor provides.
4. (Optional) Select one of the options to define the scope of the new function.
   - Select "Visible to scripts in this object only " to create the script for the current object.
   - Select "Visible to all scripts in the document" to create the script in (Globals).
5. Click OK to create the new function.
6. Enter your statements in the new function.

If you create the function in (Globals) and you did not delete the Option Public statement in the (Options) script, your new procedure can be called from any procedure in your document.

If you create the function for an object, the procedure can be called from the object's Initialize or Terminate sub, event procedures, and other procedures that you create for the object.

Note If you add a Sub, Function, Property Get, or Property Set statement inside a sub or function that defines a class method or property, the IDE does not move the statement but displays an error message to let you know that the definition is illegal in that scope.

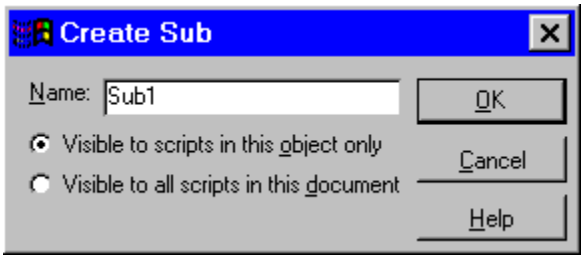
Creating Subs in the Script Editor

A sub is a named script that performs a specific task without returning a value (unlike a function, which does return a value). Subs begin with the Sub keyword followed by the name of the sub and its arguments. Subs end with the line End Sub.

Creating a sub

You can use the Create menu to add new subs to your document. To create a sub, do the following:

1. Navigate to the object, including (Globals), for which you want to create the sub.
2. Choose Create - Sub to display the following dialog box:



3. Enter a new name for the sub or accept the default name that the Script Editor provides.
4. (Optional) Select one of the options to define the scope of the new sub.
   - Select "Visible to scripts in this object only " to create the script for the current object.
   - Select "Visible to all scripts in the document" to create the script in (Globals).
5. Click OK to create the new sub.
6. Enter your statements in the new sub.

If you create the sub in (Globals) and you did not delete the Option Public statement in the (Options) script, your new procedure can be called from any procedure in your document.

If you create the sub for an object, the procedure can be called from the object's Initialize or Terminate sub, event procedures, and other procedures that you create for the object.

Note If you add a Sub, Function, Property Get, or Property Set statement inside a sub or function that defines a class method or property, the IDE does not move the statement but displays an error message to let you know that the definition is illegal in that scope.

Overview: Creating (Declarations) in the Script Editor

You can create (Declarations) in (Globals) or for individual objects. In the IDE, any subs, functions, or properties that you create are declared implicitly. You never need to use Declare for this purpose (except within a class).

(Declarations) statements in (Globals)

The (Declarations) script in (Globals) is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts in your document
- Public, Private, Type, Class, and Declare Lib statements (external C calls)
- Const statements for those constants that you want to be available to all scripts in your document and are not needed for Use or UseLSX statements in (Options)

By default the (Declarations) script is initially empty.

If you enter Type, Class, or Declare Lib statements in any other script in (Globals), the IDE moves them to (Declarations) automatically. If you enter Dim, Public, Private, or Const statements outside the scope of a procedure in another script, the IDE moves them to (Declarations) automatically. Const statements in (Options) are the exception to this rule.

(Declarations) statements for objects

The (Declarations) script for an object is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts for the current object
- Const statements for those constants that you want to be available to all scripts for the current object and that are not needed for Use or UseLSX statements in (Options)

By default the (Declarations) script is initially empty.

Note If you enter Type, Class, or Declare Lib statements in an object script, the IDE moves them to (Declarations) for the object automatically.

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object. To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

Creating Scripts in the Script Editor

You can create your own subs, functions, and properties in (Globals) or for any object that is listed in the Object drop-down box. Once you have created these scripts, the IDE lists them in the Script drop-down box. Scripts that you create in (Globals) can be called from any script in your document, unless you declare them explicitly as Private. Scripts that you create for objects can be called only from scripts defined for that object.

There are three ways to add scripts to (Globals) or to object scripts.

- Enter a Sub, Function, or Property statement in a script except within a class.
- Use Create - Sub and Create - Function to create new subs and functions.
- Use File - Import Script to add scripts contained in a text file to (Globals) or to object scripts.

In each case, the IDE automatically adds the name of the new script to the Script drop-down box for (Globals) or for an object. To distinguish scripts that you created from the predefined ones, the IDE displays them in bold italics in the drop-down box.

Creating scripts by entering script statements

The IDE automatically creates a script for you when you enter a Sub, Function, or Property statement in the current script that you are editing. To create a script in this way, do the following:

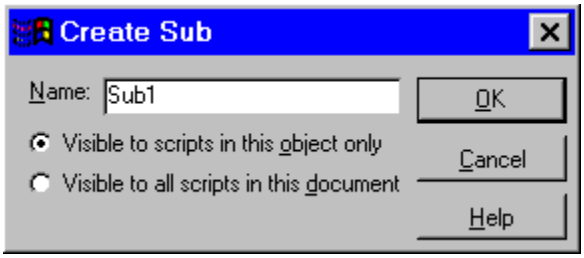
1. Select any script in (Globals) or an object to contain the new script.
2. Enter the first line of a Sub, Function, or Property statement. Make certain that the name of the new procedure does not already appear in the Script drop-down box.
3. Move the insertion point off the line. The IDE completes the procedure by appending a blank line and the appropriate End statement. The name of the new procedure appears in the Script drop-down box.

Note If you enter a Sub, Function, or Property statement within a Type definition or within a class method or property, the IDE does not move the statement and displays a message alerting you that the script definition is illegal in that context.

Creating scripts with Create - Sub and Create - Function

You can use the Create menu to add new subs or functions to your document. To do so, do the following:

1. Navigate to the object for which you want to create the sub or function.
2. Choose Create - Sub or Create - Function to display a dialog box. The following dialog illustrates the Create - Sub dialog box.



3. Enter a new name for the sub or function or accept the default name that the Script Editor provides.
4. (Optional) Select one of the options to define the scope of the new sub or function.
   - Select "Visible to scripts in this object only " to create the script for the current object.
   - Select "Visible to all scripts in the document" to create the script in (Globals).
5. Click OK to create the new sub or function.
6. Enter your statements in the new sub or function.

If you create the sub or function in (Globals) and you did not delete the Option Public statement in the (Options) script in (Globals), your new procedure can be called from any script in your document.

If you create the sub or function for an object, the procedure can be called from the object's Initialize or Terminate sub, event procedures, and other scripts that you create for the object.

Creating scripts by importing them

You can create scripts by importing them from a text file on disk. If you are working in (Globals), the imported scripts appear as new scripts in (Globals). If you are working in an object script, the imported scripts appear as new scripts for that object.

Renaming procedures

You can rename sub, function, and property procedures in your document; you cannot rename (Options) or (Declarations). To rename a procedure, do the following:

1. Navigate to the procedure.
2. Select the name of the procedure in the first line of the script.

   Example: `Sub GetNotesMail (TemplateName as String)`
3. Change the name of the procedure.

   Example: `Sub GetNotesNetMail (TemplateName as String)`

   Note You cannot have duplicate names in (Globals) or within the scripts for a particular object. If you enter a duplicate name under these circumstances, the Script Editor displays a message notifying you of the error.
4. Move the insertion point off the line. The Script Editor lists the procedure with its new name in the Script drop-down box.

Note If you rename a predefined script such as Initialize, Terminate, or an event procedure, the IDE creates two scripts. One has the new name and any statements that were contained in the original script. The other has the original name of the predefined script and is empty. Both scripts are listed in the Script list. You cannot rename (Options) or (Declarations).

Choose Edit - Find and Replace to replace references to the old name in your existing statements.

Overview: Working with Object Scripts in the Script Editor

You can enter statements in predefined scripts for an object or add new scripts, making them available to all scripts for that object.

Using predefined scripts for objects

By default, the IDE provides (Options), (Declarations), Initialize, and Terminate scripts for each product object. All predefined scripts are initially empty. Each predefined script is designed to contain certain statements or to perform a specific function with this object.

Note To indicate that you have modified one of the predefined scripts, the IDE boldfaces its name in the drop-down box.

(Options) statements

The (Options) script for an object is designed to contain the following statements:

- Option statements
- Def*type* statements
- Use and UseLSX statements
- Const statements needed for Use and UseLSX statements

(Declarations) statements

The (Declarations) script for an object is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts for the current object
- Const statements for those constants that you want to be available to all scripts for the current object and that are not needed for Use or UseLSX statements in (Options)

By default the (Declarations) script is initially empty.

Note If you enter Type, Class, or Declare Lib statements in an object script, the IDE moves them to (Declarations) for the object automatically.

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object. To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

Initialize sub

Use the Initialize sub to set up variables declared in the object's (Declarations) script. The Initialize sub for an object executes before any of its event procedures. By default, the Initialize script is empty.

Note Scripts for controls created in the Lotus Dialog Editor do not have Initialize subs.

Terminate sub

Use the Terminate sub to clean up variables that you have declared in the object's (Declarations) script. By default the Terminate script is empty.

Note Scripts for controls created in the Lotus Dialog Editor do not have Terminate subs.

Working with event procedures

If you are writing a script for an object, the Script drop-down box displays default event procedures for the selected object. In the IDE you cannot create new event procedures for an existing product object because valid events for that object are defined by the product.

To enter statements in an event procedure, do the following:

1. Click the Script drop-down box to display the list of scripts for the selected object.
2. Click the name of the event procedure.
3. Type or paste script statements between the Sub and End Sub statements.
4. Choose Script - Check Scripts to check the syntax of the statements you entered.

Creating other procedures for objects

You can create other named subs, functions, and properties for objects in addition to the predefined scripts or event procedures. Because these procedures are not in (Globals), they can be called only from other scripts for the object.

There are three ways to create object scripts.

- Use Create - New Sub and Create - New Function to create new subs and functions for an object. The IDE automatically adds the name of the new procedure to the Script drop-down box for that object.

- Enter a Sub, Function, or Property statement anywhere in a script for the current object. The IDE automatically adds the name of the new procedure to the Script drop-down box for that object.
- Use File - Import Script when you are working with object scripts to import scripts for that object. The IDE automatically adds the name of the new scripts to the Script drop-down box for that object.

Note To distinguish scripts that you created from the predefined ones and event procedures, the IDE displays them in bold italics in the Script drop-down box.

Cutting, Copying, and Pasting Text in the Script Editor

Use standard Windows conventions for keyboard accelerators, shortcuts, and menu items in the Script Editor to copy, cut, or paste text.

Copying selected text from a script

To copy selected text, do one of the following:

- Choose Edit - Copy.
- Press CTRL+C.
- Choose Copy from the Shortcut menu (RIGHT+click).

Cutting selected text from a script

To cut selected text, do one of the following:

- Choose Edit - Cut.
- Press CTRL+X.
- Choose Cut from the Shortcut menu (RIGHT+click).

Pasting text into a script

To paste text in the Clipboard, do one of the following:

- Choose Edit - Paste.
- Press CTRL+V.
- Choose Paste from the Shortcut menu (RIGHT+click).

Note You can also use the File - Import Script menu command to import scripts from text files on disk.

Pasting a keyword from a Browser list into a script

Position the insertion point in the script where you want to paste the keyword, select the keyword in the Browser, and click Paste Name at the top right of the Browser panel.

Enabling and Disabling Breakpoints in the IDE

You can enable or disable breakpoints in your scripts with the mouse or with Script or Debug menu commands. Disabling breakpoints is useful when you want to stop using breakpoints temporarily, but plan to use them again later.

Enabling and disabling breakpoints with the mouse

CTRL+click next to the breakpoint symbol in the breakpoint gutter to toggle enabling or disabling the breakpoint.

Script - or Debug - Disable Breakpoint or Enable Breakpoint

This menu item is a toggle for disabling or enabling breakpoints for a line or selection of lines.

Choose Script - Disable Breakpoint to have the IDE ignore breakpoints for the current line or selection of lines without deleting them. This menu item is available if the current line contains an enabled breakpoint or a selection of lines contains at least one enabled breakpoint. Disable Breakpoint disables all breakpoints in a selection of lines.

Choose Script - Enable Breakpoint to enable a breakpoint that you previously disabled with Disable Breakpoint. This menu item is available if the current line contains a disabled breakpoint or a selection of lines contains at least one disabled breakpoint and no enabled ones. Enable Breakpoint enables all breakpoints in a selection of lines.

Note You can also use Disable Breakpoint and Enable Breakpoint to disable or enable breakpoints displayed in the Breakpoints panel.

Script - or Debug - Disable Breakpoints for *document* or Enable Breakpoints for *document*

This menu item is a toggle for disabling or enabling breakpoints within your current document.

Choose Script - Disable Breakpoints to disable all breakpoints in the current document. This menu item is dimmed if there are no breakpoints in the current document.

Choose Script - Enable Breakpoints to enable all breakpoints in the document. This menu item is available only if all breakpoints in the document are disabled.

Entering and Deleting Text in the Script Editor

The Script Editor follows standard Windows conventions for entering and deleting text.

Entering text

To enter text in the Script Editor, do one of the following:

- Position the insertion point where you want to add text and type the text.
- Position the insertion point and choose Edit - Paste.

Deleting text

To delete text in the Script Editor, do one of the following:

- Press DEL to delete the character following the insertion point.
- Press BACKSPACE to delete the character preceding the insertion point.
- Press CTRL+BACKSPACE to delete the previous word.
- Choose Edit - Clear to delete selected text.
- Choose Edit - Clear All to delete all text in the current script.

Entering Class Statements in the Script Editor

The Script Editor completes class statements automatically as you enter them and moves them to (Declarations) if you enter them in some other script.

Here are rules for entering class statements:

- If you enter a Class statement in (Declarations) and press ENTER, the Script Editor completes the statement by appending a blank line and appending an End Class statement.
- If you enter a Class statement in a script other than (Declarations) and move the insertion point off the line, the Script Editor moves the class statement to the end of (Declarations) and completes it.

The Script Editor does not automatically move subs, functions, or properties that you enter within a class declaration. All the statements in the following class declaration would remain together:

```
Class Mortgage

   Sub MortgageCalc1
      Statements
   End Sub

   Sub MortageCalc2
      Statements
   End Sub

End Class
```

Entering Def*type* Statements in the Script Editor

Def*type* statements set the default data type for variables, functions, and properties whose names begin with one of a specified group of letters. If you defined DefStr to all variable, function, and property names beginning with the letter S, then variable names such as SCurrentDB or SResultSet in your script would automatically have the data type of string. The IDE automatically formats Def*type* statements that you enter in an object script and moves them to the end of the (Options) script for that object.

LotusScript supports the following Def*type* statements:

- DefCur (for currency)
- DefDbl (for doubles)
- DefInt (for integers)
- DefLng (for longs)
- DefSng (for singles)
- DefStr (for strings)
- DefVar (for variants)

Def*type* statements that you enter in (Options) apply only to scripts for the current object, even if you are working in (Globals). To make certain that your Def*type* statements are applied consistently throughout your document, enter the Def*type* statements in the (Options) script for every object for which you are writing scripts.

Entering Public and Private Statements in the Script Editor

The IDE creates an Option Public statement in the (Options) script in (Globals) automatically. This has the effect of making all variable, constant, procedure, class, and type declarations that you create in (Globals) public by default. These are then available to all scripts in your document. Using Public explicitly in your declaration statements has the same effect.

Use Private to limit the scope of particular declarations in (Globals). A Private variable, constant, procedure, class, or type declaration would be available to all scripts in (Globals), but not to all scripts elsewhere in your document.

Here are some examples of public and private declarations in (Globals):

Dim Var1 As String	Public by default because of Option Public
Public Var2 As String	Public explicitly
Private Var3 As String	Private explicitly

Declarations that you create in object scripts are available only to other scripts for that object. You can use Private in your declarations, but it has no effect on the scope of the declaration.

You can use Public and Private to declare class members, regardless of where the class is declared. In this context, Public and Private determine whether the class members are available outside the class declaration (Public) or restricted in scope to within class declaration (Private).

Adding Scripts to (Globals)

While you are working in (Globals), you can add scripts to make them available throughout your document. Here are three ways to add scripts to (Globals).

- Use Create - Sub and Create - Function to create new subs and functions in (Globals). The IDE automatically adds the name of the new procedure to the Script drop-down box.
- Enter a Sub, Function, or Property statement anywhere in (Globals), except within a class. The IDE automatically adds the name of the new procedure to the Script drop-down box for (Globals).
- Use File - Import Script when you are editing in (Globals) to import scripts that you want to be available to all scripts in your document. The IDE automatically adds the name of the new scripts to the Script drop-down box.

Note To distinguish scripts that you created from the predefined ones, the IDE displays them in bold italics in the drop-down box.

Using predefined scripts in (Globals)

By default, the IDE provides (Options), (Declarations), Initialize, and Terminate scripts for you. All predefined scripts are empty with the exception of (Options), which contains the Option Public statement. Each predefined script is designed to contain certain statements or to perform a specific function.

Note To indicate which of these scripts you have modified, the IDE boldfaces it.

(Options) statements

The (Options) script is designed to contain the following statements:

- Option statements

  Note (Options) contains the statement Option Public by default. This makes Const, Dim, Type, Class, Sub, Function, and Property statements public by default. You can use the Public form of these statements to make them public explicitly or the Private form to make them unavailable to other scripts outside (Globals).
- Def*type* statements
- Use and UseLSX statements
- Const statements needed for Use and UseLSX statements

Note If you enter any of these statements, except for Const, in any other script in (Globals), the IDE automatically moves them to (Options).

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object. To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

(Declarations) statements

The (Declarations) script is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts in your document
- Public, Private, Type, Class, and Declare Lib statements (external C calls)
- Const statements for those constants that you want to be available to all scripts in your document and are not needed for Use or UseLSX statements in (Options)

By default the (Declarations) script is initially empty.

Note If you enter Type, Class, or Declare Lib statements in any other script in (Globals), the IDE moves them to (Declarations) automatically. If you enter Dim, Public, Private, or Const statements outside the scope of a procedure in another script, the IDE moves them to (Declarations) automatically. Const statements in (Options) are the exception to this rule.

Initialize sub

Use the Initialize sub in (Globals) to initialize variables that you have declared in (Declarations). The Initialize sub executes before any of these variables are accessed and before any other scripts in (Globals) are executed. By default, the Initialize script is empty.

Note By contrast, use the Initialize sub for an object script to set up variables declared in the object's (Declarations) script. The Initialize sub for an object executes before the object's events are executed.

Terminate sub

Use the Terminate sub in (Globals) to clean up variables that you have declared in (Declarations) when you close your document or when you modify a script and execute it again. For example, you might use an Open statement to open a file containing data in Initialize and use a Close statement in Terminate to close it. By default the Terminate

script is empty.

Note By contrast, use the Terminate sub for an object script to clean up variables defined in the object's (Declarations) script.

Overview: Entering Options in the Script Editor

The Script Editor automatically moves the following statements to the (Options) script in (Globals) or in object scripts:

- Option statements
- Def*type* statements
- Use statements
- UseLSX statements

If you want to enter a Const statement in (Options) for use with a Use or UseLSX statement, you must enter it there explicitly, otherwise the Script Editor moves it to (Declarations).

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object, even if you are working in (Globals). To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

Entering Option Statements in the Script Editor

The IDE automatically formats Option statements that you enter in (Globals) or in any object script and moves them to the end of the appropriate (Options) script. LotusScript supports the following Option statements:

- Option Public specifies that all declarations are public by default. It is allowed only in the (Options) script in (Globals).

  Note The IDE creates an Option Public statement in the (Options) script in (Globals) automatically. This has the effect of making all variable, constant, procedure, class, and type declarations that you create in (Globals) public by default. These are then available to all scripts in your document. Using Public explicitly in your declaration statements has the same effect. Use Private in a variable, constant, procedure, class, or type declaration if you want it to be available to all scripts in (Globals), but not to all scripts in your document.
- Option Base sets the default lower bound for array subscripts to 0 or 1.
- Option Compare specifies the method of string comparison.
- Option Declare disallows implicit declaration of variables.

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object, even if you are working in (Globals). To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

Entering Type Statements in the Script Editor

Here are rules for entering Type statements in the IDE:

- If you enter a Type statement in (Declarations) and press ENTER, the Script Editor completes the statement by appending a blank line and appending an End Type statement.
- If you enter a Type statement in a script other than (Declarations) and move the insertion point off the line, the Script Editor moves the Type statement to the end of (Declarations) and completes it.

By default, the (Options) script in (Globals) contains the statement Option Public. This makes Type statements public by default.

Note You must enter at least a valid beginning Type statement. The IDE generates an End Type statement if you do not supply one.

Entering Use and UseLSX Statements in the Script Editor

The IDE automatically formats Use and UseLSX statements that you enter in any script and moves them to the end of the (Options) script. LotusScript supports the following Use and UseLSX statements:

- Use loads a product document or compiled LotusScript file (LSO) containing Public definitions.
- UseLSX loads a file containing LotusScript Extensions as Public definitions.

Overview: Executing Scripts in the Script Debugger

Once you have written your scripts in the Script Editor, you can use the Script Debugger and Script Utilities to test how they execute in your product.

The Script Debugger

The Script Debugger lets you examine your script as it executes in your product. You can set, clear, enable, and disable breakpoints to pause script execution and evaluate how your script is running in your product. The Calls drop-down box in the Script Debugger pane provides information about the name of your current procedure and any procedures that called it. The Breakpoints panel lists all breakpoints that you set in your scripts in the order that you set them, and lets you navigate to them, clear them, enable them, or disable them. The Output panel displays output generated by LotusScript Print statements that you put in your scripts. The Variables panel displays information about variables for the current procedure and lets you change their values while debugging the script.

The Calls drop-down box

The Calls drop-down box contains the name of the current procedure, the name of the procedure that called it, other procedures that called it, and so on. Each item in the drop-down box shows the name of the object that contains the procedure and the name of the procedure itself. When you select an item, the Script Debugger displays the script that contains the procedure. A white arrow in the breakpoint gutter marks the current script statement. A yellow arrow in the breakpoint gutter marks the location of a call to the procedure that contains the current statement.

Items in the drop-down box are dimmed if they are procedures contained in another product document, referenced with the %Include directive, or used with the Use statement. You cannot select these items.

When you select a script in the list, the Variables panel displays variables for the current procedure, their current values, and any global variables used by the script. You can use the Variables panel to view information about a variable or change its value.

The Breakpoints panel

The Breakpoints panel lists all breakpoints that you set in your scripts in the order that you set them, and lets you navigate to them, clear them, enable them, or disable them. The Breakpoints panel lists each breakpoint in the following format:

object: *script*: *line#*

Note (Disabled) is displayed next to the breakpoint statement if the breakpoint is disabled.

The Breakpoints panel offers a convenient way to navigate to the location of breakpoints, to enable breakpoints, or to disable breakpoints.

You can navigate to the location of a breakpoint in the Script Editor or Script Debugger in the following ways:

- Click the item in the Breakpoints panel to navigate to the breakpoint.
- Double-click the item in the Breakpoints panel to navigate to the breakpoint and to activate the Script Debugger or Script Editor.
- Select the item in the Breakpoints panel and press ENTER to navigate to the breakpoint and to activate the Script Editor or Script Debugger.

The Output panel

The Output panel displays output generated by LotusScript Print statements that you put in your scripts. Whenever the IDE encounters a Print statement, it sends output from that statement to the Output panel. Including Print statements such as the following can be a useful alternative to setting breakpoints in a script:

```
Print "Procedure SetRS has created a result set using the value "
Print "of variable CurrentResultSet: " CurrentResultSet
```

The Variables panel

The Variables panel displays information about variables for the current procedure and lets you change their values while debugging the procedure. This panel is available only while you are debugging and displays information about variables used by the procedure that is currently selected in the Calls drop-down box.

Variables are listed by name in the order they are declared. Their value and current data type are also listed. A variable that has members (arrays, lists, classes, and types) is presented as an expandable entry in the panel. Expand the entry to view its members, their values, and data types.

Working with more than one Debugger

When you step from a procedure contained in one document into a procedure contained in another document, the Script Debugger for the second document is opened or is activated to display that procedure.

If Script Editors are already open for several documents when a breakpoint encountered, Debuggers are displayed in place of the Editors, whether or not scripts for those documents are executing.

Finding and Fixing Errors in the Script Editor

You can use the Errors drop-down box to view descriptions of errors and to navigate to the line of script containing the error.

The Errors drop-down box reports details about errors in the following format:

object: *script*: *line#*: *error message*

Correcting an error in the current document

To correct an error in the current document, do the following:

1. Click the error message in the Errors drop-down box.

   The IDE navigates to the script line that contains the error.
2. Correct the error.

Tip Press F1 to view Help for the error message.

Correcting an error in an included file

To correct an error in an included file, do the following:

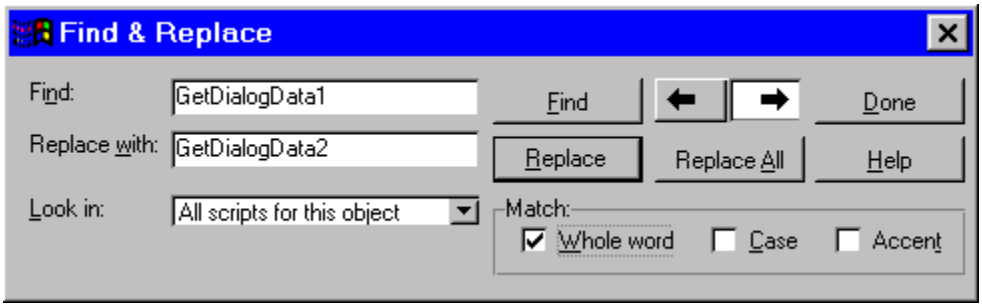
1. Click the error message in the Errors drop-down box.

   The IDE displays the line that contains the %Include directive for the file.
2. Edit the file using a text editor and correct the error.

Finding and Replacing Text in the Script Editor

You can find and replace text that you wrote in your scripts. Find and replace is particularly useful if you need to update all references to new names of variables or scripts.

Choose Edit - Find and Replace to display the following dialog box:



"Find" box
Enter the text to find.

"Replace with" box
Enter text in the "Replace with" box to replace text that you entered in the "Find" box. If there is no text in the "Replace with" box and you click the Replace or Replace All button, the Script Editor replaces text in the "Find" box with an empty string.

"Look in" drop-down box
Click this drop-down box to specify a scope for the search:

- "Current script only" searches only the script that is currently displayed in the Script Editor or Script Debugger.
- "All scripts for this object" searches all scripts for (Globals) or for the current object. This is the default.
- "All scripts for this document" searches all scripts in the current document.

"Find" button
Click this button to begin the search. The "Find" button is disabled if you did not enter text in the Find box.

Left button
Click this button to search backwards, that is, to the left of the insertion point.

Right button
Click this button to search forwards, that is, to the right of the insertion point.

Replace button
Click this button to replace the first occurrence of the "Find" text with the "Replace" text after (if searching forward) or before (if searching backward) the insertion point.

Replace All button
Click this button to replace all occurrences of the "Find" text with the "Replace" text.

"Match" checkboxes
Specify whether the Script Editor pays attention to whole words, case, and accent when searching for text. These options are not exclusive.

- Select Whole Word to match your entire search string. For example, if you were searching for "on", and did not select Whole Word, your search would find the strings "only", "honest", "on," and "Donald". If you select Whole Word, the search finds only "on".
- Select Case if you wish to match the case of your search string. If you are searching for "the", and select Case, your search will not find "The" or "tHe".
- Select Accent if you wish to match diacritical marks, such as umlauts and accents grave. Also check this box if you are searching through text written in one of the double-byte Asian languages.

Done button
Click this button to close the Find & Replace dialog box.

Overview: Fixing Errors Reported in the Script Debugger

If a run-time error occurs, the following happens in the Script Debugger:

- An error dialog box is displayed.
- The script containing the error is displayed in the Script Debugger.
- An arrow in the breakpoint gutter marks the statement causing the error.
- All Debug menu items are disabled except for breakpoint features and Stop Execution.

If you choose Debug - Stop Execution, the IDE closes the Script Debugger and displays the Script Editors in place of Script Debuggers in any open IDE windows.

Tip Press F1 to view Help for the error message.

Formatting Scripts in the Script Editor

The Script Editor provides a Smart Indenting feature that automatically provides the appropriate indentation for each script statement you enter. Smart Indenting is on by default.

Enabling automatic line indentation

If Smart Indenting is disabled, do the following to enable it:

1. Choose File - Script Preferences to display the Script Preferences dialog box.
2. Click the Basics panel.
3. Click the checkbox named Smart Indenting so it is checked.
4. Click OK to close the dialog box.

Creating blocks of script automatically

In many cases, the Script Editor automatically completes block statements based upon what you enter as the beginning statement of a block. When you enter a beginning statement such as Do, the Script Editor appends a blank line after the Do statement, appends a Loop statement to complete the block, and positions the insertion point on the blank line.

Here are the rules for block statement completion:

- If you enter a Sub, Function, or Property statement outside a class and move the insertion point off the line, the Script Editor completes the statement and creates a new procedure.
- If you enter a Sub, Function, or Property statement within a class and press ENTER, the Script Editor completes the statement.
- If you enter a Type or Class statement in (Declarations) and move the insertion point off the line, the Script Editor completes the statement.
- If you enter a Type or Class statement into any script other than (Declarations) and move the insertion point off the line, the Script Editor completes the statement and moves it to (Declarations).
- If you enter any other block statement into a script and press ENTER, the Script Editor completes the statement.

Beginning and end statements

Enter the following beginning statements to have the IDE complete the block.

Note Optional arguments are indicated in square brackets.

Beginning Statement	*End Statement*
Sub *subname* [*(arglist)*]	End Sub
Function *functionname* [*(arglist)*]	End Function
Property Get *name* [as datatype]	End Property
Property Set *name* [as datatype]	End Property
Do	Loop
For *countVar* = *first* to *last*	Next
ForAll *refVar* in *Container*	End ForAll
While *condition*	Wend
If *condition* Then	End If
Select Case *selectExpr*	End Select
With *objectRef*	End With
Type *typename*	End Type
Class *classname* [As *baseclass*]	End Class

Other automatic formatting

When you close the IDE, activate some other window, or check script syntax, the IDE completes any unterminated multiline strings, %REM blocks, and Sub, Function, Property, Type, and Class blocks.

Getting Context-Sensitive Help in the Script Editor, Debugger, and Browser

Getting context-sensitive Help In the Script Editor or Script Debugger

Do the following to get context-sensitive Help in the Script Editor or Script Debugger:

1. Select or move the insertion point on a LotusScript keyword or the name of a product class, method, or property.
2. Press F1 to display Help for that keyword or name.

   Note If the current line or the preceding line contains a syntax error and a keyword is not selected, Help displays information about the syntax error.

Getting context-sensitive Help in the Browser

Do the following to get context-sensitive help in the Browser:

1. Select an entry for a LotusScript keyword, for an OLE class or class member, or for a product class, method, or property.
2. Press F1 to display Help for the selected LotusScript, OLE, or product item.

Overview: Getting Help for Error Messages in the IDE

Context-sensitive Help is available for error messages displayed in the Errors drop-down box in the Script Editor. To view Help for one of these errors, select the error in the Errors drop-down box and press F1.

Tip You can also get Help for a syntax error by placing the insertion point on the line of the script (or the next line) that caused the error and pressing F1.

Run-time error messages are displayed in message boxes. To view Help for one of these error messages, press F1 while the message box is displayed.

You can also view Help for these error messages when you are in IDE Help by searching for "messages" and going to the IDE Error Messages topic. This topic contains a list of error messages. When you select a message, Help for the message is displayed.

Getting Help on the IDE

Displaying general information for the Script Editor or Script Debugger
Click anywhere in the Script Editor or Script Debugger pane (except inside a script block) and press F1.

Displaying general information for the Breakpoint, Browser, Output, and Variables panels
Click anywhere in a panel and press F1.

If you click an item in a Browser list and press F1, Help for the item is displayed.

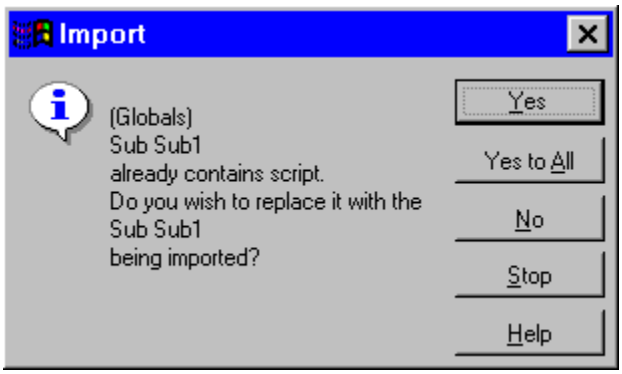
Displaying information about menus, drop-down lists, and activities you can perform
Use Help Index or the IDE Help Contents topic.

Importing Scripts in the Script Editor

You can import scripts that you wrote in a word processor or text editor into the Script Editor. Scripts are added to (Globals) in your document if you are working in (Globals) when you import them. Similarly, scripts are added as object scripts if you import them while working with a product object.

Importing a script into the Script Editor

1. Select a script in (Globals) or an object script to receive the imported script.
2. Choose File - Import Script.
3. Select the text file that you wish to import.
4. Click Open.
5. Resolve any conflicts between existing scripts and imported ones. If the text file contains scripts with the same names as those of existing scripts in your document, the Script Editor displays a dialog box asking you whether you want to replace the existing script with the imported script. Choose one of the options provided in the dialog box:



The newly-imported scripts appear in the Script drop-down box.

Overview: Managing Breakpoints in the Breakpoints Panel

The Breakpoints panel offers a convenient way to navigate to the locations of breakpoints that you set in your scripts and to manage your breakpoints.

Navigating to lines in the Script Editor or Script Debugger

You can navigate to the location of a breakpoint in the Script Editor or Script Debugger in the following ways:

- Click the item in the Breakpoints panel to navigate to the breakpoint.
- Double-click the item in the Breakpoints panel to navigate to the breakpoint and to activate the Script Debugger or Script Editor.
- Select the item in the Breakpoints panel and press ENTER to navigate to the breakpoint and to activate the Script Editor or Script Debugger.

Clearing breakpoints

To clear breakpoints in the Breakpoints panel, do one of the following:

- Press DEL or choose Clear Breakpoint in the Script or Debug menu to remove a breakpoint from the Breakpoints list.
- Choose Clear Breakpoints for *document* in the Script or Debug menu to clear all breakpoints that you set in the current document.

Disabling or enabling breakpoints

To disable or enable breakpoints in the Breakpoints panel, do one of the following:

- Choose Disable Breakpoint in the Script or Debug menu to disable the selected breakpoint in the Breakpoints list.
- Choose Disable Breakpoints for *document* in the Script or Debug menu to disable all breakpoints in the current document. This option is available only if your document contains enabled breakpoints.
- Choose Enable Breakpoint in the Script or Debug menu to enable the selected breakpoint in the Breakpoints list. This option is available only if all the breakpoints in your document are disabled.
- Choose Enable Breakpoints for *document* in the Script or Debug menu to enable all breakpoints in the current document.

Overview: Managing Text in the Script Editor

The Script Editor enhances the display of the scripts that you write so you can more easily understand and modify them.

To view and modify your current preferences for the Script Editor, choose File - Script Preferences.

Automatic formatting

The IDE automatically formats each script so that:

- LotusScript keywords appear initial-capped; directives and the Rem keyword appear in uppercase.
- Identifiers and commented lines appear exactly as you enter them.
- Lines beginning with labels are left-justified.
- If Smart Indenting is enabled, directives and lines beginning with labels are left-justified. Lines within a block are indented by one tab stop.

Entering script statements

In general, a line in a script contains one script statement. You can enter multiple statements on a single line by putting a colon between the statements. When you move off the line, the colons are converted to carriage returns and the statements are split into separate lines. The lines are formatted automatically.

If you enter a colon after an identifier, the identifier is treated as a label.

Overview: Online Help in the IDE

Help provides information about the IDE and the ways you can use it to create and test scripts. You can access this information by pressing F1 to display context-sensitive Help for the active part of the IDE window, or by using the Help menu to select a topic.

Context-sensitive Help

In the IDE, context-sensitive Help is available for:

- The Script Editor and Script Debugger panes
- The Browser, Breakpoints, Output, and Variables panels
- LotusScript keywords displayed in the Script Editor, Script Debugger, or Browser panel
- Product classes, constants, variables, and procedures displayed in the Script Editor, Script Debugger, or Browser panel
- Many OLE Automation classes (their members and properties) displayed in the Browser panel
- Syntax errors displayed in the Errors drop-down box in the Script Editor
- Run-time error messages displayed in message boxes in the Script Debugger

Help contents and index

You can also use the IDE Help Contents topic and Index to view Help for error messages and to find out how to use menus, the keyboard and mouse, the Object and Script drop-down lists, and the Breakpoints, Browser, Output, and Variables panels to create and manage scripts.

Opening LotusScript Help in the IDE

You can get context-sensitive Help for LotusScript keywords while you are working in the Script Editor, Script Debugger, or Browser panel.

Getting Help in the Script Editor or Script Debugger

1. Select a LotusScript keyword or move the insertion point on a keyword.
2. Press F1 to display Help on the keyword.

   Note If the current line or the preceding line contains a syntax error and a keyword is not selected, Help displays information about the syntax error.

Getting Help in the Browser

1. Select an entry for a LotusScript keyword.
2. Press F1.

   Help for the keyword is displayed.

Pasting from the Browser into a Script

When the Script Editor is active, you can paste a name from the Browser into the current script.

1. Select the name of a LotusScript language element, product class, constant, sub, function, or variable, or OLE class identifier In the Browser.
2. Click Paste Name at the top right of the Browser panel to paste that name into your current script.
3. Complete the statement that uses the keyword, identifier, or name in your script.

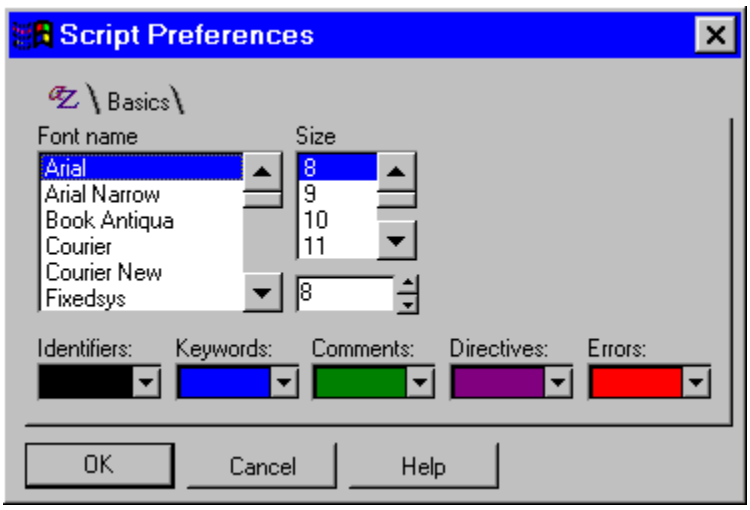
Setting Preferences in the Script Editor

You can customize the way the Script Editor displays lines of script and manages some of its automatic features.

To set these script preferences for all Script Editor windows, choose File - Script Preferences.

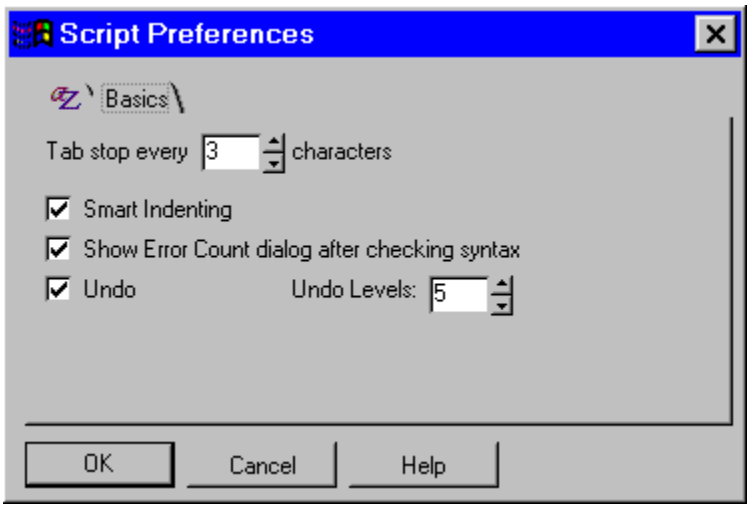
Text Preferences

Use the first panel in the Script Preferences dialog box to set default font and color preferences for lines of script in the Script Editor.



Preferences for other features

Use the Basics panel in the Script Preferences dialog box to set options for features or to enable or disable them entirely.



Tab stop every *n* characters

Specify how many characters wide a tab is when displayed in the Script Editor or Script Debugger. The default tab size is three characters; you can specify any setting between 2 and 16. You do not need to press TAB to indent lines in the Script Editor if the Smart Indenting feature is enabled; the Script Editor automatically indents lines when you complete them.

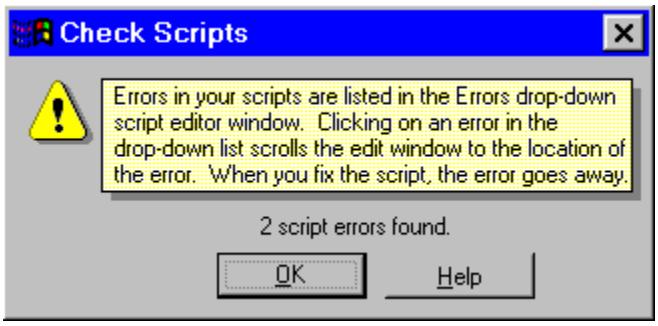
Note The IDE uses this setting when it displays scripts containing tabs that you have imported into the Script Editor.

Smart indenting

Specify whether the IDE automatically indents statements as you complete them in the Script Editor. By default, the Script Editor uses the opening and closing block keyword such as If and End If to determine indentation levels.

Show Error Count dialog after checking syntax

Specify whether the IDE reports how many errors it encounters when it checks your scripts.



Undo
Specify whether the Undo is available for the Script Editor. By default, the IDE keeps track of the last five text edits that you performed (entering, modifying, deleting, and pasting text or defining a new script). If you disable Undo in the panel, you cannot undo text edits that you perform in the Script Editor.

Note When you change objects while you are writing scripts, you cannot undo edits made to scripts for the previous object.

Undo levels
Specify the number of gestures that the IDE tracks for the current IDE window.

Renaming Procedures in the Script Editor

You can rename sub, function, and property procedures in your document; you cannot rename (Options) or (Declarations). To rename a procedure, do the following:

1. Navigate to the procedure.
2. Select the name of the procedure in the first line of the script.

   Example: `Sub GetNotesMail (TemplateName as String)`
3. Change the name of the procedure.

   Example: `Sub GetNotesNetMail (TemplateName as String)`

   Note You cannot have duplicate names in (Globals) or within the scripts for a particular object. If you enter a duplicate name under these circumstances, the Script Editor displays a message notifying you of the error.
4. Move the insertion point off the line. The Script Editor lists the procedure with its new name in the Script drop-down box.

Note If you rename a predefined script such as Initialize, Terminate, or an event procedure, the IDE creates two scripts. One has the new name and any statements that were contained in the original script. The other has the original name of the predefined script and is empty. Both scripts are listed in the Script list. You cannot rename (Options) or (Declarations).

Choose Edit - Find and Replace to replace references to the old name in your existing statements.

Saving Scripts in the Script Editor

The IDE saves your scripts when you save its product document.

Saving a script

1. Choose File - Save Scripts for *document.*

The IDE then notifies your product to save the current document.

Tip If you wish to save your scripts without displaying the Save Scripts dialog box, select "Do not display this message in the future".

Overview: Selecting and Copying Text in the Script Debugger

Statements displayed in the Script Debugger are read-only. You can select statements, copy them to the Clipboard and paste them back into the Script Editor when it is displayed. You cannot edit text or paste text in the Script Debugger.

Selecting an Object to Work With in the Script Editor

There are two ways to select an object for scripting.

Selecting an object while working in the IDE

To select the scripts associated with an object, do the following:

1. Click the Object drop-down box to display a list of product objects currently in your product document.
2. Click the right arrow or down arrow controls to navigate the hierarchy of objects displayed in the drop-down box.
3. Click the name of the object to select it and to close the list.

Selecting an object in your product

1. Navigate to the object that you want to script in your product.
2. Click the object to select it or activate it.
3. Right-click to display the shortcut menu for the selected object.
4. Choose Show Script Editor for *object* (or the equivalent) from the shortcut menu.

Note You can double-click an object in the Lotus Dialog Editor to activate the IDE (if it is not currently active) and to edit scripts for that object.

Selecting a Script in the Calls Drop-down Box in the Script Debugger

To select a script in the Calls drop-down box, do the following:

1. Click the Calls drop-down box to display a list of procedure in the current call stack.
2. Click the procedure to display it in the Script Debugger and to display information about its variables in the Variables panel.

The Script Debugger is read-only; you cannot modify scripts while the Script Debugger is active. To edit scripts, you must stop script execution and make changes in the Script Editor. You can change the value of variables in the Variables panel while debugging a script.

Selecting Text in the Script Editor

To select text in the Script Editor, you can use the mouse or the keyboard.

Selecting text with the mouse

- Double-click a word to select the word, .
- Click the beginning of a block and drag the highlight to the end of that block to select it.
- SHIFT+click to select text from the insertion point to the mouse pointer.

Selecting text with the keyboard

SHIFT+RIGHT	Selects the character to the right of the insertion point.
SHIFT+LEFT	Selects the character to the left of the insertion point.
SHIFT+UP	Selects from the position of the insertion point to the corresponding position in the previous line.
SHIFT+DOWN	Selects from the position of the insertion point to the corresponding position in the next line.
SHIFT+HOME	Selects text from the insertion point to the beginning of the line.
SHIFT+END	Selects text from the insertion point to the end of the line.
SHIFT+CTRL+RIGHT	Selects the next word or the rest of the word containing the insertion point.
SHIFT+CTRL+LEFT	Selects the previous word or the beginning of the word containing the insertion point.
SHIFT+CTRL+UP	Selects from the insertion point to the beginning of the first line displayed in the pane.
SHIFT+CTRL_DOWN	Selects from the insertion point to the beginning of the last line displayed in the pane.

Selecting a Script in the Script Drop-down Box in the Script Debugger

To select a script in the Script drop-down box in the Script Debugger, do the following:

1. Click the Object drop-down box to display a list of objects in the document.
2. Select the object whose script you want to display.
3. Click the Script drop-down box to display a list of scripts for the object.
4. Select the script you want to look at.

   The script is displayed in the Script Debugger.

Note The Script Debugger is read-only; you cannot modify scripts while the Script Debugger is active. To edit scripts, you must stop script execution and make changes in the Script Editor. You can change the value of variables in the Variables panel while debugging a script.

Setting and Clearing Breakpoints in the IDE

You can set or clear breakpoints in your scripts with the mouse or with Script or Debug menu commands. The IDE sets breakpoints for individual lines; you cannot set breakpoints for a group of selected statements. You can clear breakpoints for individual statements, for a group of selected statements, or for all statements in your document.

Setting and clearing breakpoints with the mouse

To set or clear a breakpoint, do one of the following:

- Click next to the line in the breakpoint gutter in the Script Editor or Script Debugger to set a breakpoint on a line. If the line cannot legally have a breakpoint, the IDE does not set one.
- Click the breakpoint symbol in the breakpoint gutter to clear a breakpoint.

Script - or Debug - Set Breakpoint or Clear Breakpoint

This menu item is a toggle for setting or clearing breakpoints. The Script menu and Debug menu display either Set Breakpoint or Clear Breakpoint based upon the contents of your current selection of lines. If you have selected multiple statements that contain one or more breakpoints, for example, the IDE displays Clear Breakpoint in the Script or Debug menus. If the selection contains no breakpoints and the current line in the selection can legally have one, the IDE displays Set Breakpoint in the Script or Debug menus.

Choose Set Breakpoint in the Script or Debug menu to insert a breakpoint on the current line or on the current line in the selection of lines in the script. This menu item is available if there is no breakpoint set for the current statement (or in the current selection of statements) and it can legally have a breakpoint.

Choose Clear Breakpoint in the Script or Debug menu to remove breakpoints from the current line or from a selection of lines. This menu item is available if the current line contains a breakpoint or the current selection of lines contains one or more breakpoints. Clear Breakpoint clears all breakpoints in a selection of lines.

Note If the Breakpoints panel is active, choose Clear Breakpoint to remove the selected breakpoint in the list of breakpoints.

Script - or Debug - Clear Breakpoints for *document*

Choose Clear Breakpoints in the Script or Debug menu to delete all breakpoints that you set in the current document. This menu item is dimmed if there are no breakpoints set in your document.

Clearing breakpoints in the Breakpoints panel

To clear a breakpoint displayed in the Breakpoints panel, select it and press DEL.

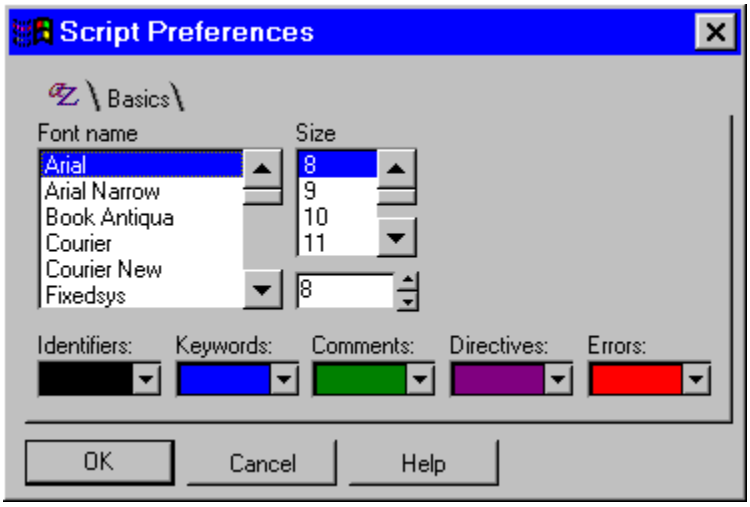
Setting Script Preferences in the IDE

You can customize the way the Script Editor displays lines of script and manages some of its automatic features such as tab widths, Smart Indenting, and Undo.

To set these properties for all Script Editor windows displayed on your system, choose File - Script Preferences.

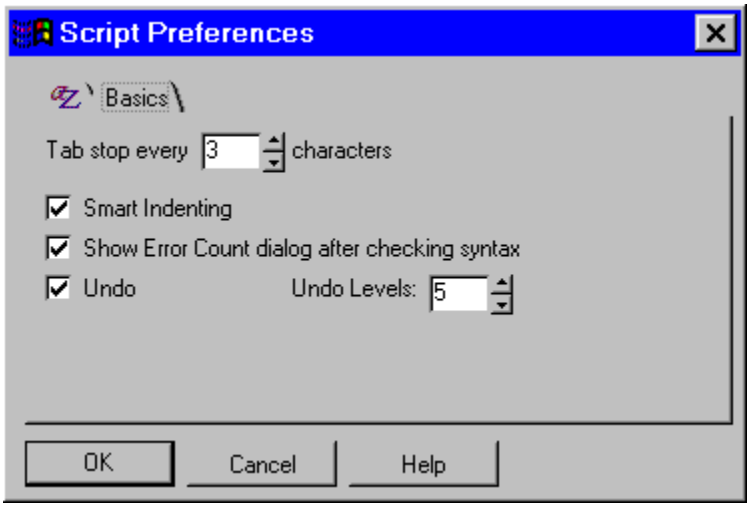
Text Preferences

Use the first panel in the Script Preferences dialog box to set default font and color preferences for lines of script in the Script Editor.



Preferences for other features

Use the Basics panel in the Script Preferences dialog box to set options for features or to enable or disable them entirely.



Tab stop every *n* characters

Specify how many characters wide a tab is when displayed in the Script Editor or Script Debugger. The default tab size is three characters; you can specify any setting between 2 and 16. You do not need to press TAB to indent lines in the Script Editor if the Smart Indenting feature is enabled; the Script Editor automatically indents lines when you complete them.

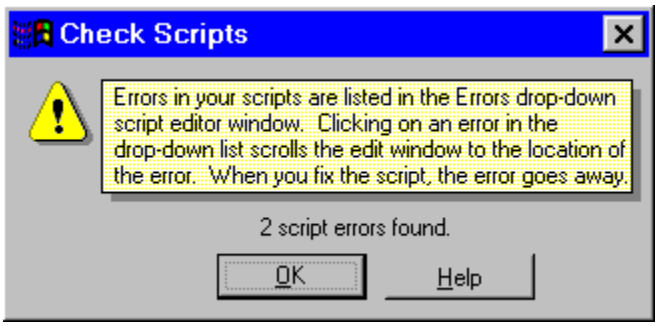
Note The IDE uses this setting when it displays scripts containing tabs that you imported into the Script Editor.

Smart Indenting

Specify whether the IDE automatically indents statements as you complete them in the Script Editor. By default, the Script Editor uses the opening and closing block keywords such as If and End If to determine indentation levels.

Show Error Count dialog after checking syntax

Specify whether the IDE reports how many errors it encounters when it checks your scripts.



Undo

Specify whether Undo is available for the Script Editor. By default, the IDE keeps track of the last five text edits that you performed (entering, modifying, deleting, and pasting text or defining a new script). If you disable Undo in the panel, you cannot undo text edits that you perform in the Script Editor.

Note When you change objects while you are writing scripts, you cannot undo edits made to scripts for the previous object.

Undo levels

Specify the number of text changes that the IDE tracks for the current IDE window. You can specify any setting between 0 and 10.

Showing and Hiding Panes in the IDE Window

There are several ways to show, hide, or activate panes in the IDE:

- Press F6 to toggle activation between the Script Editor or Script Debugger pane and the panel in the Utilities pane that was active most recently.
- Press F4 to activate the Browser panel.
- Drag the Pane Splitter to resize the relative height of a pane in the IDE window.
- Drag the Pane Splitter to the top or bottom of the IDE window to maximize either the Script Editor or Script Debugger pane or the Utilities pane.
- Double-click the Pane Splitter to toggle between single and double panes.
- CTRL+click the Pane Splitter to restore the default heights for panes in the IDE window.

Stepping Through a Script in the Script Debugger

Stepping through a script

When script execution stops at a statement, either because of a breakpoint or a previous step command, choose Debug - Step. The next statement is executed. If the statement contains a procedure call, script execution stops at the first statement in the procedure.

Stepping out of the current script

When script execution stops at a statement, choose Debug - Step Exit. Script execution continues until the current procedure has completed and stops at the first statement following the call, unless a breakpoint is encountered first.

Stepping over the script called in a breakpoint line

When a script execution stops at a statement, choose Debug - Step Over. The statement is executed, and if it contains a call, the entire procedure is executed. Execution stops at the next statement unless a breakpoint is encountered first.

Switching Between IDE Windows

If you have more than one IDE window displayed, click the IDE window to activate it.

Note When you close or deactivate the IDE, it completes the following automatically: %Rem blocks, multiline strings, sub, function, and property blocks, and user-defined type and class blocks. The IDE reformats line indentations and saves all scripts.

Switching Between Panes in the IDE Window

You can the use the mouse, keyboard, or menu commands to switch between panes in the IDE window.

Using the mouse to switch between panes

Click in the alternate pane to activate it. If you are currently in the Browser panel and you want to edit a script in the Script Editor pane, click in the Script Editor pane.

Using the keyboard to switch between panes and panels

Do one of the following:

- Press F6 to toggle activation between the Script Editor or Script Debugger pane and the panel in the Utilities pane that was active most recently.
- Press SHIFT+CTRL+RIGHT or SHIFT+CTRL+LEFT to switch between panels in the Utilities Pane.

Using menus to switch between panes

Do one of the following:

- Choose View - Script Pane to switch to the Script Editor.
- Choose View - Browser, Output, or Breakpoints to switch to those panels in the Utilities pane.

  Note If the Script Debugger is active, you can switch to the Variables panel as well.
- Right-click in a panel to display its shortcut menu and choose Browser, Breakpoints, or Output.

Overview: Debugging Scripts in the Script Debugger

When an enabled breakpoint is reached during debugging, script execution is stopped and the script containing the breakpoint is displayed in the Script Debugger. You can display another procedure by selecting it in the Calls drop-down box at the bottom of the Debugger pane, or in the Script drop-down box at the top right.

Using the Calls drop-down box

The Calls drop-down box contains the name of the current procedure and the name of the procedure that called it. Each item in the drop-down box shows the name of the object that contains the procedure and the name of the procedure itself. When you select an item, the Script Debugger displays the script that contains the procedure. A white arrow in the breakpoint gutter marks the current script statement. A yellow arrow in the breakpoint gutter marks the location of a call to the procedure that contains the current statement.

When you select a procedure in the drop-down box, the Variables panel displays variables for the current procedure, their current values, and any global variables used by the procedure. You can use the Variables panel to view information about a variable or to change its value.

Items in the list are dimmed if they are procedure contained in another product document, referenced with the %Include directive, or used with the Use statement. You cannot select these items.

Using the Script drop-down box

If you want to switch to a script that is not contain a procedure in the current call stack, you can use the Object and Script drop-down lists to navigate to the script. The script you select is displayed in the Script Debugger, but the Calls drop-down box and the Variables panel continue to display information about the last script selected in the Calls drop-down box. You can return to any procedure listed in the Calls drop-down box by selecting it.

Testing a Sub in the Script Editor

Choose Script - Run Current Sub or press F5 to test how your sub behaves when it executes.

Tip Set a breakpoint in your sub so you can watch it execute a statement at a time in the Script Debugger.

Note You can only test subs that have no parameters. If you choose Script - Run Current Sub for a sub that has parameters, the IDE displays a message notifying you that you cannot run the sub.

Overview: Breakpoints, Browser, Output, and Variables Panels

The Breakpoints, Browser, Output, and Variables panels display information about scripts that you create and test in the Script Editor and the Script Debugger. You can use them to change scripts and to manage breakpoints for debugging scripts.

Browser panel

The Browser panel displays a list of LotusScript language elements, classes, procedures, variables, and constants defined by your product, and type libraries and classes for OLE Automation objects. You can select items in the Browser and paste their names into the Script Editor. You can view Browser listings when the Script Editor or the Script Debugger is active.

The Breakpoints panel

The Breakpoints panel lists all breakpoints that you set in your scripts in the order that you set them, and lets you navigate to them, clear them, enable them, or disable them. The Breakpoints panel lists each breakpoint in the following format:

object: *script*: *line#*

Note (Disabled) is displayed next to the breakpoint statement if the breakpoint is disabled.

The Breakpoints panel offers a convenient way to navigate to the location of breakpoints, to enable breakpoints, or to disable breakpoints.

You can navigate to the location of a breakpoint in the Script Editor or Script Debugger in the following ways:

- Click the item in the Breakpoints panel to navigate to the breakpoint
- Double-click the item in the Breakpoints panel to navigate to the breakpoint and to activate the Script Debugger or Script Editor
- Select the item in the Breakpoints panel and press ENTER to navigate to the breakpoint and to activate the Script Editor or Script Debugger.

The Output panel

The Output panel displays output generated by LotusScript Print statements in your scripts. Whenever the IDE encounters a Print statement, it sends output from that statement to the Output panel. Including Print statements such as the following can be a useful alternative to setting breakpoints in a script:

```
Print "Procedure SetRS has created a result set using the value "
Print "of variable CurrentResultSet: " CurrentResultSet
```

The Variables panel

The Variables panel displays information about variables for the current procedure and lets you change their values while debugging the procedure. This panel is available only while you are debugging and displays information about variables used by the procedure that is currently selected in the Calls drop-down box.

Variables are listed by name in the order they are declared. Their value and current data type are also listed. A variable that has members (arrays, lists, classes, and types) is presented as an expandable entry in the panel. Expand the entry to view its members, their values, and data types.

Overview: The Script Debugger

In the IDE Script Debugger, you can set, clear, disable, and enable breakpoints and step through scripts to locate the source of problems that may occur while script is executing.

The Script Debugger is displayed automatically when script is executing and it encounters one of your breakpoints, encounters a Stop statement in your script, or causes a run-time error.

To display the Script Debugger, do the following:

- Set a breakpoint on the statement where you want to start to debug your script.

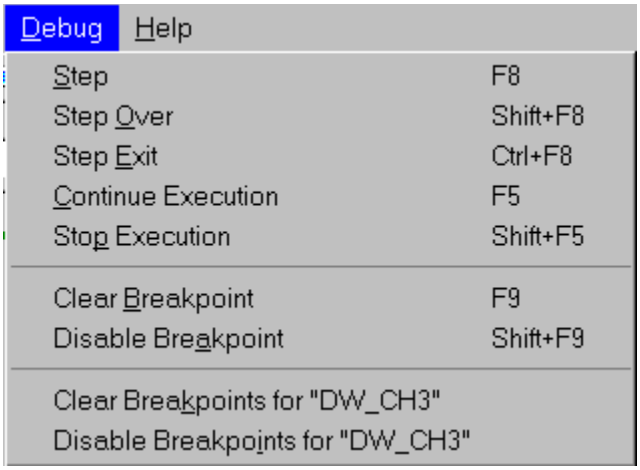
  Note If the statement continues over more than one line (using _) or contains a multiline string, set the breakpoint on the last line of the statement.
- Generate an event in your product that is required for an event procedure that you want to run. For example, to debug the Click event procedure for a Button object in your product, you first need to click the button.
- Choose Script - Run Current Sub to run a sub that has no parameters.
- Choose Edit - Script - Run (or the equivalent) in your product to run a sub in (Globals) that has no parameters.

When the Script Debugger is open:

- The Debug menu is displayed in the IDE menu.
- All scripts become read-only. You cannot edit them.
- The Browser, Output, Breakpoints, and Variables panels are accessible.
- The script containing the active breakpoint or run-time error is displayed initially.
- The breakpoint line becomes the current line.
- An arrow is displayed in the breakpoint gutter for the statement that is about to be executed or for the statement generating the run-time error.

Using the IDE Debug menu

The IDE Debug menu offers several options for debugging your scripts.



Step

Step lets you execute a script procedure one statement at a time. This lets you evaluate the effect of each statement on your product or on your script variables. If a statement in the current procedure calls another procedure, the Script Debugger also executes statements in that procedure one at a time. If another product window was active before the Script Debugger stepped to the current script statement, that product window is activated once again before the Script Debugger executes the next statement in your script. As you step through statements in your script, the Variables panel updates accordingly.

Choose Debug - Step or press F8 or for each statement that you step through.

Step Over

Step Over works the same as Step with the exception that Step Over does not stop inside procedures that are called by the current statement.

Step Exit

Step Exit completes execution of the current procedure and stops at the first statement after the current procedure

call. From there you can use other Step commands. If the current procedure was called from the product, then the script terminates normally. If another product window was active before the Script Debugger executed the current script, that product window is activated once again before script execution resumes.

Continue Execution
Continue Execution resumes script execution until the current script completes successfully or until a breakpoint is encountered. If another product window was active before the Script Debugger resumed execution of your script, that product window is activated once again before script execution resumes.

Stop Execution
Stop Execution terminates the execution of your current script at its current location. The IDE displays the Script Editor.

Note Script Debuggers that opened automatically during debugging are not closed when you stop execution.

Set Breakpoint or Clear Breakpoint
This menu item is a toggle for setting or clearing breakpoints. The Debug menu displays either Set Breakpoint or Clear Breakpoint based upon the contents of your current selection of lines. If you have selected multiple statements that contain one or more breakpoints, for example, the Debug menu displays Clear Breakpoint. If the selection of lines contains no breakpoints and the current line in the selection can legally have one, the Debug menu displays Set Breakpoint.

Choose Debug - Set Breakpoint to insert a breakpoint on the current line or on the current line in the selection of lines in the script. This menu item is available if there is no breakpoint set for the current statement (or in the current selection of statements) and it can legally have a breakpoint.

Choose Debug - Clear Breakpoint to remove breakpoints from the current line or from a selection of lines. This menu item is available if the current line contains a breakpoint or a selection of lines contains one or more breakpoints. Clear Breakpoint clears all breakpoints in a selection of lines.

Note If the Breakpoints panel is active, choose Clear Breakpoint to remove the selected breakpoint in the list of breakpoints.

Disable Breakpoint or Enable Breakpoint
This menu item is a toggle for disabling or enabling breakpoints for a line or selection of lines.

Choose Debug - Disable Breakpoint to have the IDE ignore breakpoints for the current line or selection of lines without deleting them. This menu item is available if the current line contains an enabled breakpoint or a selection of lines contain at least one enabled breakpoint. Disable Breakpoint disables all breakpoints in a selection of lines.

Choose Debug - Enable Breakpoint to enable a breakpoint that you previously disabled with Disable Breakpoint. This menu item is available if the current line contains a disabled breakpoint or a selection of lines contains at least one disabled breakpoint and no enabled ones. Enable Breakpoint enables all breakpoints in a selection of lines.

Note You can also use Disable Breakpoint and Enable Breakpoint to disable or enable breakpoints displayed in the Breakpoints panel.

Clear Breakpoints for *document*
Choose Script - Clear Breakpoints to delete all breakpoints in the current document. This menu item is dimmed if there are no breakpoints in the current document.

Disable Breakpoints for *document* or Enable Breakpoints for *document*
This menu item is a toggle for disabling or enabling breakpoints within your current document.

Choose Debug -Disable Breakpoints to disable all breakpoints in the current document. This menu item is dimmed if there are no breakpoints in the current document.

Choose Debug - Enable Breakpoints to enable all breakpoints in the document. This menu item is available only if all breakpoints in the document are disabled.

Overview: The Script Editor

In the IDE Script Editor, you can write and edit scripts, check script syntax, and set breakpoints for debugging scripts. The Script Editor initially displays a script associated with the object selected in your product.

Opening the IDE window from the Edit menu in a product

If you choose Edit - Show Script Editor (or the equivalent) in your product, the following applies:

- The Script menu is displayed in the main menu.
- The Browser, Output, and Breakpoints panels are accessible.
- A script is displayed.
  - By default, the Script Editor displays the last script that you edited.
  - If no script has been edited, the Script Editor shows the sub for the current object's default event.
  - If no default event is defined for the current object, the Script Editor shows (Declarations) for that object.

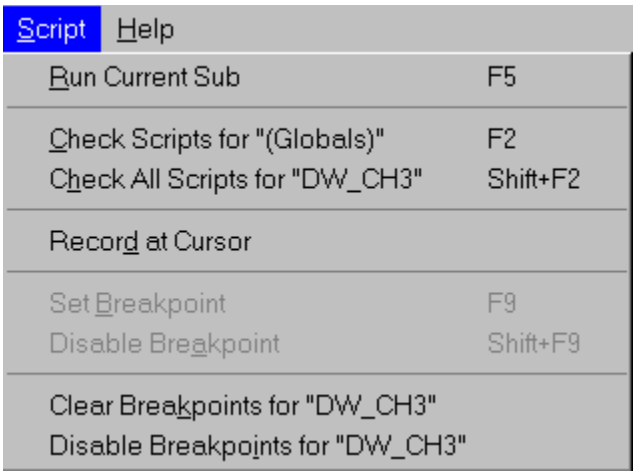
Opening the IDE window from a shortcut menu

If you select an object in your product and choose Show Script Editor (or the equivalent) from its shortcut menu, the following occurs:

- The Script menu is displayed in the main menu.
- The Browser, Output, and Breakpoints panels are accessible.
- The Script Editor displays a script. Depending on what you have selected or edited in previous sessions, the IDE displays a script relevant to your work in the document.
  - If you displayed this object's scripts earlier in your current session, the Script Editor restores the last script that you edited.
  - If you have not displayed this object's scripts during the current session, the Script Editor restores the first edited script for the object.
  - If the object does not contain any edited scripts, the Script Editor shows the procedure for the object's default event. If the object does not have a default event, the Script Editor shows (Declarations) for that object.

Working with the Script menu

When the Script Editor is active, the IDE displays a context-sensitive menu called Script.



Run Current Sub

Choose Script - Run Current Sub to execute a sub that has no parameters.

The IDE checks the syntax of all the object's scripts, all global scripts, and any scripts that they use (explicitly with the Use statement or implicitly with global scripts). If it encounters syntax errors, it reports them in the Errors drop-down box. If there are no errors, it executes the sub.

Check Scripts for *object*

Choose Script - Check Scripts for *object* to check the syntax for all the object's scripts, all global scripts, and any scripts that they use (explicitly with the Use statement or implicitly with global scripts).

The IDE completes the following automatically: %Rem blocks, multiline strings, sub, function, and property blocks, and user-defined type and class blocks. The IDE also reformats line indentations.

Check All Scripts for *document*
Choose Script - Check All Scripts for *document* to check all scripts contained in the current document and any scripts that they use with the Use statement.

Tip Use Script - Check Scripts for *object* or Script - Check All Scripts for *document* to update your current script after you have modified another script file that you reference with the %Include directive. Use Script - Check All Scripts for *document* to update your current script after you have modified an LSO file that your reference with the Use statement. Otherwise, the IDE continues to use the previous version of the referenced file.

Insert Template
Choose Script - Insert Template to insert the contents of a text file into the current script at the insertion point. Insert Template saves you time when working with comment headers or with commonly used scripts.

Note This feature may not be available for all products.

Record at Cursor or Stop Recording
Choose Script - Record at Cursor to begin recording commands that you perform in your product. You cannot see the commands while the IDE records them. Choose Script - Stop Recording to stop recording product commands and to insert the list of recorded commands into the Script Editor at the insertion point.

Note This feature may not be available for all products.

Set Breakpoint or Clear Breakpoint
This menu item is a toggle for setting or clearing breakpoints. The Script menu displays either Set Breakpoint or Clear Breakpoint based upon the contents of your current selection of lines. If you have selected multiple statements that contain one or more breakpoints, for example, the Script menu displays Clear Breakpoint. If the selection of lines contains no breakpoints and the current line in the selection can legally have one, the Script menu displays Set Breakpoint.

Choose Script - Set Breakpoint to insert a breakpoint on the current line or on the current line in the selection of lines in the script. This menu item is available if there is no breakpoint set for the current statement (or in the current selection of statements) and it can legally have a breakpoint.

Choose Script - Clear Breakpoint to remove breakpoints from the current line or from a selection of lines. This menu item is available if the current line contains a breakpoint or a selection of lines contains one or more breakpoints. Clear Breakpoint clears all breakpoints in a selection of lines.

Note If the Breakpoints panel is active, choose Clear Breakpoint to remove the selected breakpoint from the list of breakpoints.

Disable Breakpoint or Enable Breakpoint
This menu item is a toggle for disabling or enabling breakpoints for a line or selection of lines.

Choose Script - Disable Breakpoint to have the IDE ignore breakpoints for the current line or selection of lines without deleting them. This menu item is available if the current line contains an enabled breakpoint or a selection of lines contains at least one enabled breakpoint. Disable Breakpoint disables all breakpoints in a selection of lines.

Choose Script - Enable Breakpoint to enable a breakpoint that you previously disabled with Disable Breakpoint. This menu item is available if the current line contains a disabled breakpoint or a selection of lines contains at least one disabled breakpoint and no enabled ones. Enable Breakpoint enables all breakpoints in a selection of lines.

Note You can also use Disable Breakpoint and Enable Breakpoint to disable or enable breakpoints displayed in the Breakpoints panel.

Clear Breakpoints for *document*
Choose Script - Clear Breakpoints to delete all breakpoints in the current document. This menu item is dimmed if there are no breakpoints in the current document.

Disable Breakpoints or Enable Breakpoints for *document*
This menu item is a toggle for disabling or enabling breakpoints within your current document.

Choose Script - Disable Breakpoints to disable all breakpoints in the current document. This menu item is dimmed if there are no breakpoints in the current document.

Choose Script - Enable Breakpoints to enable all breakpoints in the document. This menu item is available only if all breakpoints in the document are disabled.

Overview: Using Breakpoints in Debugging

Set breakpoints in a script where you want it to pause temporarily during execution. This provides a way to examine script variables as script executes and to evaluate how the script behaves in your product.

You can set breakpoints in the Script Editor and Script Debugger on any line in a sub, function, property, or class method that contains a script statement, with the following exceptions:

- Comment lines
- The first line in a sub, function, or property definition
- Anywhere in a Terminate sub

Note If the statement continues over more than one line (using _) or contains a multiline string, set the breakpoint on the last line of the statement.

When you are editing or debugging a script, you can disable, enable, and clear a breakpoint.

Breakpoint symbols


Breakpoint


Current breakpoint


Disabled breakpoint


Current disabled breakpoint


No breakpoint

Current line indicators


Top current line


Nested current line

Combined breakpoint symbols and current line indicator


Disabled breakpoint, top current line


Current disabled breakpoint, top current line


Disabled breakpoint, nested current line


Current disabled breakpoint, nested current line


Breakpoint, top current line


Current breakpoint, top current line


Breakpoint, nested current line


Current breakpoint, nested current line

Overview: Using the IDE Breakpoints Panel

The Breakpoints panel lists all breakpoints that you set in your scripts in the order that you set them, and lets you navigate to them, clear them, enable them or disable them. The Breakpoints panel lists each breakpoint in the following format:

object: *script*: *line#*

Note (Disabled) is displayed next to the breakpoint statement if the breakpoint is disabled.

The Breakpoints panel offers a convenient way to navigate to the location of breakpoints, to enable breakpoints, or to disable breakpoints.

You can navigate to the location of a breakpoint in the Script Editor or Script Debugger in the following ways:

- Click the item in the Breakpoints panel to navigate to the breakpoint.
- Double-click the item in the Breakpoints panel to navigate to the breakpoint and to activate the Script Debugger or Script Editor.
- Select the item in the Breakpoints panel and press ENTER to navigate to the breakpoint and to activate the Script Editor or Script Debugger..

Overview: Using the IDE Browser Panel

The Browser panel displays lists of LotusScript language elements, classes, procedures, variables, and constants defined by your product, and OLE Automation type libraries and classes. You can select items in the Browser and paste their names into the Script Editor. You can view Browser listings when the Script Editor or the Script Debugger is active.

When you are working in the Browser, you can display Help for an item by selecting it and pressing F1.

Overview: Using the IDE Output Panel

When you execute scripts in a document, the IDE displays in the Output panel up to 1022 characters of output from each Print statement included in the scripts. This output is displayed in the order it is generated in the document.

You can review the output list to determine whether scripts are executing as you expect.

The IDE saves up to 500 lines of output generated by scripts running in your product. These lines are saved in your document whether the Script Editor or Script Debugger are open or not. You can run a script and open the Script Editor afterwards to view the output.

Overview: Using the IDE Variables Panel

The Variables panel displays information about variables in the current procedure and lets you change their values while you are debugging the script. This panel is available only while you are debugging and displays information about variables used by the procedure that is currently selected in the Calls drop-down box.

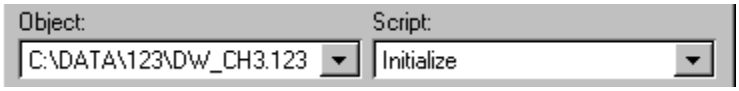
Variables are listed by name in the order they are declared. Their value and current data type are also listed. A variable that has members (arrays, lists, classes, and types) is presented as an expandable entry in the panel. Expand the entry to view its members, their values, and data types.

Overview: Using the Keyboard and Mouse in the IDE

You can use the keyboard to perform commands, move between panes and among panels in the IDE window, and select text in editable areas of the panes and panels. You can use the mouse to move quickly to any part of the IDE window, show or hide panes in the IDE window, select editable items, expand or collapse outline lists, manage breakpoints, and display shortcut menus for areas of the window.

Some general uses of the keyboard

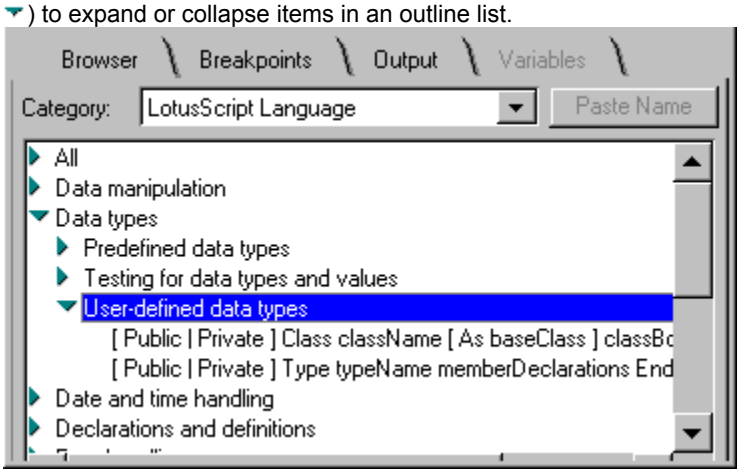
- Press F1 to display context-sensitive help.
- Press F4 to toggle activation between the Script Editor or Script Debugger pane and the Browser panel in the Utilities pane.
- Press F6 to toggle activation between the Script Editor or Script Debugger pane and the panel in the Utilities pane that was active most recently.
- Press + or - to expand or collapse an item in an outline list.
- Press SHIFT++ or SHIFT+- to expand or collapse all items in an outline list.
- Press TAB or SHIFT+TAB to move forward or backward between read-only text boxes such as Object: or Script:



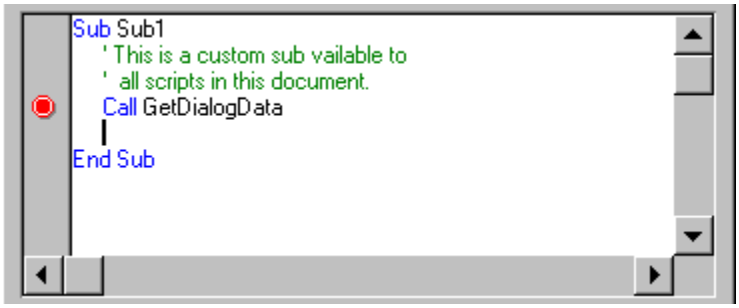
- Press CTRL+PG UP or CTRL+PG DN to navigate to the previous or next script. The IDE first navigates to the previous or next script for the current object, and then to the scripts for the next object.

Some general uses of the mouse

- Click the right arrow ( ) or down arrow ( ) to expand or collapse items in an outline list.



- Click next to a line in the breakpoint gutter to set a breakpoint on a line.
- Click the breakpoint in the gutter to clear a breakpoint,
- CTRL+click the breakpoint to disable or enable it.



Using the Keyboard in the Breakpoints, Browser, Output, and Variables Panels

Deleting a breakpoint from the Breakpoints panel
Select the breakpoint item and press DEL.

Canceling edits made to values in the Variables panel
Press ESC.

Applying a new value for a variable in the Variables panel
Press ENTER.

Moving between panels in the Utilities pane
Press SHIFT+CTRL+RIGHT or SHIFT+CTRL+LEFT.

Using the Keyboard in the Script Editor and Script Debugger

Moving the insertion point

RIGHT or LEFT	Moves the insertion point one character to the right or left.
UP or DOWN	Moves the insertion point up or down one line.
HOME	Moves the insertion point to the beginning of a line.
END	Moves the insertion point to the end of a line.
PG UP or PG DN	Moves the insertion point up or down one screen full of text.
CTRL+RIGHT	Moves the insertion point to the next word.
CTRL+LEFT	Moves the insertion point to the previous word.
CTRL+UP	Moves the insertion point to the first line displayed in the window.
CTRL+DOWN	Moves the insertion point to the last line displayed in the window.
CTRL+HOME	Moves the insertion point to the beginning of the script.
CTRL+END	Moves the insertion point to the end of the script.
CTRL+PG UP	Moves the insertion point to the previous script.
CTRL+PG DN	Moves the insertion point to the next script.

Using the Mouse in the IDE

Selecting text in a script
Do one of the following:

- Double-click a word to select it.
- Press SHIFT+click to select text from the insertion point to the mouse pointer.
- Click the beginning of a block of script text and drag the highlight to the end of the block to select a block of script text.

Viewing items in an outline list
Do one of the following:

- Click the right arrow to expand an item.
- Click the down arrow to collapse an item.

Displaying shortcut menus
Do one of the following:

- Right-click in the breakpoint gutter or in the Script Editor or Script Debugger.
- Right-click the Breakpoints or Output panel, or the values box in the Variables panel.

Controlling breakpoints from the Script Editor and Script Debugger
Do one of the following:

- Click in the breakpoint gutter to set a breakpoint on that line.
- Click a breakpoint to clear it.
- CTRL+click a breakpoint to disable or enable it.

Navigating to breakpoints from the Breakpoints panel
Do one of the following:

- Click an item in the Breakpoints list to navigate to the line of script containing the breakpoint without activating the pane.
- Double-click an item in the Breakpoints list navigate to the line of script containing the breakpoint and to activate the pane.

Overview: Viewing a List of Breakpoints in the Breakpoints Panel

The Breakpoints panel lists all breakpoints that you set in your scripts in the order that you set them, and lets you navigate to them, clear them, enable them, or disable them. The Breakpoints panel lists each breakpoint in the following format:

object: *script*: *line#*

Note (Disabled) is displayed next to the breakpoint statement if the breakpoint is disabled.

Viewing Globals for the Current Script

If you have global variables in your document, the Variables panel displays them in an outline list. To view the global variables, select and expand the Globals entry in the Variables panel.

The following variable names, values, data types are displayed:

- Variables declared in the current object's (Declarations)
- Variables defined in (Globals) (Declarations) and referenced by one of the scripts for the current object
- Variables defined in another document or LSO file specified in a Use statement and referenced in one of the scripts for the current object

Viewing Information about Products in the Browser

The Browser panel provides a view into classes, constants, and variables available for Lotus products and Lotus Dialog Editor controls. You can use the Browser when either the Script Editor or the Script Debugger is active. You can paste items from the Browser panel into the Script Editor.

You can also display Help for an item by selecting it and pressing F1.

Displaying a list of classes, constants, variables, or subs and functions

To display a list of classes, constants, variables, or subs and functions, do the following:

1. In the Category drop-down box, choose one of the product's classes, constants, subs, functions, or variables.
2. Expand or collapse items as needed.

Pasting a name into the current script

When the Script Editor is active, you can paste a name from the Browser into the current script. Do the following:

1. Select an item in the list. If the item is expandable, expand it and select one of its members.
2. Click Paste Name at the top right of the Browser panel to paste that name into your current script.
3. Complete the statement for that uses the keyword in your script.

Viewing LotusScript Language Elements in the Browser

You can display the syntax of LotusScript statements, functions, and operators in the Browser when the Script Editor or Script Debugger is active. You can also display Help for one of these elements by selecting it and pressing F1.

When the Script Editor is active, you can select an item and paste the keyword that it contains into the current script.

Displaying syntax

1. Choose LotusScript Language In the Category drop-down box.
2. To display a list of LotusScript statements, functions, and operators in alphabetical order, expand All.

   To display a list of elements in another category, expand the category.

Pasting a keyword into the current script

When the Script Editor is active, you can paste a name from the Browser into the current script.

1. Select an item from the list. If the item is expandable, expand it and select one of its members.
2. Click Paste Name at the top right of the Browser panel to paste that name into your current script.
3. Complete the statement that uses the keyword in your script.

Viewing Registered Type Libraries for OLE Classes in the Browser

The Browser panel also displays OLE information when the Script Editor or the Script Debugger is active. If you can instantiate the class with CreateObject, you can copy its application class identifier from the Browser into the Script Editor.

The Browser lists the following OLE information:

- OLE Automation classes

  Note When Help is associated with an OLE class, you can display Help for the application class identifier or OLE server method or property by selecting it in the Browser and pressing F1.
- Methods and properties for OLE servers
- Methods, properties, and events for OCX components embedded in your current product document

Displaying information about OLE classes

To dislay information about OLE classes, do the following:

1. In the Category drop-down box, choose OLE Classes.
2. From the list of registered OLE type libraries, select the one you want to look at.
3. From the list of classes in the type library, select the class you want to view information for.
4. Select Methods or Properties to view a list of methods or properties for the class.

Pasting an OLE class identifier into the current script

When an OLE class can be instantiated in a script with CreateObject, the application class identifier for the class is shown in parentheses next to the class name.

When the Script Editor is active, you can paste a name from the Browser into the current script. Do the following:

1. Select an item from an expanded category.
2. Click Paste Name at the top right of the Browser panel to paste that name into your current script.
3. Complete the statement that uses the name in your script.

Viewing Variables for the Current Script

To view the variables for your current procedure, do the following:

1. In the Calls drop-down box in the Script Debugger, select the script whose variables you want to view.
2. In the Variables panel, expand variable entries (as necessary) to display member variables, their values, and their data types.

If a value contains more than 42 characters, the first 40 characters, plus ... (ellipses), are displayed.

Note When a list displays the contents of a collection, numbers appear next to the values. These numbers represent the order of the contents; they cannot be used as indexes into the collection.

Overview: Working in the LotusScript IDE Window

Each IDE window is associated with one product document. To work with objects contained in a product document, open its IDE window and then view, write, and debug the scripts associated with those document objects. To work with scripts in a document that is not active, you must first retrieve that document in your product and then open its IDE window.

Opening an IDE window from your product

There are two ways to open the IDE window for a product document:

- RIGHT+click an object in your product to display its shortcut menu and choose Show Script Editor for *object* (or the equivalent).
- Choose Edit - Show Script Editor (or the equivalent) in your product.

Note When you open the IDE, the Script Editor and Script Utilities are displayed in separate panes on the window. When you debug a script, the IDE displays the Script Debugger and Script Utilities. The Variables panel is active while you debug your scripts.

Managing panes in the IDE window

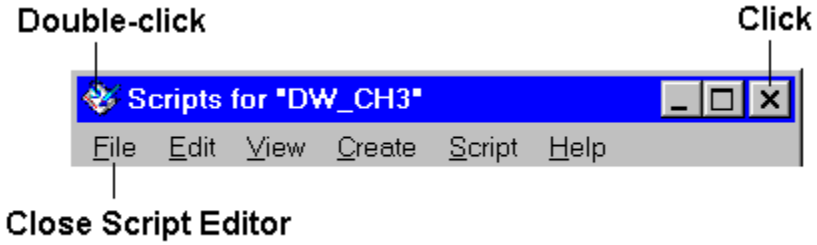
There are several ways to show, hide, or activate panes in the IDE:

- Press F6 to toggle activation between the Script Editor or Script Debugger pane and the current panel in the Utilities pane.
- Press F4 to activate the Browser panel.
- Drag the Pane Splitter to resize the relative height of a pane in the IDE window. Dragging the Pane Splitter all the way to the top or bottom of the IDE window maximizes either the Script Editor or Script Debugger pane or the Utilities pane.
- Double-click the Pane Splitter to toggle between single and double panes.
- CTRL+click the Pane Splitter to restore the default heights for panes in the IDE window.

Closing an IDE window

There are several ways to close the IDE window:

- Double-click the program icon in the top left corner of the IDE window.
- Click the Close button in the top right corner of the IDE window.
- Choose File - Close Script Editor in the IDE window.
- Close the product document associated with the IDE window.



Note When you close or deactivate the IDE, it completes the following automatically: %Rem blocks, multiline strings, sub, function, and property blocks, and user-defined type and class blocks. The IDE reformats line indentations and saves all scripts.

If you try to close the IDE while the Script Debugger containing the active breakpoint is running, it displays a message prompting you to continue debugging or to stop execution of the current script.

Overview: Working with Directives in the Script Editor

You can enter %Include and %Rem directives in scripts you create in the IDE, but not %If (%Else, %ElseIf, %EndIf) directives. If you want to use %If directives, you must enter them in a text file that you reference with the %Include directive. When entered directly in IDE scripts, %If directives generate syntax errors.

Entering %Rem and %End Rem

You can use the %Rem and %End Rem directives to enter a block of comments in any script.

If you type the %Rem directive and press ENTER, the Script Editor automatically appends a blank line followed by the %End Rem directive. If you type a %Rem directive and move the insertion point off the line, the Script Editor does not append a blank line and an %End Rem directive. In this case, the Script Editor automatically adds an %End Rem directive when you close the IDE, activate another window, or check script syntax. To make certain that a %Rem - %End Rem block is located exactly where you want it in your script, you should complete the block with an %End Rem statement while you are working in the IDE.

Working with %Include

When the IDE detects an error in a file called by an %Include statement, it reports the error at the line that contains the directive. After you edit an included file, you must get the latest changes by choosing Scripts - Check Scripts for *object* or Script - Check Scripts for *document.* Otherwise the IDE continues to use the previous version of the included file.

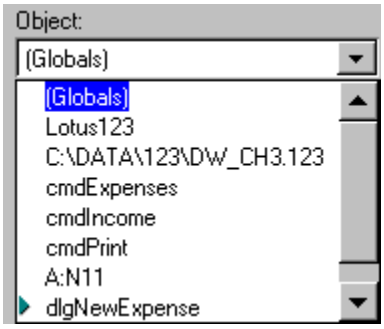
Working with (Globals) and Object Scripts in the IDE

You can use (Globals) to write scripts that are available to all other scripts in your document. Otherwise, you can limit the scope of your scripts to a particular object by writing them for that object. Use the Object drop-down box to select (Globals) or a particular object for your script.

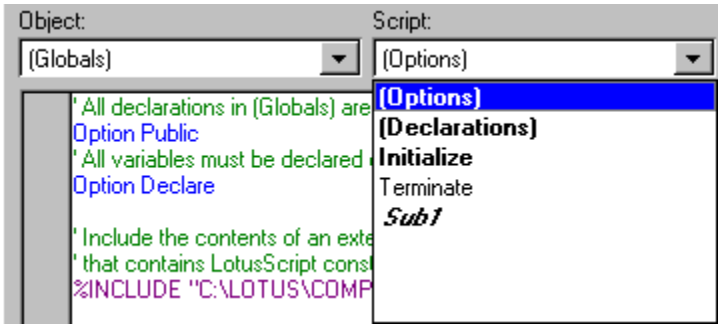
Working with scripts in (Globals)

To write scripts that are available to all scripts in your document, do the following:

1. Click the Object drop-down box.



2. Select (Globals), or its product equivalent, from the list.
3. Click the Script drop-down box to display a list of scripts in (Globals).



4. Select a script from the list.

Using predefined scripts in (Globals)

By default, the IDE provides (Options), (Declarations), Initialize, and Terminate scripts for you. All predefined scripts are empty with the exception of (Options), which contains the Option Public statement. Each predefined script is designed to contain certain statements or to perform a specific function.

Note To indicate which of these scripts you have modified, the IDE boldfaces it.

(Options) statements in (Globals)

The (Options) script in (Globals) is designed to contain the following statements:

- Option statements

  Note (Options) contains the statement Option Public by default. This makes Const, Dim, Type, Class, Sub, Function, and Property statements public by default. You can use the Public form of these statements to make them public explicitly or the Private form to make them unavailable to other scripts outside (Globals).
- Def*type* statements
- Use and UseLSX statements
- Const statements needed for Use and UseLSX statements

Note If you enter any of these statements, except for Const, in any other script in (Globals), the IDE automatically moves them to (Options).

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object, even if you are working in (Globals). To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

(Declarations) statements in (Globals)

The (Declarations) script in (Globals) is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts in your document
- Public, Private, Type, Class, and Declare Lib statements (external C calls)
- Const statements for those constants that you want to be available to all scripts in your document and are not needed for Use or UseLSX statements in (Options)

By default the (Declarations) script is initially empty.

Note If you enter Type, Class, or Declare Lib statements in any other script in (Globals), the IDE moves them to (Declarations) automatically. If you enter Dim, Public, Private, or Const statements outside the scope of a procedure in another script, the IDE moves them to (Declarations) automatically. Const statements in (Options) are the exception to this rule.

Initialize sub

Use the Initialize sub in (Globals) to initialize variables that you have declared in (Declarations). The Initialize sub executes before any of these variables are accessed and before any other scripts in (Globals) are executed. By default, the Initialize script is empty.

Note By contrast, use the Initialize sub for an object script to set up variables declared in the object's (Declarations) script. The Initialize sub for an object executes before the object's events are executed.

Terminate sub

Use the Terminate sub in (Globals) to clean up variables that you have declared in (Declarations) when you close your document or when you modify a script and execute it again. For example, you might use an Open statement to open a file containing data in Initialize and use a Close statement in Terminate to close it. By default the Terminate script is empty.

Note By contrast, use the Terminate sub for an object script to clean up variables defined in the object's (Declarations) script.

Adding scripts to (Globals)

While you are working in (Globals), you can add scripts to make them available throughout your document. There are three ways to add scripts to (Globals).

- Use Create - New Sub and Create - New Function to create new subs and functions in (Globals). The IDE automatically adds the name of the new procedure to the Script drop-down box.
- Enter a Sub, Function, or Property statement anywhere in (Globals) except within a class. The IDE automatically adds the name of the new procedure to the Script drop-down box for (Globals).
- Use File - Import Script when you are editing in (Globals) to import scripts that you want to be available to all scripts in your document. The IDE automatically adds the name of the new scripts to the Script drop-down box.

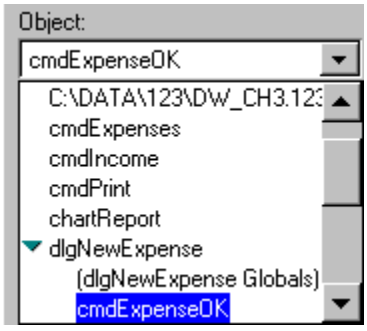
Note To distinguish scripts that you created from the predefined ones, the IDE displays them in bold italics in the drop-down box.

Working with object scripts

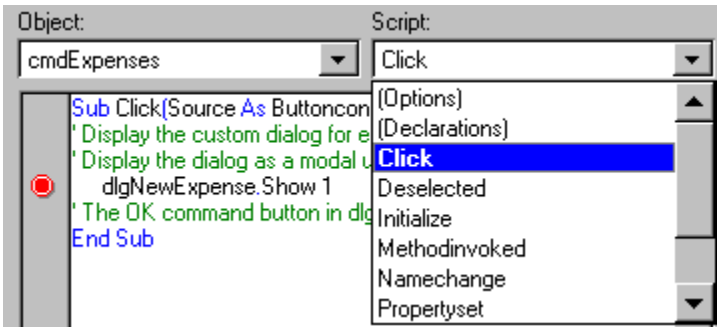
While you are working with product objects, you can enter statements for predefined scripts or add new scripts, making them available to all scripts for that object.

To select an object script, do the following:

1. Click the Object drop-down box to display (Globals) and a list of product objects in your current product document.



2. Select a product object from the list.
3. Click the Script drop-down box to display a list of predefined scripts, event procedures for that object, and user-defined scripts.



Note Each class of product object has its own set of event procedures.

4. Select the script that you want to modify.

Using predefined scripts for objects

By default, the IDE provides (Options), (Declarations), Initialize, and Terminate scripts for each product object. All predefined scripts are initially empty. Each predefined script is designed to contain certain statements or to perform a specific function with this object.

Note To indicate that you have modified one of the predefined scripts, the IDE boldfaces its name in the drop-down box.

(Options) statements

The (Options) script for an object is designed to contain the following statements:

- Option statements
- Def*type* statements
- Use and UseLSX statements
- Const statements needed for Use and UseLSX statements

(Declarations) statements

The (Declarations) script for an object is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts for the current object
- Const statements for those constants that you want to be available to all scripts for the current object and that are not needed for Use or UseLSX statements in (Options)

By default the (Declarations) script is initially empty.

Note If you enter Type, Class, or Declare Lib statements in an object script, the IDE moves them to (Declarations) for the object automatically.

Note Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object. To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

Initialize sub

Use the Initialize sub to set up variables declared in the object's (Declarations) script. The Initialize sub for an object executes before any of its event procedures. By default, the Initialize script is empty.

Note Scripts for controls created in the Lotus Dialog Editor do not have Initialize subs.

Terminate sub

Use the Terminate sub to clean up variables that you have declared in the object's (Declarations) script. By default the Terminate script is empty.

Note Scripts for controls created in the Lotus Dialog Editor do not have Terminate subs.

Working with event procedures

If you are writing a script for an object, the Script drop-down box displays default event procedures for the selected object. In the IDE you cannot create new event procedures for an existing product object because valid events for that object are defined by the product.

To enter statements in an event procedure, do the following:

1. Click the Script drop-down box to display the list of scripts for the selected object.
2. Click the name of the event procedure.

3. Type or paste script statements between the Sub and End Sub statements.
4. Choose Script - Check Scripts to check the syntax of the statements you entered.

Creating other procedures for objects

You can create other named subs, functions, and properties for objects in addition to the predefined scripts or event procedures. Because these procedures are not in (Globals), they can be called only from other scripts for the object.

There are three ways to create object scripts.

- Use Create - New Sub and Create - New Function to create new subs and functions for an object. The IDE automatically adds the name of the new procedure to the Script drop-down box for that object.
- Enter a Sub, Function, or Property statement anywhere in a script for the current object. The IDE automatically adds the name of the new procedure to the Script drop-down box for that object.
- Use File - Import Script when you are working with object scripts to import scripts for that object. The IDE automatically adds the name of the new scripts to the Script drop-down box for that object.

Note To distinguish scripts that you created from the predefined ones and event procedures, the IDE displays them in bold italics in the Script drop-down box.

Renaming objects

You cannot rename product objects such as ObjButton1 in the IDE. To rename the object, open the InfoBox for that object in your product and specify a new name for it.

If you referenced the old object name in your scripts, you must update all the references to that object name in your scripts.

Tip Choose Edit - Find and Replace to update references globally.

Renaming procedures

You can rename procedures in your document; you cannot rename (Options) or (Declarations). To rename a procedure, do the following:

1. Navigate to the procedure.
2. Select the name of the procedure in the first line of the script.

   Example: `Sub GetNotesMail (TemplateName as String)`
3. Change the name of the procedure.

   Example: `Sub GetNotesNetMail (TemplateName as String)`

   Note You cannot have duplicate names in (Globals) or within the scripts for a particular object. If you enter a duplicate name under these circumstances, the Script Editor displays a message notifying you of the error.
4. Move the insertion point off the line. The Script Editor lists the script with its new name in the Script drop-down box.

Note If you rename a predefined script such as Initialize, Terminate, or an event procedure, the IDE creates two scripts. One has the new name and any statements that were contained in the original script. The other has the original name of the predefined script and is empty. Both scripts are listed in the Script list. You cannot rename (Options) or (Declarations).

Tip Choose Edit - Find and Replace to replace references to the old name in your existing statements.

Overview: Working with Initialize and Terminate Subs in the Script Editor

The Initialize and Terminate subs are predefined for scripts in (Globals) and for all object scripts. Use these subs to set up and clean up variables you define in the (Declarations) script for (Globals) or for the object.

Initialize sub

Use the Initialize sub to initialize variables that you have declared in (Declarations). The Initialize sub in (Globals) executes before any of these variables are accessed and before any other scripts in (Globals) are executed. The Initialize sub for an object executes before any of its event procedures. By default, the Initialize script is empty.

Note Controls for the Lotus Dialog Editor do not have Initialize subs.

Terminate sub

Use the Terminate sub to clean up variables that you have declared in (Declarations). The Terminate sub in (Globals) or for an object is executed when you close your document or when you modify a script and execute it again. For example, you might use the Open statement to open a file containing data in Initialize and use the Close statement to close it in Terminate. By default the Terminate script is empty.

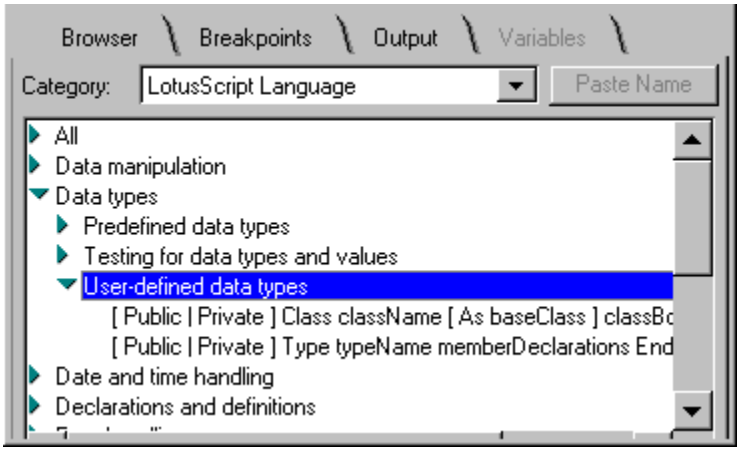
Note Controls for the Lotus Dialog Editor do not have Terminate subs.

Restrictions on using Initialize and Terminate

- There can be only one Initialize sub and only one Terminate sub per object, including (Globals).
- You cannot set debugging breakpoints in a Terminate sub.
- Initialize and Terminate must be subs.
- Initialize and Terminate cannot take arguments.

Overview: Working with Outline Lists in the IDE

Outline lists contain items that can be expanded or collapsed to display additional levels of items.



Expanding or collapsing one item in an outline list

To expand one item, do one of the following:

- Click the right arrow ( ) or down arrow ( ) to expand or collapse an item.
- Select the item and press ENTER.
- Select the item and press + (to expand) or - (to collapse).

Expanding or collapsing all items in an outline list

To expand or collapse all items, do one of the following:

- Press SHIFT++ to expand all items.

  Note In the Variables panel, SHIFT++ expands all items at the current level.
- Press SHIFT+- to collapse all items.

Can't Run This Procedure

You have an error in your script. Check your script again to see where problem is.

You may also get this error when you try to run a function or a sub containing parameters with the Run - Current Sub menu command. You can only run sub without parameters with the Run - Current Sub menu command.

Could not open file for export

There was an error when the IDE tried to create the file that was to receive the exported script.

Check the following:

- The file path is legal.
- The target directory exists.
- The file name is legal.
- The file is not write-protected.

Could not write to export file

The IDE couldn't write to the file for export. Check that the file is not write-protected and that enough disk space is available.

Could not open file for import

The IDE couldn't open the file that you wanted to import.

Check the following:

- The file exists.
- The file path is legal.
- The target directory exists.
- The file name is legal.

Could not open stream for printing

The attempt to print has failed.

Check the network connections to the printer.

Replace failed
The IDE could not replace the text.

Text not found

The string you are looking for was not found.

Check your spelling.

IDE Messages

Click a message to display Help.

.. not valid outside of class scope

A - E

Cannot forward declare user-defined class or data type

Can't run this procedure

CASE ELSE must be the last CASE in a SELECT statement

Compiler stack overflow at: <token name>

Compiler statement stack overflow at: <token name>

Could not open file for export

Could not open file for import

Could not open stream for printing

Could not write to export file

DIM required on declarations in this scope

Duplicate procedure name: <procedure name>

F - J

Illegal character after %INCLUDE directive

Illegal character after continuation character

Illegal directive

Illegal duplicate END statement

Illegal executable code in Declarations

Illegal executable code in Options

Illegal executable code outside procedure

Illegal on declarations in this scope: <keyword>

Illegal range specifier

Illegal statement

Illegal sub initialization

Illegal type suffix on keyword: <keyword>

Illegal use of escape character

Illegal use of escape character in identifier

Illegal use of parentheses

Invalid type for procedure

K - O

ME not valid outside of class scope

Name too long

Named product class instance not valid here

Out of memory

P - T

Procedure definitions illegal in this scope

Procedures may not be forward declared

PUBLIC not allowed in this module

Replace failed

SET required on class instance assignment

Statement illegal in CLASS block: <keyword>

Statement illegal in TYPE block: <keyword>
Statement is illegal in this scope
Syntax checking buffer overflow
Text not found

U - Z

Unexpected: <token>; Expected: <token>
Unmatched block terminator
Unterminated block statement
Unterminated square bracket reference
Unterminated string constant
Unterminated <keyword> block

Illegal duplicate END statement

You added an End Sub, End Function, or End Property statement to a procedure that already contains the statement. Delete the duplicate statement.

Illegal directive

Any of the following could have caused this error:

- You used an unrecognized directive. For example:

```
%EndRem    ' Illegal
%End Rem   ' Legal
```

- You nested a %Rem...%End Rem block inside another %Rem...%End Rem block.
- You used an %End Rem without a preceding %Rem.
- You entered a %If, %Else, %ElseIf, or %EndIf directive in a script you created in the IDE.

  If you want to use %If directives, you must enter them in a file that you call with the %Include directive.

CASE ELSE must be the last CASE in a SELECT statement

You used a CASE clause after CASE ELSE in a Select Case statement.

LotusScript is expecting the following sequence:

```
Select Case selectExpr
   Case conditionList
      statements
   Case conditionList
      statements
   Case Else
      statements
End Select
```

No other Case clause may follow a CASE ELSE clause. Make CASE ELSE the last clause in the SELECT Case statement.

DIM required on declarations in this scope

You declared a variable at module level without the Dim, Public, or Private keyword, or you declared a variable inside a procedure without the Dim or Static keyword. One of these is required.

Add the appropriate keyword to the declaration.

Duplicate procedure name: <procedure name>

You assigned the same name to more than one Sub, Function, Type, Property Set, or Property Get statement assigned to an object or in (Globals).

Rename the procedure so it has a unique name.

Statement illegal in CLASS block: <keyword>

You used an illegal statement in a Class...End Class block.

The only legal statements in a Class. ..End Class block are:

- Declarations of variables without the keyword Dim or Static

  A variable may be declared Public or Private, or with no leading keyword.
- Definitions without the keyword Static
- Definitions of the constructor and destructor subs (Sub New and Sub Delete) for the class
- The Rem statement
- The directives %Rem...%End Rem and %Include

By extension, when you use the %Include directive in a Class...End Class block, the file to which it refers must not contain any statements that are illegal inside a Class...End Class block.

Remove the illegal statement from the Class...End Class block.

Illegal executable code outside procedure

You entered an executable statement outside a sub, function, or other procedure. Delete the statement or move it inside a procedure definition.

If you want the statement to be executed when scripts for an object are loaded, move the statement into the Initialize sub. If you want the statement to be executed when scripts for an object are unloaded, move the statement into the Terminate sub.

Illegal character after continuation character

The line-continuation character underscore (_) is followed on the same line by a character that is not the comment character ('). The line-continuation character must be the last character on a line, except for an optional comment, beginning with the comment character.

Remove everything following the line-continuation character on the line, or insert a comment character after it to comment out the rest of the line.

Illegal on declarations in this scope: <keyword>

The following conditions could have caused this error:

- You used the keyword Dim, Public, Private, or Static when defining a member variable in a Type statement. For example:

```
Type MyType
  Public X As Integer      'Illegal: Public keyword not allowed here.
End Type
```

  Remove the Dim, Public, Private, or Static keyword.

- You used the Dim keyword when defining a member variable in a Class statement. For example:

```
Class MyClass
  Dim X As Integer         ' Illegal: Dim keyword not allowed here.
End Class
```

Remove the Dim keyword.

Illegal use of escape character

You included an escape character at the end of a line. This is not allowed. For example:

```
aString$ = "This is a tilde: "
anotherString$ = aString$~
' This is illegal
```

Remove the escape character.

Illegal use of escape character in identifier

You included an escape character in one of the following contexts in which that character is not allowed:

- In a declared name (a variable, constant, procedure, class, or user-defined data type)
- In the name of an implicitly declared variable
- In a label definition or reference
- In the name of the reference variable in a ForAll statement

For example:

```
Dim fo~x As Integer   ' Illegal
```

Remove the escape character.

Illegal executable code in Declarations

You included an executable statement in (Declarations), which is not allowed.

Only the following are allowed in object (Declarations): comments, the Private keyword, the Public keyword, the Declare (external C calls), and the Const, Dim, Type, and class statements.

Remove the invalid statement or statements from (Declarations).

Illegal executable code in Options

In (Options), you included a statement that is not allowed in the section.

Only the following are allowed in (Options): Option, Def*type*, Use, and UseLSX statements and Const statements associated with Use and UseLSX.

Remove the invalid statements from (Options).

Illegal character after %INCLUDE directive

The %Include directive is followed on the same line by something other than the name of the file to include. The name of the file to include must be the only thing following %Include on a line, except for an optional comment, beginning with the comment character.

Remove everything following %Include and the name of the file, or insert a comment character after it to comment out the rest of the line.

Named product class instance not valid here

In one of the following statements, you used the name of a product object in a context in which it is not allowed:

- An assignment statement (Let or =) in either of the following forms:

  Let *name* = ...

  name = ...

- A Set statement in either of the following forms:

  Set *name* = NEW...

  Set *name* = ...

  Set *name* = Bind...

- A Delete statement
- An Erase statement
- A ForAll statement
- A Get or Put statement
- An Input # or Line Input # statement
- An LSet or RSet statement
- A Mid or MidB statement
- A ReDim statement

Replace the name with an appropriate name, or remove the invalid statement.

PUBLIC not allowed in this module

You defined a public variable, sub, function, property, or constant for an object other than (Globals). You can do one of the following to correct this problem:

- Move the definition to (Globals).
- Leave the definition where it is, but do one of the following:
  - If you are declaring a variable, replace the Public keyword with Private or Dim.
  - Otherwise, delete the Public keyword or change it to Private.

Illegal range specifier

You used a Def*type* range in one of the following illegal ways:

- No range was specified.
- The beginning of the range was not a single character between A and Z (ASCII uppercase or lowercase), inclusive.
- The end of the range was not a single character between A and Z (ASCII uppercase or lowercase), inclusive.

Correct the error.

Illegal statement

You used a colon (:), followed by an underscore (_), to separate and then recombine statements in a line of script. This is not allowed in the IDE.

Use a colon (:) between multiple statements in a line; use an underscore (_) at the end of a line to continue the line.

Statement is illegal in this scope

You tried to enter a statement in a scope where it is not allowed. Check LotusScript Help for information about the statement you tried to enter and whether it can be used in (Globals), object scripts, a user-defined data type, or class scope.

Procedure definitions illegal in this scope

You tried to define a sub, function, or property within a class method or property. This is not allowed.

Move the definition outside the scope of the class method or property.

Statement illegal in TYPE block: <keyword>

You used an illegal statement in a Type...End Type block. The only legal statements in a Type...End Type block are declarations of variables without the leading keyword Dim, Public, Private, or Static; the Rem statement; and the directives %Rem...%End Rem and %Include. All other statements are illegal.

By extension, when you use the %Include directive in a Type...End Type block, the file to which it refers must not contain any statements that are illegal inside a Type...End Type block.

Remove the statement from the Type...End Type block.

Illegal sub initialization

You tried to create a procedure or function named Initialize or Terminate.

The routines Initialize and Terminate must be defined as subs.

Redefine your procedure or function as a sub.

.. not valid outside of class scope

You used "dotdot" syntax outside of a procedure within a class. The "dotdot" syntax is only valid inside procedures within a class. You use "dotdot" notation when referring to a procedure in a base class when the derived class has a procedure of the same name, as in the following example:

```
 CLASS BaseClass
   SUB MySub
    PRINT "In BaseClass's  MySub"
  END SUB
END CLASS

CLASS DerivedClass  AS BaseClass
  SUB MySub
   PRINT "In DerivedClass's MySub "
  END SUB

  SUB MyOtherSub
     CALL MySub                       'Print "In DerivedClass's MySub "
    CALL BaseClass..MySub           'Print "In BaseClass's MySub "
   END SUB
END CLASS
```

Remove the "dotdot" syntax and use an object reference variable in its place.

ME not valid outside of class scope

You used the keyword ME outside of a procedure within a class. Use the keyword ME only inside procedures within a class. You use Me within the definition of a class when referring to members of that class.

Remove the keyword ME. If you are referring to a class member , use an object reference variable instead of Me.

Procedures may not be forward declared

You tried to use the Declare statement to forward declare a sub, function, or property. This is not necessary because the IDE generates forward declares for you.

Cannot forward declare user-defined class or data type

You tried to use the Declare statement to declare a type or class before defining it. This is not allowed in the IDE.

Invalid type for procedure

You tried to change a product-defined procedure to another type of procedure. This is not allowed.

You cannot, for example, change a predefined sub to a function by changing the Sub statement to a Function statement. Similarly, you cannot change a predefined function or sub to another procedure type.

Name too long

The specified name is too long (it is truncated in the error message). The maximum length of a LotusScript name is 40 characters.

Shorten the name to 40 or fewer characters.

SET required on class instance assignment

You attempted to assign an object reference to a variable but omitted the SET keyword. (An object reference can be a reference to an instance of a user-defined class, a product object, an OLE automation object, or the constant NOTHING). The SET keyword is required in object reference assignments. For example:

```
Class MyClass
' ...
End Class
Dim MyObj As New MyClass
Dim varV As Variant
varV = MyObj        ' Illegal syntax
```

Insert the SET keyword in the assignment statement:

```
Class MyClass
' ...
End Class
Dim MyObj As New MyClass
Dim varV As Variant
Set varV = MyObj    ' Legal syntax
```

Out of memory

You must free enough memory to perform the operation that caused this error message. To free memory in your computer, do one of the following:

- If you have other programs in memory, end one or more of those programs.
- Reduce the amount or size of PUBLIC data.
- Activate extended memory.

Compiler stack overflow at: <token name>

The statement being compiled is too complex. It may contain a complex expression, or deeply nested block statements, such as a Do or For statement.

Reduce the nesting level, or break up the statement into multiple, less complex statements.

Unexpected: <token>; Expected: <token>

The compiler encountered an unexpected language element.

If the unexpected language element is a number appearing inside square brackets, it represents the ASCII code of an unprintable character. For example, if you enter the Backspace character in a statement where a name is expected, the following error message appears when you compile the script:

```
Unexpected: [8]; Expected: Identifier
```

For more information, refer to the list of expected language elements following the unexpected language element in the error message.

Illegal use of parentheses

You called a sub or function and enclosed its argument list in parentheses. You can only do this under the following circumstances:

- The sub or function is the target of a Call statement. For example:

```
Call MySub()                        ' Legal
Call MyOtherSub("ABC", 4)           ' Legal
Call MyFunction()                   ' Legal
Call MyOtherFunction(123, "XXX")    ' Legal
```

- The sub or function has a single parameter that the caller is passing by value. For example:

```
MySub("ABC")                        ' Legal
MyFunction(anInt%)                  ' Legal
```

- The target is a function that is included in a statement. For example:

```
X% = MyFunction(123, "XXX")         ' Legal
```

The following are illegal:

```
MySub()                             ' Illegal
MyFunction()                        ' Illegal
MyOtherSub("ABC", 4)                ' Illegal
MyOtherFunction(123, "XXX")         ' Illegal
```

Remove the parentheses from around the argument list or call the sub or function with the Call statement.

Compiler statement stack overflow at: <token>

The statement being compiled is too complex. It may contain deeply nested block statements, or single-line If statements.

Reduce the nesting level, or break up the statement into multiple, less complex statements.

Illegal type suffix on keyword: <keyword>

You included an illegal data type suffix character in the name of a LotusScript built-in function. Certain LotusScript built-in functions can end in the $ type suffix character; no other data type suffix character is valid on these functions. The names of other functions cannot end in a data type suffix character. For example:

```
Print Date()            ' Legal
Print Date$()           ' Legal
Print Date#             ' Illegal
Print CDat(Date)        ' Legal
Print CDat$(Date)       ' Illegal
```

Remove the suffix character.

Syntax checking buffer overflow

A statement you entered exceeds the limit of 32K bytes. Split the statement into multiple units.

In general:

- Each line of script (including each line in a multiline statement) can contain approximately 1000 characters.
- Each statement can contain approximately 2400 language elements.

Unterminated block statement

You omitted the ending statement for a block statement that begins with one of the following keywords:

Class

Do

For

ForAll

Function

If...Then...Else...EndIf

Property Get

Property Set

Select

Sub

Type

While

With

Enter the appropriate ending statement.

Unmatched block terminator

You omitted the statement that begins with one of the following keywords and marks the beginning of a statement block:

Class

Do

For

ForAll

Function

If...Then...Else...EndIf

Property Get

Property Set

Select

Sub

Type

While

With

Enter the appropriate beginning statement.

Unterminated <keyword> block

You omitted the keyword that marks the end of one of the following block statements:

Class

Do

For

ForAll

Function

If...Then...Else...EndIf

Property Get

Property Set

Select Case

Sub

Type

While

Terminate the block with the appropriate statement.

Unterminated string constant

You omitted the double quotation mark that signals the end of a quoted literal on a single line. Double quotation marks must be paired on the same line. For example:

```
Print "Hi,       ' Illegal because end quotation mark is missing
Martin."

Print "Hi, " _   ' Legal because string is properly quoted
"Martin."        ' Legal because string is properly quoted and
                 ' preceded by line-continuation character
' Output: Hi, Martin.
```

Terminate the string with double quotation marks on the same line where it starts.

Unterminated square bracket reference

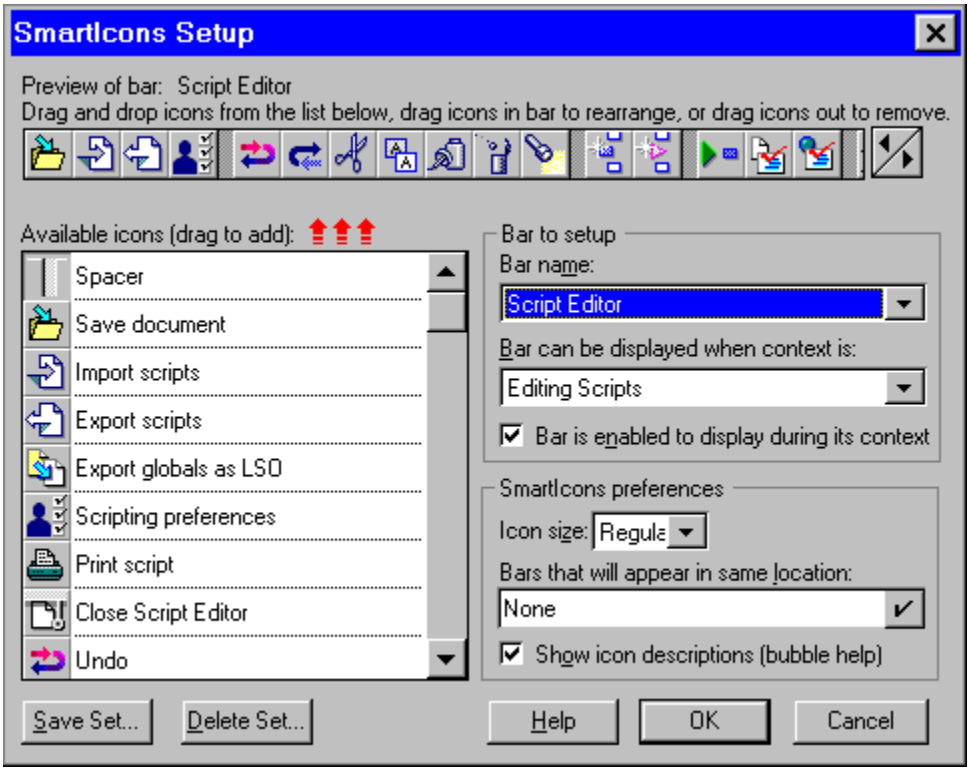
A square bracket reference was not terminated by a close square bracket (]) on the same line. Square brackets are used in some cases when referring to the names of product objects.

Terminate the square bracket reference with a close square bracket on the same line. Make sure that the product you are using supports square bracket notation for references.

Overview: Using SmartIcons in the IDE

You can review all sets of SmartIcons in this dialog box by selecting an icon set from the SmartIcons Bar name list. When you do this, the specific icon set appears at the top of the box.

Using a SmartIcons set in this dialog box, you can add, move, group, and remove the icons in a set.



Selecting and displaying a specific icon set

1. Choose File - SmartIcons Setup.
2. Select the set under "Bar name."
3. Select an option under "Bar can be displayed when context is."
4. If you want to display the set at specific times (depending on your selections in steps 3 and 4), select "Bar is enabled to display during its context."

   Note Typically, this setting should always be selected.
5. Click OK.

Sizing icons in a SmartIcons set

1. Choose File - SmartIcons Setup.
2. In the Icon size list box, select Regular or Large.
3. Click OK.

Spacing between SmartIcons in a set

1. Choose File - SmartIcons Setup.
2. Under "Bar name," select the SmartIcon set you want to work with.
3. Drag a spacer to separate the icons within the displayed set.
4. Click OK.

Note When you drag and drop icons, the IDE moves the other icons in the set forward or backward one position to accommodate the change. The SmartIcons then appear in the new order in the dialog box. You can use the left and right arrows to see icons that scroll out of sight. Save the .SMI file for the change to take effect across all sessions.

Adding an icon to a set of SmartIcons

1. Choose File - SmartIcons Setup.
2. To review the entire list of SmartIcons, use the up and down arrows in the Available icons (drag to add) list box.

   The IDE displays all SmartIcons in this list box.
3. Drag an icon from the list to the bar at the top of the dialog box.
4. To save the bar with a different name, click Save Set. To overwrite the existing set, click OK.

Saving and deleting SmartIcons sets

If you click the Save Set, Delete Set, or OK button, this what happens.

Save Set	Takes you to the Save As SmartIcons File dialog box where you can give the new icon set a name and save it in its own file. The new set name becomes part of the SmartIcons list. Click Browse to rename the .SMI file. Click OK to return to the SmartIcons Setup dialog box. Click OK again.
Delete Set	Takes you to the Delete Set dialog box where you can delete sets of SmartIcons you do not use. Click OK to return to the SmartIcons Setup dialog box. Click OK again.
OK	Displays the new SmartIcons set. The IDE displays the new arrangement every time you select this set.

Removing an icon from the set of SmartIcons

1. Choose File - SmartIcons Setup.
2. Select the set you want to modify under "Bar name."
3. Drag the icon you want to remove away from the displayed set.
4. (Optional) To remove the icon for all sessions, click Save Set.
5. Click OK.

The IDE displays the new arrangement every time you select this set.

Creating a new SmartIcons set

1. Choose File - SmartIcons Setup.
2. You can use the default SmartIcons set as a base for the new set or select another set under "Bar name".
3. Use drag and drop to add, move, group, and remove icons until the set is the way you want.
4. Click Save Set and type a new name for the set.
5. Click Browse and type a new filename for the set.
6. Click Save to return to the Save As SmartIcons File dialog box.
7. Click OK to return to the SmartIcons Setup dialog box.
8. To display this bar when you are working in a specific part of the IDE, select an option under "Bar can be displayed when context is."
9. To display this bar when the chosen context is active, select "Bar is enabled to display during its context. "
10. Click OK.

Tip You can add icons by dragging icons from the Available icons (drag to add) list box up into the new set. You can also move and rearrange icons by dragging them (including spacers) within the new set. You can remove icons from the set by dragging them away from the displayed set.

Deleting a SmartIcons set

1. Choose File - SmartIcons Setup.
2. Click Delete SmartIcons Set.
3. Select the set you want to delete.
4. Click OK.
5. The IDE asks you to confirm your selection.

   Clicking Yes deletes the .SMI file and returns you to the SmartIcons Setup dialog box.
6. Click OK.

Setting location preferences for context SmartIcons

One set of SmartIcons exists for each context supported by the IDE (Script Editor, Script Debugger, and so on).

1. Choose File - SmartIcons Setup.
2. Select the desired SmartIcons sets under "Bars that will appear in the same location."
3. Click OK.

When you place a set of context SmartIcons, its position can be used by all context SmartIcons sets.

Tip If you have more than one SmartIcons set for a specific context, only check one set. Otherwise, both sets will be in the same location, with one on top of the other. You will only be able to use one set.