W imp Topics - Debugging with the

Alan Wrigley describes some debugging techniques with the help of routines by

Recent articles in RISC User have looked at the subjects of task windows (6:6) and PipeFS (6:8). This month s Wimp Topics will bring these two useful features of RISC OS 3 together to show how they can be used to provide a handy method of debugging Basic Wimp programs. Information can be sent from a program under development, while it is running, to a pipe file and thence to a second program running in a task window, which displays the information on the screen. As anyone who has ever written a multi-tasking program knows only too well, debugging can be an enormous problem. When something goes wrong on the Desktop, you can t just stop everything and examine the state of your variables. And as for tracing the progress of your program using Basic s TRACE command, you might as well forget it since in a multi-tasking environment there is nowhere for the output to appear.

Enter the task window. As explained in Using the RISC OS 3 Task Window (RISC User 6:6), it is possible to create a Basic environment on the Desktop which multi-tasks with other applications. Couple this with the facility to use PipeFS to transfer data from one application to another (see PipeFS and Filer_Run, RISC User 6:8), and you have all the tools you need to construct a versatile debugging system. You can print out variable values in the task window, for example, or even run a redirected trace (i.e. display the line numbers currently being executed) using the extended TRACE command in the RISC OS 3 version of Basic. This provides the additional syntax:

```
TRACE TO <string>
```

which directs the trace to a stream defined in the parameter string. This could be a file, a device such as Printer: (which would give a running printout of the trace), or most usefully for our purposes, a pipe. If this pipe is then read by a second program running in a task window, the trace output will appear on the screen while the main program is running.

Once a stream has been opened in this way, all trace output will be redirected to that stream. TRACE CLOSE should be used to close the stream when you have finished with it, while TRACE used as a function returns the handle of the stream.

DEBUGGING

First of all we need a short program running in a task window, which can take and display any output sent to the pipe. This is easily achieved by opening a task window (by pressing Ctrl-F12), and running the following program:

```
10 file%=OPENIN"Pipe:$.DebugFile"
20 IF file%=0 PRINT "Error - Pipefile
doesn t exist":END
30 WHILE NOT EOF#file%
40 PRINT GET$#file%
50 ENDWHILE
60 CLOSE#file%
70 PRINT "Pipefile closed"
```

This program must run continuously in the task window while you are debugging the main program. The process of opening the task window and running the debugging program can also be automated, as will be described later.

Now we need to arrange for the program we are debugging to send the required output to the pipe. The pipe file can be opened during the initialisation procedure of the task under scrutiny. If you are going to use the pipe for trace output, you should open it as follows:

```
TRACE TO "Pipe:$.DebugFile"
```

The filename can of course be anything you choose, but it must match the filename which is being opened in the first line of the debug program above. If you open the pipe file in this way, any output from the TRACE command, for example following a TRACE ON at some point in the program, will be displayed in the task window. However, you can also output any other data to the task window provided it is in ASCII format, by using BPUT# to write to the stream you have opened, as in the following examples:

```
BPUT#TRACE,"About to call Wimp_Poll"
BPUT#TRACE,"var%="+STR$var%
```

The first example will result in a string being printed in the task window indicating which point in the program has been reached, while the second example prints a variable name (in this case var%) together with its value.

If you are not using trace output, you can simply open the pipe file with:

```
file%=OPENOUT"Pipe:$.DebugFile"
```

and write to it with:

```
BPUT#file%,<string>
```

ADDING THE DEBUG FACILITY TO YOUR OWN PROGRAMS

I mentioned earlier that it is possible to automate the process of opening the task window debugger. First of all, type in the debug program given earlier and save a copy of it as Debug inside the application directory of the program to be debugged. Create an Obey file called !RunDebug with the following lines:

```
WimpSlot -min 32K -max 32K
Run <Obey$Dir>.Debug
```

and save it to the same directory. Now add a new procedure to your main program called PROCinitdebug, and call it immediately after the main initialisation procedure (altering the line numbering to suit and also the application name in line 1030 if necessary):

```
1000 DEF PROCinitdebug
1010 TRACE TO "Pipe:$.DebugFile"
1020 file%=TRACE
1030 BPUT#file%,"Traced output from MyA
pp"
1040 BPUT#file%,""
1050 SYS "OS_ReadVarVal","Obey$Dir",blo
ck%,255 ,0,3 TO ,,len%
1060 block%?len%=13:path$=$block%
1070 name$=CHR$34+"Debug output"+CHR$34
1080 com$=CHR$34+"Obey "+path$+".!RunDe
bug"+CHR$34
1090 OSCLI("WimpTask TaskWindow "+com$+
" -name "+name$+" -display -quit")
1100 ENDPROC
```

The procedure assumes you have already dimensioned a block of at least 255 bytes as block%.

Having done this, any TRACE ON command will direct trace output to the task window, and any strings sent to file% using BPUT# will also appear in the task window. When you have finished with the debugger (for example when your application is about to quit, or when you are reporting a fatal error), you should issue a TRACE CLOSE statement.

TRACE REVISITED

It is worth elaborating a little on the TRACE command, since many newcomers to programming may not have fully explored its usefulness. We have seen above how to send the traced output to a file (and thence to a task window for immediate display). The command itself has several forms. TRACE ON switches on simple line number tracing, but you can also use the form TRACE PROC. This will instead display the names of procedures as they are called within the program. To turn any kind of trace off, just use TRACE OFF. The trace can be switched on or off at any point in your program, so you can easily isolate the section that you want to study. Bear in mind when you are tracing line numbers that lines which don t perform any actual processing (e.g. DEF PROC, ENDPROC, REPEAT, UNTIL etc.) are not shown.

Even more valuable in many situations is the ability effectively to single-step your Basic program. By adding the keyword STEP after TRACE, for example:

```
TRACE STEP ON
```

the program waits for a key to be pressed before continuing execution after each trace output item is displayed. However, you should think carefully before using this in a general way with a multi-tasking program, as it will almost certainly affect the whole operation of the Desktop. Multi-tasking depends on well-behaved applications passing control back to Wimp_Poll as soon as possible; if your application has to wait for several keypresses before it gets back to its poll loop, you will have difficulty moving windows around or accessing menus, for example. It is better to use the STEP facility within procedures to trace a route through a single operation, and turn the trace off again at the end of the procedure.

OTHER DEBUGGING