New Technical Notes

Macintosh

Developer Support

MPW Pascal Q&As

Platforms & Tools

Revised by: Developer Support Center October 1992
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

MPW Pascal converts SINGLE numbers to extended

Date Written: 1/22/91
Last reviewed: 1/22/91

Why does MPW Pascal convert SINGLE numbers to extended, create a temporary area for them, and then pass a pointer to those extended numbers on the stack?

—

The conversion of single-precision values to extended is being done to maintain accuracy. It's entirely possible to generate values of extended precision while doing the intermediate math with single-precision values, and the MPW compilers do this conversion to preserve accuracy.

If your application doesn't need the accuracy, you can declare the parameters to be of type LONGINT, and typecast them as necessary within your procedure or function. There's no way to tell the MPW compilers not to do the conversion if the parameters are declared as SINGLE.

MPW 3.2 Paslib.o library and standalone code

Date Written: 7/26/91
Last reviewed: 8/30/91

Linking a standalone code resource with the MPW 3.2 Paslib.o library generates the following link error message:

```
Link Error: Output must go to exactly one segment when using "-rt."
```

The same link works fine if the MPW 3.1 Paslib.o is used. Linking with the new MPW 3.2 Paslib.o causes Pascal modules such as %_MUL4, %_DIV4, %_MOD$, and %_CAT to be stored in a new segment called Paslib 1. However, standalone code resource should just have the one segment.

To link a standalone code resource with the MPW 3.2 Paslib.o library, use the -sn option (like -sn PASLIB=Main) to pull all of the Pascal modules into your Main segment. If your main segment is called something other than Main, you can substitute that name in the -sn option instead of Main.

For more information on segmentation, check out the "Segmentation" section of Chapter 10 of the MPW Reference Manual.

Macintosh toolbox & Pascal procedure parameters

Date Written: 6/6/91
Last reviewed: 10/15/91

The Macintosh toolbox does not support nested procedure pointers as parameters, although Pascal can support them in "native Pascal" usage. Consequently, Pascal requires that functions like the SFPGetFile filter and hook be global functions; they cannot be local to the method calling SFPGetFile nor can they be separate methods of an object calling SFPGetFile. Likewise, variables used by the filter and hook must be global. For further reading, the Macintosh Technical Note "Pascal To C: PROCEDURE Parameters" provides technicalities of this issue.

Pascal Dispose function bugs

Date Written: 5/3/89
Last reviewed: 12/17/90

I am having a hard time getting the Pascal function "Dispose" to work. It doesn't seem to be freeing up memory.

—

Surprisingly, Dispose doesn't work and has NEVER worked! As far as we know, there have been two bugs in the Dispose routine that have been in there from day 1.

The first bug has to do with the interaction between the code that the Pascal compiler creates, and the library routine (%_DISPOSE) it calls to do the actual dirty work. When the Dispose procedure gets compiled, the pointer to the memory block you want to free up and the amount of memory you want to free up are pushed onto the stack. Then a call to %_DISPOSE is made. However, this routine—written in Pascal—is declared to accept the memory pointer as a VAR parameter. This means that it is expecting a pointer to the parameter, and not the parameter itself.

The second bug occurs when trying to free a block of memory that doesn't have any free memory in front of it within its sub-heap. When this occurs, the first block in the heap gets slightly corrupted; your data is still intact, but the header that identifies it gets smashed. The result of this is that the heap routines do not properly detect when the heap is empty and should be disposed of.

These problems are fixed in MPW 3.1. In the meantime, there are a couple of things you might be able to do. The first is use the Memory Manager routines, and not the Pascal heap routines. Second, Mark and Release are bug free (and so is New), so you may try using them. Release will dispose of any additional allocated heap blocks. NOTE: Mark and Release only work if "forDispose" is FALSE.

Booleans and short circuits don't mix
Date Written: 11/17/89
Last reviewed: 12/17/90

I can't get expressions containing Booleans and short circuit in my MPW Pascal program to evaluate correctly. I have to use all of the same type. Is this a bug?

—

Your workaround is correct. Presently this isn't a bug. The MPW documentation states that mixing of short circuits and regular Booleans in the same expression IS NOT supported.

It is in Chapter 6, page 100 of the Pascal 3.0 manual.

MPW 3.2 has new syntax for forward references to objects
Date Written: 4/24/91
Last reviewed: 6/7/91

My application, which has several units and objects, compiles under MPW 3.1 but not under MPW 3.2. Do forward references to objects work differently with MPW 3.2?

—

MPW 3.2 has a new syntax for forward references to objects. Objects must be declared as externals, as follows:

```
TYPE
 TObjB = OBJECT; external;      { MPW 3.2 requires this }
 TObjA = OBJECT(TObject)
   fFwdRef:    TObjB;

   {methods}
   END;
```

Note that by default the Pascal compiler includes information such as USES in its symbol table resources, so simply using the correct USES and external declarations may not be sufficient; you may find it necessary to invoke the Pascal compiler with the -rebuild option to force it to

reconstruct its symbol table resources from scratch.

Why Pascal objects in HyperCard XCMDs is a challenge
Date Written: 4/24/91
Last reviewed: 6/14/91

Is possible to declare MPW Pascal objects in a HyperCard XCMD?

—

It's possible but very difficult to use Pascal objects in an XCMD because Object Pascal uses the jump table in its method dispatch mechanism. XCMDs and other standalone code resources don't have a jump table. See the Macintosh Technical Note "Stand-Alone Code, *ad nauseam*" for more information about this.