Computers & the Humanities 281

Debugging in Revolution

Often when we get involved in complex scripting, some of the handlers don't function quite the way we anticipate. Sometimes they don't function at all. At times like this it is difficult to determine exactly where the problem is located. It is therefore useful to have means to somehow check the handlers as they execute in order to determine the problem and solve it.

Editing Errors

Most of the errors we encounter are a result of mistyping on our part. The first line of defense against our own incompetency is the **script editor** itself. The script editor possesses several tools to help prevent a multitude of potential problems:

- Automatically completes `end` statement of handler
- Indents lines within handlers, repeat loops, if statements, etc.
- Adjusts statements when you press tab key.
- Shows unmatched end statements by not finishing the indent/outdent pattern.

We can also use the script editor to comment out questionable lines or sections of a handler. This prevents them from executing and allows us to isolate the problem.

Syntax Errors

There are also errors due to something wrong with the syntax we've input: commands, key words, labels, etc. In working with handlers, Revolution first compiles commands when you click "Apply" to register your changes. During compilation the entire script is analyzed. If a compile error is found when a script is being compiled, the entire script is unavailable for execution until the error is corrected and the script is re-compiled. Many syntax errors show up in the compile stage. Revolution then displays an error palette explaining what it *thinks* is wrong. You have already encountered this several times in working on your individual stacks. Close will get rid of the error palette, and Script opens the script editor and hilites the offending line. Don't always trust an error message as it may only reflect the effects of the error, not the cause.

Script Debug Mode

This feature is accessed from the **Development** menu and opens up the debug environment. Here we have access to the Script Editor and debug tools. We choose the object whose script we want to debug and then choose the handler we want to debug.

The handler will run completely through all its statements unless we create a break point. This is accomplished by clicking anywhere in the handler to indicate where we would like to start to watch the handler execute its commands discretely. The line turns red to indicate that it's a break point. Then we click the Send Message button. The handler then stops at the break point.

At this point we can step through the handler a line at a time with the Step Over button: keyboard shortcut Return/Enter. The field below shows all variables in the handler and their contents as the handler executes. This enables us to spot invalid values or other problems as the variables change, etc. The run button allows the handler to go on. The Trace button does an automated step from command to command with a delay between. We set the amount of delay with the Trace Delay property.

Logic Errors

These types of errors are due to something wrong with the logic of the handler. The script runs without error messages (so there is not a syntax problem), but the results are not quite right or what we were expecting. Some are quite obvious: forgetting to initialize a variable so results get added to previous run.

As you probably noticed in your own efforts, we often introduce logic errors when we add new features to an application. Consequently it pays to think through the logic each line of new code that we introduce. This often saves a lot of time debugging the logic later.

Since logic errors don't generate an error message, usually the only way to solve them is to employ break points, as described above.

Course Schedule
Main Page