

New Technical Notes

Macintosh



®

Developer Support

SADE Q&As

Platforms & Tools

M.PT.SADE.Q&As

Revised by: Developer Support Center

November 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.

SADE needs locked code resources

Date Written: 12/5/90

Last reviewed: 1/16/91

I am having difficulty in using SADE on an Omnis 5 external. I have previously succeeded in using SADE on stand-alone code resources called from a standard C application, and with MPW tools.

I place the external code resource in the Omnis 5 application resource fork, when the external executes, SADE yields:

```
Internal fatal error! Please kill the target process! at µEXTERNAL+$0010 in
"Omnis 5"
EXTERNAL
+0010 001164C8 4E45          •*TRAP          #5
```

Note that the TRAP is at the breakpoint address, which makes sense. Somehow SADE is getting confused and failing to recognize the TRAP as the breakpoint. What's happening here?

It seems like that TRAP statement you got was something from outer space because the trap-dispatching with Macintosh traps is handled by A-line traps whose opcodes start with hex A. The TRAP opcode is never used as with other operating systems, such as A/UX.

The most likely reason for this behavior is that your code resources were moved around, and SADE picked up whatever it found in that memory location—after that the CODE had moved to another place. In order for SADE to get access to the real CODE resources you need to lock them.

SADE “Cannot determine break address...” error

Date Written: 4/9/91

Last reviewed: 6/17/91

What does the SADE error “Cannot determine break address: Could not determine address from available source information” mean and how do I fix it?

Two common problems are indicated by with SADE’s “Cannot determine break address: Could not determine address from available source information” error message are:

1. not all the source modules have been compiled with -sym on
2. one of the tools that produces .SYM file information does not conform to the latest format definition

For the first problem, put breakpoints everywhere except for where the error occurs, and make sure all source code is compiled with the “-sym on” option.

If you can’t set breakpoints, check for the second problem. The MPW 3.2 tools (compilers) generate the symbolic information that the Linker puts together for one single file, so eventually when you use an old tool it will not produce good symbolic information for the final .SYM file.

Check the latest Essentials•Tools•Objects (ETO) CD for the current SADE and MPW tools.

SADE “Invalid A6...” error message

Date Written: 5/9/91

Last reviewed: 8/1/91

What’s the cause of the SADE error message, “Invalid A6, A6 must be greater than or equal to SP, and within valid stack range”?

The “Invalid A6, A6 must be greater than or equal to SP, and within valid stack range” error message that you are receiving stems from a feature that was placed into SADE 1.3a5. There is a simple fix. You just need to change the WantStackWindow variable located in the SadeStartup file from 1 to 0. (The WantStackWindow variable is about 30 lines down from the top of the SadeStartup file.)

When a user targets an application, SADE launches the application and runs until it hits an internally placed break point. At this point in time, the program is given a name and A6 is basically junk. The WantStackWindow basically tells SADE to check to see if A6 is valid or

not as the program is running. When SADE hits the break point, A6 is junk, so the user will receive this error message. As the user continues to run the application, A6 becomes valid and the “error” no longer exists.

To sum it up, when WantStackWindow is set to 1, and your program is not running you will receive this error. When WantStackWindow is set to 1, and your program is running, you will not receive this error. By setting WantStackWindow to 0, it doesn't check to see if A6 is valid; therefore, you will not receive the error message.

FPU instructions and SADE 1.3 or SourceBug on Quadra

Date Written: 11/4/91

Last reviewed: 11/27/91

Are there any known problems using SADE or SourceBug on a Quadra? My SourceBug crashes, and I get an "Unimplemented F-Line instruction" alert when stepping over a trap call..

—

The problem occurs when you execute an FPU instruction that is not implemented on the 68040, but is emulated instead in ROM. Normally, the 68040 generates an F-Line exception, which is intercepted by the numerics code and emulated. However, when the Process Manager launches SourceBug or SADE, this exception vector is patched, and F-Line exceptions are passed to the debugger instead. Therefore, when SourceBug or SADE is running, none of these transcendental FPU instructions can be emulated.

The problem will be fixed for SADE 1.3.1 and SourceBug 1.0b6.

Debugging an external CODE resource with SADE

Date Written: 5/3/89

Last reviewed: 12/17/90

How do I debug an external 'CODE' resource with SADE? I can't get it to work.

—

Lets say that the application which calls your code resource is in HD:AppDir:, and that the source code for your code resource is in HD:SrcDir:. To debug the code resource, enter the following commands in SADE:

```
Directory 'HD:AppDir:'
SourcePath 'HD:SrcDir:'
Target 'AppName' Using 'HD:SrcDir:ResourceName.SYM'
Break ResourceEntryPtName.(1)
Launch 'AppName'
```

That should do it. This is basically the same technique that is used to debug MPW tools.

There are a couple of restrictions on this technique:

1. Your code resource must be in the applications resource fork;
2. HyperCard XCMDs and XFCNs cannot be debugged in SADE.

How MPW and SADE 1.0 work together

Date Written: 5/3/89

Last reviewed: 12/17/90

Why is using SADE 1.0 to step through an MPW Tool so slow?

When MPW launches a Tool, it acts much like Finder launching an application. The MPW shell now becomes the operating system for the Tool. But unlike Finder, MPW has no hooks for a debugger like SADE. Because of this, SADE requires a huge amount of overhead for each step operation it performs. For each step operation, SADE must:

1. Save its low memory space so that MPW's and the Tool's may be swapped in;
2. Restore MPW's as well as the Tool's low memory;
3. Perform the necessary debugging operation (step);
4. (Block)Move the newly updated low memory to an area accessible by SADE (so that SADE's low memory may be restored). Once SADE is restored, it will want to report on the new low memory state. This is why low memory had to be copied to an area accessible by SADE;
5. Swap SADE's low memory space back in so that SADE can operate.

Why using SADE 1.0 to look at low memory is so slow

Date Written: 5/3/89

Last reviewed: 12/17/90

Why is using SADE 1.0 to look at low memory so slow?

Since SADE needs to use low memory just like any other application, there is some overhead involved when using SADE to view low memory. Here is what SADE has to do to view the target application's low memory:

1. Save its low memory space so that target application's may be swapped in;
2. Restore the target's low memory;
3. (Block)Move the newly updated low memory to an area accessible by SADE (so that SADE's low memory may be restored). Once SADE is restored, it can report on the new low memory state. This is why low memory had to be copied to an area accessible by SADE;
4. Swap SADE's low-memory space back in so that SADE can now analyze the low memory it copied in step #3 above.

Can I use SADE to debug my XCMD/XFCN?

Date Written: 1/1/90

Last reviewed: 1/1/90

Can I use SADE to debug my XCMD/XFCN?

—

No. SADE currently doesn't support debugging HyperCard external commands.

SADE and MacApp CString type conflict

Date Written: 8/19/92

Last reviewed: 10/11/92

I'm having problems printing a CString from memory from SADE. For example,

```
printf ("%s\n", ^CString($B64A12)^)
```

does not print the string stored at \$B64A12. What do I need to do?

The problem you encountered with getting SADE to cast a value to CString within a printf involves a type conflict between MacApp's declaration of a type named "CString" and SADE's built-in type named "CString". SADE will first match the type against declarations from the application, in a case sensitive manner. Only if it fails to match will it then match against its built-in types in a case insensitive manner. So, by casting to CSTRING instead of CString you will get the result you desired.

SADE requires 4-byte value for arithmetic operations

Date Written: 8/19/92

Last reviewed: 10/11/92

The following SADE code gives me an "### Invalid expression operand type" on line 05, unless I include line 03. Could you tell me why?

```
01 define aTemp, bTemp, cTemp, dTemp, eTemp, fTemp, gTemp;
02 aTemp := ^short(fExpandingView)^ ^ # Get Class ID
03 aTemp := aTemp * 1                 # Multiple by sizeof(short) NOT USED
04 bTemp := ^short(pClassTable)      # Get pClassTable address
05 cTemp := aTemp + bTemp             # Calculate ptr to class's jump table offset
```

The reason line 03 makes it work is that the constant one is treated as a 4-byte value. Multiplying aTemp by 1 and assigning back to aTemp therefore results in aTemp being converted to a 4-byte value. bTemp being a pointer type, SADE requires a 4-byte value as the value to be added to it.