SADE 1.3.3
Release Notes

Changes Since SADE 1.3.2

- SADE would occasionally fail while defining a certain number of user strings on systems with greater than 16MB of RAM. This has now been fixed.
- Under certain low memory conditions, SADE may not have been able to retrieve correct type information. This has now been fixed.

Changes Since SADE 1.3.1

- SADE was handling trace exceptions generated by MacsBug. While running SADE in the background, if you attempted to set an address break from MacsBug and then execute "go," SADE would take control. This has now been fixed.

Changes Since SADE 1.3

- `BreakIf` was failing under some situations. This has now been fixed.
- SADE was writing to NIL under low-memory conditions. This has now been fixed.
- Increased "preferred" memory partition size to 1.5 megabytes to be able to handle MacApp without running out of memory.
- Stepping after hitting a one-instruction source statement would stop two source

instructions later instead of at the next instruction.

- SADE now displays the correct CCR value in the registers window.
- You can now set the values of the SR/CCR register.
- SADE no longer uses the MPW Shell's small icon for the Notification Manager.
- SADE now works properly under VM on 040 machines.
- The `Extras` menu item `Show Self` has been separated into two items:
  Show **this (Command-4), and
  Show *this.

  `Show **this` functions exactly as `Show Self` did in SADE 1.3 Final. However, for C++ programmers who are not using MacApp, the `Show *this` menu item is more useful. Therefore, we have included both.

Changes Since SADE 1.3B2

- CFront generates incorrect symbols for any PascalObject with a forward declaration. In the member functions, CFront says that `this` is a `*TType`, then casts all references through `this` to `(**(TType**)this)`. To alleviate this problem some hacks were put in SADE. If your program contains the resource `%__MethTable` (as is the case with MacApp programs), and if you have a type which the symbol file specifies to be a pointer to a PascalObject (a record whose first element is a `__vtable`), SADE will make itself believe that the type really is a pointer to a pointer to that object. As a consequence of this fix, if you singly dereference a handle to a PascalObject, the pointer value displayed will be correct, but the type displayed will be a handle.

  The newly provided menu item `Twice Dereference Selection (Command-5),` explained below, makes displaying objects through handles much easier.

- A new menu called `Extras` was added. Four of the menu items under this menu make displaying objects easier, and the last one makes changing the value of objects easier.

  • `Show Value In Context (Command-2)` and

  • `Show Dereferenced Value In Context (Command-3)`

  The old style Show Value menu items have no context, they simply follow scoping rules. Thus, if you select `fNext`, it will try to show you an automatic `fNext`, a global `fNext, this^.fNext` or `this^^.fNext`.

  These new menu items put the value of the selected variable in a separate window. For subsequent selections, the name of the window provides the context. E.g. if you select

`fNext` in your source file and do a `Command-2`, you will most likely get `this^^.fNext`. However, if you had previously displayed a variable `newWindow`, and select `fNext` in that window, Command-2 will show you `newWindow.fNext`.

Since these windows are really files, when the length of the name of the thing to be displayed exceeds 32 characters, it will be displayed in the Value window instead. These menus do not work if your selection has a colon (:) in it. These commands are slightly slower than the regular show value menu items, but having a separate window for each variable far outweighs the slight speed penalty.

• `Set Value (Command-=)`
Changing variables has always been difficult in SADE . The problem is that SADE shows you structured values, but you probably need to change only a field in a struc - tured object. It would be nice to type over a displayed value in a window to the value you want. Unfortunately, it is not that simple.

If you are trying to change a simple variable, simply highlight its name and select the menu item `Set Value`. From the displayed dialog box, you can verify that SADE is going to change the variable you wanted. If the proper object is selected, type in the value you want. You can type in expressions instead of simple numbers. You can also type in the name of a complex object whose type matches that of the left hand side.

In implementing this operation, SADE first evaluates the left hand side, then evaluates the expression you typed in, and may display error messages if it finds any problems. After that SADE will try the actual assignment; that too may fail because of incompati- ble types for the left and right hand sides.

To change a field buried in a structure, you have to peel the outer levels. E.g. if you want to change

```
FirstLevel.SecondLevel->Member
```

do a `Command-2` after selecting FirstLevel. This gives you a window named `FirstLevel`. In that window, select the word `SecondLevel` and do a `Command-3`. Now you have a window named `FirstLevel.SecondLevel^`. In this window select the text `Member` and select Set Value or `Command-=`.

Two new menu commands were added to the `Extras` menu to make debugging MacApp applications written in C++ easier.

• The menu item `Show Self (Command-4)` displays `**this` in a separate window. You must be stopped inside a MacApp method for this menu to work.

C++ programmers who are not using MacApp may wish to edit the SADE routine `__ShowSelf__` in the file SadeStartup file so that this menu displays `*this` instead.

• The menu item `Twice Dereference Selection (Command-5)` makes displaying objects through a handle to them a one step process. The value is displayed in a sepa-

rate window, as in the previous commands.

This menu is implemented in the file SadeUserStartup•Extras. You can remove this file without affecting SADE's behavior in any other way.

- The order in which SADE startup files are executed is:

  • SadeStartup

  • SadeUserStartup

  • all files whose names starts with SadeUserStartup•. These files are executed in alpha - betical order.

  If you want to override anything defined in the file SadeUserStartup•Extras, create a file named something like SadeUserStartup•zzz, so that it is alphabetically later.

- SADE will place all temporary files in the sub-folder called `SADE Scratch` inside the SADE launch directory. If the folder does not exist, SADE will create it. During exit, SADE will delete all SADE created files from this directory.

  One side effect of this behavior is that SADE will no longer remember the position of the temporary windows like Values. If you prefer the old behavior, override the scratch directory by adding the following line to your `SadeUserStartup` file:

  ```
  __ScratchDir__ := SADEDir
  ```

- When SADE needs to open a source file, it tries much harder now. The symbol file provides SADE with either a full path name or just the leaf name, depending on the way the file was compiled. In addition, the modification date of the source file when it was last compiled is also provided. Here is the list of actions SADE goes through to find a source file:

  • Check if any of the open files have the same leaf name and matching date. If not,

  • If the symbol table specified a full path name, try the full path to open the file. If not,

  • Try to find a file in the default directory with the same leaf name. If not,

  • Try to find a file in any of the directories in the SourcePath. If not,

  • Check if any of the open files have the same leaf name but a wrong date. If not,

  • Ask the user to navigate the directory structure to find a file with the same leaf name.

  Check the modification date of the file found in any of the above ways, if there is a mismatch, give the user a chance to find a better match. If the user finds a file in a new directory, add the new directory to the source path.

- If you open a source file in a directory which is not in the current source path, and try to set a break point in it, if SADE finds the symbol for a file with that name, SADE will

add the new directory to the source path.

- When SADE displays a record, the name of the record is displayed in addition to its contents. Occasionally SADE may display names like `__032`, these are generated by CFront in situations like anonymous unions or structs.
- When the conditional break table is full, we shift the oldest break point out instead of deleting all old conditional breakpoints. You still get the warning.

Changes Since SADE 1.3B1

- SADE will now set the Sourcepath variable for you. If SADE cannot find the application's source files in the same directory as the symbol file, it will prompt you to find the source file. In addition, while debugging your code, if you step into a procedure defined in a separate source file and that file is in a different directory than those defined by the sourcepath variable, SADE will prompt you to search for that file as well. SADE will add the new directory indicated to the sourcepath.

  Note, however, that SADE does not clear the Sourcepath variable each time you kill your application. This way, you can continue targeting the same application without SADE prompting you each time to locate your source files.
- SC7 is a new script available in the SADE folder. When executed from the worksheet, you may use the procedure `StackCrawl7` or `sc7` defined in the script to show a possible calling chain sequence. StackCrawl7 is similar to the MacsBug command sc7. It is useful for displaying the calling sequence when the built in A6 based stack command does not work.
- MacsBug and ROM symbols will now be displayed in a disassembly. When there is no symbolic information available and your code has been compiled with MacsBug symbols, SADE will display MacsBug information, e.g.:

  ```
  BEQ.S    µINITIALIZE+$00BA   ; 00BE7B22
  ```
- SADE will now display up to the first 20 elements of an array instead of just the first element.

- When displaying the value of a pointer to a string, SADE will always display the pointer value, then the string if it meets the criteria described below.
- C does not have a built-in string type. The following heuristic is employed to determine if a signed or unsigned character array contains a string:

```
// Get past the printable characters
for (i = 1; i < len; i++)
      if (!isgraph(string[i]) && !isspace(string[i]))
            break;

// Is this a CStr?  Must terminate with a null.
if (string[i] == 0)
      // The first byte must be printable
      if (isgraph(string[0]) || isspace(string[0]))
            goto display_CStr;

// Is this a PStr?  The string[length] byte must be printable
if (string[0] < i)
      goto display_PStr;
```

  The significance of this is that if you have partially constructed strings in a char array, SADE may decide you do not have a string. If string display fails, SADE will show you the first 20 elements of your array, which may give you a clue as to why the array did not display as a string. One peculiar case is when the first element of your array contains a zero, in which case SADE will believe that it successfully displayed a null string, and will not show the rest of the array. In most cases this is the right answer, but not always.

  Also note that you can type coerce your array to display it as a PString:

```
^PString(@YourArray)
```

  Similarly, you can coerce your array to display it as a CString:

```
^CString(@YourArray)
```

  Note that Pascal strings are shown with single quotes, e.g. `'string'`, whereas C style null-terminated strings are shown as `"string"`.

XCMDs, etc.

- SADE's default behavior does not allow you to set breakpoints in resources which are not of the type 'CODE'. You can change this behavior by setting the built-in SADE variable `BreakInNonCODERsrc` to 1. Be warned, however, that as SADE must then

check every resource which is being loaded into memory, this option will slow things down.

- You can debug HyperCard XCMDs and XFCNs with SADE. To be able to debug XCMDs, follow these steps.
  (1) Compile and link your XCMD (or XFCN) with symbolics.
  (2) Using ResEdit, **remove** the XCMD resource out of the original stack and paste it into the HyperCard application itself.
  (3) In SADE, you must set the SADE variable BreakInNonCODERsrc to 1.
  (4) Set the directory to the location of the .SYM file and set the sourcepath to the location of your sources.
  (5) Next, target HyperCard using the SYM file, e.g.:

```
Target "HD:HyperCard:Hypercard" using
"HD:Examples:HyperXExample.SYM"
```

  You will see an "Initial program break encountered at..." message output to the worksheet, which indicates HyperCard itself was suspended. From here, you may open your source file, set a breakpoint and execute your program.

  Note: this is not a fully supported feature of SADE. Therefore, the above steps may not work in all instances.

- Some applications may load your XCMD-type code resources in a subheap. To be able to debug them, you need to set another built-in SADE variable, `CurTargetZone`, to the address of that subheap.

- Debugging printer drivers is analogous to debugging XCMDs and XFCNs. Build the driver and paste the code resource you wish to debug into any application. Remember to place the driver in the extensions folder (System 7.0) and select the driver from the Chooser. Set the directory and sourcepath as mentioned above. Set the variable `BreakInNonCODERsrc` to 1. Target the application using the .SYM file you just created. You may now set a break in your source file and execute the SADE command `Go`.

Known bugs

- If you target an MPW tool and then before executing the tool from the MPW worksheet, you set a break on the tool and quit SADE, the Shell may crash with a bus error.

- If you step over the last statement in an MPW tool written in Pascal, you may get an address error in the Shell.

- The SADEKey may not work until the target application has made an event call (for

instance, WaitNextEvent).

Pressing the SADEKey multiple times in a row may cause a crash.

Targeting a running application and then hitting the SADEKey may not work.

- The `Stack` command does not support the documented option `at` address.
- If the `Stack` command is part of a break action, the stack will not necessarily be dis - played when the breakpoint is encountered. A Printf command following the stack command will flush internal buffers, forcing any pending output.
- SADE shows Pascal variables containing sets as integers, instead of showing the set elements with their names.
- If you kill the target using the menu command, the watch variables are removed and the conditional break table is cleared. However, if you use the SADE command `Kill`, or quit the target, these clean up actions are not done. If you target again, the old watch variables may not be relevant.
- There is a size limit of 1,006 bytes on the value which can be assigned to a SADE vari - able. As a consequence of this, you can not use the Show Value menu to display a variable of size greater than 1,006 bytes. You can, however, enter the name of the variable you want to display in a writable window and hit enter to get its value.
- Assigning a large C structure to a SADE variable in a `define` statement may display garbage as part of the error message.
- To reference Pascal variables declared at UNIT scope, when the PC is not in the scope of the unit, use the syntax, `\`*`UNITNAME`*`.`*`variablename`*. To reference Pascal variables declared at PROGRAM scope use the syntax, `\`*`variableName`*, referencing them as `\`*`PROGRAMNAME`*`.`*`variableName`* does not work.
- When running SADE on a Classic and targeting a MacApp program, once you step into a MacApp method, SADE can't retrieve the symbol information for the source.
- If Stop Before Constructor is set and after targeting you step over main() in a C or C++ application, quitting the application will produce an error message of "Data Initialization Failed!".
- Stepping over a CHK or TRAPV instruction which will cause an exception results in a crash.
- If you enter SADE by calling a SysError() call, the condition code registers displayed will be invalid.
- Since SADE allows users to use symbol files which may not have been produced as part of the application build, SADE can not do any checking of the symbol file date relative to the application date. If you see the error message:

```
Logical end-of-file reached during read operation (OS error -39)
```

when you target a process, it probably means that your application and the symbol file are out of sync.

- The current symbolic information does not include information specific to classes. While SADE supports commands like `Break Class TSomething`, it requires SADE to scan the entire symbol file to find matching names. As a result, this command can take a long time.
- Similarly, the symbolic information does not include information about static class variables, and SADE does not display these when a class object is displayed. To see a class static variable, you have to fully qualify the name, e.g.

  ```
  TClass:fgStaticField
  ```

- If you have trace points set within the main event loop of your target program, so that they are continually displayed in the worksheet, the target application's menu may occasionally overlap the SADE menu. This is only a cosmetic problem in the Process Manager (System 7.0) and in Multifinder (System 6.0) and should not affect the functionality of your program.
- Users of Object Pascal who are not using MacApp should always specify the linker's method table optimization option `-opt on`. Otherwise, stepping over method calls will not work. This is because SADE changes the return address of the method's caller, which confuses the method dispatch procedure (%_Method). When %_Method fails, it intentionally executes a division by zero, which is what SADE reports.
- SADE cannot target background-only applications because of the way it works with the Process Manager. Workaround: Remove the background-only restriction from the application and debug. Once your application is fully debugged, make it background-only again.

System 7.0 specific bugs

- Under System 7.0, if you've targeted an application and stepped through your initialization code, even though your target is suspended, hiding SADE will cause the target to run. This is a bug in the Process Manager.
- If a file is open modifiable from a shared volume by one user, that file cannot be opened (read-only) for targeting by another user.
- Do not launch two debuggers at the same time. If you quit the second one launched, and try any operation in the first one, you will get the cryptic message:

  ```
  #   Application made module calls in improper order  (OS error -603)
  ```

System 6.0 specific bugs

- Special care needs to be taken when debugging MacApp programs under system 6.0.x. After killing a program the first time, you may not be able to retarget the application and be forced to reboot. This is because MacApp installs a VBL which is not being removed by MultiFinder after a kill. This does not occur under system 7.0. A work around is to always quit your application normally; do not kill the application from SADE.
- SADE doesn't handle script execution properly when the script is in the same folder as the target. Workaround: fully qualify the path to the script before executing it.
- There is a bug in Multifinder 6.1b9 which prevents you from being able to scroll the Apple Menu when the option key is held down.
- If you use the target command when you already have a suspended target, the old target will not be restarted.
- If you launch a target from SADE, and quit SADE without first killing it, when the target quits, Multifinder will try to send a message to a non-existent process and cause a system crash.
- The SADEKey cannot be changed under System 6.0.
- SADE does not work with Maxima™ or Virtual (both from Connectix Corp.).

Compiler and Linker symbol generation bugs:

- The compilers do not generate type information unless considered necessary. If you have a type declared but do not have any object of that type, possibly used only for casting, no symbolics are generated for the type. Workaround: none. This is a compiler feature to minimize the symbol table size.
- CFront typically generates constructor and destructor code at the beginning and end of syntactic blocks, gives them separate symbolic information records, but maps them to the same location in the source file. One observed behavior is stepping multiple times on the same source statement. Also, in this situation if you do a Break followed by an Unbreak, the location you set the breakpoint and the location you did the unbreak may not be the same, even if the selection looks identical. Workaround: none.
- If the source code contains a series of declarations with assignments, CFront produces incorrect symbol information for these. The observed behavior is that you need to single step as many times as there are declarations, but SADE keeps highlighting the first statement in the series. After sufficient number of steps, SADE will jump over all the declaration code at once. Workaround: none.

- C++ allows variables to be declared anywhere within a block. When a program is stopped in a block before a variable comes into scope, SADE will display an incorrect value for the variable. A better behavior might be to say that the variable is not in scope.

- CFront passes a temporary file `C.pipe.code` to the C compiler. The C compiler incorrectly assigns all global static variables to the unit `C`. For this reason, references to static variables in a file other than the current one does not work as expected. Workaround: instead of `\FileName.StaticName`, try `\C.StaticName`.

- SADE can not print the values of anonymous unions in C++ correctly. If you have:

```
struct Outer {
   int field;
   union {  // anonymous
         int cant_find;
         char another_cant_find;
   };
} Outer_thing;
```

  SADE is unable to display the value of `Outer_thing.cant_find`. Workaround: If you look at the struct that CFront makes, you will see a name like `__O1` for the anonymous union. If you say `Outer_thing.__O1.cant_find` then SADE will print it.

- The MPW 3.2 C compiler produces incorrect source statement information for statements with zero effect. The symptom is SADE's inability to set breakpoints in some routines. These show up most commonly in the process of compiling the output of CFront. Workaround: give `-opt off` flag to the compiler.

- Both the MPW 3.2 C and Pascal compilers may assign multiple user variables to the same register. As a consequence, if you try to display a register variable whose value is currently dead, the value displayed may be wrong. Workaround: none.

- If an enclosed Pascal procedure does not use any variable from the calling procedure, the MPW 3.2 Pascal compiler optimizes away the static link, but does not reflect this in the symbol information, resulting in incorrect display of arguments in the called function. Workaround: use `-opt on, nostatic` or the `-opt off` flags to the Pascal compiler.

- The MPW 3.2 Pascal compiler optimizes away the `LINK` instruction for functions which do not need stack storage for local variables, resulting in incorrect display of the stack. Workaround: give `-opt off` flag to the Pascal compiler.

- The MPW 3.2 Pascal compiler generates incorrect symbol information about the location of local variables that are assigned to a FPU register . Workaround: none.

- The MPW 3.2 Pascal compiler generates incorrect symbol information for enclosed functions with a procedure parameter. Workaround: none.

- If your program contains single-statement `for` loops which should execute multiple times, SADE will appear to step through the loop only once. This is due to the way

that both the MPW 3.2 C and Pascal compilers generate symbol information. Workaround: set a breakpoint on the statement in the `for` loop and execute the `Go` command multiple times.

- The MPW C and Pascal compilers occasionally generate incorrect symbol information for statement boundaries. Workaround: give `-opt off` flag to the compiler.
- The current symbolic structure is somewhat limited in the size of the program that can be compiled with symbolics. The following table lists the number of different symbolic items a symbol file can contain.

Indexes to source statements (CSNTE)	4 G lines
Number of different source files	2K
Number of Functions (MTE)	64K
Number of blocks (CMTE)	64K
Number of Variables and Constants (CVTE)	64K
Number of Types (TINFO)	64K
Number of Labels (CLTE)	64K

- Even if you stay within the above limits, the MPW 3.2 linker occasionally produces incorrect type information in large symbol tables. The `Show Value` menu item(s) normally traps all internal errors and displays the message `Undefined/Out of scope`. However, if you type the name of the object in the worksheet and hit enter, SADE may then print the error message:
  `Symbol table error: Variable type table index could not be found.`

  Workaround: The post 3.2 version of the linker shipped on the E.T.O. #5 disk fixes most of these problems. If you still find this problem, and if your program uses a rich hierarchy of classes, there is no workaround. Even with a partially broken symbol file, you may be able to do most of the things you need to do. If your program is just big, if the size of your symbol table exceeds 1 megabyte, consider compiling parts of your application without symbols, and compile only the suspect parts with symbols.

  You can use the SADE diagnostic command `List Symbol` to see the local variables and types that are defined in the current scope. This command will also show if any symbolic information associated with these objects is inconsistent.
- Types which have global scope sometimes get bound to a local scope, you can see the type definition only if the type is used in the current scope, or a containing scope. Workaround: none.

SADE 1.3 Manual Errata

- The manual's index was not generated correctly, so you may notice that each entry (after page 42) is off by two pages. Simply subtract the entry by 2 to get the correct page number.
- The dialog box displayed when the target menu is selected is shown as a generic SFGetFile dialog box. In the released product the dialog box contains the line
  `Please select an application or symbol file`.
- In the section describing the different ways to target a program, the manual states that you can drag and drop the icon of the target application on top of the SADE icon. For this feature to work, you may need to rebuild your desktop once.
- The Target menu command as described in the manual is incomplete. It states that you must select an application to debug, when in fact you may choose either an application or its .SYM file from the dialog box. Then, if both files are not in the same directory, SADE will prompt you to locate the missing file. For instance, if you select a symbol file, e.g. *AppName.SYM* ., and the application file is not in the same directory, SADE will put up a dialog asking you to locate the file *AppName*. (Your application and .SYM file names do not have to be the same, however.)
- The manual states that if your target program is suspended and you quit SADE, SADE will terminate the program. SADE will now restart your target application if it can.
- The description of the `Show Stack` menu command is incomplete. The command now displays a warning if the program counter points to a LINK instruction.
- The `Disasm` command correlates source statement numbers with assembly code instructions. Branch instructions which use the new branch island option, are shown with a ^ character to denote the indirection.
- The intentional bug introduced in the third C tutorial program, Sample2, can be more destructive than we intended. Running this program can cause your machine to hang.

  If it does not hang your system, and you display the stack when the bus error occurs, the output is more informative than documented, in that SADE displays some MacsBug symbols.
- SADE acquired the ability to prompt for source files when needed after the manual was finalized. As a consequence you may never have to use the SourcePath command. This behavior is not documented in the SADE manual, but is described in some detail at the beginning of these release notes.

- SADE now includes an optional menu called Extras which is not documented in the manual, but is described earlier in these release notes.

  Contrary to what the SADE 1.3 Manual states, you can debug Hypercard XCMDs and XFCNs with SADE. The procedure for doing this is described earlier in these release notes.