TECHNICAL QUERIES

Alan Wrigley answers your queries on Wimpslots, extending memory blocks, retrieving pointer information, and sending text files to Edit from your own

Q Dear Sir

As a newcomer to computing I am very confused by some of the jargon. The manual for a piece of software I acquired recently tells me it needs a Wimp slot of 128K in order to run. What is a Wimp slot, and how do I provide one?

E.S. Morris

A The term 'Wimp slot' or 'Wimpslot' simply means 'a chunk of memory allocated to the task within which it can run in the Wimp environment'. All multi-tasking programs need such an allocation of memory, but normally the user doesn't need to worry about this, since it is all done automatically when the program is run. There is an operating system command, WimpSlot, which allocates the requested amount of memory to the task concerned, and this command is usually issued from the application's !Run file. In the absence of any such command, the contents of the Task Manager's 'Next' slot, or the total available free memory, whichever is the smaller, will be allocated. You can see the contents of the 'Next' slot, and also the slots of all currently running applications, by clicking on the Task Manager's icon in RISC OS 3 or choosing its 'Task display' menu option in RISC OS 2.

The manual you mention is confusing because it seems to imply that you must somehow provide a Wimpslot for the program yourself. If the program has been correctly written, it will automatically request its own slot when run, and the manual is simply trying to warn you that at least 128K must be free in the machine before the program will run. If the requested amount of memory is not available, an error will be generated telling you that the application needs at least nK to run (in this case 128K). You will then have to find the required memory by quitting other applications or reducing the memory allocated to certain areas, as described in the User Guide under the section 'Optimising memory usage'.

There is certainly no excuse for using jargon such as this in a manual. Software manuals should assume that users know nothing about workings of the computer; in this particular case, a sentence such as 'You must ensure that at least 128K of memory is free before running this program' would have been far better.

Q Dear Sir

In Technical Queries (RISC User 5:4) you described various methods of claiming extra memory in order to process documents etc. What is the best way of dealing with a situation where editing the document would cause it to overrun the end of the extra memory?

John Winwick

A The methods described in 5:4 were firstly to use the keyword DIM to dimension a block of memory within the program's workspace; secondly to claim a chunk of memory from the RMA; and thirdly to increase the Wimpslot by calling SWI 'Wimp_SlotSize'.

The first method is not really suitable if you are likely to extend the document beyond the memory originally dimensioned. You cannot increase a dimensioned area directly - you can only dimension another block, which is unlikely to be contiguous with the first. If you use this second block as an extension to the first, you will have to keep pointers so that the program knows how many blocks you have, and where to make the transition from one to the next. If you abandon the first block completely and move the whole document to the second, you cannot free the memory and so it is effectively wasted.

If you are using either of the other methods,

however, the extra memory can readily be extended in a seamless fashion (provided that there is sufficient memory available in the computer). A block of memory claimed from the RMA (using OS_Module 6 as in the example given in 5:4) is under the control of the Heap Manager, and can be extended by using SWI OS_Heap with the following parameters:

R0 = 4 (reason code)
R1 = pointer to heap
R2 = pointer to block
R3 = size change relative to original

Reason code 4 indicates that you want to extend a heap block. The heap in this context is the RMA, so first you have to find where it starts by using OS_ReadDynamicArea . The pointer to the block is the pointer that was returned by OS_Module when you claimed it in the first place, while the size change in R3 is relative to the original size (i.e. a positive increment to expand it, or a negative decrement to shrink it). If the heap manager cannot extend the block in its current position, it will move it and copy the contents. The new block pointer is returned in R2, so you must read this and use it in place of the original value. This implies that all access to the memory block in your program should be relative to the pointer, and this is in any case advisable as you should not normally refer to explicit addresses.

So to expand a block pointed to by extra% from 1000 bytes to 1200 bytes, you would execute

Q following code:

```
SYS "OS_ReadDynamicArea",1 TO rma%
SYS "OS_Heap",4,rma%,extra%,200 TO
,,extra%
```

If you have used Wimp_SlotSize to claim extra memory, all you need to do to extend it is to use the call again in exactly the same way as before, but this time requesting a larger slot. Provided the memory is allocated, the operating system

A will map the extended area so that it appears to your task to be contiguous with the original memory allocation. In other words, whatever slot size your program currently has, the application workspace will always start at &8000 and extend in one block the size of the complete slot.

Dear Sir
I am trying to write a program which moves the pointer across the screen and then reads the window and icon handles underneath it using Wimp_GetPointer Info . The problem is that the pointer position returned by the call is always the position before it was moved, irrespective of where the call is made from. Can you help?

Stuart Porter

Q

Unfortunately the PRM in its description of Wimp_GetPointerInfo doesn t make it sufficiently clear that the data returned by the call relates to the moment at which your program last regained control through Wimp_Poll. In other words, the information is only updated when a Wimp poll event is returned, and any subsequent movements of the pointer will not be taken into account until

A you call Wimp_Poll again (when the data at that point will be available to the next task which is paged in).

The solution to the problem is to claim null events while you are carrying out the task in question. Having moved the pointer, you can then read its new position at the next null event, and carry out whatever actions are needed at that point.

Dear Sir
When a multi-tasking Basic program needs to display a text file, it can issue the command *Run <pathname> to load the file into Edit. However if another text file is to be displayed, a second copy of Edit is installed. Is there a better method which uses the same copy of Edit, and can the Basic program close the first text window?

Keith Vernon

When a Run command is issued for any file, RISC OS performs the run action for that filetype (as specified by the Alias$@RunType variable). In the case of text files, the default action is to load a text editor such as Edit. Any text editor that might already be installed is unaware that this is happening, and so cannot load the file itself. The solution to this problem under RISC OS 2 is to broadcast a