Release Notes MacMETH 3.2.8

Welcome to MacMETH, a "Fast Modula-2 Language System For the Apple Macintosh".

Where to Find Help?

Please note, this release contains a folder which holds the entire manual and other useful documentation in electronic form (see folder "MacMETH Docu"). We expect you to print the manual and have it handy while working with MacMETH. We fear it is not possible to work successfully with such a neat, yet powerful tool as MacMETH, unless you really consult the manual. We have put much effort into it to make it really worth-reading and hope it will be useful to you.

Get latest updates from:

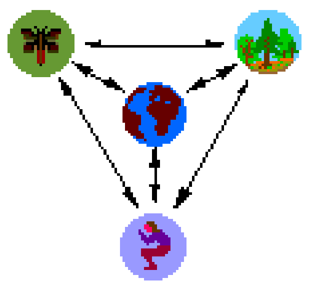
http://www.sysecol.ethz.ch

For feedbacks send E-mail to

mailto:RAMSES@env.ethz.ch

We like to hear from you. Thanks!

And now, be invited to plunge ahead. Enjoy!



A. Fischlin (& J.Thoeny) / ETH Zurich, October 2008

Getting Started Quickly

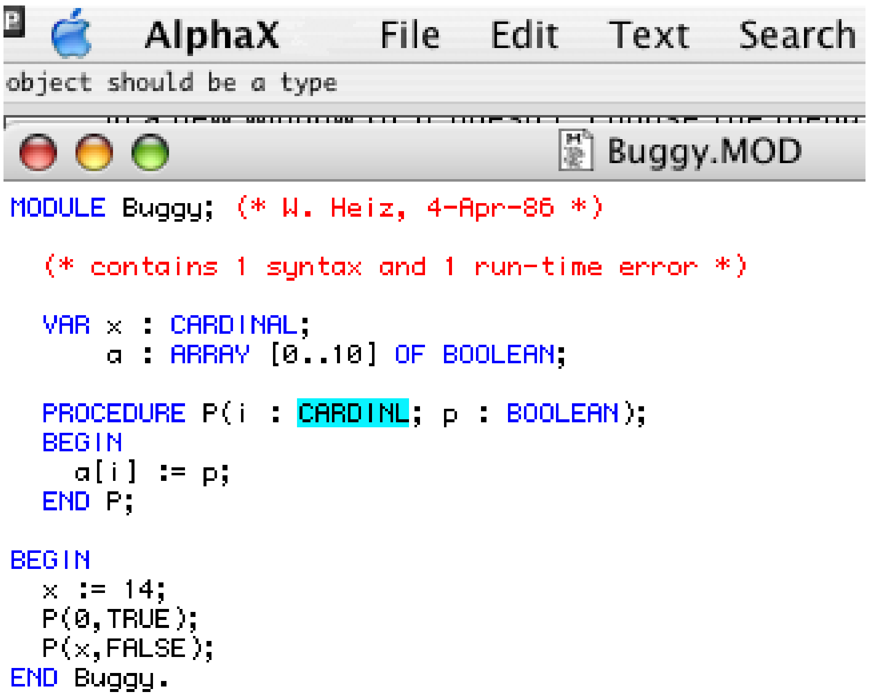
Note, if you do this in OS X, please make sure you have Classic installed or nothing will work as described here!

First aquaintence with the compiler, editor, and program execution:
Double-click the MacMETH shell, choose the menu command Compile and hit return. The example program module "Example.MOD" will be compiled. Click with the mouse or hit return and you will leave the compiler. Choose the menu command Execute and select the just compiled object file "Example.OBM" (it is in the same folder its source was, i.e. in the folder "Examples"). The example program will then execute and give you the subsequently needed instructions. Don't be surprised when the program encounters run-time errors. This is intended!

Next compile another program to learn about editing and the handling of compilation errors. In the MacMETH shell choose the menu command Compile again and just hit "tab" (instead of typing the name of the module to be compiled). In the subsequent dialog select the file "Buggy.MOD" (it resides in the same folder as "Example.MOD" does, i.e. in folder "Examples". An alternative would be to just to type "Buggy." (note the period at the end) and hit Return; MacMETH's compiler would then expand your module name by adding the default extension "MOD" and start the compilation). The compiler will compile the program and detect compiler errors. To correct them, choose the menu command Edit2 and you will launch the editor MEdit.

Since you do this the very first time, MEdit will ask you to provide some customization information. Enter your last and first name where asked and simply accept all other configuration dialogs with OK. The program "Buggy.MOD" will open in a new window (if it doesn't, choose the menu command "Open work file[s]" under menu "Macros" (remember, keyboard short cut is Cmd^0); see also chapter "Known Bugs" of this README) and the first compiler error will be displayed in an alert. Correct this syntax error (change "CARDINL" to "CARDINAL") and return to the MacMETH shell by using the menu command Clear, save and launch (Cmd^1) in the menu Macros of the editor MEdit. Once in the MacMETH shell, compile Buggy again (it's syntax should now be ok) and execute "Buggy.OBM". Module Buggy should crash due to a run-time error. At this point you may use the debugger to analyze the cause of this run-time error or simply abort the program. Note, once you abort Buggy, you have only aborted the current, so-called subprogram level, i.e. the program level of program module Buggy. But you have not aborted the MacMETH shell, which ought to be still running.

If you use an Alpha editor (see chapter «Using the Alpha Editor»), all should work in a similar way. You can use e.g. AlphaX instead of MEdit by simply choosing menu command AlphaEdit instead of Edit2. If the AlphaX editor is installed as described in the Help file for Modula-2 (available from within Alpha in the Help menu) then you should get a screen similar to this:



```
object should be a type
```

```
MODULE Buggy; (* W. Heiz, 4-Apr-86 *)

  (* contains 1 syntax and 1 run-time error *)

  VAR x : CARDINAL;
      a : ARRAY [0..10] OF BOOLEAN;

  PROCEDURE P(i : CARDINL; p : BOOLEAN);
  BEGIN
    a[i] := p;
  END P;

BEGIN
  x := 14;
  P(0,TRUE);
  P(x,FALSE);
END Buggy.
```

The the faulty code is selected and an error message is displayed in the message bar. In AlphaX it is just below the menu bar. The M2 mode of Alpha offers similar commands than those available in MEdit. As a rule, use simply the Control-key instead of the Command-key. Thus to return to the MacMETH shell choose the corresponding menu command in the "M2" menu with the keyboard shortcut Ctrl^1 (instead of Cmd^1). Searching for errors can be done with Ctrl^E (instead of Cmd^E) etc. Note, in contrast to MEdit, Alpha editors allow you to leave the source code window open while you compile. Alpha editors do also not alter the source code. They insert only temporary marks to mark code sections containing compiler errors.

Note, the entire development cycle can be done efficiently with keyboard shortcuts only. A typical sequence is: Cmd^C, Return, Return, Cmd^X, testing program, Cmd^Q, Cmd^E, editing, Cmd^S, Cmd^1 (or Ctrl^1 in Alpha). This sequence is particularly efficient if you have Default Folder from installed. Should you wish to use an Alpha editor, note that above convenient keyboard shortcuts are not fully available unless you edit the default keyboard shortcuts of the MacMETH shell. You find details on how to accomplish this in the MacMETH manual on page 9ff. (see next chapter «The Manual»).

Of course, both little programs (Example, Buggy) don't do anything useful. They just demonstrate the basic techniques needed to program in Modula-2 using MacMETH, i.e. how to edit, to compile, to deal with compiler errors plus run-time errors, and how to use the symbolic run-time debugger. Besides, the debugger can also be called anytime during program execution by pressing Command^Option^H, the so-called "User Halt" in contrast to the programmed one, i.e. statement HALT, as demonstrated by program "Example.MOD". Really useful sample programs, which function fully and which explain basic and essential programing techniques are available when you unpack the manual (see next chapter «The Manual» of this README). The manual does also explain their purpose and functioning, demonstrating the handling of events, menus, files, dialogs, text I/O, windows, graphics, strings, printing, and low-level programing including accessing the toolbox. Note, unlike many other development environments, MacMETH still runs on almost all Macintosh models, starting with Fat Macs (512Ke) up to today's Power PC Macs under a very wide range of operating systems. Moreover, on old machines it is still possible to develop programs with just a single 1.4 MB Floppy disk system (Sometimes small is beautiful!).

Should you get stuck or encounter some other difficulties, here we can't help you much more (it's just a quick introduction, right?), but help can be found elsewhere: Please consult the manual, which is provided in electronic form as a part of this release. The manual contains also a first tour using the module "Buggy.MOD" with explanations in all details plus much more useful information which helps you to make best use of MacMETH.

Enjoy!

The Manual

The manual comes as part of the ordinary MacMETH release. To obbtain it you need first to unpack it. Simply double-click the file "MacMETH Manual 1992.sea" in folder "MacMETH Docu". You will get a pdf document and a folder containing all sample programs explained in the text. The manual is 116 pages long and you ought to obtain a nice result if you can use a laser or ink-jet printer.

Alternatively you can write us if you wish to purchase the printed booklet against a minimum handling charge:

mailto:RAMSES@env.ethz.ch

Finally it is also possible to download the manual from the MacMETH home page:

http://www.sysecol.ethz.ch/SimSoftware/RAMSES/MacMETH.html

Latest Changes (not in the manual)

In chronological order:

MacMETH is now supported by 'Terrestrial Systems Ecology' ETH Zurich, Switzerland. It is available as freeware courtesy ETHZ. You can download it from our home page on the Internet at

http://www.sysecol.ethz.ch

The memory management has been improved once more considerably:

- The stack size is set to Apple's default stack size (32kB on Color QuickDraw machines and 8KB on non Color QuickDraw machines) plus an additional 10kB.

- However, you can now control the stack size by the User.Profile if you want more than just the additional 10kB. Simply write in the User.Profile the "Stack" section similar to this:

```
"Stack"
  'Size'            30kB
```

You can enter any number (in kB), provided you give a sufficient total for the application so that there is still some heap memory available. Note, the stack size can only be changed once, i.e. during application start-up. Hence, any stack changes in the User.Profile while the MacMETH, RAMSES shell or Modula-2 application is still running, will not take any effect until you restart this application.
(Note: Size is actually the amount by which the stack is increased in addition to the predefined one. The stack sizes predefined by Apple are: 8 kB for machines without and 32 kB for those with ColorQuickDraw).
   - The application heap has been expanded to the maximum possible value. Therefore the Macintosh is now able to detect stack overflows (ID=28). If you frequently encounter such errors, whith code which run more or less well under previous MacMETH versions, you have good reasons to suspect run stack overflows. Try increasing the stack size and if this helps, you got some evidence. However, now you should check your code, somewhere close to the code area where you get the ID=28 bomb, is the culprit; you should fix it. For instance, check for large objects you pass as actual parameter to an open array value parameter.
   - A safety memory cushion (20KB) is allocated at the highest possible position of the heap. This memory will be released if all the other heap memory is used up. The user will be informed, if this happens. This prevents many crashes in the context of ToolBox routines, which are using the application heap.

You may use the debugger to set break points (feature not described in the manual). This feature works only if you set item 'Break' to on in the "Traps" section of the User.Profile.

How to use it? The debugger may be launched without encountering a fatal error. For instance you may program a HALT with the statement HALT or you can manually trigger a so-called User Halt (Cmd^Option^H or Cmd^Shift^H).
   Once you have launched the debugger and a source code is displayed in the "Source" window (see manual p. 37), you can insert a break point by pointing with the mouse onto any statement where you wish to set the

break point. A little triangle pointing upwards is displayed as long as you keep the mouse button pressed and your cursor is within the window "Source". The triangle jumps to any location within the source code, where you can potentially place a break point. Hint: This location is as close to the current cursor position as possilbe, but often lags behind it. So you may find additional break points by pointing with the cursor slightly behind the actual location in which you are interested. Try it out.

Once you release the mouse button, the break point is set and program execution continues (if no fatal error has been encountered), till the break point is reached. As soon as program execution reaches the set break point, the debugger is automatically entered, where you can view the new program status and possibly set another break point.

Note, if your code encounters a programmed HALT statement, the break point setting is cleared. This means, if you set a break point, but reach first a programmed HALT statement, you have to set the break point once again, should you still wish to break at the envisaged point. Of course, you can not set break points, if a fatal error has been encountered, since any further program execution on the current level beyond the point of fatal error would take place in an undefined state. Unpredictable results would follow and are consequently usually of little interest or value.

The System works now with relative path names. This is a partial work around for the limitation of 63 characters in file names of MacMETH. We were unable to break this limitation, because it appears in the base definition modules and we didn't want to make the entire release key incompatible.

A new tool, InsertRes, available from within the MacMETH shell has been added. It handles the default access to resources in a more convenient way.
This is especially useful if you are using the 'Dialog Machine' or other RAMSES tools. A typical use is the access of 'PICT' resources via procedure 'DisplayPredefinedPicture' from module 'DMWindowIO' which will find the picture resource via the default strategy (i.e. empty file name string). Simply execute InsertRes after the compilation of the (main) program module and the resource will be inserted into the OBM-file; you need no longer to link a double-clickable application to access such resources (for details see document 'InsertRes.DOC' in the folder

'MacMETH Manual 1992').

Machines with 68LC040 processors are now handled correctly.

The compilers accept now the old syntax for type transfers in the form T(e), which is the equvalent of SYSTEM.VAL(T,e). Supports portability os source code among different computer-platforms, i.e. Macintosh, IBM-PC, and Unix workstations (for more details see the 'Dialog Machine' and RAMSES, also available via anonymous ftp from the same hosts (URLs) given above).

A new method to configure the Modula-2 run-time environment of linked, double-clickable applications. You may omit to use a User.Profile by using a so-called Modula-2 preference resource instead. The resource of type 'STR ' with ID = 7413 and name "M2 preferences" defines the following settings (for the meaning see the manual, section 1.4. «The Configuration File "User.Profile"»):

Section	Entry	Index i
"Traps"	'All'	1
	'Arithmetic'	2
	'FPU'	3
	'F-Line'	4
	'System'	5
	'Break'	6
	reserved (internal use)	7
"SANE"	'alwaysSANE'	8
	'invalidHalt'	9
	'underflowHalt'	10
	'overflowHalt'	11
	'divByZeroHalt'	12
	'inexactHalt'	13
	reserved (internal use)	14
	reserved (internal use)	15
"Stack"	'Size'	16..n

A character at index i = '1' turns the corresponding entry on, any other value off. If the string is too short to contain a character at position i, the corresponding entry is not modified and keeps its predefined default value. The integer number starting at index 16 specifies the stack size in kBytes (followed optionally by any non-digit characters).

```
   Example: 110111-110110--10kB
    indices 1234567890123456789..
```

Effect (unless overwritten by a User.Profile residing within the same folder as the Modula-2 application containing the configuration resource): All = on, Arithmetic = on, FPU = off, F-Line = on, System = on, Break = on, alwaysSANE = on, invalidHalt = enabled, underflowHalt = disabled, overflowHalt = enabled, divByZeroHalt = enabled, inexactHalt = disabled; total stack size 18 or 42 kB (depending on machine). Note, these settings are the default settings.

To simulate default settings of older MacMETH's (<=3.2), use the following settings:

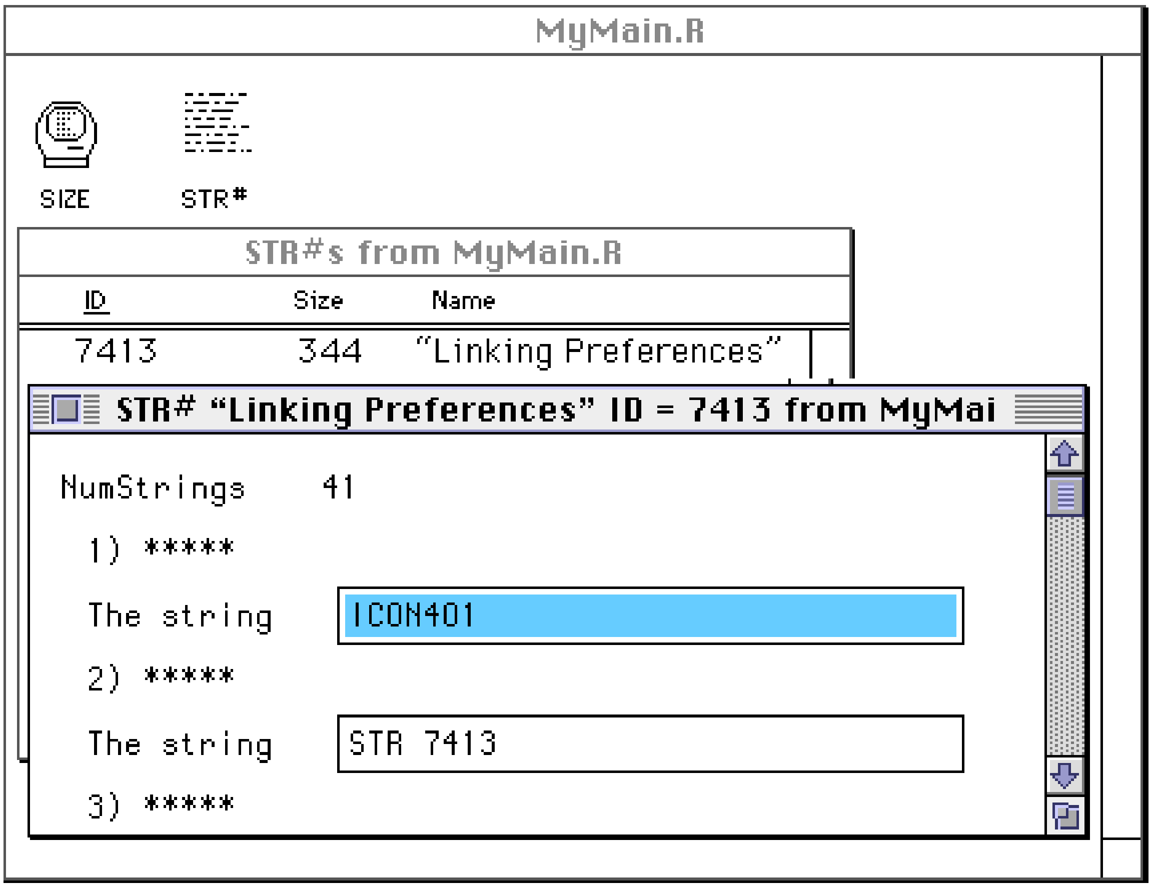
```
   Example: 110111-010110--10kB
    indices 1234567890123456789..
```

Actually, these are the settings used by the "Linker Extra" (contained in folder M2Tools) which is only the standard linker linked to a stand-alone application, but which inserts above configuration resource. However, note, this emulation of older MacMETH's behavior uses the memory still slightly differently, since there was no top-of heap memory cushion set aside between stack and heap (now 20 kB, to be used if heap gets scarce) and stack overflows were less detected, since the heap was not immediately maximally used (no call of MaxApplZone). Moreover, the stack-heap border line was set at the bottom of the block where now the memory cushion starts.
NOTE: If this resource is missing in the resource file you specify during linking, i.e. your MainModule.R-file, it will be copied into your application from the linking application (typically the MacMETH or RAMSES shell) together with all other essential resources.

MacMETH 3.2.5 and later

The linker (≥ V 3.2.4) has a new function, which allows to control the resources to be linked into an application. Your application specific resource file (the one you enter when the linker prompts "Resource file>") may optionally contain so-called linking preferences. This is a resource of type "STR#" with ID 7413. It lists all resources to be copied from the linking application's resource fork into the application you are linking.

This allows to have resources shared among several applications in its newest version in the linking application only (typically the MacMETH shell), instead of having many redundant copies, possibly in outdated versions, in many application files. Use ResEdit to edit the linking preferences. A typical linking preferences resource looks similar to this (as seen from within ResEdit)



ID	Size	Name
7413	344	“Linking Preferences”

```
NumStrings     41
  1) *****
  The string   ICON401
  2) *****
  The string   STR 7413
  3) *****
```

A particular resource is specified by its type, the first 4 characters, and by its ID, an integer number, appended in free format. Watch out, the type must be exactly 4 characters; e.g. in above example, there is no blank in the first string but one in the second string, since it denotes the type 'STR '.

Any resources not listed in the linking preferences are simply merged into the final application as before with older versions of the linker. In particular, the linking preferences are completely optional, i.e. the linker behaves exactly as before if you make no use of this additional feature.

If a particular resource listed in the linking preferences exists in both resource forks, i.e. in the one of the linking application as well as in the one of the application's resource file, the second takes preference over the resource from the linking application. Thus you may simply override a default resource with a customized variant.

Ex.: Let's assume you have a program called "MyApplication" for which you created with ResEdit a resource file called "MyApplication.R". The linking preferences (STR#,7413) in "MyApplication.R" lists the strings 'DLOG6000' and 'DITL6000', which denote the message dialog window you use when you call procedure "Message" from module "FileUtil". You can simply customize later that window, e.g. to make the window larger. Use ResEdit and copy and paste the resources DLOG, 6000 and DITL, 6000 from MacMETH's resource fork into the file "MyApplication.R". There you edit the just copied resources to your liking; that's it! Relink your application and it will use from then on the customized variants of these resources. There are some exceptions though, you can't override any of the CODE resources nor the resource ALRT, 305 and DITL, 305 respectively. They are fully reserved for internal use only.

MacMETH 3.2.6 and later

You can now control which creator is used by default for all the files module FileSystem creates (unless it is the compiler or linker). This information is stored in the Modula-2 preferences resource. It is named "M2 preferences" and is a resource of type 'STR ' with ID = 7413 (described above, since it is also used to define User.Profile data in case the file User.Profile is missing). The 4 characters after the first blank found in this resource are used to this purpose. If no such data can be found, the previously used default creator 'MEDT' for the freeware editor MEdit is used instead.

Example:

```
            110111-110110--90kB ALFA
                                  ^
    indices 1234567890123456789•1234...
```

Note the single blank after 90kB. In above example the shareware editor ALFA is denoted as being the main creator, typically for TEXT files as created by module FileSystem. There is also a special chapter in this documentation on the use of the Alpha editor together with MacMETH

(«Using the Alpha Editor»).

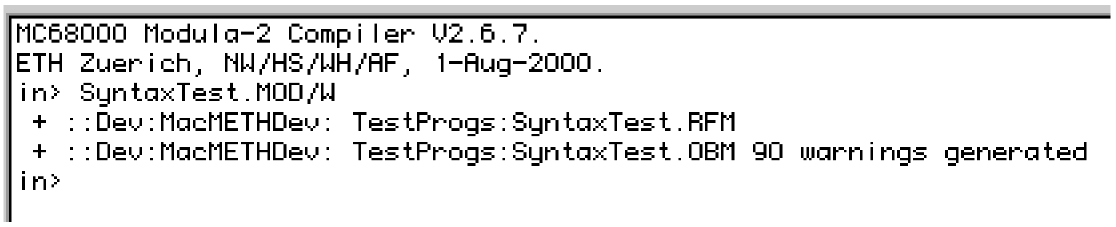
The MacMETH shell is now also high level event aware. However, this is the case only in a very limited way, i.e. MacMETH plus any loaded subprograms can now properly respond to the quit event. All other Apple core events are ignored, unless the subprogram installs a task scheduler via module EventBase, which handels such events properly.

The technique used is that the resource "mst#" (same format as "STR#") with ID 101 is used and compared (in a case sensitive way) with every menu command installed via module Menu to determine which is a quit command. If no such resource is present in the MacMETH shell, the entire mechanism is disabled, resulting in the older behavior of the software.

As soon as the MacMETH shell or any loaded subprogram level receives an Apple core event of kind 'quit', e.g. during a shutdown, MacMETH will execute all of these menu commands which have been recognized as being quit commands in reverse order of installation. The here distributed resource "mst#', 101 contains among others the common menu command "Quit", which allows for proper termination of the debugger, any English program featuring a "Quit" command plus the MacMETH shell itself. All of this should allow in most circumstances to terminate during ordinary shutdowns to terminate fully the MacMETH shell plus an entire program stack possibly running on top of it.

MacMETH 3.2.7 and later

New compilers (Compile and Compile20) 2.6.7 from 1st August 2000 (the Swiss National Day). They support the option /W while compiling a module, e.g. type module name and then "/W" similar to this:



```
MC68000 Modula-2 Compiler V2.6.7.
ETH Zuerich, NW/HS/WH/AF,  1-Aug-2000.
in> SyntaxTest.MOD/W
 + ::Dev:MacMETHDev: TestProgs:SyntaxTest.RFM
 + ::Dev:MacMETHDev: TestProgs:SyntaxTest.OBM 90 warnings generated
in>
```

If your code contains type conversions (coercions), you will get warnings.

Switch to the editor and deal with the warnings in the same way as you do with ordinary compiler errors. This helps you to locate implementation dependent code, which may cause problems while porting source codes among platforms. Note, if there are no real errors present, the compilers always generate the object code. In this respect warnings contrast with the error encountered situation where a compiler reports "errors detected" and suppresses any code generation.

MacMETH 3.2.8 and later

New compiler (Compile) 2.6.8 from June 2001: The compiler is recommended to be used on all Power PC Macs including the G5 running on an operating system later than MacOS 8.1 (By the way, it is recommended to use MacOS 9.2.2 or X with MacMETH 3.2.8). The new compiler does no longer rely on the trap mechanism for integer arithmetic exceptions, i.e. integer division by zero, range checks, and integer overflow ('Arithmetic', Trap numbers 5,6,7). Consequently the item 'Arithmetic' in the trap section in the User.Profile matters no more and is best set to off for efficiency reasons:

```
    "Traps"
      'All'          on
      'Arithmetic'   off
      'FPU'          off
      'F-Line'       on
      'System'       off
      'Break'        on
```

If you use a MacOS later than 8.1 it is recommended to use above trap settings exactly as shown (in contrast to what the manual says on p. 7). FPU traps do not occur on a PowerPC machine. Consequently it is best to set these traps to off, unless you work on a true 680x0 machine, such as a Quadra or a MacIIfx. An exception like a F-Line trap ('F-Line', Trap number 11) is usually only caused by an illegal instruction, e.g. the program register points to data due to erroneous software. If you keep this trap enabled you can learn about the program location where the error occurred. On the other hand, you may want to set this trap also to off, since the MacOS is usually unstable, once you have encountered a F-Line trap. Similarily, Access Fault, Address Error, and recognized Illegal Instruction ('System', Trap numbers 2,3,4) are set to off. They are rarely encountered on a Power PC machine. Finally, at least item 'Break' should be set always to on ('Break', Trap number 32) under MacOS 9.2.2 or X. This supports the setting of break points from within the debugger.

Note, in general most traps are best left to the operating system or an interception by MacsBug (see also next chapter of this documentation), since the interplay between MacMETH code and the newer operating systems do generally not allow for a support of traps anymore (except for 'Break', i.e. break point setting from within the debugger via Trap number 32 under MacOS 9.2.2 and X). However, these limitations, i.e. avoiding the use of traps, should be of little consequences for most programing applications. However, using the new compilers is of great advantage, since the result is an optimally stable programing environment.

Consequently, with the new compiler it is recommended to set the M2 preferences (resource type = 'STR ', ID=7413) to these defaults:

```
            110111-110110--90kB ALFA
                                 ^
        indices 123456789o123456789•1234...
```

The compiler Compile20 should no longer be used on a Power PC, since it still relies for integer arithmetics (division by zero, range checks, and integer overflow) on the trap mechanism. Of course, you can still use it, however you run the risk to obtain an unstable operating system once you have encountered a run-time error such as a division by zero, a range error, or an integer overflow (see also next chapters of this documentation for detailed hints on how to use Compile20 on a Power PC).

New compiler (Compile, Version 2.6.8 from June 20001): The compiler knows a new option Z. This option controls compiler generated checks for integer divisions by zero. The manual on p. 25 should be extended accordingly:

...
option character signals the inverse of its default value.

r	array index, subrange and pointer checks	default = on
v	integer arithmetic overflow check	default = off
z	integer division by zero	default = on

Examples:

Name.MOD/v	all checks on
Name.MOD/vr	integer arithmetic overflow checks on, range checks off,

```
                      division by zero checks on
   Name.MOD/vrz      integer arithmetic overflow checks on, range checks off,
                      and division by zero checks off.
  ...
```

If the Z-option is on, the compiler generates code which checks for integer divisions by zero and displays the corresponding Modula-2 run time error dialog if such an error is encountered. It is highly recommended to keep this option on, unless you want to trigger the trap number 5 (only possible if 'Arithmetic' in "Traps" section of the User.Profile is set to on) in case a division by zero is encountered.

You may also insert this compiler option in the source code in the usual way, i.e. insert a comment. For instance, to force this option to be temporarily off, insert a comment like this:

```
  (*$Z-*)
```

before that code where you wish to suppress any exceptions triggered by integer divisions by zero.

Known Problems and Solutions

Unless a Classic emulator becomes available, MacMETH will not function on Intel based Macintosh computers. MacMETH runs fully on all other Macintosh computers, regardless whether they are 68000, PPC, G3, G4, or G5 based. However, you can use our RASS software to run your MacMETH software on Intel based Macintosh systems:

http://www.sysecol.ethz.ch/SimSoftware/RAMSES/RASS.html

Only present in pre 3.2.8 MacMETH: Under MacOS 8.5 to 9.2.2 and X, many run time errors, such as an index range error or a division by zero crash the OS with a freeze or a bomb. Remedy: Upgrade to the latest MacMETH, in particular use the compiler 2.6.8 or newer. Recompile all your code!

By the way: We recommend highly to upgrade to MacOS 9.2.2 or X should you use an older MacOS!

However, if you are using OS X it is considerably more convenient to work with Classic. However, be aware, Classic is only supported up to OS X 10.4.x (Tiger)!

You can obtain an entire disk image containing Classic taylored to working with MacMETH and/or RAMSES from our web site:

http://www.sysecol.ethz.ch/SimSoftware/

You encounter error "215 - expression too complex (no more CPU registers available)" for code which could be compiled with compilers before V2.6.8. This happens if your code contains complex expressions. Remedy: Simplify the expression by splitting it into several expressions and using auxiliary variables to store intermediate results. Alternatively, you can disabled range checks and division by zero checks. Of course the latter solution is only recommended, if you are confident that your code is correct and robust.

http://developer.apple.com/tools/debuggers/MacsBug/

Under MacOS 9.2.2 or X your system becomes unstable after encountering the run time errors illegal instruction or a F-line trap. Remedy: First, disable in the "Traps" section of the User.Profile items 'F-Line' and 'System' (see also Manual p. 6). Install MacBugs, which is available as a free software download from

(or visit http://www.Apple.com). MacsBug allows you to exit to the Finder (shell) (type command "es" for "exit to shell" and return), avoiding the cumbersome restart of the entire OS. Then relaunch the application, typically MacMETH, which has crashed. and follow this rule: If MacBugs intercepts the same action twice, reboot, don't exit to the shell only. The draw back from this solution is of course that you do not learn which statement caused the error.

http://developer.apple.com/tools/debuggers/MacsBug/

Under MacOS 8.5 to 9.0.4 your system becomes unstable after using the break point debugger. This is a problem we can't fix easily, since Apple seems not to support properly the installation of interrupt handlers in these operating systems. The interrupt handlers of MacMETH normally deal with these traps, which are otherwise handled by Apple as so-called system errors, resulting in the highly appreciated bombs ;-). Remedy: Upgrade to MacOS 9.2.2 or X (recommended) or stick to OS 8.1 (the easiest strategy if you have a Mac which runs still this OS), or at least disable all traps in the User.Profile and install MacsBug. MacsBug is available as a free software download from

(or visit http://www.Apple.com). MacsBug allows you to exit to the Finder (shell) (type command "es" for "exit to shell" and return), avoiding the cumbersome restart of the entire OS. Then simply relaunch the application, which has crashed. To disable all traps go to the "Traps" section of the User.Profile and change the value of 'All' to off (see also Manual p. 6). The draw back from the solution where you disable all traps is of course that you do not learn which statement caused the error. Important Hint: If MacsBug intercepts the same action you try to accomplish twice, give up and reboot the system.

Using Compile20 on a Mac with a PowerPC and you encounter problems. Remedy: See next chapter «Power Macintosh».

Only present in pre 3.2.7 MacMETH: Under MacOS 9 you encounter crashes while attempting to switch to MEdit using the menu command "Edit2" in the "File"-menu. The result returned by the sublaunching macro as implemented by Apple is not 32-bit clean, thus possibly causing an index range error in module FileUtil. Remedy: Please upgrade to the latest version and you can again switch to MEdit (tool "Edit2" in folder "M2Tools") or Alpha (tool "AlphaEdit" in folder "M2Tools") without any problems.

Only present in pre 3.2.6 MacMETH: If you use the alternate editor MEdit you may have encountered problems with the automatic opening of the source file(s) via Edit2. Newer MacOS versions behave slightly differently, so that the automatic opening no longer works as described on page 22 of the manual. Remedy: To get it working again you have to change with ResEdit the resource "SIZE", ID=0 and set the flag "High level event aware" to 1 (on). Note however, this has the consequence, that automatic quitting of the MacMETH shell while shutting your computer down via the Finder does no longer work, since the MacMETH shell is not capable of responding really to the high level core event "Quit". Later versions of MacMETH (>= 3.2.6) are high level event aware.

The module "Printer" does not function with all printers or does not give you poor control over the printing process. Especially non-postscript printers such as a HP deskjet printer won't work. Sorry, but we had to give up on maintaining this module. Remedy: If you wish to print on a printer where this module prints not at all or not well, we recommend to use the "Dialog Machine" (also freeware, available as part of RAMSES from the same sites as you get MacMETH), which supports the printing of texts and graphics on most printers with the usual Page setup and printing dialogs. In case you desperately wish to use the module Printer, we are glad to provide you with the sources (if you manage to get it running on your particular printer, please give as the source back so that other users can profit from your achievements; remember, MacMETH is freeware. Thanks!).

mailto:RAMSES@env.ethz.ch

If you should learn about some more bugs, please let us know. We will either try to fix the bug or will maintain this list for your convenience

Many thanks for your help and cooperation!   :-)

Power Macintosh

MacMETH doesn't (and will probably never) produce native PowerPC code, but works fine with the 68LC040 emulation of the Power Macintosh machines. So far we found no glitch (compliments to Apple)! But a few caveats are worth-mentioning:

New key short-cuts for User HALTs: If you have been using the "Programmer's key" (used to be at the side of the machines at the back, remember?) to interrupt a running program, use the new keyboard equivalent "Command^Option^H" (H for HALT) with the U.S. keyboard layout or "Command^Shift^H" with any other keyboard (MacMETH 2.3.8 or later). DO NOT use Apple's key short-cut "Command^Power-key", which Apple has provided to replace the missing "Programmer's key", or you will obtain a system freeze (particulary if your hard disk has not been formatted with Apple's "Apple HD SC Setup")!

After pressing the key combination "Command^Option^H" or "Command^Shift^H" (MacMETH 2.3.8 or later), you obtain the usual Modula-2 HALT-dialog, where you can launch the debugger, e.g. with key "D", or continue program execution with key "C", or abort the program on the current level with "A". Note also, the dialog shows now the more precise message "User HALT" (no longer the confusing message "Programmed HALT").

Compile20 and Compile20-code: If you plan to use the Compile20 version of the compiler, you must have installed the shareware system extension "SoftwareFPU" (actually a control panel or CDEV) or the even much more efficient non-shareware "PowerFPU", both by John Neil & Associates, P.O. Box 2156, Cupertino, CA 95015 USA, (Orders: +1(800) 663-2943; Fax: +1 (415) 905-3001. As of this writing it seems that this company has gone out of business. Fortunately, Tom Pittman, the original author of "PowerFPU" is so kind to let anyone use now his software PowerFPU

href="http://www.ittybittycomputers.com/IttyBitty/PowerFPU/

The distribution of RAMSES Extras

http://www.sysecol.ethz.ch/RAMSES/RAMSES_Extras.html does now also provide this software. If all this does not help, please write us an E-mail if you need to use Compile20 without having the needed system extension. We might be able to help.

The instructions generated by the Compile20 version of the compiler are for a true MC68881 or MC68882 which were available as companion math chips in older MC68020 or MC68030 machines, respectively, or the integrated FPU in a MC68040. Executing such code on a PowerPC (besides, is also the case during starting up of Compil20 itself) crashes the 68K-emulator with an illegal instruction exception.

Note, this is not a fault of the emulator, since it emulates not a true MC68040 with a FPU integrated into it, but only a MC68LC040, which has actually no FPU built into it. "SoftwareFPU" or "PowerFPU" extend the emulator to emulate a MC68040 (please note, both system extensions are distributed as part of RAMSES Extras, see above link). Consequently, inquiries into the current system characteristics (e.g. by System.CPU and DMSystem.FPU) while one of these extensions ("SoftwareFPU" or "PowerFPU") is running, return a MC68020 CPU (which is the instruction set of the MC68LC040) and the presence of an integrated FPU (like on a true MC68040).

If you are using "SoftwareFPU" it is recommended to set

```
    'F-Line'      off
```

in the Trap section of the User.Profile. If you are using the "PowerFPU" (regardless whether its active or not) the setting in your User.Profile does no longer matter, since the "PowerFPU" takes care of that during startup of the entire machine.

Note that your Compile20 compiled obejct code will no longer generate a Modula-2 run time error for floating point exceptions. This behavior contrasts with that on a true 68020 or 68040 Macintosh. For instance, there you would get a Modula-2 run time error for a division by zero. Instead the code will quietly continue execution. Consequently, **it is recommended to refrain from using the Compile20 compiler on Power Macintosh systems**. It is better you use the ordinary compiler Compile; then you will get a Modula-2 run time error for all floating point exceptions according to your current SANE settings as defined in the User.Profile. E.g. if you use the default SANE settings, you are informed if a division by zero occurs and you may use the debugger to locate the culpable statement.

68LC040 based Macintosh (e.g. older PowerBooks)

If you plan to use Compile20 version of the compiler on a machine with a MC68LC040 CPU-chip, you must have installed either the system extension "SoftwareFPU" or "PowerFPU". Please see chapter «Power Macintosh» for further details on those system extensions.

It is also recommended to set

```
    'F-Line'     off
```

in the Trap section of the User.Profile, albeit these settings do no longer matter when using "PowerFPU".

Note that your Compile20 compiled obejct code will no longer generate a Modula-2 run time error for floating point exceptions. This behavior contrasts with that on a true 68020 or 68040 Macintosh. For instance, there you would get a Modula-2 run time error for a division by zero. Instead the code will quietly continue execution. Consequently, **it is recommended to refrain from using the Compile20 compiler on Macintosh systems with a 68LC040**. It is better you use the ordinary compiler Compile; then you will get a Modula-2 run time error for all floating point exceptions according to your current SANE settings as defined in the User.Profile. E.g. if you use the default SANE settings, you are informed if a division by zero occurs and you may use the debugger to locate the culpable statement.

See also the chapter «Power Macintosh» for more information on this issue

Using the Alpha Editor

MacMETH can now also be used with the Alpha Editor. The working technique when using Alpha with MacMETH is very similar to that as described in the MacMETH manual for using the MEdit editor. The major difference is that Alpha is not only more convenient, but also much more powerful than the donationware editor MEdit as distributed with MacMETH. Thus, it is recommended to use Alpha for any serious programing with MacMETH.

Alpha is available as shareware from the sites

http://www.kelehers.org/alpha/

or the really latest release

http://alphatcl.sourceforge.net/wikit

To make good use of Alpha you need some support i) for displaying compiler error messages from within Alpha and ii) for the programming in Modula-2. Both features are provided in a so-called mode of Alpha, i.e. the M2 mode. It establishes convenient communication between the MacMETH shell and Alpha under System 7 and newer and contains lots of templates and other features supporting Modula-2 programming. Highly recommended!

The mode M2 is released with the editor Alpha but is also available as a separately installable package. It is freeware and can be downloaded from the same site you obtained MacMETH. It comes also as part of the "RAMSES Extras" package. Use this link:

http://www.sysecol.ethz.ch/SimSoftware/#RAMSES_Extras

If you use Alpha version 7 or later (recommended), the mode M2 may already be installed or simply install it into Alpha by double-clicking file "READ to AUTOINSTALL M2" from the release and follow instructions for easy installation.

MacMETH should require no further adjustments, except for the keyboard short-cut Cmd^E in the MacMETH shell. By default it is associated with tool Edit2, which calls MEdit, instead of tool AlphaEdit, which calls Alpha. Changing this keyboard shortcut requires only a small modification in the "User.Profile": Either replace the "User.Profile" with the one provided with the release package of mode M2 or, alternatively, edit the existing "User.Profile" as described in the "READ to AUTOINSTALL M2" of mode M2. The latter method is only recommended, should you have already modified your "User.Profile" during previous work with MacMETH. Finally relaunch the MacMETH shell or call tool "ReadProfile" should you have the shell already up and running. Cmd^E should now call Alpha and display latest detected compiler errors or last successfully compiled file if no errors present.

Finally you can now also force MacMETH to create files which belong to Alpha (except for the files created by the compiler ending in the reserved extensions '.SBM', '.OBM', and '.RFM'). Simply use ResEdit to change the creator 'MEDT' in resource 'STR ', ID 7413 (M2 preferences) to 'ALFA' and most files created by MacMETH will now launch Alpha if double-clicked. For more information on this resource see also chapter «Documentation (Latest changes)» in this documentation.