C Notebook: Hints & Queries

HANDLING ERRORS

In C code examples, I have seen at least 3 different ways of handling errors wimpt_complain(), wimpt_noerr() and werr(). Can you explain when to use each.

P.R.Richards

Firstly, it is worth mentioning why the first two calls are used with so many RISC_OSLib functions. Most functions in the library do little more than make a SWI call for you, and in every case, the call is made in the X form. This means that errors are not handled directly by the operating system, nor indeed by the function. Instead most functions return a pointer to an error block if an error occurs, or a Null pointer otherwise.

So, when you call a RISC_OSLib function that returns an error pointer, you must do something about it. If any error at that point in your code will be fatal - for example a vital template might be missing - then the simplest thing to do is to wrap up your function call with wimpt_noerr(). If an error occurs, a message of the following kind will then appear:

ProgName has suffered a fatal internal error
(error message) and must exit immediately.

The program will then quit.

If the error is not so desperate, you can use wimpt_complain() instead. In this case an error box will again be put on screen, but the program will continue from the next line. This means of course that your code must detect the error, and behave accordingly. If there is an error in opening a data file for example, it is no use trying to read data from it in successive lines. Fortunately, wimpt_complain() returns the original os_error * pointer, so you can use something like the following:

```
if (wimpt_complain\
  (sprite_area_save(area,name))==NULL)
{
 -
}
```

Any code within the braces will only be executed if the call to sprite_area_save() returns without error, while if there is an error it will be correctly reported.

The final function that you mention, werr() is simply for customised error reporting. The function will put up an error box containing the message supplied when the call is made.

PREPROCESSOR QUOTE HANDLING

Bob Peterson

The preprocessor in the Acorn compiler can handle the insertion of quoted strings even when they fall next to quotation marks. As a result, the following works fine:

```
#define ICON_SET(w,i,f)
 wimpt_noerr(wimp_set_icon_state(w,i,f,f))

#define ICON_SETALL(w,i,f)
 wimpt_noerr(wimp_set_icon_state(w,i,f,~0))

#define ICON_CLEAR(w,i,f)
 wimpt_noerr(wimp_set_icon_state(w,i,0,f))

#define ICON_TOGGLE(w,i,f)
 wimpt_noerr(wimp_set_icon_state(w,i,f,0))

#define ICON_REDRAW(w,i,f)
 wimpt_noerr(wimp_set_icon_state(w,i,0,0))

#define ICON_DELETE(w,i,f)
 ICON_SET(w,i,wimp_IDELETED | wimp_IREDRAW)

 #define APP_NAME   "Test_App"

 fptr=fopen("<"APP_NAME"$Dir>.datafile"
,"r");
```

LET YOUR MACROS TAKE THE STRAIN

Dave Appleby

It is possible to define sets of macros to reduce typing and make your C code more readable. For example:

OPENING DIRECTORIES WITH fopen()

Lee Calcraft

If you use the ANSI C function fopen() supplied with CLib, it is worth knowing that