



C H A P T E R 1

Lotus Notes integrated development environment

- 1.1 Chapter objectives 2
- 1.2 Elements of the IDE 2
- 1.3 Chapter summary 8
- 1.4 Chapter review 8

1.1 Chapter objectives

- Familiarize yourself with the script pane
- Get acquainted with the class browser
- Identify default colors and what they represent in the script pane
- Change the colors
- Disable/enable LotusScript debugging
- Set breakpoints
- Watch variables
- Step through a script
- Export a script
- Import a script
- Print a script
- Identify script limitations

The integrated development environment (IDE) included in the Lotus Notes client includes features that developers love, but lacks numerous items that many truly want and/or need. The same environment is used to develop scripts/agents with @Functions/@Commands, predefined Simple Actions, Java, and LotusScript. The selection of a radio button (Simple Action, Function, Script, or Java) determines the type of code being developed. The options in the bottom pane are driven by the selection. If Script (LotusScript) is selected, you are presented with the script pane, event drop-down list, checkbox to enable/disable class browser, and a box at the bottom of the screen that shows errors encountered during development.

1.2 Elements of the IDE

The top box, titled Name, allows the developer to appropriately name his/her agent. It appears only when developing agents (see figure 1.1). The same is true for the *When should this agent run* and *Which document(s) should it act on* boxes. These allow the developer to choose when and on what an agent runs. The Shared check box signals whether this agent is for private use (not checked) or available to all (checked).

The bottom of the IDE is the same no matter what type of script you are creating. It could be an agent, `QueryClose` event on a form, or a database script. It is commonly called the script pane (figure 1.2). The very bottom of the window is where errors appear during your development. It is a drop-down list, so there can be more than one error.

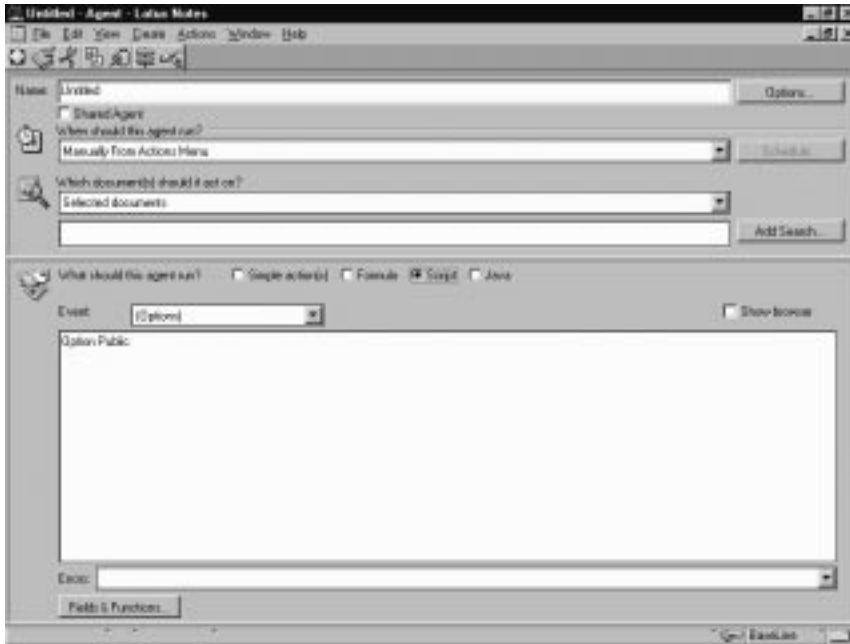


Figure 1.1 Notes 4.6 IDE

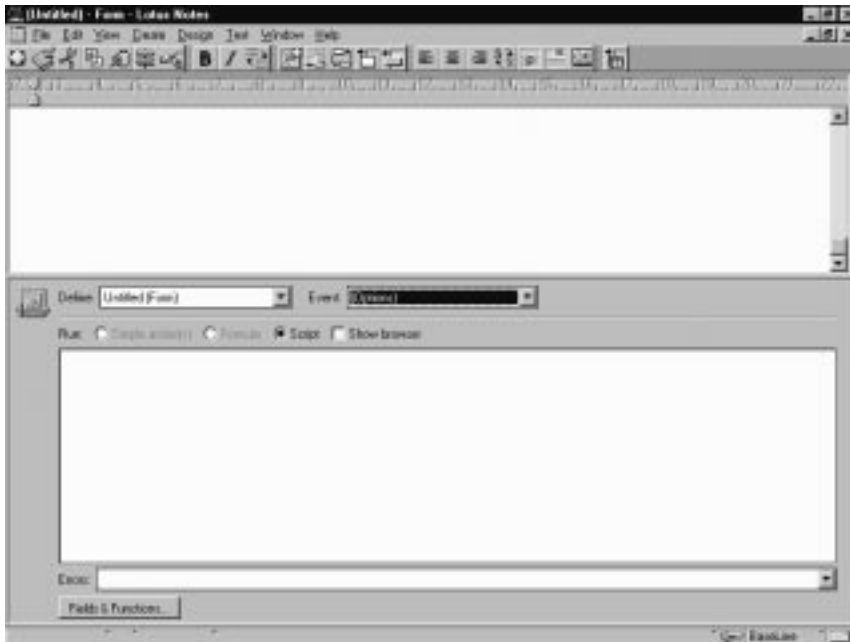


Figure 1.2 The script pane as seen when developing a form



Figure 1.3
Design Pane design properties

1.2.1 Colors

The colors used in the IDE default to black for identifiers (items created by the user). Reserved LotusScript keywords are blue, comments are green, directives (%Include) and errors are red. Right-click the script pane and the properties for the design pane appear (figure 1.3). Colors, font size, and face can be selected if changes are desired.

1.2.2 LotusScript Help

The browser checkbox (figure 1.4) allows the developer to quickly look up syntax for the LotusScript language. Select the check box and the browser pane appears. It works just like a view with twisties that expand/collapse the choices. The drop-down list, which drives the browser pane, is just above the browser pane and lets you view different sets of items. You can select All, Notes Classes, Notes Constants, Notes Subs and Functions, Notes Variables, and OLE Classes.

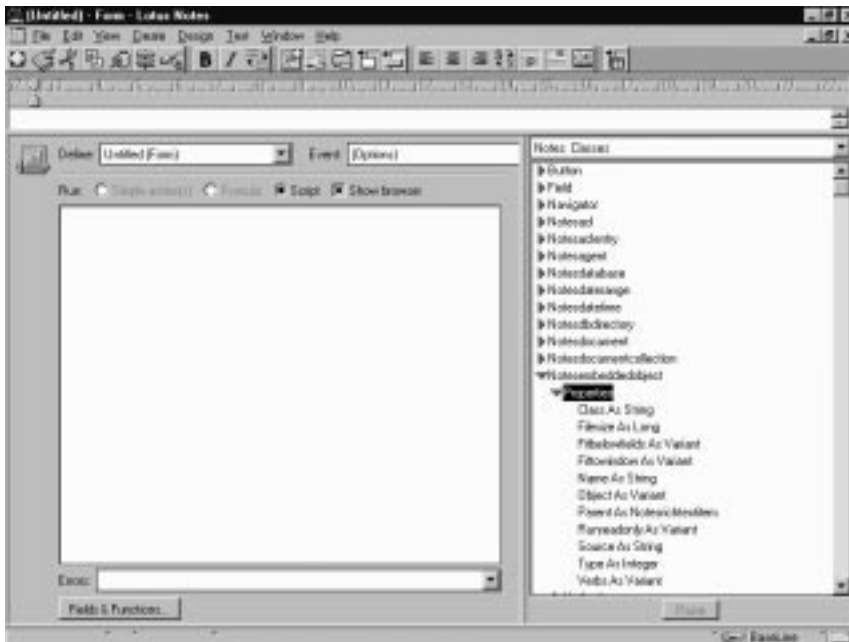


Figure 1.4 Browser pane

Once an item has been selected (highlighted) in the browser, F1 opens the Notes Help database to the topic highlighted (if Notes Help is installed). Double-clicking on an item places it into the design pane.

1.2.3 Errors

The environment does not allow a script to be saved until all syntax errors have been resolved. Syntax errors are comprised of improper use of any aspect of the language including the misspelling of statements, missing parentheses, or illegal comparisons. Remember, errors appear in red in the script pane. The error drop-down list (just below the script pane) lists all errors in the order encountered and describes each. Clicking the down arrow expands the drop-down list to show additional errors (if there is more than one). The format of the error box is:

```
<Object Name> : <Section/Event> : <Line Number> : <Error Message>
```

An error in the `Initialize` event could appear like this:

Example 2.1

```
LSTest : Initialize : 3: Unexpected: Chr; Expected: End-of-statement; _  
      Operator; ,
```

Explanation

There was an error in line 3 of the script `LSTest` in the `Initialize` event. The problem line of code looked like this:

```
MessageBox "Hello" Chr(10) & Chr(13) & "How are you?"
```

The line is missing a concatenation character (`&`) after “Hello”, so the script didn’t know what to do. Adding the proper character (`&`) fixes this error.

The object name corresponds to the section/event that contains the script. Section is the section of code containing the error (`Initialize`, `Declarations`, `Click`, etc.). The error number is an integer that corresponds to the error description. The code causing the problem shows where the error occurred in the highlighted (in red) line. The error box is not always helpful, because the description can be a little arcane. It does show where the error occurred, and double-clicking on the error in the error box highlights the questionable line in the design pane. Logical or program design errors are caught during the testing and/or debugging phase.

1.2.4 Debugging

The environment allows the developer to debug a script by stepping through its execution, watching variables, and setting breakpoints. Debugging is enabled/disabled from the File drop-down menu in the Lotus Notes client (File | Tools | Debug LotusScript). Once this option is selected, every script is opened for the user to watch its execution.

The debugging option allows the user to step through a script (move through the program one line at a time), step over a call to subroutine/function (the script will not step into and through the code in the function or subroutine), stop execution, or continue, which runs the script to the end or any breakpoints that are encountered. *Breakpoints* are points in a script where a developer wants the script to stop execution until he/she says continue. Breakpoints are set by double-clicking on a line of code. Once it has been double-clicked, a red ball/circle appears to the left of the line. The line can be double-clicked again to turn it off, at which time the red ball/circle appears with a line through it. Once breakpoints are set, they appear in the breakpoint tab in the debugging mode of the design pane. It shows all breakpoints set with line number and where it is defined (subroutine/function). Also, breakpoint functionality can be achieved with a `STOP` statement.

Breakpoints, as I said earlier, are points in a script where execution stops. The developer can set breakpoints, click the Continue button in debugging mode, and the script will run until a breakpoint or an error occurs. Once it has stopped at a breakpoint, the developer can step through the rest of the code, select the Continue button, or set more breakpoints and continue. Breakpoints can be set only on code that is executed. They cannot be set on comments or `Dim` statements.

Watching variable activity (figure 1.5) is another requirement for developers. We need to watch how objects/variables are manipulated during the execution of a script.

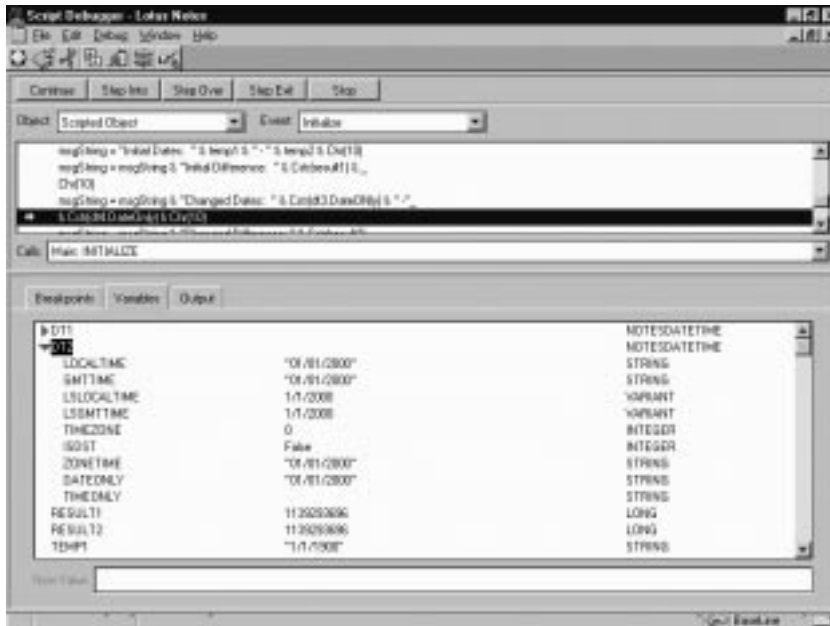


Figure 1.5 Watching variables while debugging a script

The variables tab in the debugging mode of the design pane shows all variables that are defined within the scope of the current object and their values, if any.

The last tab is labeled output. It receives any output from `Print` statements. This allows the developer to place strategic `Print` statements (for debugging) within a script and this pane will show their output.

1.2.5 Importing/Exporting



Figure 1.6 Script pane pop-up

Code can be exported from the script pane to a text file. Also, code can be imported into the environment from a text file. Right-click anywhere within the script pane, and a pop-up menu (figure 1.6) appears with the import and export choices.

Selecting Import presents the user with a file dialog box to select the file to import. Once that has been selected, the file is brought into the environment. If there are any conflicts between code in the import file and the current script (duplicate objects, methods, events, etc.), the user is presented with the Import Option (figure 1.7) dialog box.

This box describes the conflict and allows the user to replace this occurrence (Yes), all occurrences (Yes to All), skip the conflict (No), or stop the import process (Stop). Once the process is completed, the imported code appears in the script pane. It is not final until the script has been saved.

Export is the reverse of the import process. Select Export from the Script pane pop-up. Once again, a file dialog box is presented that allows the user to signify where to save the exported file and to give it a name. The standard extension for text files containing LotusScript code is `.lss`, but any extension can be used. Make sure you remember it. Once the path and name have been specified, the option to export the Current Object, Current Section, or All objects is presented in a `MessageBox` (figure 1.8).

The Current Object option exports everything associated with the currently open section of code. For example, if an action button's click method is currently open and the current object is exported, everything associated with the action button is exported. This would include the Options, Declarations, Initialize, Terminate, and Click sections along with any user-defined subroutines and/or functions.



Figure 1.7 Import options



Figure 1.8 Export options

The Current Section option exports what is currently open in the script pane. If the same action button's click method is currently open, the click section will be the only part of the script that is exported.

The All Objects selection exports everything associated with the currently open object. The object can be a script library, form, agent, subform, view, or database script. All sections and user-defined function/subroutines are exported.

1.2.6 Printing

The Export option is used to overcome a weakness in the development environment: lack of support for printing code. Developers love to get a printout of their code so they can trace the logic; besides, staring at a computer screen is very tiring on the eyes. Once the code has been exported to a text file, it can be opened and printed from any text editor such as the MS-DOS-based editor, Windows Notepad, or any word processing software.

1.2.7 Limitations

In addition to printing, the development environment has many limitations that must be recognized by the developer. As previously stated, no direct printing of scripts is supported. Also, there is no easy way to look at all of the scripts/code included in an application. Domino/Notes Release 5.0 will facilitate this through the design synopsis. The size of a script and arrays is limited to 64K. The use of script libraries is sluggish because they must be loaded when they are used.

1.3 Chapter summary

The development environment included in the Lotus Notes client is not the most full-featured development environment, but it has steadily improved with each incremental release of Domino/Notes. Many of the limitations discussed are being addressed in Domino/Notes 5.0, including support for the development of Java code.

1.4 Chapter review

- The script pane is used to develop Simple Actions, Formulas, Scripts, and Java code.
- A class pane is included in the development environment for quick look up of LotusScript syntax.

- LotusScript code can be debugged by enabling the Debug LotusScript option from the File | Tools menu. It is disabled the same way.
- Debugging gives the options of stepping through code, stepping over subroutines/ functions, or stopping execution.
- Breakpoints can be set during debugging by double-clicking on a line of code.
- The debugging pane allows the watching of variables through the variable tab.
- Exporting/importing script facilitates code reuse and printing.
- Syntax errors appear in the error box just below the script pane during code design.
- Script size is limited to 64K.