# VNC

Thomas the Richter

**COLLABORATORS**

| | | | |
|---|---|---|---|
| | *TITLE* :<br><br>VNC | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Thomas the Richter | April 15, 2022 | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# VNC

## 1.1 The PoolMem Guide

PoolMem Guide

Guide Version 1.14 PoolMem Version 1.51

Table of Contents

© THOR-Software Thomas Richter Rühmkorffstraße 10A

12209 Berlin Germany

EMail: thor@einstein.math.tu-berlin.de WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

## 1.2 The THOR-Software Licence

## 1.3 Overview: What does it?

PoolMem is a sort of "memory defragmentizer". Whenever you run a lot of programs on your Amiga, and each of them allocated memory in random order, the memory gets broken up into more and more tiny blocks. After a while, you might be unable to run a larger project or load a larger data set, even if numerically enough memory is available.

The reason for not being able to load a project is not that there is not enough free memory, but there is not enough free continuous memory in one large block. The memory will be, instead, consist of lots of tiny blocks that aren’t useful at all.

Unfortunately, the construction of the AmigaDos does not allow a memory defragmentizer in the strict sense, i.e. there is no way of "moving" memory blocks to make room for large allocations. However, the implementation of the memory allocation and deallocation functions in the AmigaOs is relatively naive, and that’s the point where PoolMem comes into the game.

PoolMem replaces the memory management functions of the AmigaOs by smarter routines that are build in a way such that the memory doesn't get fragmentized too easely. It fixes on its way several minor features (not to call them "bugs") of the memory management and replaces for that reason several other patches like "NewAllocMem" or "AllocP" and "AllocP32". It includes, too, several workarounds for "popular bugs" in programs.

After having tested PoolMem for about three years now, I would say that this version runs very stable, and I won't expect new bugs. The only problem remaining are programs that don't follow the Os rules - I've collected these programs - together with their bugs - in the compatibility section .

To make these rules available for everyone, I included the file "Developer.readme" in this archive. If you write programs for the Amiga, study it carefully. These rules aren't setup by me, they are just what can be found in the official documentation. Not following these rules causes trouble with the AmigaOs - and with PoolMem.

If you have ideas how to improve PoolMem, let me know!

## 1.4 How to install PoolMem?

Installation is quite simple. Just copy the "PoolMem" program from this archive to the C: directory. Then add the following line to your startup-sequence, right below "SetPatch", or, even better, below "SaferPatches" which should be the second patch to be installed anyways:

SetPatch >NIL: SaferPatches INSTALL >NIL: ;optionally, should be ;be run if you want to ;remove PoolMem later. PoolMem INSTALL >NIL:

Especially, PoolMem should be run IN FRONT of all other memory allocation patches, i.e. MungWall. That means for example:

RUN PoolMem first, MungWall later

Provided you installed the RexxSerDev.library, you should also apply a patch to it since this library contains a serious bug. For details check the RexxSerDevFix drawer in this distribution.

That's all. For the "fine-tuning", study the configuration chapter.

To be able to remove PoolMem later, you should run a program like "SaferPatches" as indicated below. To remove the PoolMem patch, open a shell and enter

1.SYS:> PoolMem remove

Execution of this command may take up to five seconds, so be a bit patient with PoolMem. Do NOT try this without SaferPatches.

By the way: When running PoolMem, you won't need patches like "NewAllocMem" or "AllocP" anymore. Their function is completely integrated in PoolMem. Don't install them together with PoolMem, that won't do good!

## 1.5 Configuration

The fine-tuning of PoolMem is done by shell command line options, at invokation time of PoolMem. Hence, you've to edit your startup-sequence to adjust this program; there is no graphical user interface, at least not yet.

The complete command template looks as follows:

PoolMem HELP/S,REMOVE/S,INSTALL/S,CHIPFWD/S,NOMERGE/S,SCRATCHONLY/S, PUDDLESIZE/N,SHRINKTRHES/N,F
ALLOC32/S,NORAMREVERSE/S,FREEMEMRESULT/N,KEEPSORTED/S

Here's the meaning of the various options:

HELP:

Prints a small overview of the available options.

REMOVE:

Remove a previously installed version of PoolMem. Be warned: You *should* run SaferPatches to be able to remove PoolMem safely. PoolMem does not check if a patch was installed on top of it.

CHIPFWD:

To keep the chip memory as unfragmentized as possible, PoolMem will usually allocate big memory chunks of chip memory in reverse order. That usually does only good, but if this causes trouble, you may give this option to allocate even chip memory in standard, i.e. forwards direction.

NOMERGE:

Tells PoolMem not to attempt to merge adjacent memory blocks of the same type. Even if the memory types are equal, the physical characteristics of the memory blocks might differ (e.g. 32 bit memory vs. 16 bit memory) and it is therefore not always desireable to merge them. Merged memory will cause less fragmentation, but might, too, run slower under certain circumstances because slower 16 bit memory is used instead of faster 32 bit memory.

SCRATCHONLY:

PoolMem usually splits one large memory block into two halfes, a so called "small memory pool" and a "large memory pool". Allocations for small memory blocks are taken from the "small pool", large allocations are taken from the "large pool". The boundary between the two is dynamic, i.e. PoolMem may adjust it if required. However, if this splitting causes trouble, you may disable it with this option. In that case, only the memory scratch list will be used. (See Theory of Operation .)

PUDDLESIZE:

Set the "puddle size" for the small block memory pool. If an allocation from the small memory pool fails, PoolMem attempts to enlarge the pool by this number of bytes. Defaults to 8192.

PUDDLETHRES:

Set the "puddle threshold" for the small memory pool. Allocations larger than this size are taken from the large memory pool instead of the small memory pool.

SHRINKTHRES:

Set the "merge threshold". If more than PUDDLESIZE + SHRINKTHRES bytes are unused at the end of the small memory pool, the pool shrinks and the free bytes enter the large pool. Defaults to 4096.

SMALLHEADER:

If this option is given, set the memory block of this index aside and use it as small memory pool. The argument is simply the number of the block in the memory list, counting from zero, and can be found by watching the output of ShowMem . The topmost memory block is number 0, the next below 1 and so on. This memory block MUST BE fast memory. If this option is given, PoolMem does not try to split any fast mem header, only the chip mem is split. The small memory block is sorted in just before the chip memory, but is used for the small memory block allocations as long as possible.

ALLOC32:

Align all memory allocations to 32 byte boundaries. This option shouldn't be used unless it is really required - and there is currently no real need for it. Using it is like throwing a fragmentation bomb into the memory, even WITH PoolMem.

This keyword might become useful when dealing with PPC / MC68K mixed hardware. All allocations should be aligned to 32 byte boundaries to avoid holding memory segments both in the PPC and the MC68K cache. This is currently avoided by the PPC system software.

HOWEVER, using this keyword fragmentizes the memory really badly because of the unuseful unaligned boundary snippets that neither can be allocated, nor enter the scratch list.

Even though memory returned from AllocMem() IS aligned to 32 byte boundaries, AllocVec'd memory isn't. It is still only long word aligned because the first longword of the vector is used to keep the size of the allocated memory. This doesn't hurt for the purpose this keyword was invented for; it is, even more, REQUIRED to work like this because the vector size should be either in the PPC or in the MC68K cache, but not in both.

Due to the heavy fragmentation of memory when using this keyword, it is recommended to use it only if absolutely necessary. The standard PPC system software deals with the problem of overlapping caches correctly so there's really no need for this keyword, at least not now.

This keyword works quite different from AllocP32. It doesn't use "magic cookies" in any way and is written in a way compatible to the - rather ugly - layers.library allocation methods (check the developer.readme file in this archive for details about this). The drawback of this compatibility is the rather high fragmentation, it's even higher than without PoolMem. You have been warned!

NORAMREVERSE

Do not try to allocate memory for the RAM disk in reverse order. Using this option is NOT recommended since it may increase the fragmentation.

This implements one function of the PatchRAM program into PoolMem, where it belongs.

FREEMEMRESULT

Defines a dummy result code for FreeMem() to fix broken programs. The default value is "-1" and works around an FFS bug. However, as this bug has been fixed with release 43.20 of the FFS, and the result code "-1" conflicts WITH ANOTHER BUG in the "RexxPlus" compiler and "RexxPlus" compiled programs, you might want to set the result code to something different. "-2" would be an apropriate choice.

KEEPSORTED

This implements a different management of the PoolMem scratch list for tiny memory snippets. By default, the PoolMem scratch list is a "FIFO" list, i.e. the last de-allocated memory block is the first that will be re-allocated. This simple management is very efficient, but has the disadvantage that the memory will fragmentate heavely as long as the garbage collection state is running. PoolMem will clean-up this mess later, but even then the garbage collection will be relatively slow. With "KEEPSORT" active, the memory scratch list is kept sorted by address instead, similar to the system memory lists. This means that memory deallocation will be slightly slower, but garbage collection will be faster because the memory is released in "correct order". Futhermore, the memory will not fragmentate as heavely as usual while in garbage collection. You should try this option if you run programs that really mess up the memory on exit and you can't live with the long garbage collection time.

## 1.6 Extras in the PoolMem distribution

RexxSerDevFix

This drawer contains a patch to fix a serious bug in the rexxserdev.library, namely the parameter passed into FreeMem is a word, not a long word. This may cause a completely messed up memory list, not only with PoolMem installed. The library version is upgraded from 5.2 to 5.3 and the bug is patched over.

How to apply the patch:

- Open a shell - Copy the contents of the RexxSerDevFix drawer to RAM: - Enter the following command: CD RAM: - then, enter the following command to update the library: spatch -oram:rexxserdev.library -prexxserdev.pch LIBS:rexxserdev.library - then, copy the library back to LIBS: copy ram:rexxserdev.library to LIBS:

ShowMem:

Shows the allocated/free memory in a graphical overview. For details, check the ShowMem readme and the ShowMem.guide in this archive.

PatchRAM:

Modifies and improves the RAM disk. Shows the correct size of the RAM drive which is then no longer 100% full all the time. For details, check the PatchRAM.readme. As PoolMem, it should be installed in the startup-sequence. Unlike previous versions, this patch does no longer touch the memory allocation functions that have been integrated into PoolMem.

FragMeter:

Calculates the fragmentation of your memory. The output is given separately for each memory type. A 100% fragmentation indicates that all the free memory is messed up in tiny blocks of eight bytes each which is the maximal possible fragmentation. A "Shannon-type" approach is used to measure the fragmentation: The algorithm calculates the Shannon entropy of the memory blocks with the formula

sum += log(chunk->mc_Bytes/total)

If you know a better approach for measuring the fragmentation, lemme now. This here seems at least somewhat reasonable for me as a theoretical physicist... ;-)

This program can be used to test the efficency of PoolMem. My measurements indicate that the entropy is about halved.

MemoryMess:

A program that tries to fragmentate the main memory as badly as possible by allocating and freeing memory in random order. Can be used together with the FragMeter to measure the efficiency of PoolMem or with ShowMem to watch PoolMem at its job. Can be canceled safely with ˆC (Control-C). Does nothing useful except that.

PatchReplyMsg:

Useful to find buggy programs, like XiPaint. PatchReplyMsg will create a guru 0x0100001f if a program attempts to reply an already de-allocated message. Don't install it permanently, but use it to find the reason for crashes.

PoolMemDetect:

If PoolMem is running and a guru shows up PoolMem is responsible for, run this program to get more detailed information about the crash. If possible, send the output of this program to me, including how you generated that crash - it might prove helpful.

pgs_patchmem:

This program patches AllocMem to avoid problems with Photogenics. It seems that it trashes memory outside of the domain it allocated. This patch adds a safety wall around each memory block allocated in order to avoid this problem. However, this patch cannot be stopped because the wall is then never released again. This is currently more an experimental version which you might want to check if it helps to avoid Photogenics crashes. If it does, I'll prepare a final version - even though an updated Photogenics version might be more helpful. Thanks goes to Carl Drougge for this program, and for allowing me to redistribute it here.

In case you've questions about this patch, contact Carl at

carl.drougge@2.sbbs.se

## 1.7 Compatibility

This section contains compatibility remarks and programs to be known not to run correctly with PoolMem installed. I checked all of these programs carefully and found that the reason for crashes lies usually either in a bug of the program, or in the un-ability of certain folks to read the Os documentation carefully. To help you, and to avoid future problems I collected all the Os guidelines, as far as memory allocation is concerned, in the "Developer.readme" file in this distribution. Please READ THEM!

Here's now the list of bad programs:

RexxPlus ARexx compiler, and RexxPlus compiled programs:

The compiler runtime library and its compiled programs call an undocumented rexxsyslib.library ("ReleaseRexxResource", namely) and expect a proper result code in register d0. However, what's left here is just the "result code" of FreeMem(), nothing more. Unfortunately, "RexxPlus" handles "-1" as a special error case and quits. This can be worked around by setting the FREEMEMRESULT option to something different than -1. The side effect of this would be, however, that due to another bug in the FFS releases up to 43.19 return this result code instead of DOSTRUE for a successfull Close(). This FFS bug was fixed in release 43.20.

rexxserdev.library:

Due to a bug in the memory release function of the program, only a word is passed into the FreeMem-Function, not a long word as supposed. Since the upper half of the register may contain random data, the memory release call will sooner or later trash the memory list completely. This is fixed by the RexxSerDevFix patch in this distribution.

amiboot:

The linux kernal boot program "amiboot" does not work with PoolMem installed. This program suffers from several bugs. First of all, it examines the memory list itself, without using the exec.library. (What do you think AvailMem() is good for, then?) Second, it does not even call Forbid() to do so. Third, it always expects that the chip memory comes in one continous block and is too unflexible to handle other cases, as for example the two ChipMem blocks that will be created when PoolMem is running and active. If you really MUST use this program, then use the SCRATCHONLY option of PoolMem to avoid splitting the memory in a large and a small memory pool.

The Final Odyssey:

The game demo "The Final Odyssey" (and probably the full game) crashes when PoolMem is active. Tracing this error down to its roots resulted in an aparent bug in the memory deallocation function of this game that occurs, too, with the official MungWall debugging tool instead of PoolMem installed.

Since EVERY CORRECT PROGRAM is supposed to run with MungWall, I WON'T TRY to add a workaround for this bug. Just avoid buggy programs, as usual.

xpk-Compressors:

Some xpk-compressors overwrite non-allocated memory and cause hence problems with PoolMem. Note that this is a *severe* software bug and NOT tolerated, neither by exec, nor by PoolMem. The xpkmaster.library version 5.2 or higher includes work-arounds for these bugs, though. Again, it's only a matter of good luck if this works.

Photogenics:

I got reports that this program crashes with PoolMem installed as soon as it is shut down. A bug in the shutdown code alters memory that is no longer allocated. THIS IS FATAL, even without PoolMem. It's just a matter of good luck if that does not crash the system. This hit cannot be detected with MungWall since pgs called Forbid() before this memory is released. MemSniff will do a better job in these cases...

XiPaint:

The mouse driver in the XiPaint program contains a serious bug that can cause crashes, with or without PoolMem.

The message handling of the mouse driver is broken. After a long debug session deep in the night, I found that the crashes were created by the mouse driver XMouse. It replies messages that are no longer valid and have been FreeMem()'d before. THIS IS DEFINITELY ILLEGAL AND NOT A POOLMEM PROBLEM.

To help you checking other programs as well, I included a tiny patch PatchReplyMsg . You shouldn't install it permanently, though, because it slows down the message system. If, after installation of this patch, a program crashes with guru 0x01000001f, THEN THIS PROGRAM HAS THE SAME BUG AS XIPAINT and replies illegal messages. This will crash your system anyways, with or without PoolMem. Please sent a bug report to the author, or remove these programs. As I said, THIS IS NO POOLMEM problem.

The guru 0x01000013 is rather typical for these programs - they invalidate the internal scratch list of PoolMem.

SaferPatches

There is no problem with this program, but just a side remark at this place:

To be able to remove PoolMem later on safely, PoolMem must be run AFTER SaferPatches has been installed. Same goes of course for all other patches and the SaferPatches program.

PowerUp boards and ppc.library

PoolMem should be run with the NOMERGE option on these boards, and must be launched behind the PPC library. If you fail to specify NOMERGE, PoolMem will slow down the system considerably because the PPC will then allocate motherboard memory which isn't available for full 64 bit data transfers. PoolMem cannot be run in front of the PPC library because the MMU setup of the library doesn't work with dynamic memory headers.

SetPatch, the 68040.library and ppc.boards

The 68040 library which is loaded by this program allocates MMU tables which must be aligned to certain memory boundaries. The current version of PoolMem respects this behaiviour and there shouldn't be a problem with this actually; however, I can't test this. (Unless there's a sponsor who wants to send me a PPC Amiga as a gift.... :-)

The easiest solution IF you run into trouble is to run PoolMem AFTER SetPatch, as recommended.

MemSniff: (The THOR MungWall replacement)

MemSniff patches deep into the memory allocation routines, too deep for PoolMem.

MuGuardianAngel: (The THOR GuardianAngel replacement)

Just the same goes for this program: It patches deep into the memory allocation routines to get its job done, this won't work with PoolMem. However, MuGuardianAngel is smart enough to quit PoolMem on startup.

AmiAtlas

This program doesn't use memory pools and fragmentates the memory heavely on exit for that reason. This is not really a bug, but feel prepared that the garbage collection phase might take a while. If you can't accept this, you might want to try the "KEEPSORTED" option of PoolMem to use a different memory scratch manager.

GoldEd 3.1.4

What goes for AmiAtlas goes for GoldEd as well. I don't know whether more advanced releases implement a more advanced memory management.

The following compatibility hacks and workarounds have been added to PoolMem:

PixMate:

This program relies on a correctly set zero condition code bit if AllocMem fails. There IS NO guarantee that the "z" bit is set to anything useful. However, since the fix was so easy, I added this to PoolMem.

Other assembly programs may suffer from this bug, too. (Hopefully not mine...;-)

FFS 43.xx:

The FFS/OFS ACTION_CLOSE packet does not set the correct result code. This is currently done by the PoolMem FreeMem() replacement routine. This bug has been fixed in the 43.20 release of the FFS.

Memory flush:

The memory flushing algorithm of the exec.library is - even though working in principle - a bit unuseful in praxis. Since task switching is disabled during a memory flush due to a low memory situation, libraries and devices that launched sub-tasks won't be able to inform their children to free their allocated memory in time. I don't know why the exec development team had this crazy idea.... Anyways, PoolMem adds a workaround for this and replaces therefore the "AllocP" and the "NewAllocMem" patch.

## 1.8 Theory of Operation

PoolMem splits all available memory in two blocks: One block is used for the small memory snippets (always taken from there), the other block is used for huge allocations. This partition of the main memory is dynamic, i.e. each sub-pool can grow and shrink, depending on the memory requirements.

The "public/ANY" memory gets a special treatment. PoolMem manages a "scratch" list for tiny memory blocks taken from there. Instead of taking these tiny memory blocks always from the main memory, they are taken from this (global, though) memory pool and put back into this pool if they get deallocated. A special garbage collection task - the "PoolMem.supervisor" cleans this "scratch list" from time to time, as needed. The main profit is taken from the layers.library, which uses to allocate tons of tiny snippets and is therefore the main cause for memory fragmentation.

The "chip" memory is treated a bit different. It's also split into two distinct memory pools (small and large), but memory from the large pool is allocated in reverse direction - unless you run PoolMem with the CHIPFWD keyword , of course; this happens for two reasons:

First, it helps to keep the memory defragmentated, so the big pool can't run that easely into the small pool. Second, it works around a hardware bug of my computer (the refresh of the high end chip memory in my computer seems to be a bit buggy -some bits tend to flip if they aren't frequently accessed, for example by the DMA processor as display memory.)

This is, after all, not the great, smart and final memory allocation patch, but something based on heuristic ideas that seem to prove useful at least. You might ask a real computer scientist for better algorithms and keep me informed. I'm after all only a pure (poor?) physicist.

## 1.9 Guru meditations

The following Guru meditations can be thrown by PoolMem:

0x01000013 Scratch entry illegal.

Some program invalidated the internal memory scratch list of PoolMem - by overwriting memory that has been deallocated before. An (in)famous example is XiPaint. Run "PatchReplyMsg" and a debugger (e.g. COP) to check for details.

0x0100000f MemHeader not found.

Some program attempted to free a non-existing block of memory. PoolMem wasn't able to locate the MemHeader.

0x01000011 MemHeader insane.

PoolMem found a MemHeader whose number of available bytes does not match the size of its pool.

0x01000012 Invalid DeleteHeader (internal).

Someone tried to deallocate the "large" memory pool. This is an internal guru, shouldn't happen. If it does, please inform me.

0x01000014 Memory Pools unordered.

Someone changed the order of the memory blocks, the large memory block is no longer in front of the small memory pool.

0x81000005 MemHeader corrupt.

On FreeMem(), PoolMem detected a corrupt memory header. Some program messed with the memory lists.

0x01000009 Memory freed twice.

A memory chunk already deallocated was deallocated again. Usually a bad program with a broken memory deallocation routine.

0x0100000c Memory insane.

On an AvailMem() call, PoolMem detected that the memory headers are insane, i.e. the size of free memory recorded in the memory header structure does not match the size of available memory. Some program messed with the memory list.

0x81000015 AllocMem underflow.

On a memory allocation operation, the memory free counter underflowed. Some program messed up the memory headers. You'd better check for broken programs!

0x81000016 FreeMem overflow.

On a memory release operation, the memory free counter overflowed. Some program messed up the memory header.

## 1.10 Thank you, folks!

Thanks to Andreas Kleinert for the useful discussion about his AllocP patch. My first thought (and not only mine!) was that this program is completely useless - which it isn't, due to some design flaw in the exec memory flusher.

Thanks to Dirk Neubauer for continously reporting problems or non-problems of PoolMem and for testing this program. Some of the compatibility notes are due to his observations.

Thanks to Raphael Pilarczyk for additional reports, especially the Photogenics problem, and for testing and providing additional ideas and options of PoolMem. Moreover, this guide file is more or less due to him. Hope you like it, Raphael...

Thanks to Ralph Schmidt for some discussion and giving me some insight into the PPC hardware.

Thanks to Carl Drougge for sending me the pgs_patchmem program, and for allowing me to redistribute it.

Thanks to all the other folks I'd been in discussion with and I unfortunately forgot to mention here... Urgh, please forgive me!

I do NOT want to thank Commodore for my @#%&!!! computer. (This is my 3rd computer, my 9th mouse, my 3rd fan, my 2nd SCSI controller, my 3rd disk drive, several memory problems drove me crazy, the expansion port is wrongly designed... Which idiot designed this crap? When I found out that commodore went out of business, I really ENJOYED it.)

I do NOT want to thank Gateway for trying to turn the Amiga into an intel-based machine. No folks, not this way. \begin{sarcastic} While Commodore just dig the grave for Amiga, Gateway is turning it into a zombie. \end{sarcastic}

## 1.11 History: What happened before

Version 1.19

After about three years of (interrupted) development and various ideas about PoolMem and its memory organization, and after two years of running it in my system without problems, I finally decided to publish this version in the AmiNet.

Version 1.19.1

Changed the name of the DefragMeter to FragMeter because that makes more sense. Added a chunk count to this program. PoolMem itself unchanged.

Version 1.20

Added the CHIPFWD command. Works like INSTALL but allocates chip mem in standard (forwards) mode instead of backwards mode. Might solve strange compatibility problems. Added more sanity checks, included PatchReplyMsg to create warnings for broken programs.

Version 1.21

PoolMem flushes now the memory as it should if called by AllocMem(-1,..). Except that, nothing changed.

Version 1.21.1

Included a new version of PatchRAM (check PatchRAM.readme for details), and a new version of the FragMeter program.

Version 1.22

PoolMem merges now automatically all adjacent memory lists of the same type.

Version 1.30

This version introduces a major rewrite of all critical routines, especially the replacement AllocMem() and FreeMem() routine. This might mean that either old bugs have been fixed or new bugs have been introduced. Anyways, check out for possible bugs since this is a release with a "0" in the version. This version might be much faster than the 1.2x version, due to some optimizations. WARNING: Since the new 1.30 release of PoolMem replaces the Exec memory functions completely, YOU MUST RUN POOLMEM IN FRONT OF OTHER MEMORY PATCHES. ESPECIALLY, RUN POOLMEM FIRST, MUNGWALL AFTERWARS or Mungwall will be nonfunctional.

Version 1.31

PoolMem includes a patch for AllocAbs() to be able to allocate absolute memory from the scratch list. This *should* help to make PoolMem working with the 68040/68060/ppc library, thus, it should be possible to run PoolMem IN FRONT of these libraries, i.e. "SetPatch". Would be nice if somebody could sent me a report whether this works now or not.

Additionally, new versions of the FragMeter and ShowMem program are included.

Version 1.32

Rewrote the startup segment completely. All 1.2 compatible code, argument parser etc... removed. Added a proper version check: PoolMem requires OS 3.0. Added the NOMERGE option to avoid merging of FastMem of different speed. As a result of the new argument parser, the CHIPFWD option DOES NO LONGER include the "INSTALL" option, both arguments should be given. The PoolMem algorithms hasn't been changed, though. Included new versions of PatchRAM and ShowMem. PoolMem 1.31 and above works fine on PowerUp-Systems. Thanks to Andreas Kleinert for testing! However, the current version does not yet adjust memory correctly for the PowerUp system, that's still left to the ppc.lib and AllocP32. Might change in the future, though.

Version 1.33

Removed a bug in the PoolMem shutdown code. The pre-1.33 releases did not free the scratch list completely. Added the SCRATCHONLY keyword, it installs only the "scratchlist" of PoolMem, not the dynamic pools. Added the PUDDLESIZE, PUDDLETHRES and SHRINKTHRES keywords to make the parameters of the small memory pool adjustable. Rewrote a lot of the kernal routines, mostly for optimizations.

Version 1.34

Added the SMALLHEADER keyword. A selected memory block can be set aside for the small memory blocks explicitly. Fixed a documentation error in the Developpers.readme. Thanks, Dave!

Version 1.35

Added a patch for the AvailMem() function to include the bytes in the scratch list. Added the ALLOC32 keyword. Check for caveats . You usually DO NOT want this!

Version 1.40

Integrated partially the PatchRAM AllocVec() patch. Memory requests by the RAM disk are now allocated in reverse order. This feature can be turned off explicitly by the NORAMREVERSE option, see below. Thanks to Raphael Pilarczyk for testing.

Version 1.41

Added an API for the PoolMemDetect program that extracts more information of the last guru due to PoolMem. Added the PoolMemDetect program itself.

Version 1.41.1

Included the pgs_patchmem experimental program. Check the "Extras" section to find out more about this program.

Version 1.42

Added the FREEMEMRESULT option to specify the result code of FreeMem(). The former default value "-1" worked around an FFS bug which has been fixed with release 43.20. However, "-1" conflicts with another bug in "RexxPlus" compiled rexx scripts. I would recommend to set the result code to "-2" to make "RexxPlus" working if the FFS 43.20 has been installed.

Version 1.43

PoolMem does no longer try to pool non-MEMF_PUBLIC memory. This might be of importance if virtual memory is introduced.

Version 1.44

Fixed a minor problem in the allocation function that might have resulted in non-optimal memory layout on some boards and could have caused slow-downs.

Version 1.46

Made sure that the PoolMem allocated memory headers are always in public memory. NOTE THAT THERE IS NO 1.45 RELEASE. SOMEONE WITH A STRANGE SENSE OF HUMOUR UPLOADED A TROJAN WITH A FAKED 1.45 RELEASE. DELETE IT IMMEDIATELY AS SOON AS YOU SEE IT AND REPLACE IT BY THIS 1.46.

Version 1.47:

The PoolMem memory clear routine is now a bit more effective (slightly). Allocations with MEMF_REVERSE set will be no longer taken from the memory scratch list. Added a new option KEEPSORTED that implements a slightly different scratch management which *might* help to improve the garbage collection mechanism under certain circumstances.

Version 1.48:

Added more consistency checks on memory allocation and memory release. In case you see the new gurus 0x81000015 or 0x81000016, you should better contact me.

Version 1.49:

Added a check for patches installed in front of PoolMem. It will now fail in case it finds any patches in the memory allocation routines. This rule has been valid before, but it seems nobody cared about this point. Included a new release of PatchRAM because I found that the RAM handler is too low on stack.

Version 1.50:

The patch check was a bit over-critical in the sense that it checked explicit- ly whether the library entries where in ROM which is false on machines with a (MMU-less) RAM remapped ROM. I'm now checking, alternatively, whether the entries go to the exec segment list which should avoid the problem.

Version 1.51:

Only some cosmetical changes, and updated the guide.

## 1.12 Index