READ ME RMSP1
RAMSES for P1 Modula-2

Andreas Fischlin, Dimitrios Gyalistras, and Oliver Gardi
February 2013

mailto:andreas.fischlin@env.ethz.ch
http://www.sysecol.ethz.ch
http://www.sysecol.ethz.ch/ramses/

What Is RMSP1?

This is a release of the RASS/RAMSES software (Fischlin, 1991) for the P1 Modula-2 language system (http://www.awiedemann.de/compiler/) running natively under Mac OS X. It allows you to compile and link any program made with and based on the 'Dialog Machine' (Fischlin et al., 1987; Fischlin & Schaufelberger, 1987) using the P1 Modula-2.

Note P1 Modula-2 conforms to the ISO Modula-2 standard (http://www.modula2.org/reference/isomodules/). Thus minor modifications of your sources might be needed in order to be able to compile your programs, e.g. Model Definition Programs made using RAMSES (Fischlin, 1991), ModelWorks (Fischlin et al., 1994) or RASS (Thoeny et al., 1994). However, you should strictly refrain from directly using any objects from the ISO libraries available from P1 and use only RAMSES objects (http://www.sysecol.ethz.ch/RAMSES/Objects). The latter are all strictly based on the Dialog Machine in order to keep your code platform independent (cf. Thoeny, 1994 available at http://www.sysecol.ethz.ch/Reports.html#20).

RMSP1 is very similar to RASS-OSX. The main difference is that RSMP1 is intended to be usable anywhere on your system and affects your system-wide installations only to a minimum. Thus RMSP1 can be installed in the usual, user-friendly manner Mac OS X users are accustomed to. Moreover RMSP1 contains means to generate an application with a simple user interface (making use of iHook). Whereas RASS-OSX is made like a traditional Unix software and any application made with it is typically used from a command line only. Note, however, both variants use the same underlying RAMSES software as contained in so-called libraries, i.e. they can use the same libRassxxx.a files plus the same commando tools for compilation, linking, and execution.

Installation

The recommended software consists of following parts:

a) RMSP1 (freeware courtesy ETH Zurich)
b) P1 Modula-2 (requires to purchase a license)
c) Xcode (freeware courtesy Apple)
d) AlphaX (shareware editor)
e) DeveloperSE (optional freeware courtesy ETH Zurich)

The following describes how to install and configure above componentes. This requires only some effort initially.

> *Note, component c) can be altered and you may use an alternative editor of your choice (you need then merely to do once a small change in a profile file). However, to enjoy fully writing and execution of Modula-2 programs as can be programmed using RAMSES I recommend to stick to above components, since they have been fully customized to work together smoothly. Notably, in a typical make edit cycle AlphaX supports display of compilation errors right in the program text without altering the file in addition to support editing in a Modula-2 specific manner by coloring and providing many other supports such as inserting entire statement snippets etc.*

Please make sure the following software is available and installed on your machine:

1) **RMSP1**: Simply copy the folder RMSP1 to your disk. Do not alter the folder structure.

2) **P1 Modula-2**: Install the P1 Modula-2 language system by copying the folder from the distribution without changing its structure to your disk. A convenient source for this is the folder 'MakingP1' on the SED, which I recommend to copy to your hard disk without changing anything. Everyone outside Systems Ecology needs to purchase a license (find further information here http://www.awiedemann.de/compiler/).

3) **Xcode**: Since gcc and other command line tools (CLTs) are needed by P1, you have to install Xcode. Xcode is available generally for free, from Apple (http://developer.apple.com/technology/Xcode.html). If you are a developer of RMSP1, Xcode will also allow you to run the RASS release machinery for RMSP1 under OS X.

Note, as of this writing the latest version of this software is designed to be used for Tiger (Mac OS X 10.4.11), Leopard (Mac OS X 10.5.8), Snow Leopard (Mac OS X 10.6.8), and Lion (Mac OS X 10.7.4). It should also run fine under Mountain Lion (OS X 10.8.x), but this was not much tested. But remember, under later OS X versions such as Lion and Mountain Lion, some restrictions apply in terms of backward compatibility. Notably Snow Leopard offers the greatest flexibility in this respect and is fully supported to make RMSP1 and/or RASS-OSX programs conveniently in universal binary form that run on almost any architecture, in particular on both PPC as well as Intel Macs and that can be executed under all mentioned OS X versions. Besides, we still support the PPC Macs running Tiger, because our cluster of simulation servers consists mostly of nodes with advanced G5 PPC processors (http://www.sysecol2.ethz.ch/intranet/SE_SimServers.html).

If you use Tiger Mac OS X 10.4.11, install Xcode 2.5. It is the last that works under this operating system. Its installation is straightforward and results in a fully functional environment to work with p1 Modula-2 as of this writing.

If you use Leopard Mac OS X 10.5.8 or Snow Leopard Mac OS X 10.6.8 (recommended over Leopard) then install Xcode 3.2.6. Its installation is straightforward and results in a fully functional environment to work with p1 Modula-2 as of this writing.

Optionally, if you wish to have full flexibility, that is to have both Xcode 2.5 as well as Xcode 3.2.6 available, install in addition to Xcode 3.2.6 also Xcode 2.5. Note, only as of Leopard (OS X 10.5.x) can several versions of Xcode co-exist on the same machine.

> *Unfortunately it seems an installation of Xcode 2.5 under Snow Leopard Server is not possible due to a small bug in the installer, which hangs at the very end. Therefore, to really have full flexibility, you need to, say temporarily, boot from Leopard and install Xcode 2.5. However and fortunately, once installed under Leopard, Xcode 2.5 is fully functional under Snow Leopard. Alternatively you can also obtain from us 'DeveloperSE', which gives you the same functionality as if you would have Xcode 2.5 on your system when using RMSP1 or RASS-OSX (utilities 'mk' and 'mk1' with option -X).*

If you use Lion Mac OS X 10.7.4 or Mountain Lion Mac OS X 10.8.x then install the latest Xcode, say 4.6. Its installation means moving the applcation Xcode.app to /Applications. We recommend to install additionally the so-called Command Line Tools (CLTs). Apple provides them now with a separate installer and this gets you the full Unix style CLTs of Xcode. Simply run the CLT installer and you should be set. However this is no longer required as of RMSP1 3.0 and RASS-OSX 3.6.0 and work with RMSP1 or RASS-OSX is not impeded in any way when skipping this step.

> **Hints**:
>
> **Xcode 2.5** is available from here (you need to register as developer, for free):
>
> https://connect.apple.com/cgi-bin/WebObjects/MemberSite.woa/wa/getSoftware?bundleID=19907
>
> All **other Xcode versions** are available from here (you need to register as developer, again for free):
>
> https://developer.apple.com/downloads/index.action

Note, under Tiger Xcode 2.5 installs the traditional Unix way, i.e. only some files are installed in the folder /Developer, but many binaries are installed in the system. As of Leopard Xcode 2.5 installs in the separate folder /Xcode2.5 only. The latter can be moved anywhere and /Xcode2.5 can even be replaced by a symbolic link pointing to the actual folder. Xcode 3.2.6 behaves similarly, except it also installs a minimum facility in the system to switch use among the coexisting Xcode versions (see also Apple's technical notes, i.e. Google for "Xcode Installation Details", notably consult section "Xcode Command-Line Tools Installation").

If you have multiple Xcodes installed, you can easily switch anytime between them. It is strongly recommended to use then RASS-OSX/RMSP1 utility 'switchp1'. Alternatively you could use also `xcode-select`*. These commands accomplish a switch*

```
switchp1 -2
sudo xcode-select -switch /Xcode_2.5        (to switch to Xcode 2.5)
```

or

```
switchp1 -9                                 (to switch to Xcode 3.2.6)
sudo xcode-select -switch /Developer
```

or

```
switchp1 -a /Applications/Xcode.app                 (to switch to e.g. Xcode 4.6)
sudo xcode-select -switch /Applications/Xcode.app
```

Note, after such a switch, your environment variables as needed by RMSP1/RASS-OSX are not immediately updated. The easiest is therefore to always launch a new shell (or source at least once more '~/.profile' or on servers '/etc/profile'). The use of the utility. 'switchp1' is highly recommended as an alternative to above switch commands using 'xcode-select', since it switches everything as needed and more robustly than 'xcode-select' and 'switchp1 -h' offers much help. It also informs you of any difficulties possibly present with your configuration and caveats to observe.

For RMSP1 to work well, in particular a switch among Xcodes as described above, you need also to allow the RMSP1 installation process as described below (step **A-2** or **B-2**, respectively) to install the provided profile extension, i.e. file '**~/my.RASS.profile**'. Once that file is in your home directory and you defined in there the minimum environment variables **M2HOME** and **M2EDITOR** according to your system, you need merely to make sure that your invisible '~/.profile' sources 'my.RASS.profile' as explained at the end of 'my.RASS.profile'. If you have currently no '.profile' installed, simply let the RMSP1 installer install one for you and all should be taken care of all with respect to proper environment variable settings. Note, particularly critical is for instance your environment variable **PATH**. If you use Xcode 2.5, it should list the Xcode 2.5 path first, e.g. 'echo $PATH' should give you something similar to this

```
.:./bin:~/bin:~/bin/RASS:/Xcode2.5/usr/bin:~/Library/RASS/bin
```

Otherwise any building is likely to fail.

4) **AlphaX**: This is an editor featuring the M2 mode. Its use is not mandatory, but highly recommended when working with RMSP1 or RASS-OSX. The M2 mode is entered automatically each time a Modula-2 source code file or a M2 auxiliary file such as a project file with extension .PRJ, is opened. The M2 mode offers essential capabilities such as displaying in the source code syntax errors as generated by the compiler and allows the programmer to conveniently compile, link, and edit programs during program development. AlphaX is a very powerful, open source maintained shareware editor, also of great value for many other purposes such as TeX document preparation, R scripting and many more uses. It is available at http://alphatcl.sourceforge.net/wiki/ and for members of the Systems Ecology group at ETH Zurich also in form of a special, fully customized release from the same internal server where this software is made available.

5) **DeveloperSE (optional)**: Copy it to your system (details below) but skip this if you are new to RMSP1.

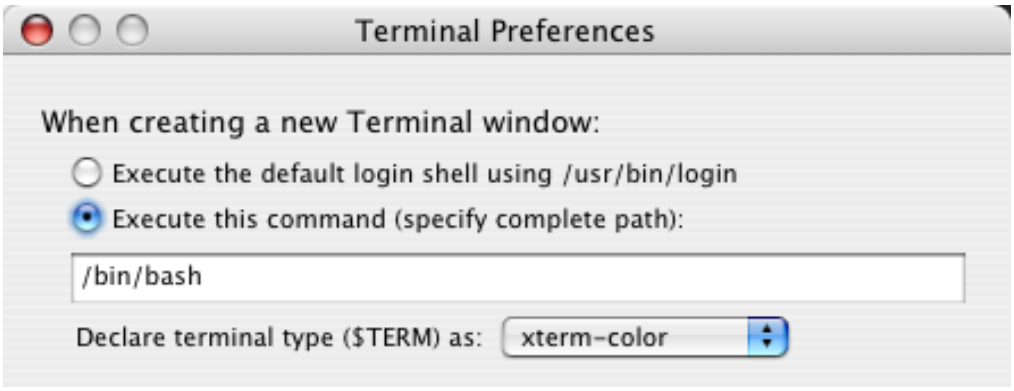
Now you are ready for performing the few last steps to complete the installation of RMSP1.

A) First Time Installation

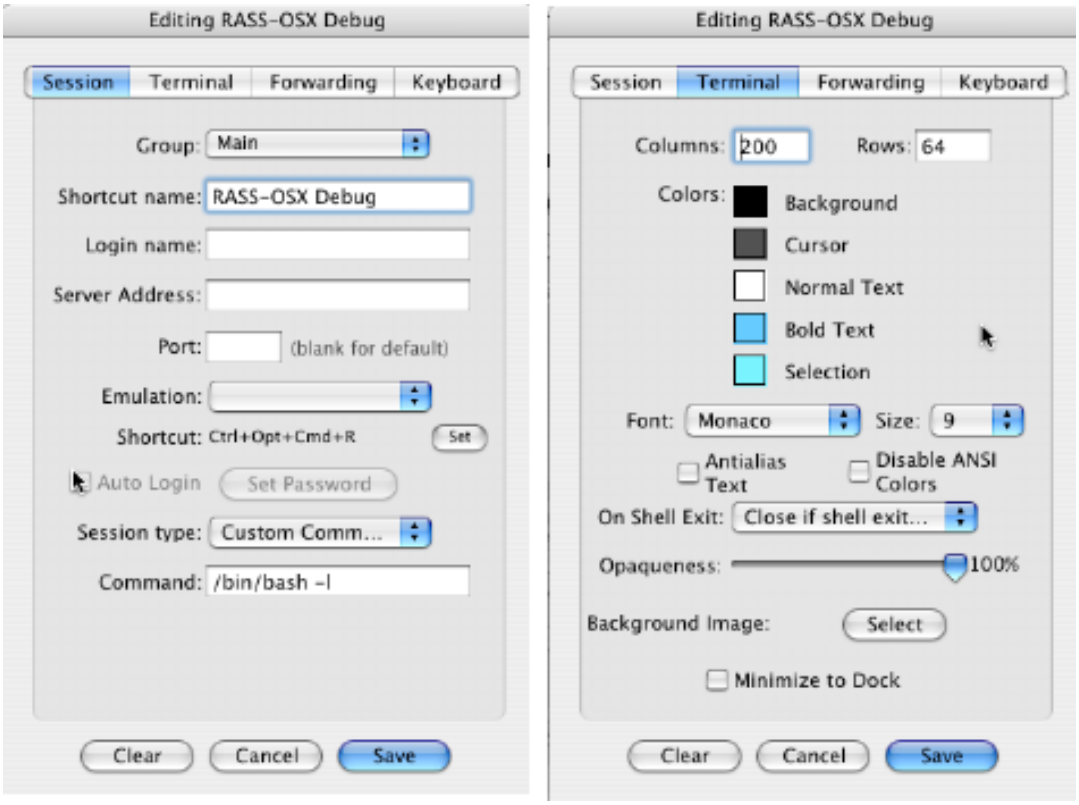
If this is not your first time installation of RMSP1 safely skip to the next step B.

Configure shell environment and Terminal: This consists of three steps A-1 to A-3:

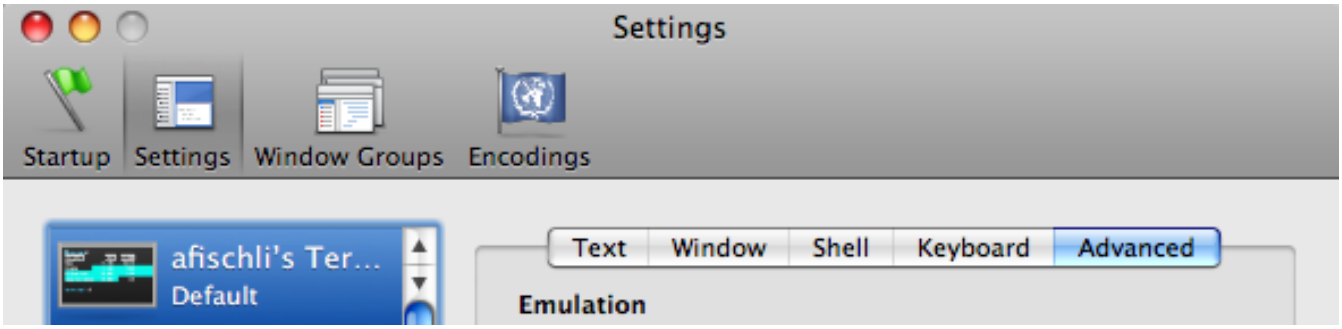
A-1) First make sure the **preferences** of your **Terminal application** are set to use the bash shell. They should look similar to this:



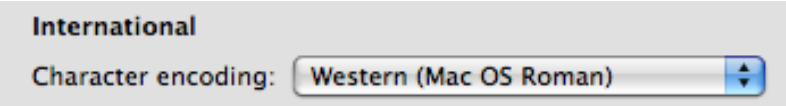
If you use Telnet Launcher (http://www.pidog.com/) it is recommended to use settings similar to the following:



Then I recommend also to set Terminal's preference for character encoding to Western Mac OS Roman, since p1 Modula-2 is made in Germany and uses internally Umlaute and other special symbols. Under Settings go to tab Advanced:



and select there from the pop-up menu Western Mac OS Roman:



A-2) Secondly run the script '**install_RMSP1.command**' from within the folder 'RMSP1' by double-clicking. Answer the asked questions accordingly to your needs. Then, open in a text editor of your choice both the file 'my.RASS.profile' as resulted from running the installer script as well as the hidden file '.profile' in your home directory. Note the hidden '.profile' needs only be opened during first installation. The purpose is to follow the instructions given at the end of file 'my.RASS.profile' and make sure '.profile' sources 'my.RASS.profile'.

> *This also means that your '.profile' is affected in a minimal way by a RMSP1 installation. If you reverse the sourcing of 'my.RASS.profile' by '.profile', this is as good as having RMSP1 deinstalled from your system (except for the copies of this software on your disk, i.e. folder 'RMSP1' and the file 'my.RASS.profile').*

In a final configuration step make sure that the values assigned to the environment variables right after the comment

```
#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-
#  BEGIN  OF  RMSP1_and-or_RASSOSX  ENVIRONMENT  DEFINITION
#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-
```

actually match the structure of your system. Most important are notably the environment variables M2HOME and M2EDITOR. Edit those paths and make sure those environment variables precisely match your installation. Although RMSP1HOME is of pivotal importance, you should refrain from defining its value, since it is automatically determined by the script 'install.RMSP1.command'. For a testing of these edits see next step.

Hint: The file '.profile' is invisible in the Finder. However, a command like 'open .profile' or 'open -a AlphaX .profile' executed from within a Terminal window may allow you to edit these files easily.

A-3) Thirdly **quit** the **Terminal application** and **relaunch** it to ensure the new .profile is read and the newly installed utilities are functional.

In general above described installation steps **A-1** to **A-3** need to be done only once, i.e. at the first installation of RMSP1. For updating a previous installation of RMSP1 see section B) below.

To test your installation and configuration, RMSP1 provides utilities. In a new terminal window run:

```
switchp1  -v
switch2using  -v
testRMSP1Installation.command
```

Remember, should you move your RMSP1 folder, rerun the installer script 'install_RMSP1.command'.

B) Updating to a New Release of RMSP1

If this is your first time installation of RMSP1 you should follow the steps described above under **A)** and you can safely skip steps **B-1** to **B-3**.

B-1) Make sure you preserve your 'Work' folder within the folder 'RMSP1' should it contain valuable sources by moving it outside of the folder 'RMSP1'.

B-2) Simply update to the new version by replacing the old folder 'RMSP1' with the new one. Then redo step **A-2**.

B-3) Move your work folder back to its original place.

B-4 optional) With some major releases, e.g. RMSP1 3.0 (RASS-OSX 3.6.0), it is recommended to also update the profile, i.e. profile extension or profile add-on 'my.RASS.profile'. The recommended

way to accomplish this goes as follows: First allow 'install_RMSP1.command' to put 'my.RASS.profile-new' in your home directory. Then edit 'my.RASS.profile-new' by merging into it from your old 'my.RASS.profile' your system specific information and your customizations (in AlphaX when both files are open: "Tools -> Compare -> Windows"). Finally replace 'my.RASS.profile'. by 'my.RASS.profile-new'.

The following explanations are only intended to trouble-shoot an installation that does not perform as expected. If all works for you, you don't need to read it. Otherwise, this is what the installation is actually doing and how the various components of RMSP1 exchange information:

Behind The Scenes (*BTS)** - ***Understanding installation/updating of RMSP1:

BTS-1 '.profile': Note, the installation script 'install.RMSP1.command' will first attempt to copy the file 'local.profile' from the RMSP1 release to your home directory and rename it to '.profile'. This happens only during a first time installation and if you have currently not installed a '.profile' in your home directory.

In case you already have such a `.profile` in your home directory, the script 'install.RMSP1.command' will ask you for permission before changing anything. If you decline permission for changes, nothing is touched but the latest versions of the files needed for the profile mechanism are stored in your home directory. This is done to allow you for a later manual fix of the installation as described in the following.

Note, by default the provided `.profile` contains almost nothing except for a mechanism, i.e. `function sourceProfileAddOn`, to source a nearby file. This is to keep any changes to an existing `.profile` to a minimum when installing RMSP1. That nearby file extending the profile for RASS-OSX or RMSP1 is named by default `my.RASS.profile` and is on purpose not hidden as `.profile` is. This is done for more convenient editing and transparency of the changes made to your system. In order to actually use `my.RASS.profile`, you need only to make sure your `.profile` sources the former, e.g. by copy/pasting function `sourceProfileAddOn` and a call to it with statement

```
sourceProfileAddOn  ~/my.RASS.profile
```

into your `.profile`. From then on the needed environment variable are made available as needed to use RASS-OSX and RMSP1, respectively. Once such a change to your `.profile` is done and `.profile` sources `my.RASS.profile`, you can forget about this and need only to consider in any future update of the RMSP1 software, whether you want to replace a newer version of `my.RASS.profile` with your current one or merge the two in case you should have done any customizations to `my.RASS.profile`.

Required as a minimum are the environment variables M2HOME and M2EDITOR. They should match your actual system and are to be defined in `my.RASS.profile`. In case you work under Leopard (Mac OS X 10.5.8 or later), you can also profit from extra environment variables such as SDKROOT. You can set these to your liking, but only towards the end of `my.RASS.profile` (location is marked). Note, other environment variables such as RMSP1HOME are also critical, but are automatically determined and stored in `my.RASS.profile` by the script 'install.RMSP1.command' depending on the folder from within which you run the script. It actually edits your `my.RASS.profile` for you. Consequently, when you move your RMSP1 folder to another location on your disk, you should always rerun the script 'install.RMSP1.command'.

That aforementioned editing is restricted to specific sections in `my.RASS.profile`. They look similar to this:

```
#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-
#  BEGIN  OF  RMSP1_and-or_RASSOSX  ENVIRONMENT  DEFINITION
#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-
```

... here are the sections that are appended/edited

```
#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-
#  END  OF  RMSP1_and-or_RASSOSX  ENVIRONMENT  DEFINITION
#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-#=-
```

Note, for a functioning RMSP1 installation the '.profile' has to define not only above listed

environment variables, but needs also to call the auxiliary script 'setenv_for_M2_RMSP1_RASS-OSX' which comes in the bin directory of the RMSP1 release. That script uses in addition to the minimum environment variables mentioned above also the temporary environment variable MYM2PATHS (allowing you to customize critical parts of environment variable PATH, should you wish to do so). Script 'setenv_for_M2_RMSP1_RASS-OSX' then defines all other environmental variables such as M2SYMS, M2LIB and notably environment variable PATH, which are all mandatory for the use of the compiler (path to program 'Modula2'; the P1 manual contains more details on this topic). Similarly many RMSP1 utilities depend on properly set environment variables. It is therefore essential you make sure your `.profile` is installed in your home directory (`~/`) and sources the RMSP1/RASS-OSX profile extension `my.RASS.profile`.

For developers there are also further environmental variables such as RAMSESDEVDIR. Some RMSP1 utilities used in a certain way, i.e. with a particular option, may require these. For instance script 'install_RMSP1.command' uses RAMSESDEVDIR if you try to install the so-called developer aliases.

Some environmental variables such as M2WORKVOL, RMSHOME, or RAMSESDEVDIR are optional. If you wish you can define them in your `my.RASS.profile` to enhance the readibility of messages regarding your installation as generated by several RMSP1 utilities. M2WORKVOL can be empty or should be set to the volume on which you normally store your Modula-2 work when these reside on a separate logical volume. RMSHOME is the RAMSES folder 'RMS' (cf. http://www.sysecol.ethz.ch/RAMSES). If you do not define them, defaults will be provided. E.g. M2WORKVOL is by default empty.

BTS-2 RMSP1 utilities: As with RMSP1 1.9 or later, the distribution of RMSP1 comes with a folder 'bin', which stores permanently all RMSP1 utilities and templates (they are no longer stored in folder `~/bin/RASS`). Thus the only parts of your system affected by the script 'install.RMSP1.command' outside of the RMSP1 folder are the files `~/my.RASS.profile` and possibly `~/.profile` (depending whether you allowed for this or not).

BTS-3 Developer aliases: If you are a developer the installation script can create symbolic links in folder `~/bin/RASS` to the master sources of the scripts, which are assumed to reside in the folder denoted by the environment variable RAMSESDEVDIR, by default '/Volumes/HD2/Dev/RASSDev/_ReleaseFiles'. This supports immediate testing while developing the scripts, since the `my.RASS.profile` defines a PATH where commands are found in `~/bin/RASS` with a higher priority than in `$RMSP1HOME/bin`. You can anytime conveniently deinstall all these aliases by rerunning the installation script and requesting a removal of these aliases (answer first to all installation questions 'no') or simply discard folder `~/bin/RASS` (cf. Deinstallation step C-4).

BTS-4 RMSP1 vs. RASS-OSX: Finally, note that RMSP1 can coexist with RASS-OSX on the very same machine. As with RMSP1 1.9 or later it makes no longer use of the same utilities but recognizes two separate sets of them. The RMSP1 utilities are stored in `$RMSP1HOME/bin`. The RASS-OSX utilities, possibly redundantly, are stored in `$RASSOSXHOME/bin`. You install RMSP1 using utility 'install.RMSP1.command', as described above. You install RASS-OSX using utility 'install_RASS_On_SimServ.command'. For your convenience this utility is not only available as part of the RASS-OSX release, but also available as part of any RMSP1 release. Note, for any reinstallation the latter utility is also available as 'doRassInstallation'.

The main differences are the following: In contrast to RASS-OSX, **RMSP1** requires a minimum of files outside the folder 'RMSP1'. The only exception of changes during installation outside the RMSP1 folder are the files `~/my.RASS.profile` as sourced by `~/.profile`. A further advantage of this architecture is the fact that ou may even have several RMPS1 folders present on your system.

In contrast to RMSP1, **RASS-OSX** is installed more in the traditional Unix style, however, here according to the more modern philosophy of OS X. While installations are done in several places those are nevertheless kept to a minimum. On a private machine, the symbol files, libraries, utilities, and templates are all installed by default in `~/Library/RASS`'(default value of env var RASSOSXHOME unless you specify it differently), which conforms to the Mac OS X philosophy. On simulation servers (http://www.sysecol2.ethz.ch/intranet/SE_SimServers.html) these files are installed in `/usr/local/lib/RASS` and `/usr/local/bin/RASS` according to the Unix tradition (environment variable RASSOSXHOME does not matter on servers).

Coexistence of RMSP1 and RASS-OSX: RASS-OSX requires no presence of a folder 'RMSP1' and vice versa. However, both require a well defined environment, i.e. files `my.RASS.profile` as sourced by hidden `.profile` in your home directory. Therefore, both installation procedures assist you to get what you need. This environment will in each case fully support the work with both RMSP1 and RASS-OSX. Note, `my.RASS.profile` will remember which method was used last and give priority to that software, i.e. either RMSP1 or RASS-OSX. After an installation the priority is given to what you have installed last. There are techniques by which you can control whether to use RMSP1 or RASS-OSX: There is always option -L of utilities 'mk' and 'mk1', to use

"temporarily" RASS-OSX, and there is utility 'switch2using' for a more permanent switching of this preference:

```
switch2using  RMSP1
switch2using  RASS-OSX
```

Note also that RMSP1 requires for technical reasons your work folder to reside within the RMSP1 folder or utilities 'mk' and 'mk1' will not work. This restriction does not apply to RASS-OSX. With the latter you can work in any folder you like. Note also, using RMSP1 it is easy to bypass this restriction: Simply create symbolic links in the work folder of the RMSP1 folder to any work directory you wish. Then your project folders can physically reside anywhere on your system you like.

If **both a RMSP1 and RASS-OSX installation are present**, scripts 'mk' and 'mk1' will use the RMSP1 files as long as you run it from a folder which resides within the RMSP1 folder. Otherwise they will fail and require you to use **option -L** (for **L**ibrary) in order to access the "more global" RASS-OSX symbol files, libraries, and templates. You can force the use of the RASS-OSX symbol files, libraries, and templates any time by using the option -L. This is particularly relevant if you have differing versions of RMSP1 and RASS-OSX, when option -L can be used to conveniently switch between the two versions.

Note, to switch also the use of the utilities and templates, use '**switch2using**' (see above). It edits the '.profile'. You are ready to use the new target system, i.e. either RMSP1 or RASS-OSX, once you have activated it (source '~/.profile') or launched a new shell (new window in application 'Terminal').

There are further **differences worth-noting**: The **RMSP1 release is basically a superset** of RASS-OSX. It contains documentations, some templates for making application bundles, and sample programs. All these are missing from the RASS-OSX installation, since they have little to do with traditional Unix based development. The only exception where RASS-OSX deviates from a traditional Unix style is the READ_ME.pdf file. It is present in both distributions, but stored in different places: In RMSP1 it is accessible via the Finder, i.e. where a user expects to find it. In RASS-OSX it is stored in the directory denoted by 'RASSOSXHOME' and is best accessed via the utility RMSP1Help.

BTS-5 Xcode repackaged as an application only: When you use only Xcode 4.3.x or later, all repackaged as an application, and have not installed CLTs, 'mk' and 'mk1' may fail because they can't find command 'make'. Simply run 'switchp1', say 'switchp1 -a /Applications/Xcode.app', at least once and this problem should be fixed. 'switchp1' creates a symbolic link '~/bin/make' to the 'gnumake' in Xcode ≥ 4.3.x (e.g. to '/Applications/Xcode.app/Contents/Developer/usr/bin/gnumake').

BTS-6 DeveloperSE: Finally, as of RMSP1 3.0 (RASS-OSX 3.6.0) there is also an optional folder '**DeveloperSE**' available. 'DeveloperSE' is needed to support 'mk' and 'mk1' with options -S and -X. 'DeveloperSE' is installed by copying the folder to your system and to edit the value of environment variable DEVELOPERSE_DIR in 'my.RASS.profile' accordingly. 'DeveloperSE' contains all from Xcode 2.5 that is needed to build backward programs back to Mac OS X 10.4.11 (Tiger) and contains all SDKs (Software Development Kits) from Mac OS X 10.4.11 to 10.8.x, offering maximal flexibility using RMSP1 3.0 or RASS-OSX 3.6.0 or later. The Xcode in 'DeveloperSE' is nicknamed Xcode 2.5SE and contains all needed CLTs such as gcc 4.0.1 etc. It preempts any need to install Xcode 2.5 in addition to Xcode 4.x, say 4.6, on more recent systems such as Mac OS X 10.7.4 (Lion) or 10.8.x (Mountain Lion) to gain full backward and flexible compatibility, including that back to Tiger.

The use of the environment variable **SDKROOT** is now also fully supported and is particularly helpful in conjuntion with DeveloperSE. It can be defined or not, this does not matter much and plays the role of a preference for a particular SDK as the user wishes. For details see below in the release notes of RMSP1 3.0 (RASS-OSX 3.6.0).

End of Behind The Scenes (*BTS) - *Understanding installation/updating of RMSP1

C) Deinstalling RMSP1

C-1) Make sure you preserve your 'Work' folder within the folder 'RMSP1' should it contain valuable files, notably your program sources or data input files, by moving 'Work' outside of the folder 'RMSP1'.

C-2) Trash the folder 'RMSP1'

C-3) Either delete '~/.profile' or restore it to its stage before installation of RMSP1 (optional step). Delete '~/my.RASS.profile' (and '~/my.RASS.profile-new' should it be present).

C-4) (If you are a developer of RASS only): Rerun 'install_RMSP1.command' actually before step C-2 and answer negatively to all questions asked. This will trigger at the end an option to deinstall first all extra utilities and templates linking your installation to RAMSESDEVDIR. In case you have not edited file '.profile' (identical to 'local.profile' in 'RMSP1HOME'), you are conveniently offered an option to trash even that as well.

To Get Started

Open the Terminal application and type

```
RMSP1Help            (or RASSHelp)
```

and visit

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

And then try the quick and dirty utility mk1, e.g. like this

```
mk1  Hello
```

while within the directory 'Work' of 'RMSP1'. Or try out a project folder, i.e. make Logistic in directory Logistic with following commands:

```
cd  Logistic  ;  mk
```

As a result the Model Definition Program Logistic.MOD will be compiled and linked with the RASS libraries and executed right away. This is the technique to work with any set of modules, as long as they reside in the same project folder or are not nested in subfolders deeper than one level. Utility 'mk' will find out about any dependencies automatically and make your programs. By default you find the simulation results in the file 'ModelWorks.DAT', ready for the RAMSES post analysis (Fischlin, 1991, http://www.sysecol.ethz.ch/publications). Enjoy!

Important Hints, Known Problems and Possible Solutions

(added in reverse chronological order)

RMSP1 3.0 / RASS 3.6.0: Most substantial release - Universal binaries can be built and run under Mac OS X from Tiger to Mountain Lion - No known bugs - Use of P1 Compiler v9.0.3:

RMSP1 v3.0 (based on RASS-OSX 3.6.0) is designed to use P1 Version 9.0.3. It supports full cross-building of RASS programs under Snow Leopard (10.6.8) back to Tiger (10.4.11). Resulting universal binaries ('mk -u', 'mk -ud') can be executed on any of these systems up to OS X Lion (10.7.4) or Mountain Lion (10.8.x). This RMSP1 3.0/RASS-OSX can also be used to with only a few restrictions in terms of backward compatibility under OS X Lion (10.7.4, tested up to Xcode 4.6) and Mountain Lion (10.8.x, tested up to Xcode 4.6).

For the first time 'mk' and 'mk1' do generate by default programs for the architecture of the host on which they are running. The philosophy to build by default always for architecture ppc has hereby been abandoned, taking into account current prevalence of the Intel architecture. Note, this was done despite the fact that many nodes of the simulation cluster run by Systems Ecology at ETH Zurich are

still PPC systems (http://www.sysecol2.ethz.ch/intranet/SE_SimServers.html). This new behavior can be overriden easily by a new option -P (for PPC) of the key utilities 'mk' and 'mk1' (previous option -P has been renamed to -H to make this possible). This new option -P ensures support for the past behavior of 'mk' and 'mk1'.

Secondly, again for the first time, do universal binaries made with the key utility 'mk' (option -u) fully function on all Mac OS X machines, including debugging without the programmer having to understand the details of compatibility among various Xcode and SDK (Software Development Kit) versions. 'mk' takes care of that. Several internal changes have made this possible. Using option -u (or -ud for debugging) is all what is needed to get fully backward compatible programs that truly work everywhere as you would expect from a reliable universal binary. Such universal binaries can be built on any system from Mac OS X 10.4.11 (Tiger) to OS X 10.8.2 (as of this writing, Mountain Lion) and run on any of these systems.

Thirdly, several previously available option combinations, that were likely to result in failures during making, are now automatically avoided by further internal changes in 'mk' and 'mk1'. Use mk' or 'mk1' with option -v to learn about the details or if some behavior of 'mk' and 'mk1' surprises you.

This release fully supports backward compatibility, i.e. it comes with a wealth of flexible options, notably all the libraries as described for previous releases (see below). These support still fully the use of Xcode 2.5 and Xcode 3.2.6 and many versions of P1 and allow for a detailed analysis of model behavior and simulation results. This backward compatibility is mostly provided for comparison reasons, e.g. when comparing previous simulation results with those obtained on new systems.

However note, for non-universal programs do still a few restrictions apply. Please clearly understand therefore the following:

(i) Any program built on Intel machines with default options, i.e. 'mk' or 'mk1 -k <program module name>.MOD', will of course fail execution on a PPC Mac (you will get error message "cannot execute binary file").

(ii) Any program built on PPC machines with default options, i.e. 'mk' or 'mk1 -k <program module name>', will of course fail execution on an Intel Mac not running Rosetta, i.e. notably under Lion (OS X 10.7.4) or Mountain Lion (OS X 10.8.x). Apple has unfortunately dropped support for Rosetta for latter operating systems.

(iii) Any PPC, non-universal program built with option for debugging and that causes a division by zero hangs on an Intel Mac when run under Rosetta (Snow Leopard 10.6.8, or Leopard 10.5.8). This is the fault of Rosetta and **not** of our software. However, since Rosetta is no longer supported by Apple, there is nothing we can do about this and this bug is destined to stay with us.

If you encounter this problem, your program will hang and becomes behind the scenes very busy and uses up a great fraction of your machines CPU power (as a result your fan on a portable Mac may start spinning at very high speed). You need then to kill your program, e.g. with the Activity Monitor or by another remote shell using command 'kill -9 <process#>' (after having learned with 'ps -a' about its <process#>). Note also, that such a program can never be debugged as intended. We recommend then to rebuild using 'mk' with options -nud and you should be able to easily locate the code that causes the division by zero on any Mac you wish to use. Note, any other exceptions than division by zero are handled normally, i.e. according to the IEEE floating point standard and the RASS software specifications (in particular see module DMFloatEnv). This means they can all be always debugged as intended.

Remember, this is only true when you use P1 9.0.3. When you use older compilers other exceptions than only division by zero can cause a hanging program that needs to be killed by you manually. Remember, utilities 'mk' and 'mk1' warn you about these facts, in particular when using option -v.

(iv) The P1 compiler version 9.0.3 contains still a few limitations that may matter for anyone deeply delving into numerics. This concerns only three aspects:

a) NaNs as produced by run time exceptions are still not encoded, i.e. there is only NaN(000). Alhtough you can esaily input other NaNs, say by using DMConversions.UndefREAL() = NaN(027) (http://se-server.ethz.ch/RAMSES/Objects/DM/DMConversions.html#107_18), backward tracing of encountered exceptions by distinguishing between original causes of the ecxeption as specified by the IEEE standard for floating point arithmetic (IEEE 754) is still not possible. This means for instance, the result of sqrt(-1) is not NaN(001), but NaN(000) or the result of 0/0 is not NaN(004) but only the very same NaN(000);

b) some results of expressions involving INF values can be questioned, i.e. the value of an expression such as (+INF) + (-INF) is on PPC +INF instead of -NaN (Sun -NaN(255), MacMETH -NaN(002), P1 Intel -NaN(000)). This is, however, rather a problem of the RISC architecture than one of the compiler alone;

c) when adding a real value to MAX(LONGREAL) the result is not always +INF as it should;

Please note, these restrictions are not new or specific to P1 9.0.3. P1 compilers have always had these limitations as far back as we have used and analyzed their results, i.e. P1 8.2. Note also, the Intel code as produced by P1 v9.0.3 is very good in these respects. According to our analysis results are almost identical to those one obtains on the numerically very reliable Suns or on older Macs using the excellent SANE (Standard Apple Numerics Environment, which is, by the way, still fully supported by MacMETH, also from RAMSES). On the other hand, remember, above limitations mean also that your results may slightly deviate when you compare them with the same results obtained by using RASS-Sun or MacMETH (see http://www.sysecol.ethz.ch/ramses for more details on RAMSES, the big software environment to which RASS belongs).

Note also that this release comes for the first time with the utility 'symdiff' and the powerful tool 'SymDiff'. 'SymDiff' allows you to compare pairwise symbols, notably numbers, between pairs of text files. The comparison is done in a meaningful manner, i.e. not done in a primitive lexical comparison as done by Unix tools such as 'diff'. Instead 'SymDiff' considers the actual numerical values. Say a numerical result such as 0.99999999999999 as written to a first file can then be compared in its actual numerical difference to 1.0 as written to a second file. A huge difference when comparing only strings, a small one when comparing actual values. At the end statistics are computed that allow you to easily locate true numerical deviations and get a solid overview. We use this extensively to test all our software before releasing and to compare any previously obtained simulation results with new ones after having changed a complex simulation model (e.g. Fischlin, 1991, http://www.sysecol.ethz.ch/publications).

(v) The C backend does not work so well that it could be used freely. Making is generally possible (except for older compilers such as P1 8.2, where making fails). When you use the C backend best observe following rule: To obtain normally functioning programs use always 'mk -b' on PPC Macs and 'mk -bP' on Intel Macs. This means also, on Intel Macs, execution requires Rosetta and therefore C backend programs can't be run on Lion or newer OS X systems. Finally, with any other options, be warned, you may have difficulties to link or the built program may simply produce no results (and no error messages either) or only confusing error messages.

Some new features have also been added, notably options **-S** (**use external SDK**) and **-X** (**use "DeveloperSE Xcode CLT tools"**) for the key utilities 'mk' and 'mk1' and the environment variables **DEVELOPERSE_DIR** and **SDKROOT.** Both options -S and -X require the presence of the so-called '**DeveloperSE**' folder on your system. The latter is optional and the options -S and -X are then simply ignored. 'DeveloperSE' serves following purposes:

1) '**DeveloperSE**' complements Xcode installations by providing a **collection of SDKs** and a set of key CLTs, the so-called "**DeveloperSE Xcode CLT tools**" as a replacement for Xcode 2.5. As a consequence from copying 'DeveloperSE' to your system and making sure the value of environment variable **DEVELOPERSE_DIR** in 'my.RASS.profile' matches the actual location, you have following advantages:

First, you can build a universal binary program for Mountain Lion under an older Mac OS X, e.g. a Snow Leopard system ('mk -S' or 'mk1 -S' while SDKROOT=MacOSX10.8.sdk) or you can build a universal binary program under Mountain Lion for a Tiger system ('mk -u')!

Secondly, you can use the DeveloperSE Xcode CLT tools instead of the currently active Xcode without having to switch anything ('mk -X' or 'mk1 -X'). This is important if a switch to a full Xcode 2.5 is wanted on a server system used by someone without administration privileges. Using 'xcode-select' requires administrator privileges, but using 'mk' or 'mk1' with option -X does not.

2) **SDKROOT** does not need to be defined, can be empty, or have a value similar to one of the following three forms:

- /Developer/SDKs/MacOSX10.7.sdk	to link with a specific SDK
- MacOSX10.8.sdk	to link with a SDK for a specific OS (regardless where it is coming from)
- /Developer/SDKs	to link with a specific SDK collection possibly outside of Xcode

3) **Option -S vs. SDKROOT**: Note, option -S is the strongest and is used, even if SDKROOT defines also a path. Option -S uses always the path given by DEVELOPERSE_DIR and can be freely combined with other options such as -t or -I. Otherwise SDKROOT is honored as much as possible. If it contains a path, that path is given higher priority than the path to the SDKs from the currently used Xcode. If you want to avoid that, simply use option -s and any use of SDKROOT is suppressed. These priority rules give you much flexibility and you can easily use several sets of SDKs, notably SDKs that are not available in the current Xcode. With this mechanism you can link with the SDK for Mountain Lion while using Xcode 3.2.6 or link with SDK for Tiger while using Xcode 4.6. Apple might not like this, but it works!

Write us if you are interested in 'DeveloperSE'.

System requirements: Following system minimum configurations are recommended: Mac OS X 10.4.11 (Tiger), Xcode 2.5; Mac OS X 10.5.8 (Leopard), Xcode 3.2.6; Mac OS X 10.6.8 (Snow Leopard), Xcode 3.2.6; Mac OS X 10.7.4 (Lion), Xcode 4.6; Mac OS X 10.8.2 (as of this writing, Mountain Lion), Xcode 4.6. On Mac OS X 10.7.4 or 10.8.2 it is recommended to also install the Xcode CLTs (Command Line Tools), but this is not really needed. For backward comatibility on Mac OS X 10.7.4 or 10.8.2, i.e. to be able to build universal binary programs that also run under Tiger, you need to install also Xcode 2.5 and/or our 'DeveloperSE' folder. It is in general recommended to install on all systems 'DeveloperSE' (and/or Xcode 2.5), since it offers maximum flexibility in terms of compatibility among all aforementioned Mac OS X. 'DeveloperSE' contains the CLTs from Xcode 2.5 (it is therefore nicknamed Xcode 2.5SE) and all SDKs (Software Development Kits) for all aforementioned Mac OS X. With all these systems it is recommended to use always AlphaX 8.2 as the editor.

Wrap up: Apart from above listed few exceptions, this RMSP1 3.0 and/or RASS-OSX 3.6.0 release is for the first time a release that fully supports productive work in a most powerful, yet flexible manner and by default without any restricitions or known bugs.

On Intel Macs native and universal binary code building of RASS programs under Snow Leopard (10.6.x) is now working correctly for the first time:
RMSP1 2.9 (based on RASS-OSX 3.5.9) is designed to use P1 Version 9.0.2 on current Intel Macs using latest Mac OS X (as of this writing 10.6.7 with Xcode 3.2.6) and on PPC Macs under Tiger 10.4.11 (Xcode 2.5). Backward compatibility is still fully provided, also thanks to the wonderful support by the author of the P1 compiler Albert Wiedemann. With two exceptions (described below) there are no known bugs and all RASS utilities have been enhanced considerably to fully support development with this new compiler.

Note, this includes cross-development among above platforms. The latter merely requires you have on the Intel Mac installed (i) Xcode 3.2.6 and (ii) both RASS releases, the standard as well the T-release (on top of each other). On the PPC Mac typically running Tiger 10.4.11 this requires to have the RASS T-release installed and Xcode 2.5. Notably universal binaries can be built on such systems that run on each other. Details how to actually accomplish this are given below.

Note, all the features that have been introduced in RMSP1 2.8 based on RASS 3.5.8 are available and have been thoroughly tested and found to work correctly. There are two exceptions though: First, any RASS programs made with option -f (fast, full code optimization) for the Intel platform do not always produce correct results. Our understanding is due to a problem with the new compiler P1 9.0.2 and therefore we discourage the use of option -f. Second, C back end RASS libraries do not function correctly on the Intel platform. For all other uses of this software there are no known bugs (and several have been fixed).

This is actually a major release representing a major step forward in having the RASS software fully available for the first time on the Intel platform. Yet the version increment is only from 3.5.8 to 3.5.9 since some smaller issues have remained unresolved (see above). The next release 3.6.0 should fix this.

For full backward compatibility under Tiger (option -t for RASS utilities 'mk' and 'mk1') there may be some renaming of P1 libraries necessary. Please install P1 9.0.2 such that the OS X 10.4. libraries are named as follows

P1 released name	RASS needed name
libm2lib.a	libm2Tlib.a
libm2dlib.a	libm2Tdlib.a
libm2libcarbon.a	libm2Tlibcarbon.a
libm2dlibcarbon.a	libm2Tdlibcarbon.a
libm2clib.a	libm2Tclib.a
libm2clibcarbon.a	libm2Tclibcarbon.a

Thanks to this renaming you can put these backward compatible libraries (made with compile option -NOXCODE3) all in the same folder, i.e. "../MakingP1/Modula2/lib" together with the more recent P1 libraries. The option -t works with the RASS utilities 'mk' and 'mk1' only if all libraries reside in the same folder. Note, you don't need to resort to Xcode 2.5 to work with these libraries. All you need is using option -t with RASS utilities 'mk' and 'mk1' to link with these libraries and the resulting program, e.g. a universal binary one, will work fine under Tiger PPC Macs. A typical command to build a universal program from an entire Modula-2 project may look like this:

```
mk -ut
```

The resulting Makefile can then be reused in subsequent edit-build-execute cycles by simply entering

make ; <myprog>

or if your program needs to know where project specific files reside

make ; mk -x

Finally note that with this version of RASS comes still the S-release that allows to work with P1 8.2. The S-release is provided mostly for enabling comparison between previously obtained results with those obtained with the new standard release of RASS. However, to work with the S-release, you need to have also Xcode 2.5 installed on the Intel Mac. Note, this release supports many software combinations: Maximum flexibility is available if you have Xcode 2.5 and 3.2.6 installed plus all three RASS releases, i.e. the S-, T- and standard release, on top of each other. Then as of this RASS you can develop RASS programs with following combinations of software components: (i) P1 8.2 and Xcode 2.5 ('switchp1 -2'), (ii) P1 9.0.2 and Xcode 2.5 ('switchp1 -95'), (iii) P1 9.0.2 and Xcode 3.2.6 ('switchp1 -9'). Target architectures can be PPC or Intel and target Mac OS X are all possible from Tiger 10.4.11 ('mk', 'mk1' option -t), Leopard 10.5.8 ('mk', 'mk1' default), and Snow Leopard 10.6.8. Programs made for Snow Leopard should also run under Lion 10.7.x, given they were made in the universal binary ('mk', 'mk1' option -u) or for the Intel architecture ('mk', 'mk1' option -i). Note, on simulation servers P1 8.2 and the S-release are preinstalled and can be used with option -2 of RASS utility 'mk'. Some further details on how to use the S-release:

- Remember, with P1 8.2 you should only build for the PPC platform.

- Moreover, running PPC code on an Intel Mac under Rosetta means as reported earlier that no debugging is possible and a crashed program is likely to lead to a hang that needs to be killed explicitly (e.g. by using Apple's utility 'Activity Monitor'). This appears to be due to bugs in Rosetta and since Apple is no longer supporting Rosetta (e.g. omitted from Mac OS X Lion 10.7.x) a fix will most likely never become available.

- While using P1 8.2 any attempt to build for Intel (option -i) or using the C back end (option -b) is discouraged and is likely to fail. For such uses we strongly recommend to use only P1 9.0.2 with RMSP1 3.5.9 and/or RASS-OSX 3.5.9.

Spurious error message «no longer needed -mlong-branch» with option -t

Option -t with RASS utilities 'mk' or 'mk1' produces PPC programs that are backward compatible with Tiger (Mac OS X 10.4.11). However, when using a standard Xcode 3.x.y under Snow Leopard following confusing error message is generated:

```
ld: warning: object file compiled with -mlong-branch which is no longer needed. To
remove this warning, recompile without -mlong-branch: /Developer/SDKs/MacOSX10.4u.sdk/usr/
lib/crt1.o
```

This warning can easily be avoided by replacing in your Xcode the file

```
/Developer/SDKs/MacOSX10.4u.sdk/usr/lib/crt1.o
```

with the version provided in RASS releases as of 3.5.8 or later in folder '/Volumes/RMSP1 v2.8/RMSP1/To Install/Xcode 3.x.y Fix/crt1.o'. Thanks to Bernhard Baehr for the hint and solution provided at http://lists.apple.com/archives/xcode-users/2009/Sep/msg00209.html

Use of P1 Version 9.0.1: Making under OS X Tiger (10.4.11), Leopard (10.5.8), and Snow Leopard (10.6.x) on PPC as well as Intel Macs:

RMSP1 2.8 (based on RASS-OSX 3.5.8) is designed to use P1 Version 9.0.1 on current Intel Macs using latest Mac OS X (as of this writing 10.6.7) and on PPC Macs under Tiger 10.4.11. RASS utility 'switchp1' works now also on simulation servers and allows to switch between P1 8.2 and P1 9.0.1 on the simulation server cluster at ETH Zurich (access restricted). In addition to the libraries available in RASS 3.5.7, RASS 3.5.8 comes with a so-called Tiger release (T-release).

RASS releases come therefore in three variants with following libraries:

Simple S-release of RASS-OSX (PPC only, P1 8.2, Xcode 2.5):

libRass357S_x.a	MSYMs (PPC only)
libRass357S.a	Standard library (full checks, SLOW, PPC only)
libRass357Snc.a	Fully optimized (minimum of checks, FAST, PPC only)

Universal binary T-release of RASS-OSX (P1 9.0.1 or later, Tiger or later, Xcode 2.5)

libRass357Tppc_x.a	MSYMs for target architecture ppc (PPC) = libRass357ppc_x.a
libRass357Ti386_x.a	MSYMs for target architecture i386 (Intel) = libRass357i386_x.a
libRass357T.a	Standard library (full checks, SLOW, universal binary)
libRass357TC.a	C back end library (universal binary)
libRass357Tnc.a	Fully optimized (minimum of checks, FAST, universal binary)
libRass357TncC.a	as previous but made with C back end (universal binary)

Universal binary standard release of RASS-OSX (P1 9.0 or later, Tiger or later, Xcode 3.x.y)

libRass357ppc_x.a	MSYMs for target architecture ppc (PPC) = libRass357Tppc_x.a
libRass357i386_x.a	MSYMs for target architecture i386 (Intel) = libRass357Ti386_x.a
libRass357.a	Standard library (full checks, SLOW, universal binary)
libRass357C.a	C back end library (universal binary)
libRass357nc.a	Fully optimized (minimum of checks, FAST, universal binary)
libRass357ncC.a	as previous but made with C back end (universal binary)

The standard or default release was made using Xcode 3.2.6, the two others with Xcode 2.5. The new T-release contains library modules that were all compiled with the same compiler as the standard release, i.e. P1 9.0.1, but with option -NOXCODE3 (since Xcode 2.5 is in use).

Again as with RASS 3.5.7 all three releases can coexist and can be easily installed on top of each other, e.g. using the installer script with option s, t or none (cr) or then using RASS utility 'install_RASS_On_SimServ.command ' (= doRassInstallation) three times, each time with another option (s, t, default). If all variants are installed, the result is three variants of MSYMs (for location see below).

Installing all three release variants on top of each other results in 3 variants of MSYMs:

Simple S-release of RASS-OSX (PPC only, P1 8.2, Xcode 2.5):

MSYMs	~/Library/RASS/SYMs (PPC only)

Universal binary T-release of RASS-OSX (P1 9.0.1 or later, Tiger or later, Xcode 2.5)

MSYMs	~/Library/RASS/SYMs/ppc	= standard release ppc MSYMs
	~/Library/RASS/SYMs/i386	= standard release i386 MSYMs

Universal binary standard release of RASS-OSX (P1 9.0 or later, Tiger or later, Xcode 3.x.y)

MSYMs	~/Library/RASS/SYMs/ppc	= T-release ppc MSYMs
	~/Library/RASS/SYMs/i386	= T-release i386 MSYMs

Note, the two universal binary MSYMs as well as the corresponding _x.a archives contain redundantly identical MSYMs. However, needed are only two sets, one for ppc, one for i386 target code, i.e. since identical you work with the last installed (for location see above). Each release comes with what is needed for that release, thus there are all in all actually only 3, not 5 variants of MSYMs.

RASS utilities 'mk' and 'mk1' are aware of all these MSYMs and libraries and use them, depending on which platform you are currently working, which P1 and/or which Xcode you currently use (use commands 'switchp1 -v' and 'switch2using -v' to learn about P1 and Xcode currently active). Use RASS utility 'switchp1' to change P1 and/or Xcode. Cross development including backward compatibility (option -t) is supported on both platforms. E.g. make an Intel program under Tiger on a PPC Mac and execute it on an Intel Mac after transfer under Mac OS X Snow Leopard (as of this writing 10.6.7) and vice versa.

RASS utilities 'mk' and 'mk1' of RASS 3.5.8 do now for the first time offer building of universal binary programs (option -u). However, not all compilation and linking variants (options -d, -i, and -b or a combination thereof) have been tested fully as of this writing, since some libraries for the underlying P1 9.0.1 are not available for full backward compatibility. The new option -u in combination with option -t (uses T-release libraries) provides particular nice support for crosscompilation. A program made with 'mk -tu' or 'mk1 -tu <program>' runs natively on the respective platform, i.e. any PPC G5 node of our simulation cluster (http://www.sysecol2.ethz.ch/intranet/SE_SimServers.html) under Tiger Mac OS X 10.4.11 or on any newer Intel Mac. However, execution of RASS Intel code is not yet really possible, since adjustments need to be made first. The next major release RASS 3.6.0 will offer this, while retaining the structure and organisation of RASS 3.5.8 releases.

If using 'install_RASS_On_SimServ.command ' (= doRassInstallation) sources and MREF files of all RASS modules can be installed as well (access restricted to RASS developers, not publicly available) and all RASS modules can be fully debugged. Note, for backward compatibility reasons with previous releases, the RASS utility 'doRassInstallation' (='install_RASS_On_SimServ.command') installs in two

locations redundantly the same set of MODs.

All three releases share the same RASS module sources
MODs ~/Library/RASS/Srcs/rass358/SLOW/*/*_MOD
~/Library/RASS/Srcs/rass358/FAST/*/*_MOD

Instllaing all three release variants on top of each other results in 10 variants of MREFs:

Simple S-release of RASS-OSX (PPC only, P1 8.2, Xcode 2.5):
MREFs ~/Library/RASS/Srcs/rass358/SLOW/*/*_MOD
~/Library/RASS/Srcs/rass358/FAST/*/*_MOD

Universal binary T-release of RASS-OSX (P1 9.0.1 or later, Tiger or later, Xcode 2.5)
MREFs ~/Library/RASS/Srcs/rass358/SLOW/T/MREFs/ppc
~/Library/RASS/Srcs/rass358/SLOW/T/MREFs/i386
~/Library/RASS/Srcs/rass358/FAST/T/MREFs/ppc
~/Library/RASS/Srcs/rass358/FAST/T/MREFs/i386

Universal binary standard release of RASS-OSX (P1 9.0 or later, Tiger or later, Xcode 3.x.y)
MREFs ~/Library/RASS/Srcs/rass358/SLOW/MREFs/ppc
~/Library/RASS/Srcs/rass358/SLOW/MREFs/i386
~/Library/RASS/Srcs/rass358/FAST/MREFs/ppc
~/Library/RASS/Srcs/rass358/FAST/MREFs/i386

The RASS utilities 'mk' and 'mk1' are aware of this and generate automatically a debug.dat that contains a path to the correct set of MREFs.

Use of P1 Version 9.0: Full making under OS X Snow Leopard (10.6.x) on Intel Macs:
RMSP1 2.7 (based on RASS-OSX 3.5.7) is designed to use P1 Version 9.0. This supports generation of code for both architectures, i.e. PPC as well as Intel (i386). It is possible to generate Intel code (option -i for RASS utilities 'mk' and 'mk1'). Debugging has now become again possible fully under latest Snow Leopard (Mac OS X 10.6.x, as of this writing 10.6.7) with both types of code generation. In case of ppc code this requires on an Intel Mac of course the presence of Rosetta (be aware of Lion, Mac OS X 10.7.x). In case of i386 code execution requires of course an Intel Mac. RMSP1 2.7 is backwards compatible and still supports the use of P1 8.2. Use RASS utility 'switchp1' to switch between P1 versions.

The release comes in two variants with following libraries:

Simple S-release of RASS-OSX (PPC only, P1 8.2, Tiger compatibility):
libRass357S_x.a MSYMs (PPC only)
libRass357S.a Standard library (full checks, SLOW, PPC only)
libRass357Snc.a Fully optimized (minimum of checks, FAST, PPC only)

Universal binary UBI-release of RASS-OSX (PPC, Intel, P1 9.0 or later, Leopard or later) (default)
libRass357ppc_x.a MSYMs for target architecture ppc (PPC)
libRass357i386_x.a MSYMs for target architecture i386 (Intel)
libRass357.a Standard library (full checks, SLOW, universal binary)
libRass357C.a C back end library (universal binary)
libRass357nc.a Fully optimized (minimum of checks, FAST, universal binary)
libRass357ncC.a as previous but made with C back end (universal binary)

The simple S-Release of RASS-OSX corresponds to the previous releases of RMSP1. The second variant is the new release featuring universal binary libraries. Note, the two variants can coexist and can be easily installed on top of each other, e.g. using the installer script with option S or using RASS utility 'install_RASS_On_SimServ.command ' (= doRassInstallation) twice, once with once without the option simple variant.

RASS utilities 'mk' and 'mk1' are aware of all these libraries and use them, depending which P1 is in use and depending on the options by which they are called (-f for fully optimized code; -b for C back end; -t for full backward compatibility under Tiger). Use 'mk' -h or 'mk1' -h (-h for help) respectively to get more details on what the new RASS utilities 'mk' and 'mk1' can do. Note, these utilities suppress the execution of programs that would lead to a crash. Yet, it is possible to make an Intel program under Tiger on a PPC Mac and execute it on an Intel Mac after transfer under Mac OS X Snow Leopard (as of this writing 10.6.7) and vice versa (if option -t was used).

Stable support for making under OS X Snow Leopard (10.6.x) on Intel Macs (22.May.2011):

RMSP1 2.6 (based on RASS-OSX 3.5.6) supports development under latest Snow Leopard (Mac OS X 10.6.x, as of this writing 10.6.7) on Intel Macs even better than its predecessors.

This means that for both Mac platforms, i.e. PPC Macs as well as Intel Macs, you can develop and execute RASS programs that work flawlessly. If you are developing on a PPC Mac the full functionality, i.e. including debugging, is available. Several other improvements and bug fixes were made such as suppressing requesting for sources by the debugger and there are currently no known bugs. Therefore P1 8.2 together with RMSP1 v2.6 can be considered a very stable and a fully functional development environment on PPC for PPC RASS programs. On Intel Macs development of PPC RASS programs (default) is also fully functional except for debugging. The RASS utilities 'mk' and 'mk1' have been improved for safer use, i.e. they suppress now the execution of a program that would just hang. This supports also a more convenient use of option -d (debugging) for cross-development on Intel Macs for later execution on a PPC Mac.

The execution of PPC programs made with RMSP1 2.6 is basically possible on both platforms regardless on which platform it was made. Note however, on Intel Macs the execution depends on Apple's Rosetta. Rosetta is for sure available up to Mac OS X 10.6.x (Snow Leopard) but is likely to be no longer available under later versions of Mac OS X. As of this writing rumors say that OS X 10.7.x (Lion) will no longer offer Rosetta.

RMSP1 2.6 can basically be used with all P1 versions currently available, i.e. P1 8.2, 8.3, 8.4 and 9.0. The new RASS utility 'switchp1' facilitates the use and coexistence of several P1 versions. However, using any other P1 later than version 8.2 on Intel Macs, is only partly supported, of an experimental nature and restrictions apply:

The use of P1 v8.3 is not recommended except for experimentations perhaps of interest for RASS developers.

P1 8.4.1 is of greater interest. RMSP1 2.6 fully supports the making of RASS and RASS programs on Intel Macs via the backend C code generation. However, the resulting code is incompatible with Rosetta and leads to a hanging DM at startup time. Only simple M2 programs not using RASS can be executed successfully. Therefore release 2.6 of RMSP1 offers no Intel (i386) variant of the RASS library and comes only with a PPC variant.

P1 9.0 (compiler 9.0b4) features for the first time Intel native code generation and debugging for the Intel Mac platform. RMSP1 2.6 supports basically the use of P1 9.0 and plain Modula-2 programs can be made for both the PPC and Intel architecture. However, Intel programs contain bugs, probably due to errors in the library that comes with P1 9.0. Consequently this release of RMSP1 comes still without an Intel variant of the RASS library and therefore Intel RASS programs can't be linked. Whereas PPC RASS programs can be made successfully with this RMPS1 using P1 9.0, this is of no practical use. All resulting programs lead to a bus error on both platforms when executed (probably due to an incompatibility between the two involved P1 versions, since the RASS library coming with this RMPS1 was made with P1 8.2).

To wrap up: It is best to continue using P1 8.2 with this release of RMSP1, regardless whether you are working on PPC or an Intel Macs and to generate PPC code only (default). As long as no run-time errors occur the resulting programs work flawlessly on both platforms. For programs that do not crash and if you can do without debugging, this software combination is highly recommended on all OS X Macs. If debugging becomes necessary, you have to resort to PPC Macs.

The next release of RMSP1 is planned to offer full functionality on latest Intel Macs using P1 v9.0. The goal is to enable the making of PPC as well as Intel RASS programs. The making of universal binaries should then become possible for maximum compatibility among all OS X platforms.

 **Making possible under OS X Snow Leopard (10.6.x) on Intel Macs**:
RMSP1 version 2.4 (based on RASS-OSX 3.5.4) supports development under latest Snow Leopard (Mac OS X 10.6.x) on Intel machines. However, the same limitations apply as described below. I tested this with latest Xcode 3.2.2 (as of 20.Apr.2010) while still having also Xcode 2.5 installed (note, you need both Xcode versions to be present on your system: Xcode 2.5 in `/Xcode2.5/` and Xcode 3.x.y in `/Developer/`; the installer of Xcode 3.x.y installs the needed utility as `/usr/bin/xcode-select`). The only imiportant thing for any RMSP1 or RASS-OSX work is to execute first following commands (after each logout or system restart):

```
sudo xcode-select -switch /Xcode2.5 ; source setenv_for_M2_RMSP1_RASS-OSX /Developer
```

 **Making possible under OS X Leopard (10.5.x) on Intel Macs, but execution remains limited**:
RMSP1 version 2.4 (based on RASS-OSX 3.5.4) makes it possible to develop under latest Leopard (Mac OS X 10.5.x) on Intel machines. However, considerably limitations exist if RMSP1 or RASS-OSX programs are run on Intel machines under Leopard: Exceptions and debugging are all non-functional, because:

- Any program made with RASS utility 'mk' using option -d crashes during startup with a bus error
- any exception handling, notably standard run-time errors, result in a hanging program

The latter can't be killed from the console anymore in which you are working. You need to open another session in application Terminal and kill the hanging process with

```
kill -9 <process id>
```

Ex.: Assuming program RangeErr hangs and you fail to kill it, neither with CTRL-C nor CTRL-\ do you succeed. Then open another Terminal window (shell) and enter:

```
afischli$ ps -a | grep RangeErr | grep -v grep
22517 ttys005     0:05.15 RangeErr
afischli$ kill -9 22517
```

Note, in contrast to previous releases of RASS-OSX and RMSP1 it does not matter whether you create your program on a PPC machine under Tiger or on an Intel machine under Leopard. If run on an Intel machine under Leopard all programs show the same behavior as described above. This means as of this release you have the advantage to be able to fully develop programs under all systems up to Leopard on Intel machines. However, running RASS-OSX or RMSP1 programs is only fine as long as they encounter no run-time errors and you can forgo debugging. If not, you need to work on a Tiger system until your application is fully debugged. As of this release p1 Modula-2 is available as version 8.3 (http://www.awiedemann.de/compiler/), but RASS-OSX / RMSP1 are not using this version of p1 Modula-2 yet.

If working with Leopard (Mac OS X 10.5.x), make also sure that you follow carefully the installation instructions as described in this document. Otherwise linking will most likely fail. Successful making including compilation, linking, and editing cycles of Modula-2 programs was possible as of this writing under Mac OS X 10.5.7 using Xcode 2.5 and 3.1.3, p1 Modula-2 8.2, and AlphaX 8.2b13 with M2 mode 4.7.1.

OS X Leopard (10.5.x) and related issues:
RMSP1 version 2.3 (based on RASS-OSX 3.5.3) works only under latest Mac OS X 10.5.x (Leopard) if you use Xcode 2.5. Fortunately Xcode 2.5 runs under both Mac OS X, i.e. Tiger (10.4.x) as well as Leopard (10.5.x). Compiling and linking using the new Xcode 3.0 (latest for Leopard as of this writing) can not be used and any resulting application will crash with a bus error or other fatal exception during startup. M2 Mode 4.6.1 or later together with AlphaX 8.2ad4 or later work well under both Tiger and Leopard.

Important hints on RMSP1 M2 Mode interaction:
While working with AlphaX 8.2.x it is recommended to create first a folder 'AlphaPrefs' within the folder where AlphaX resides if you are a user with administrative privileges. This will not only allow you to work smoothly with several AlphaX applications on the same system, but will also further facilitate the interaction between RMSP1 and AlphaX.

Should you encounter difficulties with this interaction, you need to understand its purpose: The M2 mode of AlphaX can only open the recently compiled files and display possibly present compilation errors if the compiler generated error output can be processed by the M2 mode (Command Ctrl-0, Open M2 Work Files). This requires the Rass utilities ('mk', 'mk1' as well as the 'make' using the 'Makefile' generated by 'mk') to know about a folder where to exchange that information with the M2 mode. It is stored in auxiliary files 'M2_ErrListP1.DOK', 'M2_err.ALPHA', and 'M2_err.LST'. Unless folder specifications match exactly, the M2 mode does not manage to open the Modula-2 files you ar working with, nor can it properly display compilation errors and warnings in AlphaX. The new M2 mode (>= 4.6.1) will follow a more robust strategy than in the past and uses the value of the AlphaTcl variable $PREFS to store the auxiliary files in. To match this the RASS utilities 'mk' and 'mk1' as of RMSP1 2.2 or later follow a similar behavior and use the 'AlphaPrefs' folder if present (see above), or they use by default the AlphaTcl Cache folders within the ~/Library/Preferences folder (AlphaX <= 8.1.x uses folder '~/Library/Preferences/Alpha-V8', AlphaX >= 8.2.x uses ~/Library/Preferences/Alpha-v8.2a1).

Note, option -v of 'mk' and 'mk1' informs about the used folder (m2_P1AuxFileCacheFolder). Note also, the following preference (m2_P1AuxFileCacheFolder) of the M2 mode specifies the used folder (in the middle) as shown in this example:

These techniques allow you to easily check whether folder specifications match and whether a common folder is in use in case you encounter difficulties. Using button "Set..." the M2 mode also allows you to set the folder to the same the RASS utilities use in case of a mismatch.

In case you do not have administrative privileges on the machine you are using you might encounter still difficulties. This is because the RASS utilities may lack write permission in the folder used. To circumvent this, you can use the new option -2 of the RASS utilities 'mk' and 'mk1'. It will

automatically create and use a folder 'M2Mode' inside your personal preferences folder where every user has sufficient permissions. Note, the drawback from that option is that this same folder will be shared by any AlphaX present on your machine. If you have several present, this technique may cause several AlphaX to confusingly open the same M2 working files. In such cases 'AlphaPrefs' folders are preferable.

As a rule of thumb: Default strategies should work fine for most users, and any variant should work for users with administrator privileges. The new option -2 is recommended in case you encounter any difficulties in the interaction between the M2 mode and RMSP1, especially if you are an ordinary user without administrative privileges. Awlays ensure the M2 mode preference 'M2_P1 Aux File Cache Folder' points at a folder for which the user has write permission and that it is actually the very same folder as used by 'mk' or 'mk1'.

RMSP1 version 2.2 and later support now also the use of AlphaX 8.2.x, the generation of universal binaries of AlphaX running natively on Intel Macs. Note, however, this requires to upgrade to the M2 mode version 4.6.1 or later. The working with RMSP1 2.2 and AlphaX 8.2a2d2 using M2 mode 4.6.1 has been tested and found to give you a smooth interaction between the updated RASS utilities 'mk' and 'mk1' and AlphaX:

In the new M2 mode the global M2 shell launching commands CTRL-2 has been improved: Not only will this command bring you to Apple's Terminal application, but it does also change to the directory of the file of the currently topmost window in AlphaX. Moreover, it can even open in the Finder that directory (new M2 mode preference "Open Also Dir In Finder'). Of course its simpler counterpart, now CTRL-1, is also still available and switches merely to the Terminal application serving as the M2 shell.

RMSP1 version 2.0 and later support now also the compilation on Intel Macs and offers also better support for the M2 mode of AlphaX. Within AlphaX you can now use the global M2 shell launching commands (CTRL-1 and CTRL-2). By default this will bring you back to Apple's Terminal application. If you prefer another application, simply use the M2 preference dialog (button "Set..." for Compilation preference "P1 M2 Shell Appliation Name") to choose another application. Enahcements/bug fixes allow also to more consistenly use the global "M2 -> Open Work Files" command (CTRL-0).

If you move your RMSP1 folder, you need also to rerun RMSP1 installation utility 'install.RMSP1.command'. BE WARNED: Otherwise your RMSP1 installation will no longer work!

Important note as of RMSP1 version 1.9 or later: **Which utilities** are used if both RMSP1 and RASS-OSX are present is determined only by the environment variable PATH. The latter depends on which variant has been installed last. It sets the value of a local '.profile' variable 'giveFirstPriorityTo'. That variable is used to determine the priority in the PATH, by which either utility set from RMSP1 or RASS-OSX, respectively, is to be used. If these definitions are missing (because you use a not fully uptodate '.profile'), the RMSP1 set is used. E.g. in case 'giveFirstPriorityTo' is assigned the value 'RMSP1' and you source the '~/.profile', the effect is that a utility like 'mk' is first searched in '$RMSP1HOME/bin' before it is searched in '$RASSOSXHome/bin'.

You can easily change this behavior: To switch between RMSP1 and RASS-OSX utilities use the new RMSP1/RASS-OSX utility **'switch2using**'. E.g. type

switch2using RASS-OSX ; source ~/.profile

To make the switch immediately effective, don't forget to source the '.profile' in the current shell or launch a new shell, e.g. open a new window in application 'Terminal'. Utility '**switch2using**' edits your '~.profile' by assigning the value of the first argument, e.g. RASS-OSX, to the variable 'giveFirstPriorityTo'. Need a reminder? Type

switch2using help

RMSP1 version 1.9 and later use a set of utilities, libraries, and templates which is renundant from that of RASS-OSX (no more the same in ~/bin/RASS). **Option -L** fully controls which set of symbol, library, and template files is to be used.

RMSP1 version 1.9 and later are incompatible with the installation scheme of earlier versions.

Make sure you deinstall any remnants from previous versions (e.g. run first old RMSP1 installation utility '**install.RMSP1.command**' in the deinstallation mode by first **answering to all installation questions 'no'** and then deinstall all). Only then should you run the new 'install.RMSP1.command' from the new RMSP1 release. For details see above, section on installation (cf. "Behind the Scenes").

RMSP1 version 1.9 and later allows to have multiple RMSP1 folders present on your system. Each can be used independent from the other. However, they all share a common '~/.profile' and common environment variables, notably **RMSP1HOME**. You need to alter the latter, either by manually editing '~/.profile' or by conveniently rerunning the RMSP1 installation utility '**install.RMSP1.command**' from the RMSP1 folder with which you wish to work.

RMSP1 version 1.8 and later expect by default **P1 release 8.2**. The display of compilation error and warning messages has fundamentally changed with P1 8.2 and later versions. The format is Xcode compatible. The RASS-OSX utilities 'mk' and 'mk1' as of RMSP 1.8 take care of that and fully support Xcode formatted compilation messages. Note, you can still use older P1 compilers by using new option -O with 'mk' or 'mk1'. With this version of P1 comes also a debugger. It is now available in its final form and is no longer a prerelease.

With P1 8.2 it has in principle become possible to generate **universal binaries**. However, the RASS-OSX libraries are not yet released with a universal binary variant. Note, you need also OS X 10.4.x as well as the latest Xcode 2.x to generate universal binaries with this development environment.

Note, RASS-OSX utility 'mk' recognizes only files with the extensions .**DEF** and .**MOD**. Notably be warned, that any files with extensions .def and .mod are ignored.

Open erroneous work files for editing in AlphaX' M2 mode: RASS utilities 'mk' and 'mk1 -o' (as of RMSP1 v1.3 and later) do now fully support editing using the M2 mode from AlphaX. Source files containing compiler errors or triggering warning messages are opened in the editor of your choice (option -o for opening files; cf. environment variable M2EDITOR for specifying your editor in file '~/.profile'). Compiler errors and warnings are displayed within all source files (CTRL-e) in the same manner as with the MacMETH or RAMSES shell. Make sure you have at least the latest M2 mode 4.2 installed or this feature will not work (check it out from the CVS using tag 'HEAD').

Note a new preference flag called 'Open P1 Work Files' has been introduced in the M2 mode. It remembers by which mechanism AlphaX was called the last time, i.e. whether it has been called via the RASS-OSX utilities 'mk' / 'mk1 -o' or whether via a Modula-2 shell, i.e. either the MacMETH or RAMSES shell. Any subsequent menu command "M2 -> Open Work Files" (or CTRL-0) will open the work files either for MacMETH / RAMSES or RMSP1/RASS-OSX accordingly. You can still anytime control the behavior of the menu command "M2 -> Open Work Files" (or CTRL-0) from within AlphaX, i.e. without using a RASS utility or a Modula-2 shell. To switch its behavior simply use the usual preferences dialog (F12) from within the M2 mode.

If AlphaX is not available, either the OS X application TextEdit (interactive usage) or the the OS X editor 'pico' (remote ssh usage, e.g. on a remote simulation server) may be called instead ('pico' (Panther) is now 'nano' (Tiger)).

Enhanced mk and mk1: For advanced users who wish to use merely the standard Unix 'make' command can now do so. The latest RASS utility 'mk' (as of RMSP1 v1.4 and later) generates a 'Makefile' which fully supports the use of your editor independent of 'mk'. Unless you change the module structure, calling standard Unix 'make' will do the job. Use 'make help' to learn more about this.

Utility 'mk1' allows also to make only single targets, i.e. a particular program module out of several, in addition to the standard 'make all' issued by 'mk'.

The 'mk' (as of RMSP1 v1.4 and later) offers now also two new possibilities: (i) Option -m makes all of your programs, but without regenerating the 'Makefile' (unless it does not exist) and without executing anything. Thus 'mk -m' is the same as 'make' (= 'make all') and is very efficient. You can force the generation of a new 'Makefile' anytime by 'mk -n' instead of 'mk -m'. (ii) Option -k ("keep closed") offers the possibility to suppress the immediate use of the editor during the making process. This is typically of interest if you wish to defer the editing of erroneous source files to a later time, e.g. to complete first

some work in the current shell. Then the command 'mk -e' allows you to switch anytime later to the editor, open all files containing errors, and displaying them (M2 mode of AlphaX only). By the way, this is the same as switching to the editor 'AlphaX' yourself and issuing there the keyboard shortcut CTRL-0 or choosing menu command "M2 -> Open Work Files".

Access files outside current folder (M2PATH): RMSP1 offers also a mechanism to define paths by which you can access files. This allows you to program in your RASS application a file access by merely specifying the file's name without a path (cf. module 'DMFiles' procedure 'Lookup'). This is actually the recommended way to do this. The path denoting the actual file relative to your application can then be provided later at run-time via this path mechanism. This offers much flexibility, since files can be offered to the program by different directory structures, which can evolve while you use the very same program.

The mechanism works by defining the environmental variable 'M2PATH'. 'M2PATH' has to contain a list of path strings separated by ':' like any other path environmental variable such as 'PATH' or 'MANPATH'. RASS offers basically two techniques by which you can specify the needed path strings.

a) If you use the RASS utility 'job', you have to use the file 'config.M2PATH' in which you can specify your path strings. RASS utility 'getm2config' lets you create such a 'config.M2PATH' file anytime (In RASS-Sun you have also the RASS utilities 'prepare' and 'reprep' which use 'getm2config' to accomplish this).

b) If you use the RASS utilities 'mk' or 'mk1', you have to use the file 'config.M2PATH.sh' in which you can specify your path strings. This file is automatically created for you by 'mk' and 'mk1' by using option -P. All options which execute your program, notably the default option and the -x option, will observe the paths specifications you enter into 'config.M2PATH.sh'.

If you have a file 'config.M2PATH.sh' present, any possibly present settings of the environment variable 'M2PATH' will be ignored and only those paths specifications used as given in the file 'config.M2PATH.sh' are used.

Note, once the file 'config.M2PATH.sh' deviates from the original form (as given by file 'RMSP1HOME/bin/config.M2PATH.sh.TEMPLATE' or the one from 'RASSOSXHOME' if option -L is used), your paths specifications will be preserved. For instance, 'make cleanall' will rename the file 'config.M2PATH.sh' to 'config.M2PATH.sh.MSTR' (MSTR stands for Master). Future usage of utilities 'mk' and 'mk1' will use that file 'config.M2PATH.sh.MSTR'. Note also, before first use utilities 'mk' and 'mk1' will rename it to 'config.M2PATH.sh' to be properly recognized at run-time.

Program execution: RASS utilities 'mk' and 'mk1' can work together. RASS utility 'mk' can make several modules. By default the first will be executed. Thus you can use commands such as

```
mk -m
```

to make all without any execution and

```
mk1 -x Prog3
```

to execute only 'Prog3'. Note, option -x implies for 'mk1' to activate implicitly option -l. Thus a command like 'mk1 -x Prog' is similar as calling 'Prog' (except for additional handling of paths as given by 'config.M2PATH.sh'). If your program needs no path specifications, then it does not matter whether you type 'Prog' or 'mk1 -x Prog'.

The **debugger** is available only as a prerelease. The debugger is not available by default. To use it you need to use option -d of the RASS-OSX utilities, e.g. 'mk -d' or 'mk1 -d'. This will then link your program with another library, i.e. "libm2dlib.a" instead of "libm2lib.a".

The debugger's functionality is still limited. First it can display the current program state. As of RMSP1 version 1.41 it features also interactive usage. The debugger responds to keyboard commands (no mouse). The following rules apply:

- Select a menu by CTRL– <*letter*> where <*letter*> denotes the displayed bold, first letter of the menu
- Select a menu command from the current menu by cursor up and down
- Choose (issue) the current menu command by <return>

or

- Choose (issue) a specific menu command by SHIFT– <*letter*>, where the <letter> is that displayed as 'Apple – <*letter*>' for those menu commands a shortcut is available

- The currently active window pane has a yellow background
- Activate the next window pane by TAB
- Scroll in the currently active window pane by cursor up and down

The debugger is able to read commands from the optional file 'debug.in'. In the current version the debugger is simply left if such a file is present.

Find similar hints also at http://www.sysecol2.ethz.ch/intranet/Unix_Hints.html#Machines.

The call of the debugger is either triggered by an exception or the statement SYSTEM.BREAK. Displayed are the following: The line of source code by which the debugger was called, the procedure chain, local and global variables of the involved procedure and module, a list of modules, a memory dump, and the registers' content. Too long names are abbreviated with a '?', not the usual ellipsis symbol "...". Enlarge the terminal, if you can't read the information. Since the latter requires a rerun, make the Terminal wide from the very beginning if you plan to use the debugger.

Note, the full information can only be displayed if source code (*.MOD) and reference (*.MREF) files are made available to the debugger. The utilties 'mk' and 'mk1' generate the needed file 'debug.dat', which contains paths to RASS files. It uses a debug.dat.TEMPLATE file from your $RMSP1HOME/bin directory (or $RASSOSXHOME/bin if using option -L).

For developers this template assumes also that the RASS-OSX sources and reference files are available, either in a directory named '~/Library/RASS/Srcs/rassXYZ', or in directory /usr/local/RASS/rassXYZ (on clustered simulation servers), where XYZ stands for the corresponding RASS-OSX version. If your installation differs, you may have to edit that template file. Note, that the debugger uses the latest path definition, not the first! If you used RASS utility 'doRassInstallation' (or 'install_RASS_On_SimServ.command') to install your RASS-OSX, then you already have all those needed files present. Otherwise run the utility 'doRassInstallation' to obtain those files.

The debugger can be left, e.g. to continue execution of the program, by pressing SHIFT-G. In case of a SYSTEM.BREAK ordinary program execution follows. In case of an exception, the exception handler is executed (e.g. all statements after keyword 'EXCEPT' in the procedure, where the exception occurred).

Note, if you do not properly link with "libm2dlib.a", i.e. don't use option -d for 'mk' or 'mk1', then a programmed SYSTEM.BREAK will lead to a break in the program execution, but without calling the debugger. Continue with any key.

Finally note, the debugger is only called by run time exceptions intrinsic to the Modula-2 language and the P1 implementation, but not by a programmed exception using ISO-Modula-2 statement RAISE. This also means that exceptions encountered during execution of external C-libraries (e.g. math functions), may not cause an exception, regardless whether you made the exception hot, e.g. by using DMFloatEnv. DMFloatEnv only controls the behavior of the Modula-2 code and not necessarily of external C code. Note also, that DMFloatEnv can not control the exception divideByZero in the current P1 compiler. E.g. the call

```
DMFloatEnv.DisableHalt(divideByZero)
```

is not sufficient to really disable this exception. You need the following compiler instructions (pragmas) around your offending code:

```
(* IF VERSION_P1 *) (*.
<* PUSH; ASSIGN(DivZeroCheck, FALSE) *> (* disable extra P1 div by 0 checking *)
.*) (* ENDIF VERSION_P1 *)
Your offending code such as      y := x / zero;
(* IF VERSION_P1 *) (*.
<* POP *> (* resumes previous setting *)
.*) (* ENDIF VERSION_P1 *)
```

For devlopers only: As of RMSP1 version 1.42 you can also debug any library provided as part of RASS-OSX. However, to view those files, you need to install the result of the release machinery onto your machine (~ 350 MB). The RASS utility 'doRassInstallation' (or 'install_RASS_On_SimServ.command') helps you to get the needed files. This kind of debugging is also available on any of the clustered simulation servers, e.g. the superior.ethz.ch (of course only when you execute code linked with the -d option). If you execute code on a clustered simulation server, which you have linked locally on your own machine, you must make sure that you use exactly the same libraries as those installed on the cluster of simulation servers. Be warned, otherwise you risk to see most confusing wrong information (with no warning)!!

Hint: If the debuggers interferes with your ordinary window settings of your Terminal, simply choose menu command 'File -> Send Hard Reset' (shortcut COMMAND-OPTION-R) from the application

'Terminal' and you should be back to normal display settings. Application Terminal under Leopard (OS X 10.5.x) differs and this command is now found under menu 'Shell' .

To debug **heap errors**, use a text file with the name 'BreakForADR.txt' beside your program. It will trigger the debugger as soon as you try to allocate or deallocate a heap block with the address given in that file. The address has to be a hexadecimal number consisting merely of hexadecimal digits (not case-sensitive). Note, do not put the Modula-2 'H' at the end. Ex.: 1109a90.

CTRL-C does not flush: The output generated by your RMSP1 program may be very misleading, if you kill the process with CTRL-C. This is because in all current versions of RMSP1 the flushing of standard output/standard error does not to work if you kill a process with CTRL-C or CTRL-\. Fortunately using HALT statements flushes all outputs well. Use HALT and you may see much more output than what you would see if you kill the process with CTRL-C (or CTRL-\) at the very same program location. This behavior can be very confusing and you are likely to suffer from tedious and time consuming programing cycles. Thus be warned and rely only on HALT when interpreting outputs from a buggy program which requires to be killed by CTRL-C. Of course I will make efforts to remedy this situation in the future. For the time being all what can be done is a plea to listen to this warning.

If you use a libRASSxxx.a, (xxx stands for a version number such as 320) which contains some modules with bugs, there is a technique to **override specific library modules**. It allows you to use fixed implementations in place of those contained in the libRASSxxx.a. There are two things you need to do to accomplish an override: (i) I recommend to have a folder, e.g. named 'RMSUpdate', nested within your work folder, where you keep the new sources, i.e. MOD files, with fixes for all those faulty implementation modules you wish to override. Make sure all these files have the extension **.MOD**, **not .mod**! (ii) Import all these fixed modules within your main program module at the end of the import list. E.g. similar to this example where the modules SysDep, DMFiles, and FileNameStrs are to be overriden with new implementations:

```
IMPORT SysDep, DMFiles, FileNameStrs;
```

Utility 'mk' will compile those modules automatically and the linking will use the new variants in place of the faulty ones from the library. Note, this behavior is different from that on a Sun using RASS. There you need to have the DEFs as well as the MODs in the same folder where all other sources are contained. Secondly you need not import those modules into the main program module.

You can also override definition modules. You then need to compile the definition module manually similar to this example

```
Modula2 DMFiles.DEF
```

This may work or may not work, depending on the incompatibilities you might create between definitions (cf. section 1.6 «Built-In Version-Compatibility Check» of the P1 compiler manual).

It may also be convenient to use symbolic links to the actual sources instead of copies. This is particularly useful for developers. Ex.:

```
ln -s <drag via Finder source file here from Dev> ModuleName.MOD
```

Finally note, if you wish to stop overriding library modules, simply move them to a folder nested sufficiently deep, i.e. at least 2 levels below the main program or rename them with extension .mod. As a consequence utility 'mk' will ignore those modules when called the next time (don't use make, since the Makefile is not consistent with the new setup unless you regenerate it!).

A new RASS-OSX utility '**mknewlib**' is available. It allows to create a new libRASSxxx.a library from the existing one. You can simply run this script and it will recreate the library. Once you have done that at least once, you will also obtain the folder 'RMSP1/lib/work'. It contains all *.o files from the libRASSxxx.a library. To replace a faulty *.o file with a new one (possibly generated by utility 'mk' within your project folder containing a RMSUpdate folder), simply overwrite in 'RMSP1/lib/work/' the *.o files you wish to update. Then run 'mknewlib' and you quickly get a new updated libRASSxxx.a. This utility is designed to be used mainly by developers of RMSP1 who wish to prepare a new release of RMSP1. However, it allows any user to add permanently new implementations to the library libRASSxxx.a, allowing you to give up on the more cumbersome, overriding technique described above. This is particularly useful if you work on several projects, where each needs the overriding, i.e. the new implementations of some library modules. Utility 'mknewlib' will not affect any installation of

RASS-OSX.

Making an OS X Application With a User Interface

Under some circumstances it may be of interest to have an application bundle for your RMSP1 program. This can now be achieved by using the provided template 'Template RMSP1 Bundle.sit'. Simply compile and link your application in the usual manner using RMSP1 utility 'mk'. Once you have completed this step create a duplicate from the template and move the RMSP1 program together with possibly needed resources to the Resources folder of the project and then add them to the project. The result is an application which looks very much like any other OS X native application. However, it is usable only for batch processing of data. There are two approaches available: One will still use the Terminal for displaying the text output as generated by routines from DMWindIO. The other uses iHook and behaves like a real OS X application (iHook can be obtained from http://sourceforge.net/projects/ihook/ or older versions from http://rsug.itd.umich.edu/software/ihook/). The Apple script provided as part of the template contains many more details on the technique to be used to get your application bundle working, should you encounter difficulties.

Hint: iHook can be used to make a real OS X application bundle with RMSP1, given the program does only some batch processing.To indicate progress of processing, simply insert these statements:

```
DMWindIO.WriteString("%"); DMWindIO.WriteInt(percentAccomplished); DMWindIO.WriteLn;
```

in your program code.

Cited References and Further Reading

(Most references are conveniently available from http://www.sysecol.ethz.ch/publications (see links Publications and Systems Ecology Reports). Visit also the home page of RAMSES offering a web based documentation of all RAMSES objects used by RMSP1 at http://www.sysecol.ethz.ch/ramses/Reference).

Cellier, F.E. & Fischlin, A., 1982. Computer-assisted modelling of ill-defined systems. Hemisphere Publishing Comp.: Washington a.o. Progress in cybernetics and systems research, VIII: 417-429.

Fischlin, A., Mansour, M.A., Rimvall, M. & Schaufelberger, W., 1987. Simulation and computer aided control system design in engineering education. In: Troch, I., Kopacek, P. & Breitenecker, F. (eds.), Simulation of Control Systems. Pergamon Press, Oxford a.o., pp. 51-60.

Fischlin, A. & Schaufelberger, W., 1987. Arbeitsplatzrechner im technisch-naturwissenschaftlichen Hochschulunterricht. Bull. Swiss Electrotech. Assoc., 78(1): 15-21.

Fischlin, A., 1991. Interactive modeling and simulation of environmental systems on workstations. In: Möller, D.P.F. (ed.), Analysis of Dynamic Systems in Medicine, Biology, and Ecology. Springer, Berlin a.o., pp. 131-145.

Fischlin, A., 1992. Modellierung und Computersimulation in den Umweltnaturwissenschaften. In: Schaufelberger, W. (ed.), Computer im Unterricht an der ETH Zürich, Bericht über das Projekt IDA (Informatik Dient Allen) 1986-1991. Verlag der Fachvereine, Zurich, Switzerland, pp. 165-178.

Fischlin, A., Gyalistras, D., Roth, O., Ulrich, M., Thoeny, J., Nemecek, T., Bugmann, H.K. & Thommen, F., 1994. ModelWorks 2.2: An interactive simulation environment for personal computers and workstations. Systems Ecology Report No. 14, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 324 pp.

Fischlin, A. & Thommen, F., 2003. On the Dialog Machine. Electronic Document obtainable from http://www.sysecol.ethz.ch/ramses/layer/DM, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 52 pp.

Keller, D., 2003. Introduction to the Dialog Machine, 2nd Edition. Systems Ecology Report No. 30 (Electronic Document also obtainable from http://www.sysecol.ethz.ch/publications/reports#30, Department of Environmental Sciences, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 35 pp.

Thoeny, J., 1995. Practical considerations on writing portable Modula-2 code. Systems Ecology Report No. 20, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 6 pp.

Thoeny, J., Fischlin, A. & Gyalistras, D., 1994. RASS: Towards bridging the gap between interactive and off-line simulation. In: Halin, J., Karplus, W. & Rimane, R. (eds.), CISS - First Joint Conference of International Simulation Societies, San Diego, Cal. 92177, USA, August 22-25, 1994; Zurich, Switzerland, The Society for Computer Simulation International, P.O. Box 17900, pp. 99-103.

Thoeny, J., Fischlin, A. & Gyalistras, D., 1995. Introducing RASS - the RAMSES simulation server. Systems Ecology Report No. 21, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, 32 pp.

Vancso, K. & Fischlin, A., 1988. Automated construction of interactive learning programs in Modula-2. Comput. Educ., 12(4): 507-512.

Vancso, K., Fischlin, A. & Schaufelberger, W., 1987. Die Entwicklung interaktiver Modellierungs- und Simulationssoftware mit Modula-2. In: Halin, J. (ed.), Simulationstechnik. Springer, Berlin, pp. 239-249.

Vancso-Polacsek, K., Fischlin, A. & Schaufelberger, W., 1988. SAM - Simulation and modelling software for working stations based on modelling theory. In: Vichnevetsky, R., Borne, P. & Vignes, J. (eds.), Proc. of the 12th IMACS World Congress (5 Vols.), Scientific Computation, Paris, France, July 18-22 1988, pp. 761-763.

Wirth, N., Gutknecht, J., Heiz, W., Schär, H., Seiler, H., Vetterli, C. & Fischlin, A., 1992. MacMETH. A fast Modula-2 language system for the Apple Macintosh. User Manual. 4th, completely revised ed. User Manual Department of Computer Sciences (ETH), Zürich, Switzerland, 116 pp.