palis

Hans Bühler and Codex Design Software

COLLABORATORS

	TITLE : palis	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Hans Bühler and Codex Design Software	January 19, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

palis

1.1 Palis V1.02 - How to make things twice ..... ;^(

```
                                ·C·O·D·E·X· ·D·E·S·I·G·N· ·S·O·F·T·W·A·R·E·
                                 presents:

                               PALIS V1.02
                          the patchlib solution


                  Introduction$^1$

                  Requirements & Installation

                  Bugs & Problems

                  ViewPALIS

                  PatchTest

                  How to patch a library$^1$

                  The source$^1$

                  Using Palis in a production$^1$

                  History

                  Copyrights ?

                  Author
                    $^1$: These paragraphs had been changed since the last release  ↩
                        of Palis.
        Please read them even if you already know about Palis.

        written 17.1.1995 by Hans Bühler, Codex Design Software
                  This program might be freely distributed.
```

1.2 Introduction to Palis...

INTRODUCTION

Well, since I had some talking with a guy called Osma I know that there are several other programs that simply do the same as Palis.
Therefore the introduction is obsolete for you if you know about

SaferPatches
or any other Patch-Libraries-Manager.
For all others: Palis allows programs to remove their patches even if these patches have already been overwritten.

1. Who needs Palis ?

Everybody who doesn't have another Manager or who doesn't love it.

2. What's the problem ?

In simplicity, Palis tracks all patches made by exec/SetFunction() and will remember them. Then, when a program tries to remove that patch, Palis will ensure that former calls to the patched function will always work as they're meant to.

Click here for an example story...
You may take into account that Palis will _not_ use much more memory than needed, actually. Since it's a very small program you should use it to prevent your system from beeing crashed by such things as described above.

3. Everything fine ?

Well, I think Palis helps to solve the problem described above. There's another kind of problem that Palis cannot solve for you:
Is there any task still using your function ?
That means: If you savely remove a patch from any library-function it's not sure whether there might be some program that is still running your function !

1.3 What happens with and without Palis...

Here is a little example (for all who don't knwo what I'm talking of):

SETTING:
Two programms: "first" is run at first,
"second" afterwards.

PLOT:
1. Program "first" makes a patch to intuition.library/OpenWindow() (-204)
It sets its function 'firstOpenWin()' there.
This function does something and jumps into the original function,
I gonna call it 'firstOpenWin()' from now on.

2. Program "second" makes another patch to OpenWindow() and installes its function 'secondOpenWin()'~there.
   This functions again does something cool and calls the old function
   => it calls 'firstOpenWin()'~since this function was set to intuition.library before by "first".

3. Any programm opens a window.
   It calls 'secondOpenWin()' which calls 'firstOpenWin()' which calls 'intOpenWin()'. Everything is fine since "first" and "second" are good programs and do not cause any trouble.
4. Window is closed.

Now, pay attention...

5. User thinks that the "first" program isn't a good program and removes it.
   "first" has to remove its patch to intuition/OpenWindow() and sets ITS old pointer as returned from exec/SetFunction() to intuition.library.
   => now the original pointer is restored.
6. User starts a program that opens a window and wants program "second" to make some stuff with that window.
   Unfortunately, "first" has re-installed the original 'intOpenWin()'.
   Therefore 'secondOpenWin()'~won't be called since 'intOpenWin()'~is now set to the original vector of intuition.library.
   The user may wonder why "second" doesn't do its job anymore.
7. "Well,", user says, "it doesn't work... then I gonna quit that program "too."
   => Program "second" tries to de-install its patch... it sets ITS old function 'firstOpenWin()'~(which isn't there anymore since "first" has been removed times ago...).
   It installs 'firstOpenWin()'~to intuition.library because "second" doesn't know that the program "first" is dead.
8. Now user opens a window... the program calls 'firstOpenWin()'...
   BUT THERE ISN'T ANY 'firstOpenWin()'~anymore... ;-(
   => BOOOOOOM

RESULT:
    Software failure
    Task held...

Of course there're are various tricks to avoid such crashes. Most programs will not install their own funcion pointer to the library but a pointer to a small piece of memory which looks like this (assembly):

```
$00 UWORD jmp
$02 APTR  func
```

where 'func' points to the function which the program wants to install. If it removes itself now, the 'func'~entry is set to the previous (old) function returned from exec/SetFunction(). That works but each time you install such a patch some memory (8 Bytes) will remain in memory unused. I doesn't bother about that 8 bytes but due to the handling of AllocMem() (AllocVec() uses 12 bytes) it will cause memory-fragmentation.
Other programs (like MagicMenu from Martin Korndörfer) send a request to the user allowing him to choose between simply deactivating the program or forcing the program to remove itself (what this means is been explained in my little story above ;^).

And Palis... ?

Here's where Palis comes into effect: Palis will in fact do nothing else like I explained above: It will catch each exec/SetFunction() call and will not install the function which is to be installed but a 'dummy' function as described above, automatically.
But it has a great advantage: Due beeing able to recognize _each_ exec/SetFunction() call it will note if a program wants to remove its patch using exec/SetFunction(). Moreover it can detect whether the entry (the dummy function block above) is needed anymore.
Again the same story as above during Palis is active (Palis jobs are marked by '>'):

```
SETTING:
  Two programms: "first" is run at first,
                 "second" afterwards.

PLOT:
  1. Program "first" makes a patch to intuition.library/OpenWindow() (-204)
     It sets its function 'firstOpenWin()' there using exec/SetFunction()
     which points to a Palis-function.
     > Palis will recognize that this is a true patch (no attempt to remove
     > a previously installed function) because it tracks all functions that
     > have been made to intution/OpenWindow().
      > Hence it will generate a little dummy-function (see above...)
     > which will be installed to intuition/OpenWindow() which will simply
     > call 'firstOpenWin()'.
     > Moreover Palis will store the result from the original
     > exec/SetFunction().
     > This old funtion 'intOpenWin()'~will be returned to "first" thus it
     > can execute the original function if it needs to do that.
     The new function does something and jumps into the original function,
     I gonna call it 'intOpenWin()' from now on.
  2. Program "second" makes another patch to OpenWindow() and installes its
     function 'secondOpenWin()'~there.
     > Again, Palis notes that it is a new patch.
     > 'secondOpenWin()'~will be installed as 'firstOpenWin()'~has been.
     > Because the function currently set to intuition/OpenWindow() is _not_
     > 'firstOpenWin()'~but a dummy function from Palis this functionaddress
     > will be returned to "second".
     This functions again does something cool and calls the old function
     => it calls the dummyfunction installed by Palis which calls
     'firstOpenWin()' since this function was set to intuition.library
     before by "first".

  3. Any programm opens a window.
     > It calls a dummyfunction which calls 'secondOpenWin()'.
     > 'secondOpenWin()'~does something and calls a dummyfunction which calls
     > 'firstOpenWin(). 'firstOpenWin()' calls the original 'intOpenWin()'.
     => Everything is fine since "first" and "second" are good
     programs and do not cause any trouble.
  4. Window is closed.

 Now, pay attention...

  5. User thinks that the "first" program isn't a good program and removes
     it.
     "first" has to remove its patch to intuition/OpenWindow() and sets ITS
```

```
     old pointer as returned from exec/SetFunction() to intuition.library.
     > Palis finds that old pointer in its internal lists thus finds out
     > that "first" wants to remove its function from the library.
     > Additionally, Palis notes that there's another patch which had been
     > installed after "first" made its patch.
     > Therefore Palis won't do anything with intuition/OpenWindow() but
     > replaces the 'func'~pointer in the dummyfunction by the function
     > which had been replaced by "first". This is 'intOpenWin()'.
  6. User starts a program that opens a window and wants program "second"
     to make some stuff with that window.
     > Since the dummyfunction for 'secondOpenWin()'~is still set to
     > intuition/OpenWindow() this dummy-function is been called which
     > jumps right into 'secondOpenWin()'. 'secondOpenWin()'~does its work
     > and calls the function from which it assumes to be the original
     > function. Actually, this function is the dummyfunction for the "first"
     > patch. It jumps in there but the dummyfunction does nothing else than
     > jumping into the original 'intOpenWin()'... luckily !
     The user may wonder why "second" did its job well.
  7. "Well,", user says, "fine, it works ... but..." and removes "second"
     from his system.
     > "second" calls exec/SetFunction() with ITS old function (which is
     > the "first" dummyfunction)
     > Palis notes that "second" wants to remove its patch. Moreover
     > it recognizes that no further patches are followed by that one.
     > That means that the "second" dummyfunction won't be needed anymore.
     > Additionally, the "first" dummyfunction is not needed !
     > Palis will install the original function 'intOpenWin()'~to
     > intition/OpenWindow() and will free all memory having been used to
     > track all these patches.
     > => no more memory is spent for 'dead'~dummyfunctions.
  8. Now user opens a window... the program calls 'intOpenWin()'...
     => Everything is fine !

RESULT:
     A new window.
```

1.4 Requirements & Installation

```
                              SYSTEM REQUIREMENTS

                                    Palis

                  Palis needs Kickstart V2.04 or higher to run.

                                  ViewPALIS

  ViewPALIS (an additional program) needs the following libraries to run:
                  icon.library V37+, diskfont.library V37+
       reqtools.library (c)Nico François will be used if available
                  (whereby it won't use it very often ;-)


                                INSTALLATION

- Copy Palis somewhere in your path.
```

- Copy ViewPALIS somewhere in your path.
- Make sure that Palis will be started _before_ any other programs that patch libraries (execpt those that you won't remove ever - like Setpatch etc.)
- Copy this poor guide somewhere you want.

The installation script which comes along with the archive will check whether there's another PatchManager running or not.
In the first case it will not install PALIS - you have to do it yourself since I don't know which program to remove (even if _you_ think it is worth to do so ;^).

1.5 Bugs & Problems when using Palis

BUGS

I think that Palis itself will do its job without any bugs. Due its a simple program and I was using it for three month now I expect it wouldn't have problems anymore. Please note that Palis won't do any patch-tracking when it runs out of memory. Then it returns to the original system-behaviour.

PROBLEMS

Alien software

There could occur several problems with alien software having not prepared to work with Palis:

a) A program uses its own dummy-function to avoid conflicts with its patches. That means that this memory won't be freed and on the other hand that Palis will never remove its own dummy-function while assuming that the program hasn't tried to quit yet.
Therefore some memory will be wasted - nothing else.

b) Some programs check the current library-vector before they remove their patches. They check whether these vectors are unlike their own function-addresses and will reject to quit if they think so. They will of course NOT find their own funtion-addresses their due Palis has put its very own functions in there in the case they perform a direct check like:

```
if ( myFunc == *( (APTR) &((char *)lib)[offset+2] ) )
  ok = TRUE;
else
  ok = FALSE;
```

Programs that use another way (as suggested by some people from c.s.a.programmer):

```
Forbid();
dummy = SetFunction(lib,off,oldfunc);

if(dummy == myFunc)
  ok = TRUE;
else
```

```
    {
      SetFunction(lib,off,dummy);
      ok = FALSE;
    }
    Permit();
```

will work properly execpt that "ok" will always be TRUE (thus they will never again refuse to quit ;^)

Click here to see how to code this.
Palis problems

c) Palis will _not_ save your patches of other things than beeing overwritten and stuff. Note that _you_ are still responsible for the cases that other programs do actually execute your function while you are about to remove it.
See
example code
for such things.

d) You are not allowed to quit Palis !!!!!!
For developpers: Send a ctrl-c to the program in order to terminate it. Non-developpers must not do this !

e) Making patches to exec/ Obtain[Attempt]Semaphore(), exec/ Obtain[Attempt]SemaphoreShared, exec/ ReleaseSemaphore(), could be dangerous since Palis will use them itself. Check out whether it works.

1.6 ViewPALIS - information for Palis users

ViewPALIS

I added this little external program to the archive using it you may determine how Palis works or what it had done for you that far.

GUI

Here's the GUI (designed using GadToolsBox V2.0c ©Jaba Development):

```
  +-+------------------------------+-+-+
  +·| ViewPALIS V1.00 hotkey = xxx | | | <= title. hotkey might be adjusted
  +-+------------------------------+-+-+    using tooltype CX_HOTKEY=xxx
  |                                    |
  |           Current patches:         |
  |                                    |
  | +------------------------------+-+ |
  | +                              | | |
  | +                              | | | <= list of currently known patches.
  | +                              | | |    patches that have been removed
  | +                              | | |    and which are still simulated
  | +                              | | |    are marked <removed>
  | +                              |#| |
  | +                              |#| |
  | +                              |#| |
  | +                              |#| |
```

```
  | +-------------------------------+-+ |
  |                                     |
  |   [Hide] [Update][About] [Close]    | <= action gadgets...
  |                                     |
  +-------------------------------------+
```

```
Hide:   Closes the window but keeps ViewPALIS active.
Update: Since ViewPALIS just takes a copy of Palis internal data,
        you may want to "update" the list.
About:  ;^)
Close:  Ends up ViewPALIS
```

TOOLTYPES

These tooltypes are known to ViewPALIS:

```
CX_POPUP:  Open window when ViewPALIS is been started.
CX_HOTKEY: Hotkey to re-open the gui if ViewPALIS is "hidden"
           (Default: "lalt lshift p").
CX_PRI:    Priority to load ViewPALIS (when loading workbench)
           and pri of commodities job.
WINX,WINY: Last window position (will automatically be saved for you).
DONOTWAIT: Workbench shouldn't wait for ViewPALIS having been finished
           (You cannot disable that ;^)
```

1.7 Patchtest V1.00 - find out whether there's a patch manager

[copy of patchtest.doc]

·C·O·D·E·X· ·D·E·S·I·G·N· ·S·O·F·T·W·A·R·E·
presents:

PatchTest V1.00

INTRODUCTION

PatchTest is a little program that is been designed to work along with the Installer from AT.
It is able to check whether a patch manager like SaferPatches is running on the current system.
Therefore an installation script is able to decide whether a patch manager should be installed on the target system or not.

REQUIREMENTS

None.

REALIZATION

PatchTest will do the following: It will SetFunction() a function twice and will try to remove these "patches" in reverse order, afterwards. If that fails, no patch manager is active.

See source for more information.

HOW TO USE IT

General CLI-Use

PatchTest expects the following parameters:

```
LIBRARY/A  : Here you specify the library which will be patched for a test.
OFFSET/N/A : This is the function offset where PatchTest will install its
             test patch.
QUIET/S    : No output (for the final use in the Installer-script).
```

Along with PatchTest, you find the "patchtest.library" in the archive. This library has only one function (offset -30) which does nothing. Since it could be dangerous to patch _any_ library at _any_ offset, please use this library for the test:

"Patchtest LIB=PatchTest.library OFF=-30"

Note that since OS2 OpenLibrary() is able to open a library even by relative path-names:

Let's say, PatchTest & patchtest.library is placed in a subdirectory "adds" in your archive, you can even use

"adds/PatchTest LIB=adds/patchtest.library OFF=-30".

=> You don't have to copy "PatchTest.library" to LIBS: or stuff !

Usage with Installer

PatchTest will return errorcode 0 if a patch manager is been detected or 5 if none had been found (or 10/20 if a heavy error like "Can't open library" occured).
For Kick2.04+, the Installer allows to use it this way:

```
(if (= 5 (run "PatchTest PatchTest.library -30 QUIET"))
    (
    (if (askbool (prompt "No patch-manager found !\n"
                         "Do you want one ?")
                 (help   @askbool-help)
                 (default 1)
        )
        (  (copyfiles .... copy patch manager here
        )
    )
)
```

WARNING

Since PatchTest patches a library it might be dangerous to use it. Please be careful and do not use it if you do not know enough about the AmigaOS.

DISCLAIMER

The author cannot be held liable for the suitability or accuracy of this manual and/or the program(s) it describes. Any damage directly or indirectly caused by the use or misuse of this manual and/or the program it describes is the sole responsibility of the user her/him self.

AUTHOR

TestPatch V1.00 and TestPatch.library had been written by Hans Bühler and are Freeware (=> do what you want ;^).
TestPatch is part of the aminet:dev/misc/Palis.lha archive where you may find the source of it, too.

Hans Bühler
codex@stern.mathematik.hu-berlin.de
buehlhan@kadewe.artcom.de
http://kadewe.artcom.de/~codex

1.8 How to patch a library

This paragraph is has completely changed since V1.01 of Palis.

SOLUTIONS FOR A SMALL PLANET

Well, since I unfortunately found out that there are many or at last one program that does exactly the same as Palis I think it isn't worth paying attention to Palis only if you want to make patches to your system.
Therefore I developped an object offering three functions to you which you may use to install various patches to various libraries.

Please read cdxMakePatches.h for further information !

The files can be found in the archive, directory "cdxMakePatches/".

These functions will no longer stuck on palis itself to remove a patch. In simplicity, they do the following:

cdxPutFunction(...)

Will install a patch using SetFunction() and will track the patch using a global variable. Therefore you won't need to remember anything you can remove _all_ patches (of one heap) by

cdxRemAllFunctions(...)

This function will do the following (simple example because it is only ONE function):
Let's say, you installed

```
    oldfunc = cdxPutFunction(...,lib,off,newfunc);
```

calling cdxRemAllFunctions() will now do:

```
   Forbid();

   dummy = SetFunction(lib,off,oldfunc);

   if(dummy == newfunc)                  // manager active or no second patch
   {
     ok = TRUE;
   }
   else
   {
     SetFunction(lib,off,dummy);
     ok = FALSE;
   }

   Permit();

   return ok;
```

...i.e. it will return TRUE if it was possible to remove the patch.

The advantages of cdxRemAllFunction() are:

- It will check whether ALL functions set using cdxPutFunction() can be removed together. If it isn't possible, cdxRemAllFunctions() will restore ALL original pointers. Therefore a program can try to remove its patches and can inform the user that it didn't work (if so) without taking care of any function that might already been removed.
- cdxPutFunction() is able to add a little piece of code to your function(s). Using it you can remove the function even if it isn't possible normally. The only point is that in that case a leck of 10 bytes per function must be taken into account. cdxRemAllFunctions() will do so only on demand (option "force").
- cdxRemAllFunctions() is capable of a simple trick that shall protect your code from beeing run by another task after you removed it:
cdxPutFunction() can initialize a exec/struct Semaphore for you.
Then, _after_ cdxRemAllFunctions removed a function it will try to ObtainSemaphore() this semaphore. Since I suggest your "new" function code should look like

```
int NewFunc(...)
{
   ObtainSemaphoreShared(NewFuncSem);

   .....

   ret = result(...);

   ReleaseSemaphore(NewFuncSem);

   return ret;
}
```

that will only be possible if no task runs the code between the ObtainSemaphoreShared()/ReleaseSemaphore() pair (in that case ObtainSemaphore() from cdxRemAllFunctions() will wait until the smaphore is released using ReleaseSemaphore() from inside the function.
Since after ReleaseSemaphore() from "NewFunc" not much code to run is left for the other task a quick dos/Delay() after calling cdxRemAllFunctions() will allow you to free the memory occupied by your NewFunction(s) nearly without any danger to the running process.
Note that this feature should be used along with time-consuming functions only.

1.9 Palis/ViewPALIS source code available.

You may want to have a look at the source of Palis/ViewPALIS. ↩
Note that
any access to internal data structures is forbidden execpt when following the rules defined in Palis.h. However, I don't think that you need to access Palis itself. Keep your fingers off !

```
src/
  Palis.h         - Include for programs that do want to access Palis.

  Include.h       - Includes used by Palis (little more since I always copy
                    the same file...;^)
  pl.h            - Include for Palis.exe

  Basic.c         - Basics as requesters and stuff.
  Com.c           - commodities stuff.
  Main.c          - Mainloop (actually doesn't do much)
  pl.c            - Initialisation object.
  SetMan.c        - Palis installation/work/etc.
  SCOPTIONS       - Options for SAS/C

src/vpl/
  plView.h        - include.
  PalisViewGUI.c  - src from GadToolsBox; fixed using gtp 1.07
  PalisViewGUI.h  - include from GTB
  basic.c         - requesters and stuff.
  Main.c          - mainloop.
  Com.c           - commodities stuff.
  Gui.c           - managing the gui.
  Lists.c         - list work.
  plView.c        - main.
  ttype.c         - argument parsing (object not yet included; might be
                    copied from the author if needed).
```

See copyright notes !

1.10 Using Palis - what do you have to care for ?

SUPPORT Palis !

If you want to support Palis within a program of you feel free to do so. No copyright will restrict its use.
Here are some suggestios how to realize that:

1. Do not rely on running with Palis

I already stated that it wouldn't be wise to _assume_ that Palis is active. Therefore please use the
object
supplied with this archive.
This object will make your patches safe even if there's no patch-manager.
We cannot command others to use Palis (...unfortunately ;^).
From V1.01.1 on there's an object file in "cdxMakePatches/" which might be linked to your program. See the Read.me file for more information !

2. Which parts of Palis.lha should be distributed ?

I suggest that you distribute the whole archive since others might want to know what Palis does, too.
In all cases, please let the user know where to find it (aminet:dev/misc).

3. Installing Palis with using the Installer

I assume that it wouldn't be wise to install Palis or another Patch-manager on the target-system.
Therefore please follow these points:

Use this construction to find out whether there's already a patchmanager:

Archive:

```
:
:
palis/palis                  [executable]
palis/
             patchtest
                           [executable]
palis/patchtest.library      [library for patchtest]
palis/palis.guide            [this guide]
palis/palis.guide.info
:
:
```

Script:

```
(if (= 5 (run "palis/PatchTest palis/PatchTest.library -30 QUIET"))
    (
    (if (askbool (prompt "No patch-manager found !\n"
                         "Do you want one ?")
                 (help   @askbool-help)
                 (default 1)
        )
        (  (copyfiles (prompt "Copying Palis V1.02...")
                      (help   "This will ensure that patches can be removed !")
                      (source "palis/Palis")
```

```
                    (dest   "C:")
                    (optional "force" "nofail")
                    (safe)
          )
          (startup "Palis" (prompt "Auto-run palis when starting...")
                           (help   "This is neccessary...")
                           (command "C:Palis")
          )

          (... ask whether to copy guide and copy it ...)

          (message "Palis is installed now."
                   "Note that it will start to work when you reset"
                   "your machine."
                   "Until then, you may work with out it !"
          )
        )
    )
)
```

1.11 History

```
                                          History of Palis

V1.00   Initial release.
V1.01   Little bug fixed: If you started Palis V1.00 twice, it would
        not recognize that it is already running !
        That's now fixed.
V1.01.1 Documentation partly rewritten since I noted some people thought
        that Palis is like other programs (e.g. SetManager).
        Added two objects which include the described functions for patching
        libraries with support of Palis.
V1.02   Several bugs fixed. Palis is now as Saferpatches.
        Another
                information tool
                , but the same !
        No direct support availanble.
        Source< does work with _any_ patch manager now.

                        History of ViewPALIS

V1.00   Initial release.
V1.02   Seconde release (V1.02).
```

1.12 Copyrights

```
                Disclaimer

 The  author  cannot be held liable for the suitability or accuracy of this
manual  and/or  the  program(s)  it  describes.  Any  damage  directly  or
indirectly caused by the use or misuse of this manual and/or the program it
```

describes is the sole responsibility of the user her/him self.

Copyrights

Palis V1.02 & ViewPALIS V1.00 have been written by
Hans Bühler
for common
use. This is real freeware. Do what you want.

1.13 Alt F4 !

Was sagt ein Intel-Entwickler zu Neujahr 1996 ?
- Schönes neues 1995,99978899988999889999999.....
oder:
- was ist kooperatives Multitasking ?‘
antwort: scheiße.

1.14 How to waste time efficiently

1.15 Have a chat with me.

h a n s b u e h l e r : c o d e x d e s i g n s o f t w a r e

[email]
codex@stern.mathematik.hu-berlin.de (Hans Bühler)
buehlhan@kadewe.artcom.de (Hans Bühler)

[www]
http://kadewe.artcom.de/~codex

[smail]
Hans Bühler
Kirchstr. 22 - 10557 Berlin 21
Germany

:
:
:
:
.
.

Alt-F4 !

.