Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Intermediate
Works with: Notes/Domino
Updated: 01-Jul-2003

by
Andre
Guirard

The Domino development environment includes a debugger for LotusScript code—a very handy tool. The **Domino Designer help** describes how to use this debugger. In this first of a two-part article series, we examine the debugger in detail with illustrations and examples. In addition, we list several common LotusScript error messages and show the coding errors that often cause them. In Part 2, we'll explain how to improve error reporting in your applications. We'll also discuss situations in which the debugger *doesn't* help.

This article has an accompanying sample Notes database that you can download from the **Sandbox**. This database contains examples of problem code you can debug, so you can see for yourself some of the error conditions discussed here. (The database also includes examples described in Part 2 of this series.) Also note that the screen shots and names of menu items shown in this article are based on Notes/Domino 6.5 (Beta version). However, except as noted, everything discussed here should also work in earlier versions as far back as R5.0.2.

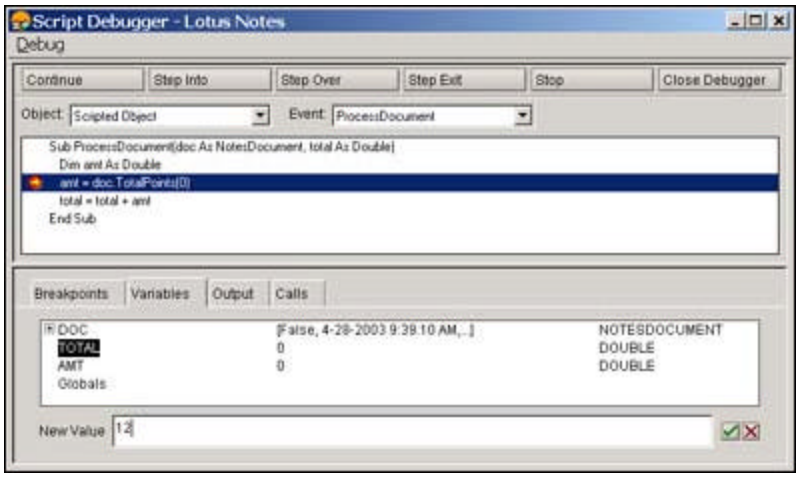
This article assumes that you're an experienced Domino application developer and at least a little familiar with LotusScript. For more information, consult the following Domino Designer help topics:

- Run-time error processing
- Using the LotusScript debugger
- Agent Manager debugging information
- NotesLog class
- Viewing the agent log
- Troubleshooting agents
- Using the Remote Debugger

A gentle introduction to the LotusScript Debugger

The LotusScript Debugger lets you pause LotusScript code in the middle of execution to see which line it's on, how you got to where you are (the “call stack”), and what the values of variables are. To activate the debugger, select File - Tools - Debug LotusScript.pt. The debugger window will open the next time you start to run a LotusScript agent or event script:

© Copyright IBM



Note: If a form or view window is already open, you cannot debug its events by activating the debugger. You must close and reopen that window before it "notices" that the debugger has been turned on.

You can do anything in the debugger only while it's paused. It pauses once when you start a program to let you set breakpoints or to "step" through the code. A breakpoint is a flag you put on a statement to tell the debugger to pause when it reaches that statement. Depending on which buttons you push, the code might run away with you and not stop again—if it gets into an infinite loop that has no breakpoints in it, there's no way to break into the debugger. You need to have arranged in advance for it to pause inside the loop.

In the preceding illustration, the red stop sign icon shows an active breakpoint. To add a breakpoint, double-click the line on which you want to pause. You can only set a breakpoint before a line that actually performs an action during execution. For instance, you can't break on the statement Dim amt As Double because it only contains information for the compiler about the name and datatype of a variable. However, if a Dim statement uses New, it performs an action (creates a new object), so you can break before that statement.

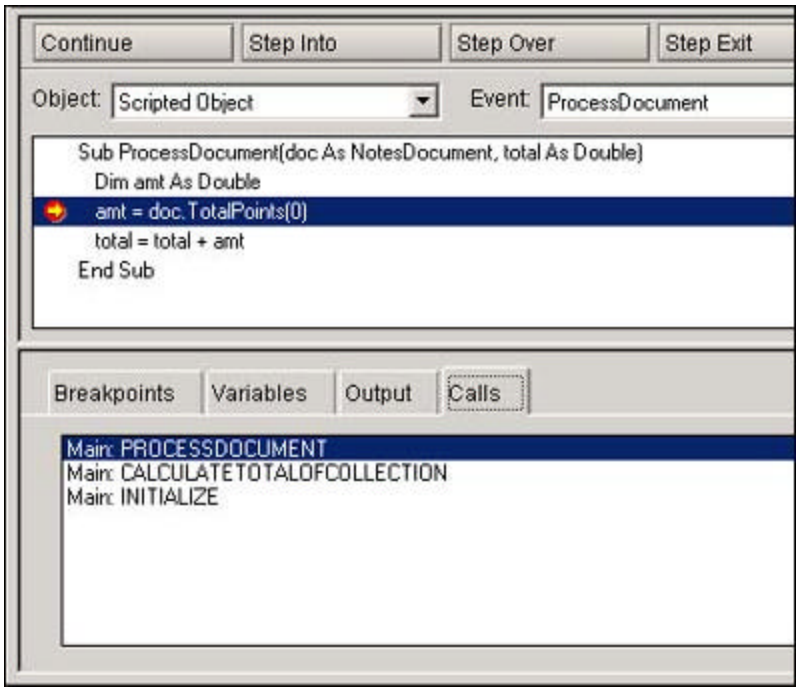
The yellow arrow (overlaid on the breakpoint icon in the preceding figure) shows which statement executes next.

To follow along with this article and view this same code in the debugger, open the sample database, turn on LotusScript debugging, highlight the rusty wand document, and use the action total \ 1. original. Click Step Into until you reach the line shown here. You won't have set any breakpoints, so you won't see the breakpoint stop sign, just the yellow arrow.

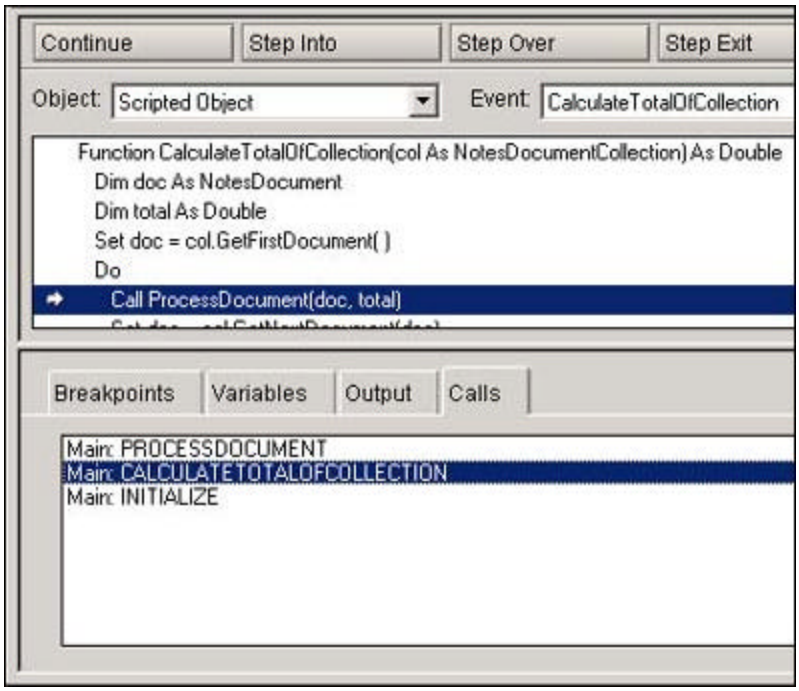
In the bottom panel, the Variables tab is selected, displaying the values of variables defined in this subroutine. The column on the right displays the datatypes. Click the plus sign (+) next to DOC to expand the values of its fields and properties. The line at the bottom shows the process of changing the value of the variable TOTAL from 0 to 12. Note that the Globals line lists values of variables available to the entire script defined in the Declarations section. There are no globals in this agent, or there would be a plus sign like the one next to DOC.

Next click the Calls tab:

© Copyright IBM



This tells us that the current subroutine (ProcessDocument) was called by a module (module is shorthand for subroutine or function) named CalculateTotalOfCollection, which was called from the Initialize subroutine of the agent. Click the second line in the Calls tab to show the code in CalculateTotalOfCollection. The following screen appears. (Note that in R5 the call stack is shown in a dropdown list field, but it works the same way.)



The line highlighted with an arrow is the current line in this function, the line that called ProcessDocument. The Variables tab now shows the local variables of the CalculateTotalOfCollection function (for example, col) instead of the ProcessDocument variables.

Because variables may be used in loops and If statements, changing them (as we did in the first illustration)

© Copyright IBM

may indirectly affect which statements get executed. Other than that, there's no way to affect the order in which statements execute; you can't tell the debugger to skip to a different statement. All you can do is set a breakpoint so that the debugger stops when it reaches that point in the normal flow of execution.

The debugger pauses:

- Just before the first statement of the script. This gives you a chance to set breakpoints before proceeding with the script.
- Whenever it comes to a Stop statement.
- Whenever it comes to a breakpoint (marked with a stop sign). There's also such a thing as a disabled breakpoint (a stop sign with a yellow diagonal line through it). The debugger doesn't pause here.
- After it has executed the amount of code you stepped over using one of the Step buttons.
- Any time an error condition occurs, unless there's an On Error statement to trap that error. It sometimes makes sense to comment out On Error statements in code before you debug, so that the debugger automatically stops at the location of the error.

It's not possible to continue execution of the script after it pauses because of error; only the Stop button works. But you get to see the line of code, the values of the variables, and the call stack, which are usually helpful. Here's how the different buttons instruct the debugger when to pause:

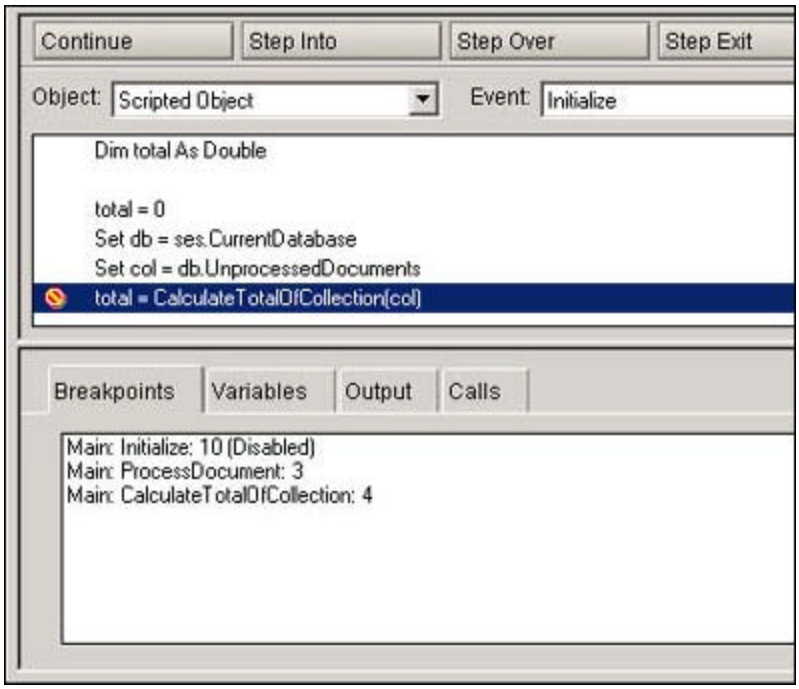
- Step Into tells the debugger to execute one statement and to pause. If the statement contains a subroutine or function call, the debugger pauses before the first executable statement of the called module. You are stepping "into" the module.
- Step Over tells the debugger to pause before the next statement in the current module. If the statement you're executing contains a call, the debugger will not pause in the module being called (except for a Stop statement, breakpoint, or error). If there's no call in the current line, Step Into and Step Over do the same thing.
- Step Exit tells the debugger not to pause again until it's done executing the current module (unless it hits a Stop, breakpoint, or error). The next pause comes after returning to the module that called this one. If this module wasn't called from another module (for instance, if it's the Click event of a button you clicked), the debugger will not pause again.
- Continue tells the debugger to complete the script without pausing again (unless it hits a Stop, breakpoint, or error).

An important side effect of the Continue button in R5: After you click Continue, if you're in a form event, the debugger will not pause again for any of the LotusScript on that form until you close and reopen the form. Before you press Continue, make sure you've set breakpoints in whatever other code you may want to debug. This is why we usually prefer to use Step Exit instead of Continue.

The Output tab shows any information your script outputs with Print statements. In client-side LotusScript, these also appear in the status bar of the Notes client, but you can't use the Notes client while you're paused in the debugger, so it's handy that you can also see it here. You can also scroll to see more output than the history on the status bar shows.

The Breakpoints tab shows where all the breakpoints are in all your modules:

© Copyright IBM



There are three breakpoints set in this agent. The one in the Initialize module is disabled, which means that execution won't pause there. The only difference between a disabled breakpoint and not having a breakpoint is that the disabled breakpoint shows in this list. You can display the code at the breakpoint by clicking the line in the breakpoints tab. Disabled breakpoints are a good way to "bookmark" lines in a large script so that you can find them quickly.

When to use the Stop statement

The Stop statement in client-side LotusScript has no effect unless the debugging mode is active. If the debugger is on, Stop makes execution pause at that point so that you can debug. In other words, the effect is the same as setting a breakpoint in the code with the disadvantage that you can never turn it off except by editing the code.

So why would you want to do this? In R5, when a debugging session ends, the debugger forgets the locations of your breakpoints. It gets tiresome to have to set the same breakpoints over and over, so you may instead choose to edit the code, insert Stop statements, and take them out when you're done debugging. Notes 6 remembers your breakpoints from one session to another, so you don't need Stop statements for that purpose.

Another use for Stop statements (in either R5 or Domino 6) is "conditional breakpoints" that pause on a certain line, but only if a condition is met.

Suppose you have an agent that processes many documents with no problem, but you know that on the document with ID code RW-2039, it calculates an incorrect result. Here's the loop where a document is fetched and processed:

```
Set doc = col.GetFirstDocument( )
Do Until doc Is Nothing
      Call ProcessDoc(doc)
      Set doc = col.GetNextDocument(doc)
Loop
```

You want to watch what the agent does when it reaches document RW-2039. You could set a regular breakpoint on the Call ProcessDoc line and just keep pressing Continue until you reach the document you want, but if the agent processes lots of documents, that could take some time. To save work, edit the agent and add the line in bold as follows:

© Copyright IBM

```
Set doc = col.GetFirstDocument( )
Do Until doc Is Nothing
      If doc.IDCode(0) = "RW-2039" Then Stop
      Call ProcessDoc(doc)
      Set doc = col.GetNextDocument(doc)
Loop
```

After you solve the problem, remove that line. Perhaps someday the debugger will let you specify a breakpoint condition. In the meantime, though it's not perfect, this technique may save you some time.

You can also use Stop to make a server agent pause to wait for you to start the Notes 6 remote debugger. For instance, if you want to pause for debugging at the very top of your agent, you may begin the Initialize event as follows:

```
Sub Initialize
    ' remove the following three lines for production.
    Print "Waiting for remote debugger."
    Stop
    Print "Debug wait ended."
```

Set the length of the pause by editing the Server Configuration document in the Domino Directory.

Even if you don't have a Stop statement, the agent manager automatically pauses to wait for the remote debugger to attach before starting the agent. When it pauses, the message "Agents delayed *xx* seconds to provide an opportunity for the remote debugger to attach" appears on the server console. The message doesn't tell you which agent is waiting, and anyway you usually don't want to have to wait until the agent you want to debug runs on its own. To tell the server to run your agent immediately, use the server console command tell amgr run databasepath 'agentname'.

The error doesn't happen when I debug!
Your code causes an error whenever you run it, but when you use the debugger it works just fine! What's happening?

For example, see the Test Conduit agent in the sample database. When you run the agent normally, you get the error "Type mismatch on external name: REPOLARIZE." But when you run it in debug mode, you can step through the entire agent without any error. This is usually caused by a script library being edited more recently than the code calling it. When you save changes to LotusScript code, Notes compiles the code. If you later change a script library that the script calls, its definition may become incompatible with the compiled agent code, hence the error. To fix the problem, edit and re-save the agent. You may have to make some trivial change (such as add and remove a space) to force a recompile.

In our example, the agent uses the script library Conduit which contains the function Repolarize. The definition of Repolarize changed after the agent was written. It used to be Sub Repolarize(c As Long); now it's Function Repolarize(Byval c As Double) As Double.

Why don't you get the error when you debug? If you start running a piece of LotusScript code and the debugger is active, the debugger recompiles the code. This special debug compile lets the debugger insert breakpoints and examine variables. The compiled code is held in memory during the debug; when you exit the debugger, it goes away. Depending on how the script library has changed, the compile may not work. For instance, if the change to Repolarize added a new parameter, the recompile would fail because the agent is passing only one parameter. It's actually unfortunate in this case that the agent does compile because it is syntactically correct but not functionally correct. When you change a script library, it's a good idea to also find and correct all calls to any changed functions. In this case, the call should have been changed from Repolarized(r) to r = Repolarize(r).

Note: Domino Designer 6 has a function to recompile all LotusScript code in your application, starting with the lowest-level script libraries and working upwards (Tools – Recompile all LotusScript). This is an easy way to

© Copyright IBM

make sure all your code is compiled in the correct order.

Errors in indirectly called Event code
You may encounter the mysterious situation in which you see an error and debug it, but the line that the error is on doesn't seem remotely likely to have caused this error. For instance, we recently came across a situation where the following line in an agent was giving a Type mismatch error:

```
uidoc.GotoField "Price"
```

This was very strange. Type mismatch (described later in the list of common errors) means there's been an error converting a value to the appropriate type. We can see from the debugger's Variables pane that uidoc is of the right type (NotesUIDocument) and the GotoField method takes a string argument (which we gave it), so there's no type mismatch there.

As it turned out, the problem wasn't in the agent at all, it was in the Entering event of the Price field. The debugger can only hold one script in memory at a time—in this case, the agent we were running and all the script libraries included in that agent. When another script executes as a side effect of the script you're running, it runs outside of the debugger; the debugger is already busy with your original script. Therefore, any LotusScript command that triggers an event script may cause errors that (like this one) seem to make no sense.

Part 2 of this article series will include a discussion of techniques for adding information to error messages to make it clear where they're occurring, even if the debugger can't show the actual line. In this case, if we could have had a message saying, "Type mismatch on line 15 of Entering" instead of just "Type mismatch," that would have saved a lot of time and brow furrowing.

Recognizing the most common errors

This section lists the most common error messages in Notes and Domino and describes their usual causes.

Type mismatch
This LotusScript error message means that you're using a value of the wrong type, and the value can't be converted to the right type. For instance, the expression Cint("b12") causes a Type mismatch. The most common cause of this problem, even for experienced developers, is the use of NotesDocument.Fieldname to refer to a field value in a NotesDocument without specifying an array index. For instance:

```
memo.Subject = "Your request, number " & doc.RequestID & ", has been received."
```

The expression doc.RequestID returns an array. The concatenate operator & cannot work on an array. You should instead write:

```
memo.Subject = "Your request, number " & doc.RequestID(0) & ", has been received."
```

Object variable not set
The variable whose name comes before a period (.) has no value. For instance, if the error occurs on this line:

```
Set doc = byDateView.GetFirstDocument( )
```

the problem is that the variable byDateView has the value Nothing. The problem in your code isn't really on this line, it's in your previous code, which failed to assign a value to byDateView. For instance, perhaps you had an earlier statement:

```
Set byDateView = db.GetView("AllByDate")
```

If there is no such view as AllByDate or if you can't find it for some reason (for example, because the ID running the script doesn't have access to the view), this statement sets byDateView to Nothing, setting you up for the later error. Insert code to test for a Nothing value and display a more meaningful message, for example:

```
Set byDateView = db.GetView("AllByDate")
```

© Copyright IBM

```
If byDateView Is Nothing Then
    Messagebox |View “AllByDate” was not found.|…
    Exit Sub
End If
Set doc = byDateView.GetFirstDocument( )
```

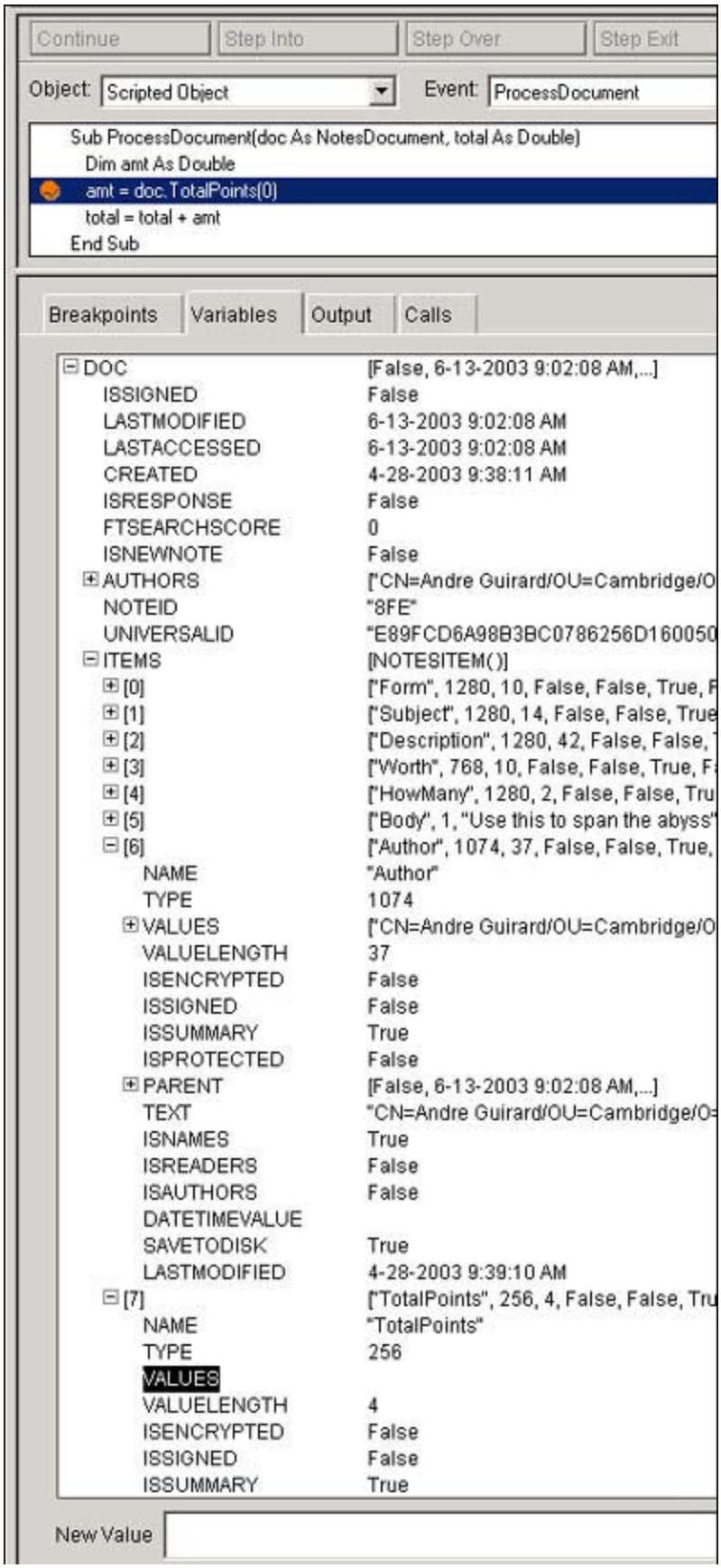
Variant does not contain a container

A container is a datatype that contains other data, such as an array, list, or OLE data collection object. This LotusScript error complains that you used an expression such as variablename(index) where variablename is not an array or list. To get this error, variablename would have to be declared as a Variant (or not declared) because for variables of a declared type, the compiler would catch this problem. This can occur if your code assumes that a particular expression always has an array value when, in some cases, it may not. These are the situations where this error is especially common:

- Code refers to doc.Columnvalues(n)(0). The values of the Columnvalues array may be arrays themselves, but sometimes they are not.
- Code refers to doc. Fieldname (0) and the field value is of a type whose value is not returned as an array, for instance, a rich text field or error value.

To see an example of this in the sample database, turn on debug mode, highlight the crystal wand document, and run the totals \ 1. original action. This document's TotalPoints field has an error value stored in it caused by an incorrect formula on the form:

© Copyright IBM



© Copyright IBM

We've highlighted the line near the bottom where the value of the TotalPoints field would normally appear. Compare that with the Author field, which has a single-element array to store its text value.

The functions Isarray, Islist, Isempty, and Isnull are useful in "bulletproofing" your code against this type of error. For instance, in the previous example, we could have written:

```
If Isarray(doc.TotalPoints) Then
        amt = doc.TotalPoints(0)
    Else
        amt = 0
    End If
```

Of course, whether or not the effort of bulletproofing is necessary depends on your estimate of the likelihood of receiving bullets. Your form design can affect this, as we'll see in Part 2 of this article series.

Entry not found in index or view's index not found
Returned by macro language when you're doing an @DbLookup, this error says that the key you're searching for was not found. If you're certain the key does appear in the view, check the following:

- Does the view contain a sorted column, and is the first sorted column the one that contains the value you're looking up?
- Is the datatype of your key the same as the datatype of the column? For example, if the column is sorted by date you must specify a date value as a key, not a string containing a date in text form.
- Is the lookup value an exact match for the column value? To use dates as an example again, if you use @Today as your key value, and the sorted column contains the formula @Created, you will not get a match because the column contains a time and your key does not. Change the column formula to @Date(@Created) to find matching documents.
- If the sorting column is not categorized and is not set to sort multivalues onto separate rows, then you can only match the first entry of a multivalued column. In other words, if one row contains the two values "Ogre" : "Troll", a search for Ogre will succeed, but Troll will fail.

If you supply multiple keys to @DbLookup, you get this error only if *none* of the keys match the first sorted column. If one or more keys match, @DbLookup returns the values from the matching rows and ignores the non-matching keys.

No Resume
This LotusScript error means that you wrote an error trap using On Error, an error occurred and your code trapped it, but there was no instruction in your code what to do after handling the error. For example, suppose you just want to intercept the error and print a more informative message:

```
On Error Goto Oops
    ...
    Exit Sub
Oops:
    Messagebox "Error " & Err & " on line " & Erl & ": " & Error, 0, "Error trap!"
    ' missing instruction here!
End Sub
```

When your code traps an error, it transfers control to the Oops label, executes the Messagebox statement there, and then comes to the End Sub. That's when the No Resume error occurs; LotusScript expects you to end your error trap in one of the following ways:

- A Resume statement (Resume Next or Resume label)
- An Exit Sub or Exit Function statement
- An Error statement, which causes another error to be passed up to the function that called this one
- An End statement (just the word End), which terminates the script

© Copyright IBM

Notes Error – field didn't pass validation formula
This LotusScript error occurs when you call NotesUIDocument.Save, Send or Refresh, or NotesDocument.ComputeWithForm, and one of the fields on the form fails validation. Because that field also displays a validation error message, ordinarily you first see the message from an input validation formula's @Failure clause ("You must enter a subject" for example) then the Notes Error... message. This most often happens in a form Querysave event or Action formula, when you call Refresh to make sure all the computed field values are up to date. To prevent your code from displaying a second error message, use On Error to intercept the error and take care of it silently. For instance, in a Querysave event, you may write the following:

```
Sub Querysave(Source As Notesuidocument, Continue As Variant)
    On Error Goto validationFailure
    Call Source.Refresh(False) ' recalculate computed fields
    On Error Goto ErrorTrap
    ' ...
    Exit Sub
validationFailure:
    Continue = False ' don't try to save because that'll just get us the same error
    Exit Sub
ErrorTrap:
    ' display a more informative error message than the Notes default.
    Messagebox "Error " & Err & " line " & Erl & ": " & Error
    Continue = False
    Exit Sub
End Sub
```

Set required on a class instance assignment
When you assign an object value to a variable, even if the variable you're assigning is not explicitly declared as an object, you must use the Set keyword, for instance:

```
Set doc = view.GetFirstDocument( ) ‘ right
```

not

```
doc = view.GetFirstDocument( ) ‘ wrong!
```

Incorrect data type for operator or @Function: xxx expected
This macro language error message occurs when you use a value of a datatype that doesn't work in the context. For example, in the following formula:

```
@TextToTime(ReqDate)
```

you get this error if ReqDate is not a text value. The xxx in the message tells which datatype Notes was looking for there, in this case Text.

Invalid or Nonexistent Document
This is one of the more obscurely worded error messages you're likely to see. The "document" referred to here is usually a form, subform, or page. For instance, if you use @DialogBox and give the name of a form that doesn't exist, you get this error.

Error Creating Product Object
If seen in a Domino 6 server agent, this is generally caused by an attempt to create a Notes UI object, such as NotesUIWorkspace. Because there's no user interface for the server agent, these classes are not available and cause an error if you try to use them. (In Notes 5, if you use UI objects anywhere in your code, you get an error before the agent even starts to run.) This error may also be caused by trying to use OLE automation to communicate with a program that's not installed on the computer running the script.

Unknown LotusScript error

© Copyright IBM

In Notes 5, this is usually caused by an attempt to use a UI object in a server or background agent. (See the *LDD Today* article, "**Troubleshooting agents in Notes/Domino 5 and 6**.") This error happens on loading the agent, before it even starts to run, so an On Error statement will not help you.

Error loading Use or Uselsx module: XXX
In client-side script, this error usually means that there's a Use or Uselsx statement that tries to use a nonexistent script library or LSX library. In R5 server or background agents, the problem may also be that the script library contains a reference to UI objects.

Type mismatch on external name: XXX
The function, subroutine, or variable XXX is defined in a script library. The script library has changed more recently than the code that uses it. The calling code needs to be recompiled. See the previous section "**The error doesn't happen when I debug!**"

Conclusion

In this article, we've examined the basics of debugging LotusScript. We introduced the LotusScript Debugger and discussed several common error situations and what you can do about them. Next month we'll look at macro formulas, dialog boxes, scheduled and Web agents, and how to get errors to return more meaningful information. See you there.

ABOUT THE AUTHOR
Andre Guirard is a member of the **Enterprise Integration team** of IBM Lotus Software, the developers of Lotus Enterprise Integrator (LEI) and other products that let you connect disparate data sources with each other and with Lotus Notes. Andre has made occasional appearances as a speaker at IBM conferences, and his articles have previously appeared in *The View* magazine and elsewhere. This is his first publication in *LDD Today.*

© Copyright IBM