How do I add comments?

You can add comments to a script by using a single quotation mark, the REM directive, or the %REM directive.

Using a single quotation mark

Use ' (a single quotation mark) to start an entire line of comment text or to add comments to the end of a line of code.

The following are two examples of using a single quotation mark.

```
'Comments can preceed a statement, but not on the same line.
Print "Two"
Print "One" 'Comments can also follow a statement.
```

Using the REM directive

Use REM to start an entire line of comment text or to add comments to the end of a line of code. All text after REM is part of the comment.

The following are two examples of using REM.

```
REM Comments can both preceed and follow a statement.
Print "Three" REM But the statement always comes first.
```

Using the %REM directive

Use the % REM directive to add comments in a block style.

Put the keyword %REM at the beginning of the block and the keywords %END REM at the end of the block. For example:

```
%REM
   ============================================================
   This sub was written on 01/23/98.
   Edited on 03/01/99 by JAM -- redefined ButtonColor909
   Edited on 06/08/99 by KHD -- added NotButtonColorBlue
   ============================================================
%END REM
```

The %REM and %END REM directives can designate the beginning and end of one or more lines of comment text. %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 line after the initial directive must be followed by one or more spaces, tabs, and newline characters before any more comment text appears. %REM blocks cannot be nested.

{button ,AL(`;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_INDENTING_SCRIPT_STATEMENTS_STEPS;H_IDE_WORKING_WITH_DIRECTIVES_IN_THE_SCRIPT_EDITOR_OVER',0)} See related topics

Checking scripts 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 IDE does not report errors for the following when it checks a single line:

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

To check an individual sub

Choose Script - Run Current Sub in the IDE to test all statements in the current sub and, if it contains no errors, run it.

To check all scripts for an object

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

To check all scripts for a document

Choose Script - Check All Scripts *for document* to check all statements in all scripts for the current document.

Tip Use Script - Check Scripts or Script - Check All Scripts to update a script after you have modified a file referenced with the %Include directive. If you do not update the script, the IDE continues to use the previous version of the included file. If you have changed an .LSO file referenced in a Use statement, you can use Script - Check All Scripts to update the script that contains the reference.

{button ,AL(`;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_MONITORING_CALLS_IN_YOUR_SCRIPTS_OVER;H_IDE_RUNNING_A_SCRIPT_IN_THE_IDE_STEPS;H_IDE_TYPE_STATEMENTS_OVER;H_IDE_WORKING_WITH_DIRECTIVES_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_WORKING_WITH_GLOBALS_IN_THE_IDE_STEPS',0)} See related topics

Overview: Creating (Declarations) in the Script Editor

(Declarations) is a script, predefined by the IDE, in which you declare the types of variables. You can create a (Declarations) script for the (Globals) object or for individual objects. In the IDE, any subs, functions, or properties that you create are declared implicitly. You do not need to use Declare statements to declare subs, functions, and properties, except within a class.

(Declarations) statements for the (Globals) object

By default the (Declarations) script for (Globals) is initially empty.

The (Declarations) script for (Globals) can contain:

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

If you enter Type, Class, or Declare Lib statements in any other script for the (Globals) object, 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

By default the (Declarations) script for an object is initially empty.

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)

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.

Details: Creating functions and subs

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 open a new procedure window, but instead displays an error message to let you know that the definition is illegal in that scope.

If you create a procedure 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.

If you create a procedure 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.

{button ,AL(`H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS',1)} Go to procedure

{button ,AL(`H_IDE_CREATING_FUNCTIONS_AND_SUBS_EX',1)} See example

Example: Creating functions and subs

This is a function definition.

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

Creating functions and subs

A function is a named script that performs a specific task and returns 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.

A sub is a named script that performs a specific task without returning 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.

A procedure is a named script. It can be either a sub or a function.

Note It is illegal to add a procedure to a sub or function that defines a class method or property.

1. Select the object, including (Globals), for which you want to create the function or sub.
2. Choose Create - Function or Create - Sub.
3. Enter a new name for the function or sub, or accept the default name that the Script Editor provides.
4. (Optional) Do one of the following:
   - Select "Visible to scripts in this object only "to create the script for the current object.
   - Select "Visible to all scripts in this document" to create the script in (Globals).
5. Click OK to create the new script.
6. Enter code statements in the new script.

{button ,AL(`H_IDE_CREATING_FUNCTIONS_AND_SUBS_DETAILS',1)} See details

{button ,AL(`H_IDE_CREATING_FUNCTIONS_AND_SUBS_EX',1)} See example

{button ,AL(`;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_IMPORTING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_RECORDING_A_SCRIPT_AT_THE_CURSOR_STEPS;H_IDE_SAVING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

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.

1. Place the insertion point where you want to put the new script.
2. Enter the first line of a Sub, Function, or Property.

   Make certain that the name of the new procedure is unique within the current document.
3. Move the insertion point off the line.

   The IDE moves the new procedure into a new, separate window, adds a blank line, and adds the appropriate End statement. The name of the new procedure appears in the Script drop-down box.

Note It is illegal to enter a Sub, Function, or Property statement within a Type definition or within a class method or property.

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_SAVING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_SELECTING_AN_OBJECT_TO_WORK_WITH_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_TYPE_STATEMENTS_OVER',0)}
See related topics

Creating scripts in the Script Editor

You can create subs, functions, and properties for (Globals) or for any object listed in the Object 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 specific objects can be called only from scripts defined for that object.

You can add scripts to (Globals) or to object scripts.

- Enter a Sub, Function, or Property statement in a script anywhere except within a class.

  See Creating scripts by entering script statements.
- Use Create - Sub and Create - Function to create new subs and functions.
  See Creating functions and subs.
- Use File - Import Script to add scripts contained in a text file to (Globals) or to object scripts.
  See Importing scripts in the Script Editor.
- In the IDE choose Script - Record at cursor to record mouse movements and keyboard strokes into the current sub.
  See Recording a script at the cursor.

  Note You can only record scripts in Lotus 1-2-3, Approach, and Word Pro. To record complete subs within the product, choose Edit from the product menu and the Script sub menu. In some products, such as Word Pro, you can choose more than one record option.

In the first three cases, the IDE automatically adds the name of the new script to the Script drop-down box. To distinguish scripts that you created from the predefined ones, the IDE displays the name of scripts that you create in bold italics in the drop-down box.

Displaying global variables for the current script

If you have global variables in your document, the Variables panel displays them in a list when the Debugger is active.

To view the global variables, select and expand the Globals entry in the Variables panel.

The following variable names, values, and 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

---

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
```

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER',0)} See related topics

Importing scripts in the Script Editor

You can import text that you intend to use as script code into the Script Editor. These imported 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.

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 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 to replace the existing script with the imported script.
6. Choose one of the options provided in the dialog box.

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

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_RECORDING_A_SCRIPT_AT_THE_CURSOR_STEPS',0)} See related topics

Overview: Initialize and Terminate subs in the Script Editor

The Initialize and Terminate subs are predefined scripts for (Globals) and for all object scripts.

- You can have only one Initialize sub and one Terminate sub per object, including (Globals).
- You cannot set breakpoints in a Terminate sub.
- Initialize and Terminate must be subs; you cannot rename them as functions.
- Initialize and Terminate cannot take arguments.

Initialize sub

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

Note Dialog Editor Controls do not have Initialize subs.

Terminate sub

Use the Terminate sub to clean up variables that you declare in (Declarations). The Terminate sub in (Globals) or for an object runs 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 the Initialize sub, then use the Close statement to close the file in the Terminate sub. By default, the Terminate sub is empty.

Note Dialog Editor Controls do not have Terminate subs.

{button ,AL(`H_IDE_CHANGING_VARIABLES_STEPS;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_DISPLAYING_GLOBAL_VARIABLES_FOR_THE_CURRENT_SCRIPT_STEPS;H_IDE_RENAMING_FUNCTIONS_AND_SUBS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Printing a script

The IDE allows you to print the script in the current window, all scripts for the current object, or all scripts for a product document.

1. Choose File - Print Script.
2. In the Print dialog box, select what you want to print.
3. Click Print or OK.

{button ,AL(`;H_IDE_CHANGING_FONTS_AND_TEXT_COLORS_IN_THE_IDE_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_INDENTING_SCRIPT_STATEMENTS_STEPS;H_IDE_SAVING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Overview: Public and Private statements

The IDE creates an Option Public statement in the (Options) script in (Globals) automatically. The Option Public statement makes all variable, constant, procedure, class, and type declarations that you create in (Globals) public by default. These declarations 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 is 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):

Declaration	Explanation
`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 within the class declaration (Private).

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_ENTERING_CLASS_STATEMENTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_THE_OPTIONS_SCRIPT_OVER;H_IDE_WORKING_WITH_GLOBALS_IN_THE_IDE_STEPS',0)} See related topics

Recording a script at the cursor

When you are writing a script and you want to include the LotusScript equivalent of a certain task, but do not know what that equivalent is, you may want to record a script and store it in the Script Editor at the insertion point.

Note This feature is available in Lotus 1-2-3, Approach, and Word Pro.

1. In the Script Editor, place the insertion point in the desired sub or function at the point where you want to insert the recorded task.
2. Choose Script - Record at Cursor.
3. Minimize or move the Script Editor.
4. Perform any task(s) you want to record.
5. In 1-2-3 or Word Pro, choose Edit - Scripts & Macros - Stop Recording.
   In Approach, choose Edit - Stop Recording.

   The Script Editor displays the recorded task.

{button ,AL(`;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_IMPORTING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Renaming functions and subs in the Script Editor

You can rename subs, functions, and properties in your document. You cannot rename the (Options) and the (Declarations) scripts.

Note If you rename a predefined script (such as Initialize or Terminate) or an event procedure, the IDE creates two scripts. One script has the new name and any statements that were contained in the original script. The other script has the original name of the predefined script and is empty. Both scripts are listed in the Script drop-down list.

1. Display the procedure in the Script Editor.
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, the Script Editor displays an error message.
4. Move the insertion point off the line.

   The name of the new procedure appears in the Script drop-down box.
5. Choose Edit - Find & Replace to replace references to the old name in your existing statements.

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS;H_IDE_INITIALIZE_AND_TERMINATE_SUBS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_THE_OPTIONS_SCRIPT_OVER;H_IDE_WORKING_WITH_GLOBALS_IN_THE_IDE_STEPS',0) } See related topics

Running a script in the IDE

1. From the product's Edit menu, open the Script Editor.
2. Select the script to run.
3. Choose Script - Run Current Sub.

   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). Syntax errors are reported in the Errors drop-down box. If the script has no errors, the IDE executes it.

{button ,AL(`;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_OPENING_AND_USING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

Saving scripts in the Script Editor

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

- Choose File - Save Scripts.

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

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

{button ,AL(`H_IDE_PRINTING_A_SCRIPT_STEPS;H_IDE_RUNNING_A_SCRIPT_IN_THE_IDE_STEPS;H_IDE_SELECTING_A_SCRIPT_IN_THE_IDE_STEPS;H_IDE_SAVING_A_SCRIPT_AS_AN_LSS_FILE_STEPS;H_IDE_SAVING_A_SCRIPT_AS_AN_LSO_FILE_STEPS',0)} See related topics

Selecting an object to work with in the Script Editor

Selecting an object while working in the IDE

1. Click the Object drop-down box to display a list of product objects.
2. Click the name of the object to select it and to close the list.

Note If the object raises an event but it does not have a name, the object may not appear in the Object drop-down list. To name an object, go to the Misc. tab of the InfoBox for that object and enter a name in the "Name" box.

Selecting an object in your product

1. Go to the object that you want to script in your product.
2. Click the object to select it.
3. From the product's Edit menu, choose to display the Script Editor

   The Script Editor displays the scripts for the selected object.

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

{button ,AL(`;H_IDE_DISPLAYING_GLOBAL_VARIABLES_FOR_THE_CURRENT_SCRIPT_STEPS;H_IDE_SELECTING_A_SCRIPT_IN_THE_IDE_STEPS;H_IDE_WORKING_WITH_EVENTS_STEPS',0)} See related topics

Overview: The (Options) script

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

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

Option statements

Option statements that you enter in (Options) apply only to scripts for the current object, even if you are working in (Globals). Enter the appropriate statement in the (Options) script for every object for which you are writing scripts so that it is applied consistently throughout a document

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).
- (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).
- 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.

Def*type* statements

Def*type* statements set the default data type for variables, functions, and properties whose names begin with one of a specified group of letters. For example, if you specify *S* in a DefStr statement, then variable names such as SCurrentDB or SResultSet in your script are defined automatically with the data type of String.

Def*type* statements that you enter in (Options) apply only to scripts for the current object, even if you are working in (Globals). Enter the appropriate statement in the (Options) script for every object for which you are writing scripts so that it is applied consistently throughout a document

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)

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_ENTERING_CLASS_STATEMENTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_PUBLIC_AND_PRIVATE_STATEMENTS_OVER;H_IDE_WORKING_WITH_GLOBALS_IN_THE_IDE_STEPS',0)} See related topics

The Script Editor

In the IDE Script Editor, you can write and edit scripts, check script syntax, and set breakpoints for debugging scripts.

Choose one of the following tasks:

- Selecting an object to work with in the Script Editor
- Working with events
- Checking scripts in the Script Editor
- Creating functions and subs
- Creating scripts by entering script statements
- Setting breakpoints in the IDE
- Clearing breakpoints in the IDE

Overview: Type statements

If you enter a Type statement in (Declarations) and press ENTER, the Script Editor completes the statement by inserting a blank line and appending an End Type statement.

If you enter a Type statement in a script other than (Declarations) and physically move the insertion point off the line (for example, if you click on an object in the product), 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 Script Editor does not generate an End Type statement if you do not supply a valid beginning one.

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_PUBLIC_AND_PRIVATE_STATEMENTS_OVER;H_IDE_THE_OPTIONS_SCRIPT_OVER',0)} See related topics

Using a template in the Script Editor

In Approach and Word Pro, the Script Editor has several sets of frequently used scripts as script templates. You can insert these templates into the current script.

1. In the Script Editor, display the script to which you want to add the script template.
2. Place the insertion point in the script where you want to insert the template.
3. Choose Script - Insert Template.
4. Select a script template.
5. Click Insert or OK.

   The Script Editor places the script or scripts and comments in the current script at the insertion point. For some of the script templates, the Script Editor places variable names and remarks in the (Declarations) script for the Globals object. You can and will probably need to modify the code that was inserted from the script template because the templates contain example text and/or variable names.

{button ,AL(`;H_IDE_ADDING_COMMENTS_TO_A_SCRIPT_OVER;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS;H_IDE_WORKING_WITH_GLOBALS_IN_THE_IDE_STEPS',0)} See related topics

Overview: Working with directives in the Script Editor

%If

If you want to use an %If directive, you must enter it in a text file that you reference with the %Include directive. When entered directly in IDE scripts, %If directives generate syntax errors. The IDE also generates syntax errors when it encounters the %Else, %ElseIf, and %EndIf keywords.

Note You cannot enter the %If directive directly into a script.

%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 upload the latest changes by choosing Script - Check Scripts for *object* or Script - Check Scripts for *document.* Otherwise, the IDE continues to use the previous version of the included file.

%REM and %END REM

You can use the %REM and %END REM keywords to enter a block of comments in any script.

If you type %REM and press ENTER, the Script Editor automatically inserts a blank line followed by %END REM, which completes the directive.

If you type %REM and then move the insertion point off the line, the Script Editor does not insert a blank line and an %END REM. In this case, the Script Editor automatically adds an %END REM directive when you close the IDE, activate another window, or check script syntax.

{button ,AL(`;H_IDE_ADDING_COMMENTS_TO_A_SCRIPT_OVER;H_IDE_CHECKING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER',0)} See related topics

Working with events

An event is an action performed by the user, an application, or the system. For example, saving a document is an event, as is clicking a button.

Many objects have a set of predefined events with names that match the action of the event. For example, clicking a button is defined by the Click event of the Button object.

The IDE provides a sub for an event, and the name of the sub matches the name of the event. For example, the Click sub is where you write the event script for the Click event. When a button is clicked, the Click event of the Button object occurs, and the code inside the Click sub is executed. Thus, you can control what operations occur as a result of an event taking place.

1. Click the Object drop-down box to display the list of objects available.
2. Select an object.
3. Click the Script drop-down box to display the list of scripts for the selected object.

   Note 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.
4. Click the name of the event procedure.
5. Type or paste script statements between the Sub and End Sub statements.

{button ,AL(`;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_IMPORTING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_INITIALIZE_AND_TERMINATE_SUBS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_RECORDING_A_SCRIPT_AT_THE_CURSOR_STEPS',0)} See related topics

Working with (Globals) 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.

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 the list of scripts for (Globals).
4. Select a script from the list.
5. Modify the script as desired.

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

{button ,AL(`;H_IDE_CREATING_DECLARATIONS_IN_THE_SCRIPT_EDITOR_OVER;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_PUBLIC_AND_PRIVATE_STATEMENTS_OVER;H_IDE_SAVING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Overview: Breakpoint symbols

Current breakpoint indicators

Breakpoint
Current breakpoint
Disabled breakpoint
Current disabled breakpoint
No breakpoint

Current script statement indicators

Current script statement
Call to the procedure that contains the current script statement

Combined breakpoint and current script statement indicators

Disabled breakpoint in the current script statement
Current disabled breakpoint in the current script statement
Disabled breakpoint in the call to the procedure that contains the current script statement
Current disabled breakpoint, call to the procedure that contains the current script statement
Breakpoint in the current script statement
Current breakpoint in the current script statement
Breakpoint in the call to the procedure that contains the current script statement
Current breakpoint in the call to the procedure that contains the current script statement

---

{button ,AL(`H_IDE_DISABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_ENABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_NAVIGATING_A_SCRIPT_USING_THE_BREAKPOINTS_PANEL_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER',0)} See related topics

Details: Changing variable values

Variable data types
When you change a variable value, you can enter the following kinds of values:

- Strings enclosed in " " (double quotation marks)
- Numbers in a standard format for scripts
- Variant variables except the constants TRUE, FALSE, and NULL

Viewing variables in the Variables panel
The Variables panel displays the following information:

- Variables listed by name in the order they are declared
- The value and current data type for each variable

A variable that has members (arrays, lists, classes, and types) is presented as an expandable entry in the panel. Click on the right twistie (▸) to expand the entry to view its members, their values, and data types.

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.

Navigating to a different script
If you want to switch to a script that contains a procedure not 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.

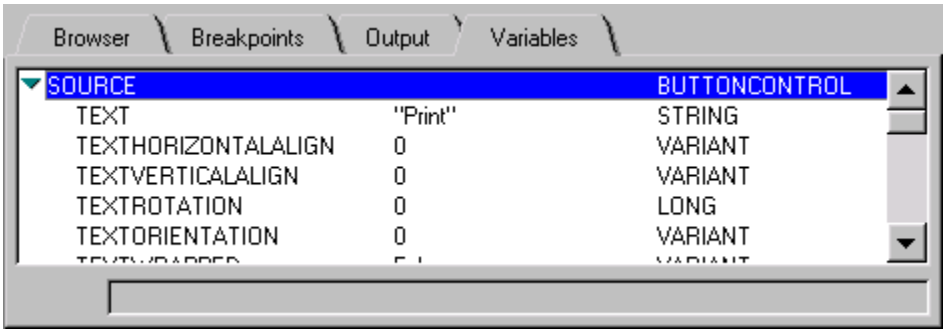
{button ,AL(`H_IDE_CHANGING_VARIABLES_STEPS',1)} Go to procedure

Changing variable values

The Variables panel lists the current value of variables used in scripts that are running. You can monitor how your scripts are using variables by changing their values while the script executes.

1. In the "Calls" 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, at the bottom of the Variables panel, is enabled. The following illustration shows where you could change the value of the Text property of the command button from "New Expense" to something else.



3. In the value box, enter a new value for the variable.
4. If a variable stores more than 1024 characters, you can view and edit only the first 1024 characters in the box.
5. Press ENTER or click the check mark next to the "value" box to change the value of the variable.

{button ,AL(`H_IDE_CHANGING_VARIABLES_DETAILS',1)} See details

{button ,AL(`;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_THE_BREAKPOINTS_PANEL_CS;H_IDE_THE_BROWSER_OVER;H_IDE_THE_OUTPUT_PANEL_OVER',0)} See related topics

Clearing breakpoints in the IDE

You can clear breakpoints for individual statements, for a group of selected statements, or for all statements in your document.

1. Open the IDE.
2. Open the desired script.
3. Do one of the following:
   - Click the breakpoint symbol in the breakpoint gutter.
   - Choose Script - Clear Breakpoint.

   You can do this 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.
   - If the Breakpoints panel is active, you can select the breakpoint in the Breakpoints panel and press DELETE.
   - To delete all breakpoints in the current document, choose Script - Clear Breakpoints for *document*.

   Note This menu item is dimmed if there are no breakpoints in your document.

{button ,AL(`H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_DISABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_ENABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_NAVIGATING_A_SCRIPT_USING_THE_BREAKPOINTS_PANEL_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER;',0)} See related topics

Closing the Script Debugger

Choose Debug - Continue Execution to continue script execution to the next breakpoint or the end of the script. When the current script ends, the IDE returns to the Script Editor.

Choose Debug - Stop Execution to stop script execution with the current statement and return the IDE to the Script Editor.

{button ,AL(`;H_IDE_OPENING_AND_USING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_THE_SCRIPT_DEBUGGER_OVER;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER',0)} See related topics

Disabling breakpoints in the IDE

When you first set a breakpoint, it is enabled by default.

1. Move the insertion point to the line containing the breakpoint.
2. To disable a breakpoint for the selected line(s), choose Script - Disable Breakpoint in the Script Editor or choose Debug - Disable Breakpoint in the Script Debugger.

   To disable all breakpoints in the current document, choose Script - Disable Breakpoints for *document* in the Script Editor or choose Debug - Disable Breakpoints for *document* in the Script Debugger.

{button ,AL(`H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_ENABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_NAVIGATING_A_SCRIPT_USING_THE_BREAKPOINTS_PANEL_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER;H_IDE_CLEARING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

Enabling breakpoints in the IDE

1. Move the insertion point to the line containing the disabled breakpoint.
2. To enable breakpoints for selected line(s), choose Script - Enable Breakpoint in the Script Editor or choose Debug - Enable Breakpoint in the Script Debugger.

   To enable breakpoints for the current document, choose Script - Enable Breakpoints for *document* in the Script Editor or choose Debug - Enable Breakpoints for *document* in the Script Debugger.

{button ,AL(`H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_DISABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_NAVIGATING_A_SCRIPT_USING_THE_BREAKPOINTS_PANEL_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER;H_IDE_CLEARING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

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*

```
(Globals)31: End statement expected
```

Correcting an error in the current document

1. Select 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

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

   The IDE navigates to the script line that contains the %Include directive for the file.

2. Edit the file using a text editor and correct the error.

{button ,AL(`;H_IDE_CHECKING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Overview: Monitoring calls in your scripts

The Calls drop-down box provides a view into calls between procedures, that is, which procedure makes what calls to other procedures as your script executes. The Calls drop-down box has an entry for each call between procedures. Each item contains the name of the current procedure, the name of the procedure that called it, other procedures that called it, objects associated with each procedure, and so on.

When you select an item, the Script Debugger displays the script that made the procedure call. 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 in another product document that are 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.

Navigating to a different script

If you want to switch to a script that contains a procedure not 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.

{button ,AL(`;H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_CHANGING_VARIABLES_STEPS;H_IDE_THE_BROWSER_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Navigating a script using the Breakpoints panel

To display the breakpoint in your script, click the item in the Breakpoints panel.

To display the breakpoint in your script and to place the insertion point on the same line as the breakpoint, double-click the item in the Breakpoints panel or select the item in the Breakpoints panel and press ENTER.

{button ,AL(`H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_DISABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_ENABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER;H_IDE_CLEARING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

Details: Opening and using the Script Debugger
When script execution stops at a statement, you can do one of the following:

- To step through a script, 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.
- To step out of the current script, choose Debug - Step Exit.

  Script execution continues until the current procedure has completed and stops at the first statement following the current procedure call, unless a breakpoint is encountered first.
- To step over the procedure called in a breakpoint line, chose 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.

{button ,AL(`H_IDE_OPENING_AND_USING_THE_SCRIPT_DEBUGGER_STEPS',1)} Go to procedure

Opening and using the Script Debugger

1. In the Script Editor, select a script to debug.
2. Set one or more breakpoints.

   See Setting breakpoints in the IDE
3. Choose Script - Run Current Sub.

   Script execution stops when the IDE reaches the first breakpoint. The Debugger pane appears and the Debug menu replaces the Script menu.

   Note You cannot edit a script in the Debugger. To return to the Script Editor and edit the script, choose Debug - Stop Execution.
4. Choose Debug - Continue Execution.

   The IDE continues script execution to the next breakpoint or the end of the script. When the script ends, the IDE returns to the Script Editor.

{button ,AL(`H_IDE_OPENING_AND_USING_THE_SCRIPT_DEBUGGER_DETAILS',1)} See details

{button ,AL(`;H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_CLOSING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_OPENING_AND_USING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_SETTING_AND_CLEARING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

Pasting from the Browser into a Script

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

1. Open a script in the Script Editor.
2. Move the insertion point to the location where you want to paste.
3. Click the Browser tab to open the Browser.
4. Select a category from the "Category" box.
5. Click the right twistie (▸) to expand levels of information or the down twistie (▾) to collapse levels of information about the selected entry in the list.
6. Select the item to paste in the Browser panel.
   You can display syntax information for an element by selecting it and pressing F1.
7. Click the Paste Name button to paste that item into your current script.

{button ,AL(`;H_IDE_FINDING_SYNTAX_INFORMATION_IN_THE_BROWSER_STEPS;H_IDE_THE_BROWSER_OVER;H_IDE_VIEWING_ITEMS_IN_THE_BROWSER_LIST_STEPS',0)} See related topics

Selecting a script in the IDE

To edit or debug scripts in the IDE:

1. Open the IDE.
2. Select the object whose script you want to display from the Object drop-down box.
3. Select the script you want to display from the Script drop-down box.

{button ,AL(`;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_SELECTING_AN_OBJECT_TO_WORK_WIT H_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Setting breakpoints in the IDE

You can set 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.

Setting breakpoints

1. Open the IDE.
2. Open the desired script.
3. Do one of the following:
   - Click in the breakpoint gutter next to the line in the Script Editor.

     If the line cannot legally have a breakpoint, the IDE does not set one.
   - Choose Script - Set Breakpoint.

     You can do this if no breakpoint is set for the current statement (or in the current selection of statements) and it can legally have a breakpoint.

Note You can also set breakpoints in the Debugger by clicking in the breakpoint gutter or by choosing Debug - Set Breakpoint.

{button ,AL(`H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_DISABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_ENABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_NAVIGATING_A_SCRIPT_USING_THE_BREAKPOINTS_PANEL_STEPS;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER;H_IDE_CLEARING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

The Breakpoints panel

The Breakpoints panel lists all breakpoints in the order that you set them. You can use the panel to navigate to breakpoints in your script, where you can clear, enable, or disable breakpoints.

Note The breakpoint statement is followed by (Disabled) if the breakpoint is disabled.

The Breakpoints panel lists each breakpoint in the following format:

```
object: script: line#
```

For example,

```
(Globals): FillLine 31
```

Choose one of the following tasks:

- Navigating a script using the Breakpoints panel
- Enabling breakpoints in the IDE
- Disabling breakpoints in the IDE
- Clearing breakpoints in the IDE

Overview: The Output panel

When you execute scripts in a document, the IDE displays in the Output panel up to 1024 characters of output from each Print statement included in the scripts. This output is displayed in the order it is generated in the document.

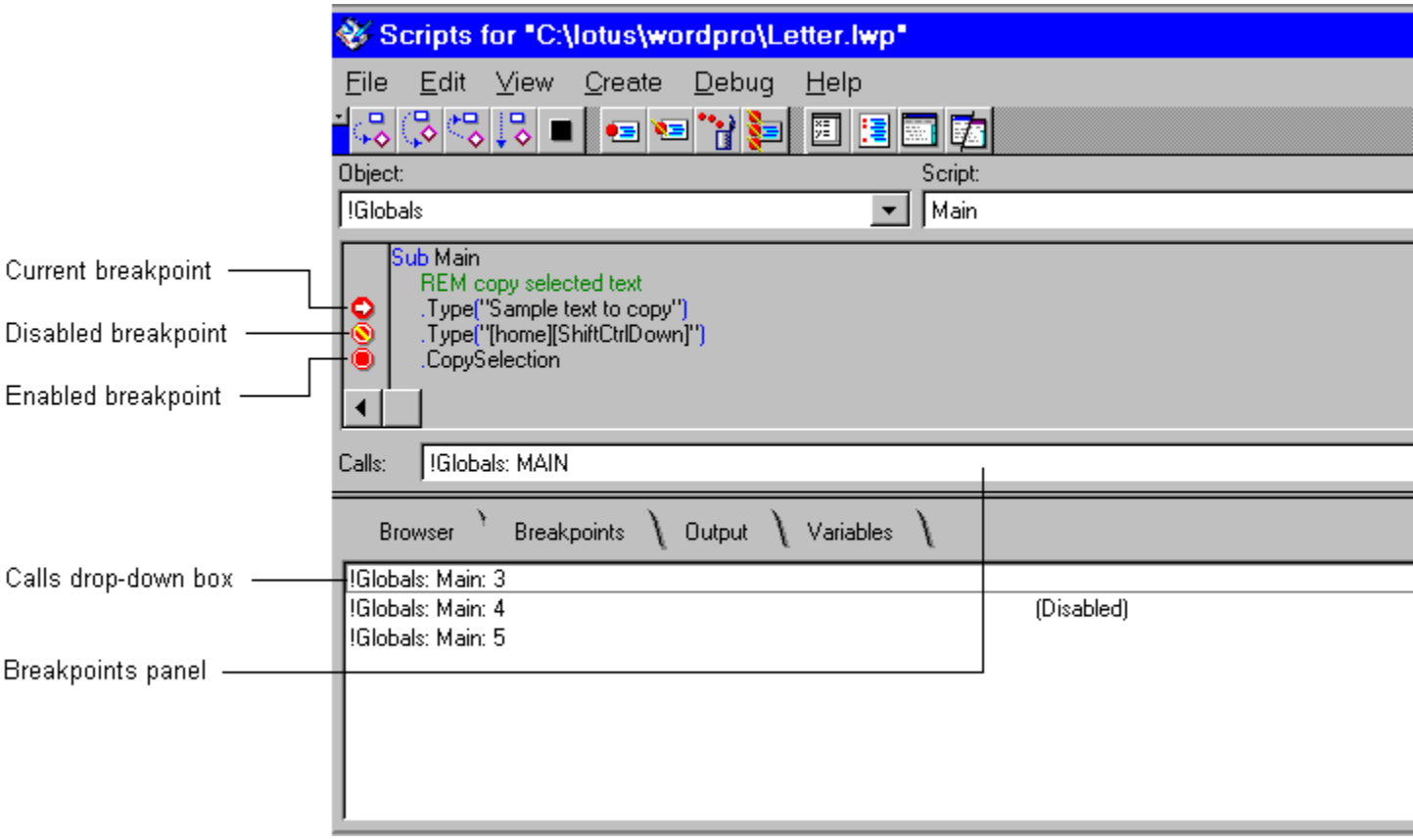
You can review the output list to determine whether scripts are executing as you expect.

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

{button ,AL(`;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_RUNNING_A_SCRIPT_IN_THE_IDE_STEPS;
H_IDE_SAVE_SCRIPTS_STEPS;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Overview: The Script Debugger

The IDE Script Debugger appears automatically when a script is executing and encounters a breakpoint, a Stop statement, or a run-time error. While the Script Debugger is displayed, you can set, clear, disable, and enable breakpoints. You can also step through scripts to locate the source of run-time errors.



When the Script Debugger is open:

- The Debug menu is displayed in the IDE menu.
- All scripts become read-only. You cannot edit them.

  Note 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.
- 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.
- The "Calls" drop-down box provides information about the name of the current procedure and any procedures that called it.
- The Breakpoints panel lists all breakpoints that you set in the order that you set them. It also lets you navigate to breakpoints, clear them, enable them, or disable them.
- The Output panel displays output generated by LotusScript Print statements.
- The Variables panel displays information about variables for the current procedure and lets you change their values while debugging the script.

Encountering run-time errors

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

- Displays an error dialog box.

  Tip Press F1 to view Help for the error message.
- Displays the script containing the error.

- Marks the statement that caused the error with an arrow in the breakpoint gutter.
- Disables all Debug menu items except for breakpoint commands and the Stop Execution command.

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 is encountered, Debuggers are displayed in place of the Editors, whether or not scripts for those documents are executing.

{button ,AL(`;H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_CHANGING_VARIABLES_STEPS;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_THE_LOTUSSCRIPT_IDE_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER',0)} See related topics

Overview: The 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 dataType 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.

{button ,AL(`;H_IDE_CHANGING_VARIABLES_STEPS;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_THE_BREAKPOINTS_PANEL_CS;H_IDE_THE_BROWSER_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Overview: Using breakpoints in debugging

Breakpoints are markers that you set in your scripts to assist in the debugging process. When you set a breakpoint on a statement, the Script Debugger pauses at that line when it executes your script. You can get an accurate picture of how your script is executing by placing breakpoints at key points in your script. If you do not want a breakpoint to pause execution of your script, you can disable it. In effect, the breakpoint is still on the line in case you need it again, but the Script Debugger does not pause execution at that line.

With a few exceptions, 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.

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

You cannot set breakpoints for the following:

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

You can disable, enable, and clear a breakpoint when you edit or debug a script.

Setting breakpoints vs. enabling breakpoints

A breakpoint, when set, is enabled (in operation) by default. The IDE stops script execution when it reaches an enabled breakpoint.

Clearing breakpoints vs. disabling Breakpoints

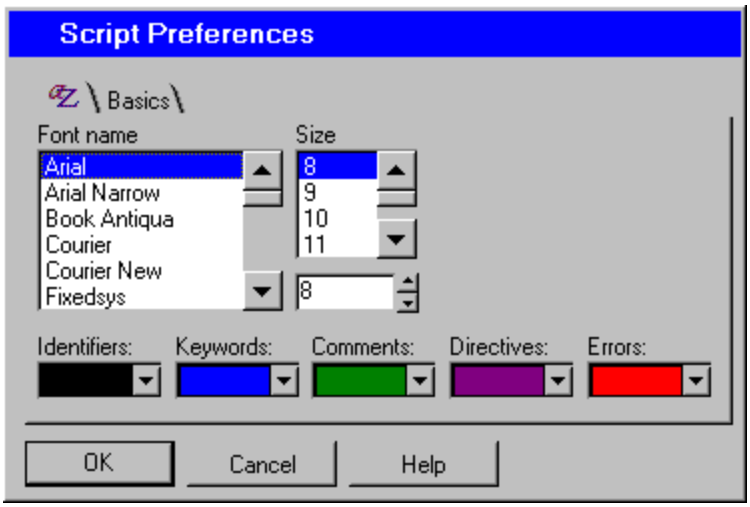
Clearing a breakpoint removes it from the script. When you disable a breakpoint, it remains in the script. However, the IDE ignores it when the script executes. Disabling breakpoints is useful when you want to stop using breakpoints temporarily but plan to use them again later.

{button ,AL(`H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_DISABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_ENABLING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_NAVIGATING_A_SCRIPT_USING_THE_BREAKPOINTS_PANEL_STEPS;H_IDE_SETTING_BREAKPOINTS_IN_THE_IDE_STEPS;H_IDE_CLEARING_BREAKPOINTS_IN_THE_IDE_STEPS',0)} See related topics

Changing text fonts and colors in the IDE

The IDE identifies keywords, comments, directives, identifiers, and errors in a script by displaying them in a specified color. You can change these colors, and you can also change the font of the IDE display.

1. Choose File - Script Preferences.



2. In the Text Format panel, select a color for each language element.
3. Select a font and a size for all text.
4. Click OK.

{button ,AL(`H_IDE_INDENTING_SCRIPT_STATEMENTS_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_SETTING_UNDO_LEVELS_IN_THE_IDE_STEPS;H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS;',0)} See related topics

Creating a SmartIcons set in the IDE

You can specify a name for a bar of SmartIcons and a file name in which to save the SmartIcons.

1. Choose File - SmartIcons Setup.
2. Select the desired SmartIcons set in the "Bar name" box.

   You can use the default Script Editor SmartIcons set as a base for the new set or select another set from the "Bar name" box.
3. Drag the icons to add, move, and remove them until the set is the way you want it.
4. Select an option in the "Bar can be displayed when context is" box.
5. Click Save Set then click Save As New.
6. Type a name in the "SmartIcons bar name" box.
7. Specify a name in the "File to save to" box.

   You must save a new set of SmartIcons under a different name to prevent it from overwriting an existing set.
8. Click OK to return to the SmartIcons Setup dialog box.
9. Click OK.

{button ,AL(`H_IDE_CREATING_A_SMARTICONS_SET_IN_THE_IDE_STEPS;H_IDE_SELECTING_AND_DISPLAYING_SMARTICONS_IN_THE_IDE_STEPS;H_IDE_CHANGING_FONTS_AND_TEXT_COLORS_IN_THE_IDE_STEPS;H_IDE_CLOSING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_STEPS',0)} See related topics

Deleting a SmartIcons set in the IDE

1. From the IDE File menu, choose SmartIcons Setup.
2. Click Delete Set.
3. Select one or more sets in the "Bar(s) of SmartIcons to delete" box.
4. Click OK.
5. Click Yes to confirm.
6. Click OK to close the SmartIcons Setup dialog box.

{button ,AL(`H_IDE_CREATING_A_SMARTICONS_SET_IN_THE_IDE_STEPS;H_IDE_SELECTING_AND_DISPLAYING_SMARTICONS_IN_THE_IDE_STEPS;H_IDE_CHANGING_FONTS_AND_TEXT_COLORS_IN_THE_IDE_STEPS;H_IDE_CLOSING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_STEPS',0)} See related topics

Displaying the Error Count dialog box

The Error Count dialog box reports how many errors the IDE encounters when it checks a script. The Error Count dialog box is enabled by default.

1. Choose File - Script Preferences.
2. Click the Basics tab.
3. Choose one of the following:
   - Select "Show Error dialog after checking syntax" to have the IDE display the Error Count dialog box after checking a script.
   - Deselect "Show Error dialog after checking syntax" to have the IDE not display the Error Count dialog box after checking a script.
4. Click OK.

{button ,AL(`H_IDE_CHECKING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS;H_IDE_RUNNING_A_SCRIPT_IN_THE_IDE_STEPS;H_IDE_SETTING_TAB_WIDTHS_IN_THE_IDE_STEPS;H_IDE_SETTING_UNDO_LEVELS_IN_THE_IDE_STEPS;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Details: Finding and replacing text in the IDE
When you find and replace text in the IDE, you can set a scope for the search by selecting an option in the "Look-in" box.

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 option is the default.

All scripts for this document
Searches all scripts in the current document.

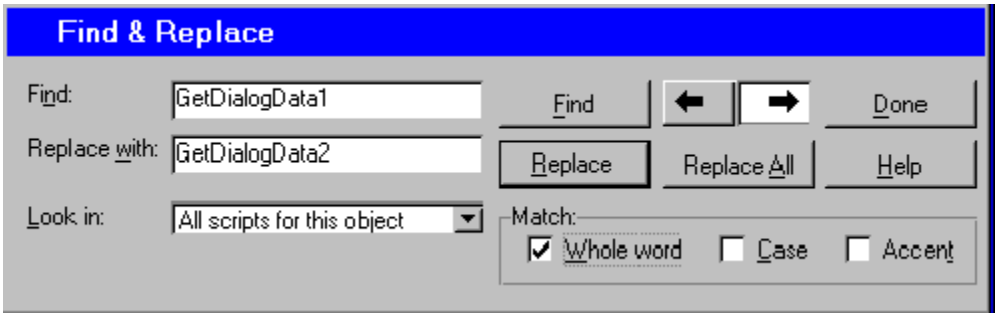
{button ,AL(`H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS',1)} Go to procedure

Finding and replacing text in the IDE

Find and replace is a useful feature if you need to update all references of variables and scripts to new names.

Note The IDE is divided into two panes: the upper pane which can contain either the Script Editor or the Debugger and the lower pane which contains the Utilites panels. You can use Find to search for text in the Editor or the Debugger; however, you can only replace text in the Editor.

1. Choose Edit - Find & Replace.



2. Enter the characters you want to find in the "Find" box.
3. Enter the new characters in the "Replace with" box.
4. (Optional) Select options in the "Look in" and "Match" boxes.

   The "Look in" drop-down box sets a scope for the search.
5. Click Find.

   The IDE highlights the first occurrence of the characters.
6. Click Replace.

   The Script Editor replaces the characters and moves to the next occurrence.
7. (Optional) To replace the next occurrence of the characters, click Replace again; to replace all occurrences of the characters, click Replace All.
8. Click Done.

{button ,AL(`H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_DETAILS',1)} See details

{button ,AL(`;H_IDE_CHECKING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_RENAMING_FUNCTIONS_AND_SUBS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Details: Finding syntax information in the Browser
You can find syntax information in the Browser for the following:

- LotusScript language elements
- All available Lotus product constants and variables
- All classes and the properties, methods, and events for those classes
- Lotus Dialog Editor control classes and the methods, properties, and events for these classes
- OLE Automation classes and the properties and methods for these classes
- Methods, properties, and events for ActiveX components embedded in your current document

---

{button ,AL(`H_IDE_FINDING_SYNTAX_INFORMATION_IN_THE_BROWSER_STEPS',1)} Go to procedure

Finding syntax information in the Browser

To find syntax information for elements listed in the Browser.

1. Click the Broswer tab.
2. Select an item in the Category box.
3. Click a twistie to display the desired element.
4. Select the element.
5. Press F1.

When the Script Editor is active, you can select an item in the Browser and click Paste Name to paste it into the current script.

{button ,AL(`H_IDE_FINDING_SYNTAX_INFORMATION_IN_THE_BROWSER_DETAILS',1)} See details

{button ,AL(`;H_IDE_IDE_HELP_OVER;H_IDE_THE_BROWSER_OVER;H_IDE_VIEWING_ITEMS_IN_THE_BROWSER_LIST_STEPS',0)} See related topics

Overview: Formatting block statements in the IDE

The Script Editor automatically completes block statements based on what you enter as the beginning statement. For example, when you enter a beginning statement such as Do, the Script Editor inserts 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 ending statements

When you enter the following beginning statements, the Script Editor completes the block.

Note Optional items are enclosed in square brackets.

If you enter...	The Script Editor enters...
Sub *subName* [*(argList)*]	End Sub
Function *functionName* [*(argList)*]	End Function
Property Get *name* [As *dataType*] [*(argList)*]	End Property
Property Set *name* [As *dataType*] [*(argList)*]	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 another window, or check script syntax, the Script Editor completes any unterminated multiline strings, %REM blocks, and Sub, Function, Property, Type, and Class blocks.

{button ,AL(`;H_IDE_CREATING_FUNCTIONS_AND_SUBS_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS;H_IDE_INDENTING_SCRIPT_STATEMENTS_STEPS',0)} See related topics

Overview: IDE Help

While you are in the Lotus IDE, you can use the Help menu or press the F1 key to access help and syntax information.

Help menu

Three types of help are available from the Help menu.

- LotusScript Help -- reference Help for the LotusScript language
- Script Editor Help -- Help for the IDE
- Lotus *product* Object Help -- reference for the product's specific object scripts

F1 Help

To display context-sensitive Help, press F1. For example, if you type a keyword in the Script Editor and press F1, the IDE displays reference information on that keyword.

Context-sensitive Help is available for the following:

- The Browser, Breakpoints, Output, and Variables panels
- LotusScript keywords displayed in the Script Editor, Script Debugger, or Browser panel

  If the current line or the preceding line contains a syntax error and a keyword is not selected, pressing F1 displays information about the syntax error.
- Product classes, constants, variables, and procedures displayed in the Script Editor, Script Debugger, or Browser panel
- Many OLE Automation classes 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

---

{button ,AL(`;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_THE_BROWSER_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER;H_IDE_USING_BREAKPOINTS_IN_DEBUGGING_OVER',0)} See related topics

Indenting script statements

Smart Indenting automatically indents statements as you complete them in the Script Editor.

1. Choose File - Script Preferences.
2. Click the Basics tab.
3. Do one of the following:
   - Select "Smart Indenting" to turn on Smart Indenting.
   - Deselect "Smart Indenting" to turn off Smart Indenting.
4. Click OK.

By default, the Script Editor uses opening and closing block keywords, such as If and End If, to determine indentation levels.

{button ,AL(`;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_SETTING_TAB_WIDTHS_IN_THE_IDE_STEPS;H_IDE_SETTING_UNDO_LEVELS_IN_THE_IDE_STEPS',0)} See related topics

Keyboard shortcuts in the IDE

You can use the keyboard to select text and move the insertion point in the IDE.

Keyboard shortcut	Task
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.
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.

{button ,AL(`;H_IDE_CHANGING_FONTS_AND_TEXT_COLORS_IN_THE_IDE_STEPS;H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS',0)} See related topics

Details: Opening and closing the IDE

What happens when the IDE opens
The Browser, Output, and Breakpoints panels are accessible.

The Script Editor displays a script, if one exists for the current object.

- If you displayed the scripts for the current object earlier in the current session, the Script Editor restores the last edited script.
- If you have not displayed the scripts for the current object during the current session, the Script Editor displays the first editable script for the object.
- If the current 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.

What happens when the IDE closes
The IDE reformats line indentations and saves all scripts.

When you close the IDE, it completes the following automatically:

- %REM blocks
- Multiline strings
- Sub blocks
- Function blocks
- Property blocks
- User-defined type and class blocks

Other ways to close the IDE
- 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.

{button ,AL(`H_IDE_OPENING_AND_CLOSING_THE_IDE_STEPS',1)} Go to procedure

Opening and closing the IDE

To open the IDE, you must first run a Lotus product and then open a document in that product.

Note An IDE window is associated with one product document.

To open an IDE window

From the product's Edit menu, open the Script Editor.

To close an IDE window

Click the Close button in the top right corner of the IDE window.

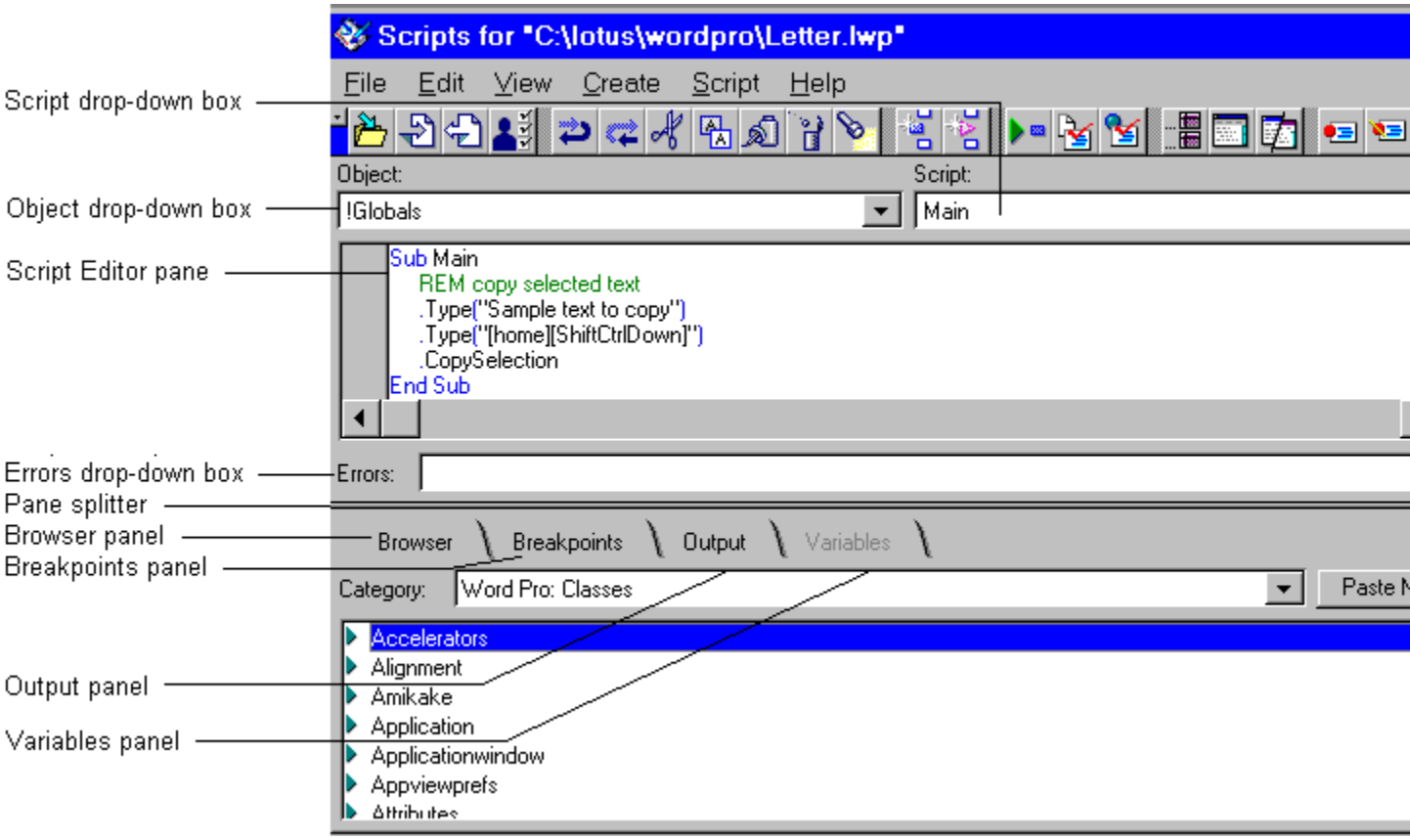
Note If you try to close the IDE while the Script Debugger is running and contains an active breakpoint, the IDE displays a message prompting you to continue debugging or to stop execution of the current script.

{button ,AL(`H_IDE_OPENING_AND_CLOSING_THE_IDE_DETAILS',1)} See details

{button ,AL(`;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER;H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_STEPS;H_IDE_THE_LOTUSSCRIPT_IDE_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Overview: Parts of the IDE window

The IDE provides several tools in one window.



- The Object drop-down box lists all objects for the current document.
- The Script drop-down box lists all scripts for the current object.
  - Either the Script Editor or the Script Debugger is displayed in the top pane, depending on which is currently active.
  - The Script Editor lets you write scripts and check their syntax. It also lets you set, clear, disable, and enable breakpoints.
- The Script Debugger lets you set, clear, disable, and enable breakpoints. It also lets you step through scripts to locate the source of problems that may occur while a script is executing.
- The Pane Splitter lets you resize, display, or hide panes in the IDE window.
- The Errors drop-down box displays any errors the IDE finds in the current script.
- The Script Utilities pane has four panels, each containing a tool:
  - The Browser panel lists LotusScript keywords; classes, constants, procedures, and variables 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.

{button ,AL(`;H_IDE_BREAKPOINT_SYMBOLS_OVER;H_IDE_THE_LOTUSSCRIPT_IDE_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER;H_IDE_USING_BREAKPOINTS_IN_DEBUGGI

NG_OVER',0)} See related topics

Saving a script as an .LSO file

If you need to execute your script within multiple SmartSuite products, you can save the object code as an .LSO file.

In the IDE, choose File - Export Globals As LSO.

The IDE saves the code as an .LSO file in the specified directory. Because an .LSO file contains object code, you cannot edit it directly. To change an .LSO file, you must edit the source code contained in the corresponding .LWP, .MWP, or .LSS file.

Note When you export scripts to an .LSO file, the Script Editor only exports compiled code that is part of the !Global object.

If you want to call a sub or function in an .LSO file, use the following syntax:

Use "filename.lso"

```
MyProcedure
```

{button ,AL(`;H_IDE_SAVING_A_SCRIPT_AS_AN_LSO_FILE_STEPS;H_IDE_SAVING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_CREATING_SCRIPTS_BY_ENTERING_SCRIPT_STATEMENTS_STEPS',0)} See related topics

Saving a script as an .LSS file

Sometimes you may want to save your script as an external text file. In this case, you can export your script to an .LSS file.

In the Script Editor, choose File - Export Script.

The IDE saves the current script or all the scripts for an object in the current document as an .LSS file in the specified directory.

If you want to call a sub or function in an .LSS file, use the following syntax to include the .LSS file:

```
%Include "filename.lss"
```

{button ,AL(`;H_IDE_SAVING_A_SCRIPT_AS_AN_LSS_FILE_STEPS;H_IDE_SAVING_SCRIPTS_IN_THE_SCRIPT_EDITOR_STEPS',0)} See related topics

Details: Selecting and displaying SmartIcons in the IDE

Bar can be displayed when context is
You can display a set of SmartIcons at certain times.

When you are working in the context you select here, the IDE adds this SmartIcons set to the available list of sets displayed when you click the Bar button. If you are not working in this context, the IDE automatically hides this SmartIcons set.

For example, you are working in the Script Editor and want a custom set of SmartIcons to display.

- First, select the custom set of SmartIcons in the "Bar name" box.
- Then select "Script Editor" in this box.

Now you can display the custom set when you're working in the Script Editor by selecting it from the list when you click the Bar button.

Bar is enabled to display during its context
This option acts as an on/off display switch. If selected, it displays a specific set of SmartIcons whenever you're working in the selected context. Deselect this option to turn off the display.

Selecting this option is the equivalent of selecting the SmartIcons set using the Bar button.

For example, you customize a set of SmartIcons for use in the Debugger and name it under a separate file name. You always want that set to appear when you work in the Debugger

- First, select the custom set in the "Bar name" box.
- Second, select "Script Debugger" in the "Bar can be displayed when context is" box.
- Then select this option to display the set whenever you work in the Debugger.

{button ,AL(`H_IDE_SELECTING_AND_DISPLAYING_SMARTICONS_IN_THE_IDE_STEPS',1)} Go to procedure

Selecting and displaying a SmartIcons set in the IDE

You can choose which set of SmartIcons to display and when each set is displayed.

1. Choose File - SmartIcons Setup.
2. Select the desired SmartIcons set in the "Bar name" box.
3. Select an option in the "Bar can be displayed when context is" box.
4. Select "Bar is enabled to display during its context" if you want to display the set at specific times.
5. Click OK.

{button ,AL(`H_IDE_SELECTING_AND_DISPLAYING_SMARTICONS_IN_THE_IDE_DETAILS',1)} See details

{button ,AL(`H_IDE_DELETING_A_SMARTICONS_SET_IN_THE_IDE_STEPS;H_IDE_CREATING_A_SMARTICONS_SET_IN_THE_IDE_STEPS;H_IDE_CHANGING_FONTS_AND_TEXT_COLORS_IN_THE_IDE_STEPS;H_IDE_CLOSING_THE_SCRIPT_DEBUGGER_STEPS;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_STEPS',0)} See related topics

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.

Choose a task:

- Changing fonts and text colors in the IDE
- Setting tab widths in the IDE
- Indenting script statements
- Displaying the Error Count dialog box
- Setting Undo levels in the IDE

Setting tab widths in the IDE

Well-chosen tabs increase the readability of your script.

1. Choose File - Script Preferences.
2. Click the Basics tab.
3. Enter a tab stop number.

   The IDE default tab size is 3 characters; you can specify any setting between 2 and 16.
4. Click OK.

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

{button ,AL(`;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_INDENTING_SCRIPT_STATEMENTS_STEPS;H_IDE_SETTING_UNDO_LEVELS_IN_THE_IDE_STEPS',0)} See related topics

Setting Undo levels in the IDE

By default, the IDE keeps track of the last five text edits that you performed (entering, modifying, deleting, or pasting text). However, you can change the number of edits the IDE tracks for Undo.

1. Choose File - Script Preferences.
2. Click the Basics tab.
3. Specify the desired number of "Undo Levels."

   You can specify a number from 0 through 10.
4. Click OK.

Note To disable the Undo feature, deselect "Undo" in the Basics panel. Deselecting "Undo" is the same as setting the option to 0.

{button ,AL(`;H_IDE_DISPLAYING_THE_ERROR_COUNT_DIALOG_BOX_STEPS;H_IDE_FINDING_AND_REPLACING_TEXT_IN_THE_IDE_STEPS;H_IDE_FORMATTING_BLOCK_STATEMENTS_IN_THE_IDE_OVER;H_IDE_SETTING_TAB_WIDTHS_IN_THE_IDE_STEPS',0)} See related topics

Details: Switching and resizing panes in the IDE

Another way to switch panes
Press F6 to toggle activation between the Script Editor or Script Debugger pane and the current panel in the Utilities pane.

Other uses of the Pane Splitter
You can double-click the Pane Splitter to toggle between single and double panes and you can press CTRL and click the Pane Splitter to restore the default heights for panes in the IDE window.

{button ,AL(`H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_STEPS',1)} Go to procedure

Switching and resizing panes in the IDE

To switch panes
Click in the pane to activate it.

To resize a pane
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.

{button ,AL(`H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_DETAILS',1)} See details

{button ,AL(`;H_IDE_THE_BROWSER_OVER;H_IDE_THE_LOTUSSCRIPT_IDE_OVER;H_IDE_THE_OPTIONS_SCRIPT_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER',0)} See related topics

Overview: The Browser

The Browser provides syntax information for the following:

- The LotusScript language
- Classes, constants, procedures, and variables available for Lotus products
- Classes, constants, and variables available for Dialog Editor controls
- OLE Automation classes
- Methods and properties for OLE servers
- Methods, properties, and events for OCX components embedded in your current SmartSuite product document

---

{button ,AL(`H_IDE_FINDING_SYNTAX_INFORMATION_IN_THE_BROWSER_STEPS;H_IDE_SWITCHING_AND_RESIZING_PANES_IN_THE_IDE_STEPS;H_IDE_VIEWING_ITEMS_IN_THE_BROWSER_LIST_STEPS;H_IDE_PARTS_OF_THE_IDE_WINDOW_OVER',0)} See related topics

Overview: The LotusScript IDE

The LotusScript Integrated Development Environment (IDE) provides a powerful set of tools for creating and debugging scripts in Lotus products.

The separate tools of the IDE are described in Overview: Parts of the IDE window.

Features of the IDE

- Each IDE window contains menus, SmartIcons, an editor, a debugger, a browser and other utilities.
- The IDE is closely integrated with documents in a Lotus product. For each active document in a product, you can display an IDE window that displays the scripts for 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.
- The IDE saves your scripts in product documents. For example, scripts developed in the IDE for a document named INVEST.LWP are saved in INVEST.LWP. To save your scripts separate from product documents, you can export them as .LSS text files or compiled .LSO files.
- The IDE uses context-sensitive menus and SmartIcons to focus your programming tasks.

What is a script?

A script is a sequence of one or more LotusScript statements. A script can be a complete application or part of an application.

What is a document?

A document is the product file.

If you are working in...	Then your document is...
1-2-3	a workbook file (a .123 file)
Word Pro	a document (an .LWP file)
Freelance	a presentation (a .PRZ file)
Approach	a database (a .APR file)

{button ,AL(`;H_IDE_IDE_HELP_OVER;H_IDE_THE_BROWSER_OVER;H_IDE_THE_LOTUSSCRIPT_IDE_OVER;H_IDE_THE_OPTIONS_SCRIPT_OVER;H_IDE_THE_OUTPUT_PANEL_OVER;H_IDE_THE_SCRIPT_DEBUGGER_OVER;H_IDE_THE_SCRIPT_EDITOR_CS',0)} See related topics

Viewing items in the Browser list

To view items in the Browser list, you can use any of the following keyboard or mouse procedures:

You can...	To...
Press +	Expand an item in the Browser list.
Press -	Collapse an item in the Browser list.
Press SHIFT and +	Expand all items in the Browser list.
Press SHIFT and -	Collapse all items in the Browser list.
Click a right twistie ()	Expand an item in the Browser list.
Click a down twistie ()	Collapse an item in the Browser list.

{button ,AL(`;H_IDE_KEYBOARD_SHORTCUTS_IN_THE_IDE_STEPS;H_IDE_THE_BROWSER_OVER;H_IDE_THE_LOTUSSCRIPT_IDE_OVER',0)} See related topics

Can't run this procedure

You have a syntax error in your script. Use the menu command Script - Check Scripts to see where the error is.

If you use the menu command Script - Run Current Sub to run a function or a sub containing parameters, you will also get this error. You can only use Script - Run Current Sub to run a function or sub without parameters.

Could not open file for export

The IDE could not 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 could not 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 could not 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 failed.

Check the network connections to the printer.

Replace failed

The IDE could not replace the text.

Check that the text is not in a read-only file, and that you have enough memory.

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

{button ,AL(`H_IDE_IDE_ERROR_MESSAGES_OVER_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ALREADYHASENDER_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;', 0)} See related topics

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 an %If directive in a script you created in the IDE.

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

{button ,AL(`H_IDE_STR_BADDIR_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_CASEELSE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

DIM required on declarations in this scope

You declared a global variable 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.

{button ,AL(`H_IDE_STR_DIMREQ_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_DUPLICATESECTION_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLCLSTMT_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLCODEOUTSIDE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

Illegal character after continuation character

The line-continuation character underscore (_) is followed on the same line by a character that is not ' (single quotation mark), 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.

{button ,AL(`H_IDE_STR_ILLCONTINUE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLDECLSCOPE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)}
See related topics

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.

{button ,AL(`H_IDE_STR_ILLESCAPE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLESCAPEID_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

Illegal executable code in Declarations

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

Only the following are allowed in (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).

{button ,AL(`H_IDE_STR_ILLEXECUTABLECODE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLINCLUDE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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 one 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.

{button ,AL(`H_IDE_STR_ILLPRODINST_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

Edit the Def*type* statement.

{button ,AL(`H_IDE_STR_ILLRANGE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLSTMT_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLSTMTINSCOPE_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)}
See related topics

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.

{button ,AL(`H_IDE_STR_ILLSUBPROG_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_ILLTYSTMT_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

Illegal sub initialization

You tried to create a procedure named Initialize or Terminate.

The routines Initialize and Terminate must be defined as subs.

Redefine your procedure as a sub, or rename it.

{button ,AL(`H_IDE_STR_INITTERMSUB_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPSH_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_STR_ILLSTMT>OVERVIEW;',0)} See related topics

.. 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.

{button ,AL(`H_IDE_STR_INVALIDDD_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_INVALIDME_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_INVFWD_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_INVFWDCL_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_NAMETOOLONG_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)}
See related topics

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
```

{button ,AL(`H_IDE_STR_NEEDSET_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_OUTOFMEMORY_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_PARSESTACKOVERFLOW_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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 [8], the ASCII code for a 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.

{button ,AL(`H_IDE_STR_SIMPLESYNTAX_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)}
See related topics

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.

{button ,AL(`H_IDE_STR_SPROGPAR_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_STMTSTACKOVERFLOW_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_SUFFKEYWORD_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)}
See related topics

Syntax checking buffer overflow

A statement you entered exceeds the limit of 32K. 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.

{button ,AL(`H_IDE_STR_TOKENOVERFLOW_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_UNMATCHEDBLOCKBEGIN_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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.

{button ,AL(`H_IDE_STR_UNMATCHEDBLOCKENDER_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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

With

Terminate the block with the appropriate statement.

{button ,AL(`H_IDE_STR_UNTERMBLK_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics

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 a double quotation mark on the same line where it starts.

{button ,AL(`H_IDE_STR_UNTERMSCONST_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)}
See related topics

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.

{button ,AL(`H_IDE_STR_UNTERMSQB_RT;H_IDE_FINDING_AND_FIXING_COMPILE_TIME_ERRORS_IN_THE_SCRIPT_EDITOR_STEPS;H_IDE_FIXING_ERRORS_REPORTED_IN_THE_SCRIPT_DEBUGGER_OVER;H_IDE_IDE_ERROR_MESSAGES_OVER;H_IDE_OPENING_LOTUSSCRIPT_HELP_IN_THE_IDE_STEPS;',0)} See related topics