AMCAF Extension V1.40 Manual

AMCAF Extension V1.40 Manual by Chris Hodges.

AMCAF V1.50beta4 final FREEWARE release. Sorry, but I didn't have time to update the manual.
You'll have to find out the new commands since V1.40 yourself (or by looking at the history).

Welcome

Welcome to AMCAF.
You will certainly be very satisfied with this product. This extension contains over 200 commands and functions for various areas.
The AMCAF extension is more than 42 KB in size which makes it the biggest extension for AMOS Professional known to me.

Some commands may work better or even require Kickstart 2.04 or higher. If you want to take full advantage of the AMCAF extension then you should really think about upgrading to 2.0 or 3.1, if you haven't already.

By the way, the extension must be entered at slot number 8 in the AMOS Pro extension list, but this is done by the installation software for you!

Don't forget to read the Notes, too!

No program is absolutely bug free and for that reason I ask you to please report any bugs as quickly as possible. However, it's rather useless, if you only write: The command xxx crashes my machine.
Please give me a detailed description of your computer and enclose the part of the program that causes the problem, but only if you like doing that kind of thing.

I welcome any suggestions that you may have, any problems that you may encounter, bug reports for certain, or even any hints and tips that you may have for me!

Contact address
If you have got access to a ftp site, there will be free updates coming up regularly on Aminet (dev/amos). Make sure you download the English version and not the German one.

Enjoy AMCAF and may the force be with you!
Chris Hodges

Notes

If you haven't already installed AMCAF, you should do so as soon as possible, because no example program runs without the AMCAF extension (which is quite obvious ;-)).

The installation process is quite easy and quickly done. Just read the instructions carefully which appear when you start the installation program.

During the process you will be asked to enter your name, if you are using a registered version. Please enter it correctly. It will be encrypted and saved into the extension file. Remember that you MUST NOT copy the registered version of AMCAF to anybody else. YOU have paid for this software, why should someone else get it for free?

AMCAF can either be written directly onto hard disk or onto a copy of your AMOSPro_System disk. When installing to disk, AMCAF allows you to either delete some files directly on the disk to get some free space for the extension or to copy the system files to a new disk (preferably FFS), which is formatted by the program directly.

Due to lack of space on the examples disk, the file "mod.no good" has been written onto the installation-disk. The only program that needs this file is the demo "NoGood.AMOS". It automatically loads the file either from the install disk or hard disk.

To unleash the power of the command "Set Rain Colour", the main Amos Pro library must be modified a bit (or better: a byte). So please first start the program "SetRainColourPatch" and then restart AMOS Professional. This patch remains permanently so you have only to start it once.

The examples haven't been written for you to start them, look at them for a moment and then load the next. Most of the programs are well documented and allow you to learn something new. Take your time, it's worth it.

Some commands are very sensitive against abuse. Do save your programs regularly, especially when you've added a new routine.

Contact

If you have any questions or even found a bug, then leave me a mail or call me directly.

Don't forget to enclosed a DETAILED DESCRIPTION OF THE ERROR with the configuration of your computer and the piece of code which doesn't work correctly if you've encountered a bug.

Snail-Mail:
Chris Hodges
Kennedystraße 8
D-82178 Puchheim
West Germany
Tel: +49-89/8005856 (Voice/Modem)

Bank account:
Christopher Hodges
Account 359 68 63
Sparkasse Fürstenfeldbruck
BLZ 700 530 70

E-Mail: platon@cu-muc.de

WWW: http://www.platon42.de

IRC: platon42 on #amiga, #amigager (Efnet)

Bugs

This is a small list of AMOS bugs you must be aware of:

1. The AMOS string management is not bugfree. Be especially careful with AMAL programs!

2. The AMOS Default routines are not called: If you load AMOS and then start a program, the extension default routines are not called. I.e if a music is running from the previous call, it will not be stopped.
Conclusion: - Call any accessory (e.g the help-accessory) once.

3. Programs cannot be compiled: The compiler prints out the error 'Not an AMOS program'. If you use memory banks, all bank lengths must be even. If this is not the case, the programs do work from the editor but not compiled.
Conclusion: - Extend odd banks by one byte using Bank Stretch.

4. A program that uses data lines does not work correctly when compiled: There must not be any comments after a data line. Data commands must be completely alone on a line. Otherwise the compiler will interpret these comments as data.
Conclusion: - Write the comments into a separate line.

Even worse: a friend of mine was faced to a guru when trying to compile a program that uses approx. 500 KB of data lines. As for now, I found no solution for this problem.

Bugs that are interesting for extension programmers:

1. Your commands or functions can only have a maximum of 9 parameters. Everything above this will not be put on -(a3).

2. All routines, that are jumped to directly by a token must be placed in the first 32 KB of code. Otherwise the command will crash. Additionally the distance of a RBras or RBccs jump must not exceed 32 KB.

Copyright

Greetings

First, best regards to everybody who has registered AMCAF, thank you very
much.

Special greetings fly to:

My parents (Hallo!)
My brother (Come on! I want to see a wonderful raytracing animation!)
My sister (Hi pumpkin!)
Hans Peter Obermeier (Anything new? ;-))
Ralf Schulz (Oh, I cannot beleve you've bought an ugly MS-Tinbox... :-( )
Markus Ungerer (Schreib mal wieder eine Kurzmail!)
Bernd Ungerer (Danke fürs Beta-Testen und für die guten (?) Vorschläge!)
Michael Ufer (You little hobby magician! Thx for the many refreshing mails)
Oliver Ufer (Thx for the huge amount of suggestions ;-))
Oliver Seibert (Surprise had been the best box known to me!)
Oliver K.
Dirk Drießen (Man gönnt sich ja sonst nix ;-))
Ralph Bernecker (Hello jMS/Dr.Feelgood/Striker/fELON)
Alexander Kunz (Thx 4 da kewl tunes and support!)
Omer Sasic (+++)
Claude Müller (Greetings to Swiss! I hope your back from the army soon!)
Dirk Schulten (Hallo Maus-User! ;-))
Andre Panser (Schreib mal wieder)
Andreas Duncker (Thx for your supporting mail)
Mathias Mischler (Thx für your support and logement!)
Andreas Zymny (Das Quotezeichen bleibt UNKONFIGURIERBAR! Basta! ;-))
Kriegsheld (Jaja! Eigentlich ist ja heute schon morgen, gelle?)
Henning Baron
Rainer Benda
Robert Rothhardt (Thx for the best time in all my school life!)
Florian Fackler (Strato Impact rulez!)
Thomas Nölker (Good luck with your AMIPS-Extension!)
Jürgen Schäfer (Sorry wegen der Gif-Geschichte, vielleicht kommt das noch)
Greg Cox (Keep up the good work!)
Michael Cox (Thanks for adding me to the mailing list ;-))
Marco Eberhardt (Thanks for your nice mails)
Carsten Albert
Paul Hickman (death to all tennis players! ;-))))
Andy Church
Ben Wyatt (Greetings! ;-))))
Petri Hakkinen (nice 3D engine!)
Roy Antonsen (stop drinking so much booze ;-))
Martijn Wehrens (nice mails from ya! Keep that up)
Semprini (you're a REAL friend!)
Mark Wellington (nice stories from ya, keep that up!)
Daniel Rädel
Thomas Nokielski
Jari Jokivuori
and all the others, who know me and I forgot by mistake.

In addition, more greetings go to my modem friends:

Magic, Lemming, VIP, Schneemann, Killer, Marvin, Harry, Holger,
Caboose, Blue Shogun, Ralli, REYem, Nosy and Kily.

Merlin, WotaN, Vinzenz, Fritz, Braumeister, Amigaman and Kai.

Dr.Dre, Tomy, Brocker, Case, Omer and Guru.

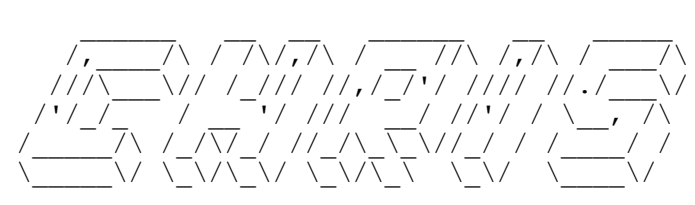
Curses to:

Hendrik Heimer (You know why, Mr. Software Society)

François Lionet
Jester (Christoph Steinecke)
Nobody (Horst Bressem)
Intel, IBM and MicroSoft
German Telekom
Europress
Data Becker and Sybex
R2B2 ($$%$&%$$)
Phase 5

Good... I hope, I've forgotten anyone... erm yes... greetings to all
members of the RAMSES-Computerclub.

CU



AMCAF Extension History

V1.50beta 11-Jan-98

- Found some wrongly assembled lines in the protracker vibrato table.
  If someone encountered some scarce problems the modules using
  fine-vibrato, these should be gone now.
- New c2p routine by Mikael Kalms. Up to 20%-80% faster and now also
  supports plane depth from 4 to 6!
- Fixed a little bug that wouldn't play samples longer than 64KB
  correctly (just a lsr.w instead of a lsr.l).
- Probably fixed a lot of bugs and spurious crashes introduced with the
  last beta.
- Fixed a newly introduced bug in Lzstr$ and Lsstr$.
- Found something new out about AMOS and fixed some bugs with editor-
  compiled programs.
- Added new transition commands:
  - Alloc Trans Source bank
  - Set Trans Source bank/address
  - Alloc Trans Map bank,width,height
  - Set Trans Map bank/address,width,height
  - Alloc Code Bank bank,size
  - Trans Screen Runtime scr,bitplane,ox,oy
  - Trans Screen Dynamic scr,bitplane,ox,oy
  - Trans Screen Static NOT YET IMPLEMENTED
  - Trans C2p chkbuf NOT YET IMPLEMENTED
  Some last words: Allocating the Code bank to small will cause memory
  overwrites. Wrong or stupid parameter values are not checked for
  validity. The Trans commands are still a bit slow (3 VBLs for a 256x256
  pixel transition with MC68060 and all caches disabled, 0.8 VBLs for the
  same transition with all caches turned on).
  Maybe the C2P Transition will be a bit better. Still need to update the
  c2p to a state-of-the-art routine (by Kalms).
- New for ALL PT sample commands: Entering a negative value as channel
  bitmask will trigger the Pt Free Voice function which will search for
  the best channel number to use for sample replaying. Really handy
  indeed! Pt Free Voice is rather complex:
  - it first checks if the bitmask is zero and if so, it returns 0.
  - then it checks, if only one bit was set in the mask and returns
    the same.
  - after that, it finds out if all four channels (or the ones given in
    the bitmask) are currently playing a sound. If so, it returns the
    channel bit of the sound which will cut off least.
  - if there are free channels, it verifies, if music is playing at all.
    If not, it returns the first free channel.
  - so for music is playing, it looks if the user has masked out a
    channel from the music (and which is currently free).
  - then it checks for the remaining channels are playing looping sounds
    for those will be not taken if necessary.
  - the last step finds out the shortest sound currently played on the
    channels, and then returns its channel bit.
  So you'll get the most of your game sfx and music if you're using the
  free voice function!
- New functions:
  - =Pt Free Voice[(bitmask)]
- Lzstr and Lsstr now return '9999...' if number is too large to fit into
  the given amount of digits.
- Sample&music-mixer changed to be more accurate (samples could have been
  cut off some VBLs too early if the music was playing at higher cia
  speeds than 125bmp) with cia-timed mods (at least this should have been
  the case in theory).
- The sample replayer now uses a 2 byte chip mem buffer to kill the
  high pitched beep sound sometimes appeared on samples which started
  with 2 non-zero bytes. FixSamBank should now no longer be required.

- Pt Raw Play can now play looping samples by giving a negative length.
- Implemented Vu Meters to AMAL. BUT it will ONLY work, if you REMOVE
  the original AMOS Music Extension. No other way to get it work!
- Added support for negative numbers in Lzstr$ and Lsstr$.

V1.44 27-Jul-96

- Fixed a bug in Ssave.
- Fixed a minor bug in Pt Play (cia speed is now reset to 125).
- Fixed a bug in Qsqr. Now bigger numbers than 0-65535 can be used.
  (Warning: starting with numbers >65535 it gets a bit inaccurate!
  Remember that the resolution of the result remains 8 bit!)

V1.43 02-Nov-96

- Added Sload/Ssave. Just the same commands like in the music
  extension. Now you can really remove it!
- Misc bugfixes.

V1.42 18-Oct-96

- Added support for Set Tmpras for Fellipse&Fcircle commands.
- Added type check for Protracker modules.

V1.41 29-Mar-96

- Fixed bug in Xfire.
- Misc changes.

V1.40 26-Dec-95

- New commands:
  - =Pt Cpattern
  - =Pt Cpos
  - =Pt Cinstr(channel)
  - =Pt Cnote(channel)
  - Pt Sam Freq channel,freq
  - =Vclip(v,lower To upper)
  - =Vin(v,lower To upper)
  - =Vmod(v,upper)
  - =Vmod(v,lower To upper)
  - =Insstr$(a$,b$,pos)
  - =Cutstr$(a$,pos1 To pos2)
  - =Replacestr$(a$,search$ To replace$)
  - =Itemstr$(a$,item)
  - =Itemstr$(a$,item,sep$)
  - =Qarc(dx,dy)
  - =Even(val)
  - =Odd(val)
  - =Ham Point(x,y)
  - Set Object Date file$,date,time
  - =Aga Detect
  - Pal Spread c1,rgb1 To c2,rgb2
  - =Ct String(time$)
  - =Cd String(date$)
  - C2p Convert st,wx,wy To screen,ox,oy
  - C2p Shift st,wx,wy To st2,shift
  - C2p Fire st,wx,wy To st2,sub
- Pal Set Screen now does a 'View' automatically.
- Another bug in Ptile Paste has been fixed.
- Sine-Table moved and shortened, so I save about 1536 Bytes,
  added Arctan-Table.

V1.31 03-Oct-95

- Actually, the bug hasn't been in Change Bank Font but in Make Bank Font.
  Although this bug is now fixed, I suppose you have to recreate all old bank fonts.
- Finally removed Rnc Unpack and =Rnp.
- Fixed a bug in Pt Stop which cut off the channels, even if no music had been playing.
- New commands:
  - Pt Continue
- Fixed a bug in Change Bank Font.

V1.30 31-Aug-95

- NEXT OFFICIAL UPDATE.
- Various changes to get the extension under the 32 KB limit.
- Best Pen now supports EHB mode.
- Found a bug in the PlaySample routine and removed it.
- Removed a little bug in Pt Raw Play.
- Obvious bug in Extpath$ fixed (Thx Ben!)
- New commands:
  - =Best Pen($RGB[,c1 To c2])
  - Bzoom s1,x1,y1,x2,y2 To s2,x3,y3,m
  - =X Smouse
  - =Y Smouse
  - Limit Smouse [x1,y1 To x2,y2]
  - Smouse X x
  - Smouse Y y
  - Smouse Speed speed
  - =Smouse Key
  - =Xfire(port,button)

V1.19 30-Jun-95

- New commands:
  - Exchange Bob i1,i2
  - Exchange Icon i1,i2

V1.18 21-Mar-95

- NEXT OFFICIAL UPDATE.
- Fixed a very scarcely appearing bug in Turbo Plot.

V1.17 16-Feb-95

- Little bug in the protracker routines: mask for FineSlideUp was $D
  instead of $F, what resulted in ignoring $2 slides. (Thx Patrick)

V1.16 29-Jan-95

- Little flaw appeared with the protracker cia replay code: Pt Cia Speed
  did not work correctly in compiled programs (but I don't know why!).
- Forgotten a Pt Sam command by mistake:
  - Pt Sam Volume [voice,] volume
- Reworked code a bit to ensure the demo version is working (32 KB limit!).
- New palette handling commands:
  - Pal Get Screen palnr,screen
  - Pal Set Screen palnr,screen
  - =Pal Get(palnr,colindex)
  - Pal Set palnr,colindex,colour

V1.1 28-Dec-94

- Adapted protracker replay code to handle samples correctly.
- New commands:
  - =Pt Data Base
  - =Pt Instr Address(samnr)
  - =Pt Instr Length(samnr)
  - Pt Bank bank
  - Pt Raw Play voice,address,length,freq
  - Pt Instr Play samnr
  - Pt Instr Play voice,samnr[,freq]
  - Pt Sam Bank bank
  - Pt Sam Play samnr
  - Pt Sam Play voice,samnr[,freq]
  - Pt Sam Stop voice
- Tiny optimizations on Ham Fade.
- One more bug in Ptile Paste removed.
- Bug in Speek: Speek did only allow even addresses.
- Silly bug in Lsstr$ and Lzstr$. Often they did produce trashed strings
  or didn't return from the call.

V1.0 16-Oct-94

- VERY FIRST RELEASE VERSION!
- Removed a few little bugs in the registration code.

V1.0ß 06-Oct-94

- Imploder Load didn't free the lock on a file if the loading process was
  successful. Fixed.

V1.0ß 22-Sep-94

- Every string allocation reserves two bytes more. Now there seem to be no
  problems with the string buffer anymore.

V1.0ß 28-Aug-94

- Ptile Paste had a bug. Removed.
- =Extpath$ did overwrite 32 KB of variable buffer using empty strings!!!
  Argl...
- =Ham Best completely rewritten. Works even better now.
- Added one more error check to Coords Read and =Count Pixels.
- Coords Bank reserved a bank which was not completely used up and could
  hold exactly one pair of coordinates less.
- Coords Bank corrupted the memory or crashed when trying to create a bank
  with zero coordinates.
  The same with Splinters Bank. Bug removed.
- =Ham Best rewritten. Now it is much quicker as before!
  Hint came from Dr. Peter Kittel, now ex-employee of Commodore, Class 95.
- Protracker: Vumeter did not support the C-Command, for that reason
  =Pt Vu returned 64 as volume most of the time. Bug removed.
- Protracker: If a channel had been turned off, neither the signal could
  be received nor have speed and other commands been taken notice of.
  Implemented subroutine to handle this case.
- Renamed Secexp to Binexp and Seclog to Binlog.
- Protracker: added more init code. Now there should not be any flaws with
  funks or patternrepeats when playing many modules after each other.
- Moved database from chipram to fastram (really should have done this much earlier!).
- Due to the improvements named above, the code became to long to execute
  a certain branch command. This caused AMOS to crash. Bug removed.

V0.992ß 27-Jul-94

- Yeah, got some vacations at last and can now fully concentrate on the
  manual. Release date of V1.0 was set to 1.9.94.
- Extension restructured, so ensure that all functions, that do have a
  token, are placed into the upper 32 KB. Now everything should work again
  correctly.
- Now you can switch between CIA and VBL timing while playing a module.
- Encountered a small bug in Bank Copy which was killed immediately.
- Re implemented some commands: Rnc Unpack and =Rnp.
- New commands:
  - =Qsqr(value)
  - Bcircle x,y,r,c
- Added clipping for Turbo Plot, Shade Pix and Turbo Point. Now they are
  as secure as the normal Plot and Point commands.
- Improved the speed of Ham Fade a little.
- When reaching the end of a song, Pt Signal now reports $FF.

V0.991ß 18-Jul-94

- New commands:
  - Set Rain Colour rainnr,colour
  - Rain Fade rainnr,colour
  - Rain Fade rainnr To rainnr

V0.990ß 01-Jul-94

- Removed some command to shrink the size of the extension:
  - Rnc Unpack
  - =Rnp
- Deleted a few error messages from Io Error$.

V0.990ß 04-Jun-94

- 'Created' a lethal bug by resorting the command groups. Seems to be an
  AMOS interior bug, but when I write Audio Lock and Audio Free at the end
  of the extension every other command should work. However, I could not
  test if every command works, so be warned! Strangely enough, all
  commands do work when compiled, which indicates the existence of this
  AMOS bug.
  DO NOT USE AUDIO LOCK OR AUDIO UNLOCK IN THE INTERPRETER MODE!!!
- New commands:
  Blitter Copy Limit screen
  Blitter Copy Limit x1,y1 To x2,y2
  Blitter Copy sc1,pl1 [,sc2,pl2 [,sc3,pl3]] To sc4,pl4 [,minterm]

V0.989ß 03-Jun-94

- Discovered bugs in Blitter Fill and Blitter Clear which have caused some
  strange structure faults (e.g quitting from For-Next loops)

V0.989ß 03-Jun-94

- Made Blitter Fill more secure.
- New commands:
  - Blitter Clear screen,plane
  - Blitter Clear screen,plane,x1,y1 To x2,y2
  - Blitter Wait
  - flag=Blitter Busy
  - Shade Pix x,y
  - Shade Pix x,y,planes

V0.988ß 02-Jun-94

- At last Turbo Draw does now support clipping. Even implemented some
  special check routines for the blitter mode.

V0.988ß 31-May-94

- Hurray!!! Splinters do now work totally correctly.
- Various optimizations on Splinters and Td Stars.
  - Reduced memory consumption from 32 to 22 bytes per Splinter.
  - Reduced memory consumption from 16 to 12 bytes per Td Star.

V0.987ß 29-May-94

- New commands, to satisfy Markus:
  - Make Bank Font bank
  - Change Bank Font bank
- New function:
  - =Cop Pos
- Little flaw: Pt Play did not reset the signal to zero.
- New functions:
  - =Vec Pos Z(x,y,z)
  - =Vec Pos Z
- Removed private commands.

V0.986ß 27-May-94

- Added three quite private temporary commands (for a maildisk)
  Will be removed immediately after completing the maildisk.
  - Private A bank1,bank2,bitplane,maxrand
  - =Private B(bank2)
  - =Private C(bank2)
  DO NOT USE!!! Wrong use will crash the computer!!!
- New function:
  - =Qrnd(value) as replacement for Rnd... does not need Randomize and is
    faster.
- Error in the token list caused a wrong syntax of Blitter Fill to be
  converted into Pt Play. Funny :)
- Found and removed an error in the Blitter Fill commands. Blitter Fill
  filled the screen one line to deep -> memory got corrupted.
- There was a bug in the vector rotation calculation with negative
  positions.

V0.986ß 25-May-94

- New command to stop Bernd from complaining:
  - Change Print Font bank

V0.986ß 24-May-94

- Completed the Vec Rot commands. Removed =Vec Rot Adr again.
- Vec Rot commands:
  - Vec Rot Pos posx,posy,posz
  - Vec Rot Angles angx,angy,angz
  - Vec Rot Precalc
  - =Vec Rot X(x,y,z)
  - =Vec Rot X
  - =Vec Rot Y(x,y,z)
  - =Vec Rot Y

V0.985ß 24-May-94

- Removed a bug in Cd Date$. (Thx Bernd)
- Discovered and corrected a cheap bug in Blitter Fill screen,plane.
- Removed the bug in Turbo Draw when using the blitter mode.
- New commands:
  - =Qsin(angle,factor) angle must be between 0 and 1023
  - =Qcos(angle,factor) factor is the value that is
                         multiplied with the sinus value.
  - Vec Rot Pos midx,midy,midz Positions for the vector rotation.
  - Vec Rot Angles angx,angy,angz Rotation angle.
- Still buggy:
  - Vec Rot Precalc Creation of the matrix
  - =Vec Rot X(x,y,z) Calculation of the new x-value
  - =Vec Rot Y(x,y,z) Calculation of the new y-value
- Added the function =Vec Rot Adr for testing reasons.

V0.984ß 21-May-94

- New syntaxes for Blitter Fill:
  - Blitter Fill screen,plane
  - Blitter Fill screen,plane,x1,y1,x2,y2
  - Blitter Fill s1,p1 To s2,p2
  - Blitter Fill s1,p1,x1,y1,x2,y2 To s2,p2
- Turbo Draw doesn't draw on non-existing bitplanes anymore. (Thx Marc)
- Change Font now adds '.font' automatically, if needed. (Thx Markus)

I'm afraid to say that there are no history entries for the other half of
the year.

Bank Commands

AMCAF contains many commands that are dedicated to AMOS memory banks.
With all these commands it's important to use EVEN addresses, if there are some demanded.
Otherwise you will crash any computer with MC68000 processor.
In addition, the lengths of every bank must be even, or the compiler will report a "Not an AMOS program" error, but this is a problem of AMOS and not of AMCAF.

AMOS Memory Banks

AMOS banks are a linear block of memory (there are two Exceptions). In these chunks various kinds of data is stored. AMOS uses them mostly for packed graphics, sound, music, AMAL programs, menus, resources and other data.

Generally, there are four main types of memory banks:

- Banks, that are stored in Chip ram and remain there permanently.
- Banks, that are stored in Fast ram and remain there permanently.
- Banks, that are stored in Chip ram and remain there temporarily only.
- Banks, that are stored in Fast ram and remain there temporarily only.

Permanent and Temporary Memory Banks

BANK PERMANENT
instruction: make a memory bank permanent
Bank Permanent bank number

If you defined a bank as 'Work', but afterwards want this bank to be Permanently in memory and in the program, you could use Bank Permanent.
This will make the bank stay resident in your program until it is erased.
This command also has use on MED-Modules, which were loaded with the Med Load command and on Power and Imploder unpacked banks.

BANK TEMPORARY
instruction: make a memory bank temporary
Bank Temporary bank number

Changes a bank to Temporary-type, i.e it will be erased on the start of your program or when calling the Default-Command.

BANK TO FAST
instruction: move a memory bank to fast ram
Bank To Fast bank number

Moves a bank into Fast ram, if any is available. Naturally, the bank will get a new starting address.
Bank To Fast won't work with Icon and sprite bank.
Warning: Do not try to replay music or sound that exist in fast ram.

BANK TO CHIP
instruction: move a memory bank to chip ram
Bank To Chip bank number

Bank To Chip is the reverse command to Bank To Fast. It moves a bank into Chip ram. Obviously, the starting address will therefore change.

BANK STRETCH
instruction: extend the size of a bank to a new length
Bank Stretch bank number **TO** length

Extents the bank numbered 'bank' to the given new length 'length'.
During this process, the starting address of the bank is changed. This command does not work on Icon and sprite banks.

BANK COPY
instruction: copy a memory bank
Bank Copy sourcebank **TO** targetbank
Bank Copy startaddress,endaddress **To** targetbank

Creates a identical copy of the bank with the number 'sourcebank' in the bank numbered 'targetbank'.
The second version copies a specific part of the source bank into the target bank, creating a temporary bank of same content.

BANK NAME
instruction: rename a memory bank
Bank Name bank number,name$

This command renames a bank to the 8 characters long name 'name$'. Most AMOS commands ignore this ID, but e.g the AMOS Tracker commands require a bank named 'Tracker'.

Encoding and Decoding Memory Banks

Using these commands you can encode banks in many different ways to protect them from unauthorized access and insight. Especially manual copy protections can be made more secure by encoding the specific keyword bank.

Each command comes in two versions, one with the suffix .b and one with .w. By using the .b version the codenumber can range from 1 to 255, the .w version allows codes from 1 to 65535.
However, the rotational commands are an exception as the codes may only reach from 1 to 7 and from 1 to 15 respectively.

Every command has the following syntax

Bank Code xxx.y code,bank
Bank Code xxx.y code,startaddress To endaddress

BANK CODE ADD
instruction: encode or decode a memory bank using addition
Bank Code Add.b code,bank number
Bank Code Add.b code,startaddress **To** endaddress
Bank Code Add.w code,bank number
Bank Code Add.w code,startaddress **To** endaddress

Encodes the bank using the key code 'code'. To decode the bank, the same instruction has to be used with the negative key code.

Note: This encoding routine works by adding the value and is therefore very easy to decode.

BANK CODE XOR
instruction: encode or decode a memory bank using xor
Bank Code Xor.b code,bank number
Bank Code Xor.b code,startaddress **To** endaddress
Bank Code Xor.w code,bank number
Bank Code Xor.w code,startaddress **To** endaddress

Bank Code Xor encrypts a bank in a similar way to Bank Code Add using another algorithm. Each byte or word of the bank is combined by a 'logical exclusive or'.
To decode the bank simply use the same command along with the same key code.

Note: A Xor encryption is not so easy to crack without the right code. Good codes are $AA and $55.

BANK CODE MIX
instruction: encode or decode a memory bank using mix
Bank Code Mix.b code,bank number
Bank Code Mix.b code,startaddress **To** endaddress
Bank Code Mix.w code,bank number
Bank Code Mix.w code,startaddress **To** endaddress

The third possibility to encode a bank, is with Bank Code Mix. So coded banks should be hard to decode.
To decode a bank you should use the same key code as seen with Bank Code Xor.

BANK CODE ROL
instruction: encode or decode a memory bank using rol
Bank Code Rol.b code,bank number
Bank Code Rol.b code,startaddress **To** endaddress
Bank Code Rol.w code,bank number
Bank Code Rol.w code,startaddress **To** endaddress

Using this command every bit in each byte or word is rotated by 'code' bits to the left. (Rol=Rotate Left). This results in a restriction of the 'code' parameter from 1 to 7 on '.b' and 1 to 15 on '.w' command version.
To decode a bank either use the negative code with the same instruction or the same key code along with the Bank Code Ror command.

BANK CODE ROR
instruction: encode or decode a memory bank using ror
Bank Code Ror.b code,bank number
Bank Code Ror.b code,startaddress **To** endaddress
Bank Code Ror.w code,bank number
Bank Code Ror.w code,startaddress **To** endaddress

Similar to Bank Code Rol, but this time the bits are shifted to the right instead of left.

Delta Encoding and Decoding

BANK DELTA ENCODE
instruction: encode a memory bank using delta algorithm
Bank Delta Encode bank number
Bank Delta Encode startaddress **To** endaddress

Bank Delta Encode encodes a memory bank with the so-called delta algo.
This is not packing, however, but yields better pack ratios on 8-bit sound samples.
Delta encoding just stores the difference from one byte to the next, so it is certain full curve patterns in samples can be seen more 'clearly' for packing algorithms.

```
On Protracker Modules, you could use:
Encode samples only:
Pt Bank 3
Bank Delta Encode Pt Instr Address(1) To Start(3)+Length(3)

Decode:
Pt Bank 3
Bank Delta Decode Pt Instr Address(1) To Start(3)+Length(3)
```

BANK DELTA DECODE
instruction: decode a memory bank using delta algorithm
Bank Delta Decode bank number
Bank Delta Decode startaddress **To** endaddress

This command decodes a previously delta encoded memory region.
See Bank Delta Encode for more details on delta encoding.

Functions

BANK CHECKSUM
function: calculate the checksum of a memory bank
number=**Bank Checksum**(bank number)
number=**Bank Checksum**(startaddress **To** endaddress)

This function calculates a checksum of a bank with specific contents. Using this checksum you can find out if the contents of a bank has been changed.
The second version of this command calculates the checksum from the memory area from startaddress to endaddress.

BANK NAME$
function: get the name of a memory bank
name$=**Bank Name$**(bank number)

The function Bank Name$ returns the name of a memory bank.

Basic Graphic Commands

AMCAF expands the basic command set of AMOS by some important commands.

BZOOM
instruction: magnify a region of the screen
Bzoom s1,x1,y1,x2,y2 **To** s2,x3,y3,factor

Using this command you can zoom a region on the screen to a integer multiple very fast. As result the graphics are double, four times or eight times as wide and from 1 to 15 times as high as before.
The command zooms the rectangular area from x1,y1 to x2,y2 on screen s1 onto the screen s2 at the coordinates x3,y3. The ending coordinates result from the zooming factor.

The coordinates x1 and x2 are rounded down to the next multiple of eight, x3 is even rounded to the nearest multiple of 16. Moreover, you must ensure that the area to be zoomed fits on the target screen, because no clipping is done at all.

The factor parameter defines how much the original image should be zoomed. It is divided into two nibbles: $yx. y may be a value from 1 to F, x must be either 1,2,4 or 8.

```
Example factors:
$11 copies the image without zooming.
$22 zooms the picture to double width and height.
$34 zooms the graphics to 4 times width and 3 times height.
```

HAM FADE OUT
instruction: fade out a screen
Ham Fade Out screen number

As HAM screens cannot be faded out with the normal Fade commands, AMCAF provides you with a HAM fader! Although you can only fade out pictures.
Ham Fade Out darkens the screen by one single step. After calling it 16 times, the Ham screen is completely black. Technically, it's not possible to fade in a ham screen without enormous processor power, but for fading out, a modified Shade Bobs routine is used.

CONVERT GREY
instruction: convert a screen to grayscale
Convert Grey sourcescreen **To** targetscreen

Using this command you can convert any screen into a grey scale image, even HAM and ExtraHalfBright screens are supported.
The only thing you have to do, is to open the screen and set the correct palette.
The parameter 'sourcescreen' defines the screen to be grey scaled, 'targetscreen' is the number of the screen where the grey image is to be created on.The number of colours in the target screen will be taken in account, but it makes no sense to open a HAM screen for that purpose.

TURBO PLOT
instruction: draw a dot on the screen
Turbo Plot x,y,c

Draws a dot at the coordinates x,y using the colour 'c'.
This command is about 3-14 factors faster than the AMOS plot command, but does not care about logical combinations which could be defined with Gr Writing.

TURBO DRAW
instruction: draw a line on the screen
Turbo Draw x1,y1 **To** x2,y2,c
Turbo Draw x1,y1 **To** x2,y2,c,bitplanes

This instruction replaces the AMOS Draw command. The x1,y1 coordinates represent the starting point of the line and x2,y2 the ending point.
The coordinates needn't to be on the screen, the line is clipped automatically.

The colour of the line 'c' must be given, whereas the 'bitplanes' parameter is optional. It determines, into which Bitplanes the line should be drawn.
So Bit 0 represents bitplane 0, bit 1 represents bitplane 1 etc. If the corresponding bit is set, a line will be drawn in the bitplane, otherwise not.
If 'bitplanes' is omitted, every bitplane is drawn into. Turbo Draw supports a line pattern, which can be changed using the AMOS Set Line command.

With Turbo Draw, you can draw even more and special lines, so-called Blitter Lines. These lines are only drawn with one dot in each horizontal line and are used to create polygons which can then be filled with the blitter chip.
To switch to this mode, the 'bitplane' parameter must be given as negative number, everything else can remain as it is.
Please note that an extra line is drawn automatically if the blitter line leaves the right boundary of the screen.

FCIRCLE
instruction: draw a filled circle on the screen
Fcircle x,y,r

Fcircle draws a filled circle with the radius 'r' and center x,y.

FELLIPSE
instruction: draw a filled ellipse on the screen
Fellipse x,y,rx,ry

This command draws an filled ellipse, similar to the AMOS Ellipse command.

RASTER WAIT
instruction: wait for the raster beam position
Raster Wait y
Raster Wait x,y

This command can be used to wait for a specific position of the raster beam.
The x and y parameters contain the hardware coordinates from which the program should continue.

Basic Graphic Functions

TURBO POINT
function: return the colour of a pixel
c=**Turbo Point**(x,y)

Like the Point instruction, this fast function requests the colour of the pixel at the coordinates x,y.

X RASTER
function: return the current X position of the raster beam
x=**X Raster**

This function returns the current X position of the raster beam in hardware coordinates.

Y RASTER
function: return the current Y position of the raster beam
y=**Y Raster**

Y Raster returns the current Y position of the raster beam.
This function can be used to see how many raster lines a specific part of a program takes.

Effects Commands

A big part of AMCAF contains graphical effects.
If you only want to have a few stars or simply want to make a logo explode or create some neat effects using shade bobs, AMCAF makes (nearly) everything possible!

Pix Shift Commands

Using these commands you can increase or decrease the colour indexes in a rectangular area.
These instructions work pixel wise and therefore makes it possible to limit the colours, and so the Pix Shift commands are slower than Shade Bobs.

MASK COPY
instruction: copy an area of a screen to another screen
Mask Copy screen1,x1,y1,x2,y2 **To** screen2,x3,y3,maskaddress

Copies a part of a screen to an other, just like Screen Copy.
However, a mask bitplane can be given. 'maskaddress' represents the startaddress of the Bitplane.

PIX SHIFT UP
instruction: increase the colour index of an area of a screen
Pix Shift Up screen,c1,c2,x1,y1 **To** x2,y2
Pix Shift Up screen,c1,c2,x1,y1 **To** x2,y2,bank

Using this command, you can increase the colour indexes in the rectangular area from x1,y1 to x2y2.
The 'screen' parameter therefore contains the number of the screen on which this regions is situated. 'c1' and 'c2' hold the border colours, which should be taken into account for the colour cycling, other colours are not affected.

If you supply the optional parameter 'bank', you can use a previously calculated mask which can be created with the Make Pix Mask command.
However, if the colour indexes override 'c2', they are set back to 'c1' again.

PIX SHIFT DOWN
instruction: decrease the colour index of an area of a screen
Pix Shift Down screen,c1,c2,x1,y1 **To** x2,y2
Pix Shift Down screen,c1,c2,x1,y1 **To** x2,y2,bank

Using this command, you can decrease the colour indexes in the rectangular area from x1,y1 to x2y2.
The 'screen' parameter therefore contains the number of the screen on which this regions is situated. 'c1' and 'c2' hold the border colours, which should be taken into account for the colour cycling, other colours are not affected.

If you supply the optional parameter 'bank', you can use a previously calculated mask which can be created with the Make Pix Mask command.
However, if the colour indexes go below 'c1', they are set back to 'c2' again.

PIX BRIGHTEN
instruction: increase the colour index of an area of a screen
Pix Brighten screen,c1,c2,x1,y1 **To** x2,y2
Pix Brighten screen,c1,c2,x1,y1 **To** x2,y2,bank

Using this command, you can increase the colour indexes in the rectangular area from x1,y1 to x2y2.
The 'screen' parameter therefore contains the number of the screen on which this regions is situated. 'c1' and 'c2' hold the border colours, which should be taken into account for the colour cycling, other colours are not affected.

If you supply the optional parameter 'bank', you can use a previously calculated mask which can be created with the Make Pix Mask command.
In difference to Pix Shift Up, the colour indexes don't override 'c2'.

PIX DARKEN
instruction: decrease the colour index of an area of a screen
Pix Darken screen,c1,c2,x1,y1 **To** x2,y2
Pix Darken screen,c1,c2,x1,y1 **To** x2,y2,bank

This command decreases the colour indexes in the rectangular area from x1,y1 to x2y2.
The 'screen' parameter therefore contains the number of the screen on which this regions is situated. 'c1' and 'c2' hold the border colours, which should be taken into account for the colour cycling, other colours are not affected.

If you supply the optional parameter 'bank', you can use a previously calculated mask which can be created with the Make Pix Mask command.
In difference to Pix Shift Down, the colour indexes don't go below 'c1'.

MAKE PIX MASK
instruction: grab an area of a screen
Make Pix Mask screen,x1,y1 **To** x2,y2,bank

Grabs a specific part of the screen and saves it into the bank with the number bank. This command can be used to create a mask for the Pix Shift instruction.
If you use such a mask, the size must be exactly the same like the limits you specify with the Pix Shift commands.

SHADE PIX
instruction: increase the colour value at the given point of a screen
Shade Pix x,y
Shade Pix x,y,number

This instructions increases the colour value at the given point x,y on the current screen. If the highest colour is reached, the colour is reset to zero.
The optional parameter 'number' holds the number of Bitplanes to be cycled and may range from 1 to 6.

Shade Bobs Commands

Shade Bobs aren't really bobs, but "in the scene" this is the common name for such effects. Shade Bobs increase or decrease colour values of the pixel they are placed on. This is rather similar to colour cycling, however using Shade Bobs the pixels on the screen are affected and not the palette entries!
When using Shade Bobs you cannot limit the colours to a certain range but only the amount of bitplanes that will be used to be cycled through. Additionally, Shade Bobs may leave the screen boundaries.

SHADE BOB MASK
instruction: set image to use on shade bob
Shade Bob Mask flag

Using Shade Bob Mask you may specify, if the mask of a object or the first bitplane is to be taken for drawing.
If flag is set to zero, bitplane 0 is used for the Shade Bob Up and Shade Bob Down instructions, other values for flag instruct the commands to use the mask of the object.

SHADE BOB PLANES
instruction: set number of bitplanes to use on shade bob
Shade Bob Planes number

Set the number of Bitplanes to be cycled through. This can be used to protect the graphics in higher bitplanes from the influences of Shade Bobs.
'number' sets the number of bitplanes, that should be drawn in and must be a value between 1 and 6.

SHADE BOB UP
instruction: increase the colour index and draw a shade bob
Shade Bob Up screen number,x,y,image

This instruction draws a Shade Bob on the screen at the coordinates x,y. This bob only increases the colour indexes only.
'image' holds the image number of the sprite bank which should be used to draw the bob.
Either the mask or the first bitplane of the object is used for this process, according to the setting of Shade Bob Mask.
Remember, that this command supports the hot spot of the bob image.

SHADE BOB DOWN
instruction: decrease the colour index and draw a shade bob
Shade Bob Down screen number,x,y,image

This command is similar to Shade Bob Up and decreases the colour index only.

Td Stars Commands

Using these commands you not only can realize 3D star effects, because the commands are rather versatile.

TD STARS BANK
instruction: reserve a bank for 3d stars
Td Stars Bank bank,stars

This instruction reserves a bank for 3D stars. The 'bank' parameter holds the number of the memory bank which should be used for the stars. 'stars' contains the number of stars to be saved in this bank.
Each star consumes 12 bytes of memory.

TD STARS PLANES
instruction: set the bitplanes to use for 3d stars
Td Stars Planes bitplane 1,bitplane 2

Td Stars Planes is used to specify the Bitplanes which the stars should be drawn on.
Normally, these are the bitplanes 0 and 1, but these defaults can be modified with this instruction.

TD STARS LIMIT
instruction: limit the area of a screen to use for 3d stars
Td Stars Limit
Td Stars Limit x1,y1 **To** x2,y2

Limits the stars into a specific area on the screen. If no parameters are given, the full screen sizes are used.
x1,y1 and x2,y2 create a rectangular region, in which the stars will be drawn in.
These coordinates must lie WITHIN the screen dimensions, otherwise the stars could corrupt your memory.

TD STARS ORIGIN
instruction: set the starting point for 3d stars
Td Stars Origin x,y

Sets the origin, where stars start from, as soon as they have left the screen. The coordinates x,y must lie on the screen and within the drawing area, you have defined using Td Stars Limit earlier.

The Td Stars Limit instruction automatically places the origin in the middle of the specified area. Therefore, this command has to be placed after Td Stars Limit.

TD STARS GRAVITY
instruction: set the direction for 3d stars
Td Stars Gravity sx,sy

Set the direction which the stars shall drift into. 'sx' contains a value which is added each step to the horizontal speed and may be a positive number for right drift or a negative one for left movement. Same with 'sy', but this one has effect on the vertical speed.

Normally, these values are set to 0, but you could use other values in conjunction with Td Stars Accelerate to create an effect of spreading sparks.

TD STARS ACCELERATE
instruction: toggle acceleration for 3d stars on or off
Td Stars Accelerate On
Td Stars Accelerate Off

Determines, if the stars are to be accelerated during the flight.

TD STARS INIT
instruction: initialise the 3d stars
Td Stars Init

Initialises the stars. That means, that the stars are moved by random values to avoid that they all start in the origin.
This command should therefore be called once after all parameters have been set.

TD STARS SINGLE DO
TD STARS DOUBLE DO
instruction: process and draw the 3d stars
Td Stars Single Do
Td Stars Double Do

Clears, draws and moves all the stars in one single step. For Double Buffered screens you have to call the command Td Stars Double Do, for normal ones Td Stars Single Do.
To manually control the draw process you have to call the commands Td Stars Single Del or Td Stars Double Del, then Td Stars Move and Td Stars Draw in this order.

TD STARS SINGLE DEL
TD STARS DOUBLE DEL
instruction: delete the 3d stars from the screen
Td Stars Single Del
Td Stars Double Del

Wipes the stars from the screen. You have to distinguish between Single and Double Buffered screens. Use Td Stars Double Del on Double Buffered
Note: The background is not saved during the drawing process.

TD STARS MOVE
instruction: move the 3d stars on the screen
Td Stars Move
Td Stars Move star number

The Td Stars Move instruction moves all the stars by on step. To see the result, you have to draw the stars with a call to Td Stars Draw. Optionally you can move a single star.

TD STARS DRAW
instruction: draw the 3d stars to the screen
Td Stars Draw

This command draws all the stars onto screen. To clear the stars again you have to use either Td Stars Single Del or Td Stars Double Del according to the screen type.

Splinters Commands

Splinters are similar to Td Stars, but they don't destroy the background and use the colour of the pixel they have removed and Splinters require a list of coordinates.
Each coordinate requires 4 bytes, i.e already a field of 16x16 coord consumes 16 KB of memory.

COORDS BANK
instruction: reserve a bank to store coords
Coords Bank bank number,coords
Coords Bank bank number

Reserves the bank to store a coordinates array. These banks are utilized e.g for Splinters. 'coords' holds the amount of coordinates the bank should be able to hold.
Each coordinate requires 4 bytes. If this parameter is omitted the existing bank will only be switched to without erasing it. So you can jump between predefined banks.

COORDS READ
instruction: read coords into a bank
Coords Read screen,colour,x1,y1 **To** x2,y2,bank,mode

Read the coordinates for Splinters into a bank. The memory bank with the number 'bank' must have been defined previously with a call to the Coords Bank command.
The 'screen' parameter holds the number of the screen on which the pixels are. 'colour' represents the background colour, that will be left out when reading in the dots.
The rectangle which is described by the coordinates x1,y1 and x2,y2 will be then read in and all dots, which don't have the colour 'colour', will be stored in the bank.
'mode' can be either 0, if you want the coords to be read in strict order, or 1, if you want the coords to be shuffled.

SPLINTERS BANK
instruction: reserve a set number of splinters into a bank
Splinters Bank bank number,splinum

Reserves a memory bank for a maximum of 'splinum' Splinters. Each Splinter requires 22 bytes of memory.

SPLINTERS COLOUR
instruction: set splinter colour to use
Splinters Colour bkcolour,planes

Determines, which colour is to be left, after a Splinter has released a dot from that point. The 'planes' parameter describes the number of bitplanes to be taken for the Splinters operation.
Normally, this value should be equal to the number of available bitplanes. However, it can be useful for some effects to reduce this value.

SPLINTERS LIMIT
instruction: limit splinter area on the screen
Splinters Limit
Splinters Limit x1,y1 **To** x2,y2

Sets the limits for the Splinters to a rectangular area on the screen. If you don't give any parameters, AMCAF uses the limits of the current screen.

SPLINTERS MAX
instruction: change max number of splinters
Splinters Max number

Changes the max. number of new Splinters to appear on each step. Therefore a pulsing effect can be avoided.
If the number is set to 0, no more Splinters are created. When set to -1, there won't be a limit.

SPLINTERS FUEL
instruction: set the time for splinters on screen
Splinters Fuel time

This command is used to set the amount of time the Splinters move over the screen until they disappear automatically. The 'time' parameter holds the number of steps the splinters are moved before they vanish. If you set 'time' to 0, the Splinters only disappear at the edges of the screen.

SPLINTERS GRAVITY
instruction: set the direction for splinters on screen
Splinters Gravity sx,sy

Set the direction which the Splinters shall drift into. 'sx' contains a value which is added each step to the horizontal speed and may be a positive number for right drift or a negative one for left movement.
Same with 'sy', but this one has effect on the vertical speed.

SPLINTERS INIT
instruction: initialise the splinters
Splinters Init

Initialises the Splinters. They are fed with the coordinates and speeds you have specified earlier.

SPLINTERS SINGLE DO
SPLINTERS DOUBLE DO
instruction: draw the splinters to the screen
Splinters Single Do
Splinters Double Do

Clears, draws and moves all Splinters in one single step. For Double Buffered screens you have to call the command Splinters Double Do, for normal ones Splinters Single Do.
To manually control the draw process you have to call the commands Splinters Single Del or Splinters Double Del, then Splinters Move, Splinters Back and Splinters Draw in this order.

SPLINTERS SINGLE DEL
SPLINTERS DOUBLE DEL
instruction: clear the splinters from the screen
Splinters Single Del
Splinters Double Del

Clears the splinters from the screen again. As the clearing process must either wipe the pre-last pixels from the screen (when using Double Buffering), or the last pixels (with Single Buffered screens), you have to take the appropriate command for the right screen type.
The background is automatically restored.

SPLINTERS MOVE
instruction: move the splinters on the screen
Splinters Move

This command moves the Splinters one step.

SPLINTERS DRAW
instruction: draw the splinters onto the screen
Splinters Draw

Draws the Splinters onto the screen.

SPLINTERS BACK
instruction: save the background for next splinter draw operation
Splinters Back

Saves the background, on which the Splinters are to be drawn in the next step.

Splinters Functions

COUNT PIXELS
function: count the number of pixels in an area
number=**Count Pixels**(screen number,colour,x1,y1 **To** x2,y2)

This function returns the number of pixels in the rectangular area from x1,y1 to x2,y2 in the screen, that don't have the colour index colour.
This function can be used to first acquire the number of points in this area and then to reserve a coordinates bank.

SPLINTERS ACTIVE
function: return the number of active splinters
number=**Splinters Active**

This function returns the number of active Splinters at this time. This can be used to know when the animation has finished.

Font Commands

AMCAF provides you with powerful instructions for font managing.
For example the Change Font command allows you to set a font without having to get the full font list using Get Disk Fonts etc. and therefore reduces the time to access disk considerably.
Hard disk drive users have the unbelievable opportunity to store any disk font in an AMOS memory bank and so are totally independent of the fonts directory and the diskfont.library.

CHANGE FONT
instruction: load a font from disk
Change Font "fontname.font"
Change Font "fontname.font",height
Change Font "fontname.font",height,style

Opens a font directly from disk. This font will be taken for future text print outs. 'height' represents the vertical size of the font, a default height of 8 will be filled in if this parameter is omitted. The 'style' parameter is only required in special cases and sets the type of the requested charset (see Set Text).

Since Amiga OS 2.0 it's even possible to load multicoloured fonts. The poor guys on Kickstart 1.3 can use the program ColorText (which e.g is included on the DPaint disk) to patch the graphics.library to handle these fonts as well.
The diskfont.library (length=51200 bytes) found on the Fonts disk can scale your font to your desired size.

If you open many fonts, you should call the command Flush Libs from time to time to remove unused fonts from memory.

With the Make Bank Font command you can convert this font into a memory bank, so you don't require the diskfont.library and the font file anymore.
Additionally, you can change the font which is used for the Print instruction using Change Print Font.

MAKE BANK FONT
instruction: create a font bank
Make Bank Font bank number

Using this powerful command you can store any Amiga font in a memory bank. The current font on the current screen will be taken to create the font bank numbered. To change the current text font simply use the Change Font instruction.

These banks don't require the 'diskfont.library' or other disk access any more, once they have been created. To load a font from a bank use the Change Bank Font command.

CHANGE BANK FONT
instruction: set the screen font from a bank font
Change Bank Font bank number

This command sets the text font on the current screen to the one saved in the memory bank numbered.

If you change the font using this command, no diskfont.library is required any more.
This command is also quite important if you write your programs to be installed on hard disk that use your own fonts. Bank Fonts can be created easily with the Make Bank Font command.

CHANGE PRINT FONT
instruction: change the font that is used by print
Change Print Font bank

Change Print Font changes the charset which is used for the Print command. This font always 8x8 pixels big and contains 256 characters, which is stored in a memory bank of exactly 2 KB (=2048 bytes).
Therefore the 'bank' parameter has to point exactly to such a bank.

These Print-Font banks can be created by hand (reserve the bank, write 8 bytes per chars) or with the help of the Font-Editor supplied on the Accessories disk (AMOSPro_Accessories:Font8x8_Editor.AMOS) and then can be loaded using Dload.

FONT STYLE
function: return the current text styles
style=**Font Style**

This functions replaces the AMOS function Text Styles, because this one does not return the multicoloured font bit (Bit 6).
Apart from this, Font Style is totally identical with the AMOS function.

Colour Commands

Playing with colour was often linked with complex formulas. But now there is a series of helping commands - only AMCAF makes it possible!

RAIN FADE
instruction: fade out a rainbow to another rainbow
Rain Fade rainbow number,$RGB
Rain Fade rainbow number **To** target rainbow

Using this powerful instruction you can fade out the numbered rainbow to other colours. With the first version of Rain Fade you may specify the target colour '$RGB' to which all colours of the rainbow will be faded.
The other version fades exactly to the colours of the target rainbow.

Rain Fade works step by step only. Therefore you need a maximum of 16 calls to reach the new colour values.

SET RAIN COLOUR
instruction: change the colour index of a rainbow
Set Rain Colour rainbow number,new colour

With the help of this command you can change the colour index of a rainbow which has been set previously by Set Rainbow to a new value.
This means that you can remove the irritating limit to the first 16 colours and are now able to access all 32 colours. But there are even more possibilities!

A colour index of -63 enables you to alter the hardware scrolling register, so you can create fancy water and wobble effects.
For more effects you need to install the SetRainPatch.

PAL SPREAD
instruction: spread the colour index from one to another
Pal Spread colour 1,rgb1 **To** colour 2,rgb2

Creates a smooth blend between the two colours rgb1 and rgb2. The resulting colour set will be stored between colour 1 and colour 2.

PAL GET SCREEN
function: save the palette of a screen
Pal Get Screen palnr,screen number

This function stores the complete palette of a screen in the internal palette memory number 'palnr'.
'palnr' must be a value between 0 and 7.
This command is used to quickly store a specific palette of a screen in a buffer. To restore the old palette, you can call the Pal Set Screen command.

PAL SET SCREEN
instruction: set the palette of a screen
Pal Set Screen palnr,screen number

This command restores the previously stored colour palette 'palnr' to the numbered screen.

PAL SET
instruction: change an entry of a saved palette
Pal Set palette number,index,colour

This command changes the colour entry numbered 'index' in the selected palette to the colour value 'colour'.
palette number must be a value between 0 to 7.

AMCAF AGA NOTATION ON/OFF
instruction: toggle aga amiga colour format
Amcaf Aga Notation On
Amcaf Aga Notation Off

Normally, AMCAF commands, which take colour values as parameter, use a normal 12 bit colour value in the format $RGB.
Since machines like the A4000/A1200 there are 24 bit colour values in the Format $RRGGBB.

After calling Amcaf Aga Notation On, all AMCAF commands and functions take 24 bit values, except Rrggbb To Rgb and Rgb To Rrggbb.
The default setting is 12 bit.

Functions

PAL GET
function: read a saved palette entry
colour=**Pal Get**(palette number,index)

This function reads the value of the colour with the number index 'index' from the internal stored palette buffer.
palette number must be a value between 0 to 7.

RED VAL
function: return the red value of a colour
r=**Red Val**(rgb)

This function acquires the red value of a colour. Together with the other Val functions, you can separate the colour into its three contents.

GREEN VAL
function: return the green value of a colour
g=**Green Val**(rgb)

This function acquires the green value of a colour. Together with the other Val functions, you can separate the colour into its three contents.

BLUE VAL
function: return the blue value of a colour
b=**Blue Val**(rgb)

This function acquires the blue value of a colour. Together with the other Val functions, you can separate the colour into its three contents.

GLUE COLOUR
function: return a colour using the three colour values
colour=**Glue Colour**(r,g,b)

Combines red, green and blue value of a colour to a normal colour value.

MIX COLOUR
function: return a colour my mixing colours
colour=**Mix Colour**(colour 1,colour 2)
colour=**Mix Colour**(oldrgb,addrgb,lower rgb **To** upper rgb)

The first version of the function mixes the two colours colour 1 and colour 2 and returns the resulting colour value.

The second version behaves a little bit differently: The colour 'addrgb' is added to the colour value 'oldrgb', if 'addrgb' is a positive value or subtracted, if the value is negative.
'lower rgb' defines the lower and 'upper rgb' the upper boundary of the allowed resulting colours.

BEST PEN
function: return the nearest pen for $RGB
pen=**Best Pen**($RGB)
pen=**Best Pen**($RGB,colour 1 **To** colour 2)

This function acquires the pen which is nearest to the colour $RGB. Optionally, you can give the range to search for the best colour. The Best Pen function can be used to recolour pictures with limited palette.

HAM POINT
function: return the rgb value of a ham pixel
rgb=**Ham Point**(x,y)

This function simplifies the process of finding the right colour of a point at the coordinates x,y on a HAM screen.
There's no need to proceed from the left to the right until the right pixel value is found, although the Ham Colour method is a little bit faster when processing a picture from left to the right.
Ham Point can access any point on the screen individually without preprocessing.
If the point x,y is not on the screen, rgb will contain -1 instead of the red-green-blue value of a pixel.

HAM COLOUR
function: return a colour in ham mode
rgb=**Ham Colour**(c,oldrgb)

This function calculates the new colour value, that is created, when plotting a pixel in colour 'c' directly behind the last point.

HAM BEST
function: return the best colour in ham mode
colour=**Ham Best**(newrgb,oldrgb)

As you cannot achieve the desired colour by plotting only one pixel in HAM mode, you can use the Ham Best function to search the best colour for the next dot.
'newrgb' holds the target colour and 'oldrgb' the colour of the pixel exactly before the current dot.
Note: the current screen must be a HAM screen.

RGB TO RRGGBB
function: return an aga colour from an ecs colour
rrggbb=**Rgb To Rrggbb**(rgb)

This function calculates a 12 bit colour value into a 24 bit AGA value. The missing bits are set to zeros.
Using this colour format you can assign 256 different colour values each to the red, green or blue part, that is 16777216 different values in total.

RRGGBB TO RGB
function: return an ecs colour from an aga colour
rgb=**Rrggbb To Rgb**(rrggbb)

This function converts a 24 bit AGA colour value into a 12 bit ECS colour value.
The other 12 bits will be discarded.

Blitter Commands

AMCAF makes it possible to control the Blitter by hand and to clear, fill, copy or modify Bitplanes directly.

BLITTER COPY
instruction: copy and modify a bitplane
Blitter Copy sourcescreen,sourceplane **To** targetscreen,targetplane
Blitter Copy sourcescreen,sourceplane **To** targetscreen,targetplane,miniterm
Blitter Copy s1,p1,s2,p2 **To** targetscreen,targetplane
Blitter Copy s1,p1,s2,p2 **To** targetscreen,targetplane,miniterm
Blitter Copy s1,p1,s2,p2,s3,p3 **To** targetscreen,targetplane
Blitter Copy s1,p1,s2,p2,s3,p3 **To** targetscreen,targetplane,miniterm

With the help of Blitter Copy you can copy a Bitplane of a screen to another or the same bitplane on another or the same screen.
The Amiga Blitter chip is used to fulfill this action.
If two or three source screens and bitplanes parameters are given, the graphics can be combined by so called miniterms and the result is then written into the target screen.

Important

1. Before you can call Blitter Copy, you **must** set the limits of the operation using Blitter Copy Limit.

2. Take care that **all** specified screens have these dimensions and **all** screens are big enough.

The optional parameter miniterm contains a bitmask which defines the way the source planes should be combined.

If these values are omitted, these default values are used:

```
One source   : the bitplane is copies normally             (=%11110000).
Two sources  : the bitplanes are combined using logical OR (=%11111100).
Three sources: the bitplanes are combined using logical OR (=%11111110).
```

Frequently used miniterm values:

```
Sourceplanes one, two and three represent A, B and C.

1: %11110000: copy bitplane                : A
1: %00001111: invert bitplane              : NOT A
2: %11111100: use logical OR on the planes  : A OR B
2: %00000011: OR planes and invert them     : NOT (A OR B)
2: %11000000: the two planes are ANDed      : A AND B
2: %00111111: planes are ANDed and inverted : NOT (A AND B)
2: %00111100: the planes will be XORed      : A XOR B
2: %11000011: planes are XORed and inverted : NOT (A XOR B)
3: %11111110: OR all planes (create mask)   : A OR B OR C
3: %00000001: OR all planes and invert them : NOT (A OR B OR C)
3: %10000000: Do a logical AND on all planes: A AND B AND C
3: %01111111: AND all planes and invert them: NOT (A AND B AND C)
```

If you want to create other combinations, simply try out the values you like (nothing can happen except of bad looking screens).
If you have some knowledge on boolean algebra, read Blitter miniterms.

BLITTER COPY LIMIT
instruction: set the blitter copy area
Blitter Copy Limit screen
Blitter Copy Limit x1,y1 **To** x2,y2

Using this command you define the rectangular area which will be used for the Blitter Copy command.
If you only specify the 'screen' parameter, the full screen dimensions of the screen numbered 'screen' will be taken.

Otherwise x1,y1 represents the upper left corner and x2,y2 the lower right corner of the region which will be affected by the copying and combining process.
This area will be used for **all** screens, therefore you must ensure that every screen is at least as big as the lower right corner of the specified limits.

BLITTER FILL
instruction: fill polygons using the blitter
Blitter Fill screen,bitplane
Blitter Fill screen,bitplane,x1,y1,x2,y2
Blitter Fill sourcescreen,sourceplane **To** targetscreen,targetplane
Blitter Fill sourcescreen,sourceplane,x1,y1,x2,y2 **To** targetscreen,targetplane

With Blitter Fill you can fill polygons. However, there are some limitations: the filling algorithm of the Blitter chip is very simple.
It only fills the gap between two dots of a horizontal line. Therefore the limiting lines may only be one pixel thick.
These lines can be either created using Turbo Draw or Bcircle.

The 'screen' and 'bitplane' parameter specify the screen which contains the area to be filled.
Additionally you can give a rectangular region which is defined by the coordinate pairs x1,y1 and x2,y2.

You can also give a second screen 'targetscreen' and a second bitplane number 'targetplane' into which the filled figures are written.
In this mode, the source screen is not altered in any way.

If more than one filled figure is to be drawn and these don't overlap, you can draw all limiting lines first and then fill all figures in one cycle.

The Blitter processes the screen from the lower right to the upper left.
Due to this feature you should draw on **Double Buffered** or **hidden screens** to avoid flickering.

BLITTER CLEAR
instruction: clear a bitplane using the blitter
Blitter Clear screen,bitplane
Blitter Clear screen,bitplane,x1,y1 **To** x2,y2

The Blitter chip can be used to clear Bitplanes as well.
In comparison to the AMOS command Cls, Blitter Clear allows you to wipe single bitplane instead of all.

Optionally you may include the coordinates of a rectangular region where the command should have an effect on.

BLITTER WAIT
instruction: wait for blitter to finish a task
Blitter Wait

This command waits until the Blitter has finished his work.
You are advised to use this command **before** calling e.g Print or any other command that does not require the blitter, that will draw onto a screen, which is currently altered by the blitter.
You should use Blitter Wait in **front** of Wait Vbl, too.

TURBO DRAW
instruction: draw from one coordinate to another
Turbo Draw x1,y1 **To** x2,y2,colour
Turbo Draw x1,y1 **To** x2,y2,colour,bitplane

This instruction replaces the AMOS Draw command. The x1,y1 coordinates represent the starting point of the line and x2,y2 the ending point.
The coordinates needn't to be on the screen, the line is clipped automatically.

The colour of the line must be given, whereas the 'bitplane' parameter is optional.
It determines into which Bitplanes the line should be drawn. So Bit 0 represents bitplane 0, bit 1 represents bitplane 1 etc. If the corresponding bit is set, a line will be drawn in the bitplane, otherwise not. If 'bitplanes' is omitted, every bitplane is drawn into.
Turbo Draw supports a line pattern, which can be changed using the AMOS Set Line command.

With Turbo Draw, you can draw even more and special lines, so-called Blitter Lines.
These lines are only drawn with one dot in each horizontal line and are used to create polygons which can then be filled with the blitter chip.
To switch to this mode, the 'bitplane' parameter must be given as negative number, everything else can remain as it is.
Please note that an extra line is drawn automatically if the blitter line leaves the right boundary of the screen.

BCIRCLE
instruction: draw a circle for the blitter to fill
Bcircle x,y,radius,bitplane

Draws an empty circle around x,y with a radius into the bitplane.
This line is really only one pixel thick to ensure the circle can be filled by the Blitter chip.

BLITTER BUSY
function: return the current blitter state
flag=**Blitter Busy**

This function returns -1 (True), if the Blitter chip is currently busy and working on a job, e.g is currently clearing a bitplane or drawing a line etc.
This can be used to wait for the end of the blitter activity, although Blitter Wait is preferred in this case.
You could use Blitter Busy to decide, if you want to do some more calculations using the processor or better start the next blitter activity.

Miscellaneous Graphic Commands

SET NTSC/PAL
instruction: switch to the 60hz ntsc mode or 50 mhz pal mode
Set Ntsc or **Set Pal**

Set Ntsc command switches to 60Hz NTSC screen mode. Set Pal returns to the 50Hz Pal screen mode

SET SPRITE PRIORITY
instruction: change the sprite priority in dual playfield mode
Set Sprite Priority bitmap

Set priority of a sprite in a playfield to 'bitmap'. 'bitmap' is a bit mask in the following format:

```
Bits 0-2: 0 = all sprites      will be displayed behind         playfield 1
          1 = the sprites 0-1  appear in front of               playfield 1
          2 = the sprites 0-3  are drawn in front of the first  playfield
          3 = the sprites 0-5  appear in front of               playfield 1
          4 = all sprites      will be displayed in front of    playfield 1
Bits 3-5: 0 = all sprites      will be displayed behind         playfield 2
          1 = the sprites 0-1  appear in front of               playfield 2
          2 = the sprites 0-3  are drawn in front of the second playfield
          3 = the sprites 0-5  appear in front of               playfield 2
          4 = all sprites      will be displayed in front of    playfield 2
```

SCRN RASTPORT
function: return the screen rastport address
rastport=**Scrn Rastport**

Fetches the address of the graphics/rastport structure of the current AMOS screen.

SCRN BITMAP
function: return the screen bitmap address
bitmap=**Scrn Bitmap**

Fetches the address of the graphics/bitmap structure of the current AMOS screen.

SCRN LAYER
function: return the screen layer address
layer=**Scrn Layer**

Fetches the address of the graphics/layer structure of the current AMOS screen.

SCRN LAYERINFO
function: return the screen layer info address
layerinfo=**Scrn Layerinfo**

Fetches the address of the graphics/LayerInfo structure of the current AMOS screen.

SCRN REGION
function: return the screen region address
region=**Scrn Region**

Fetches the address of the graphics/region structure of the current AMOS screen.

Primary Disk Commands

AMCAF gives you some very easy to use commands to load, save, and modify files.

FILE COPY
instruction: copies a file
File Copy sourcefile$ **To** targetfile$

This command copies the file with the name 'sourcefile$' to the file 'targetfile$'.
This command allows you to even copy a file of 3 MB in size, even if you only got 100 KB of free memory.

WLOAD
instruction: load a file temporarily
Wload file$,bank

This command loads the file named 'file$' completely into memory, storing it in bank number 'bank'.
The bank is defined as 'Work'.
If 'bank' is a negative number, the file is loaded into Chip ram instead.

Wload could be replaced by the following commands:

```
Open In 1,FILE$ : LE=Lof(1) : Close 1
Reserve As Work BANK,LE
Bload FILE$,BANK
```

DLOAD
instruction: load a file permanently
Dload file$,bank

Just like Wload, this command loads a file into memory, but this time a Permanent 'Data'-Bank will be created.

WSAVE
instruction: save a file to disk
Wsave file$,bank

This command saves the bank with the number 'bank' as file named 'file$' onto the current disk drive.
This file contains no AMOS overhead, that means that only the pure binary data is saved.

DSAVE
instruction: save a file to disk
Dsave file$,bank

Dsave is exactly the same as Wsave in every aspect.

PROTECT OBJECT
instruction: modify the protection bits of an object
Protect Object pathfile$,prot

This command changes the Protection Flags of the Object 'pathfile$' to the bitmapped value 'prot'.

SET OBJECT COMMENT
instruction: set the filenote of an object
Set Object Comment pathfile$,comment$

This command sets the comment of the Object 'pathfile$' to the string 'comment$'.

SET OBJECT DATE
instruction: set the date of an object
Set Object Date pathfile$,date,time

This command changes the date of the Object 'pathfile$' to the given date and time stamp.
This command only works on OS2.0 and higher.

LAUNCH
instruction: start a new process
Launch file$
Launch file$,stack

The Launch instruction starts a new task which is saved on disk with the name 'file$'.
The optional parameter 'stack' holds the size of the stack memory to be allocated (default value is 4096).

Note: Many programs don't work when started from AMOS, as they are not part of a workbench or CLI environment.

Disk Object Commands

The AMOS functions Dir First$ and Dir Next$ are neither very reliable nor very easy to use.
Using these commands you could only get the name and the length of the object but what could you do if you wanted to know even more?
Thankfully, AMCAF provides you with an easy solution for these problems.

Commands

EXAMINE OBJECT
instruction: get all information about an object
Examine Object file$

Examine Object supplies you with all available information about the object named 'file$'.
This data can then be requested using the Object functions without any parameters.

EXAMINE DIR
instruction: initialise the reading of a drawer
Examine Dir directory$

This command loads all information about the drawer 'directory$' into the FileInfoBlock.
Additionally, the contents of the directory can be read out by Examine Next$.

EXAMINE STOP
instruction: stop the reading of a directory
Examine Stop

Aborts the reading process of a directory. After this command, you may not make any further calls to Examine Next$.

Functions

EXAMINE NEXT$
function: read the next entry in a directory
file$=**Examine Next$**

Gets information about the next object in the directory and returns its name.
More information can be acquired using the object functions. If the end of the directory list is reached, 'file$' will contain an empty string and the drawer will be closed.

OBJECT NAME$
function: return the name of an object
file$=**Object Name$**
file$=**Object Name$**(pathfile$)

Object Name$ returns the name of the object 'pathfile$'. This function is similar to the Filename$-Function, but acts quite differently as the existence of the file is checked.
More information about the object is stored in the info block and the name is read out with correct case.
The first version of Object Name$ gets the name of the object from the info block which has been read out earlier using Examine Object.

OBJECT TYPE
function: return the type of an object
type=**Object Type**
type=**Object Type**(pathfile$)

This functions finds out if 'pathfile$' or the object whose data is currently in the info block is a directory or a file.
'type' is greater than 0 for a file, less than 0 if the object is a drawer.

OBJECT SIZE
function: return the length of a file
length=**Object Size**
length=**Object Size**(pathfile$)

This function returns the size of an object. The first version copies the size directly from the current info block.
The second version fills the info block with other data of the object 'pathfile$'.

OBJECT BLOCKS
function: return the length of a file in blocks
number=**Object Blocks**
number=**Object Blocks**(pathfile$)

Object Blocks returns the number of blocks the current or the object 'pathfile$' uses on a specific volume.

OBJECT PROTECTION
function: return the protection flags of an object
prot=**Object Protection**
prot=**Object Protection**(pathfile$)

Object Protection returns the Protection flags of an object.
The Object Protection$ function converts this numeric value into a string in the format "hsparwed".

OBJECT TIME
function: return the time of creation of an object
time=**Object Time**
time=**Object Time**(pathfile$)

Object Time returns the time of the last write access to an object.
The value 'time' can be converted into hours, minutes, seconds and ticks using the time functions.

OBJECT DATE
function: return the date of creation of an object
date=**Object Date**
date=**Object Date**(pathfile$)

Object Date returns the date stamp of an object as numeric value. This can also be separated into year, month, day and weekday using the date functions.

OBJECT COMMENT$
function: return the file note of an object
comment$=**Object Comment$**
comment$=**Object Comment$**(pathfile$)

This function returns the file note of an object.

Disk Support Functions

These are various functions to help you with disk access and filenames.

IO ERROR
function: return the last dos error code
errnumber=**Io Error**

The Io Error function gives back the number of the last DOS-error.

IO ERROR$
function: return a dos errorstring
error$=**Io Error$**(errnumber)

This function returns the string of the corresponding DOS error message.
If no string for the error number exists, an empty string will be returned.
When using Kickstart 2.0 not the interior list is taken but the localized strings of the system.

FILENAME$
function: return the filename of a full path
file$=**Filename$**(pathfile$)

This function cuts the filename out of an mixed path and file string.

```
Example: Filename$("DH2:AMOS/AMOSPro") results in "AMOSPro".
```

PATH$
function: return the directory of a full path
pathway$=**Path$**(pathfile$)

Path$ returns the pathname part of a mixed path-filename string. Could be used as a kind of Parent$-Function too.

```
Example: Path$("DH2:AMOS/AMOSPro") returns "DH2:AMOS".
```

EXTPATH$
function: append a forward slash to a path if required
newpath$=**Extpath$**(directory$)

Adds a slash to a pathname, if it is a directory. Handy if you want to add a filename to a path.

```
Examples: Extpath$("DH2:AMOS")+"AMOSPro" returns "DH2:AMOS/AMOSPro",
          Extpath$("DH2:")+"AMOS"        returns "DH2:AMOS".
```

OBJECT PROTECTION$
function: return an objects protection flags as a string
prot$=**Object Protection$**(prot)

Object Protection$ returns the Protection flags of an Object.
The Object Protection$ function converts this numeric value into a string in the format "hsparwed".

PATTERN MATCH
function: compare a string with a certain pattern
flag=**Pattern Match**(sourcestring$,pattern$)

Pattern Match checks, if the string 'sourcestring$' matches the pattern 'pattern$'.
If this is the case, then True (-1) will be returned otherwise False (0).
The pattern may contain any regular DOS jokers, an asterisk (*) will be converted into '#?' automatically.

This command only works on OS2.0 and higher.

DISK STATE
function: return the state of a disk device
flags=**Disk State**(directory$)

Disk State acquires the current state of a disk drive. 'flags' is a bitmap which holds two bits:

```
Bit
0=0: The disk is not write protected, you can write onto the disk.
0=1: The volume is write protected or is currently being validated.
1=0: Currently the disk is not in use, nobody is reading or writing.
1=1: The disk is currently in use.
```

If no disk is in the drive, it normally should return -1, but I'm afraid in this case a requester will be opened with the title "No disk in drive xxx:" which creates the error 'File not found'.

Please use the following workaround:

```
Request Off
Trap FLAGS=Disk State(DIRECTORY$)
Request On
If Errtrap Then FLAGS=-1
```

DISK TYPE
function: return the type of a volume
type=**Disk Type**(directory$)

The Disk Type function returns the type of a directory. This value can be:

```
0: It is a real device              (e.g. 'DF0:')
1: The path is a directory (assign) (e.g. 'LIBS:')
2: It is the name of a volume       (e.g. 'Workbench2.0:')

Using this function you can filter specific disk types out of a device list.
```

DOS HASH
function: calculate the hash value of a file
hash$=**Dos Hash**$(file$)

This function return the hash value of a file. Only for advanced users who want to read directly from dos disks.

Packer Support

Packer are nearly the most important tools, if you try to fit a lot of files onto a disk.
Powerpacker and File-Imploder are two of the best and how else could it be, AMCAF does support them to some extent.

PPFROMDISK
instruction: load and unpack a powerpacked file
Ppfromdisk file$,bank

This command loads the file named 'file$' into memory bank number 'bank' and decrunches it, if it has been packed using PowerPacker.
If 'bank' is a negative value, the file is unpacked into Chip ram instead.
If the file is Imploder packed, the Imploder Load command will be called, and if it is not packed, it will be loaded normally using Wload.

PPUNPACK
instruction: unpack a powerpacked file
Ppunpack sourcebank **To** targetbank

This command decrunches a powerpacked file in 'sourcebank' into the bank number 'targetbank'.
If 'targetbank' is a negative value, it will be decrunched into Chip ram.

Note: Powerpacked files can be identified by the first Longword containing 'PP20'.

PPTODISK
instruction: pack and save a file as pp20
Pptodisk file$,bank
Pptodisk file$,bank,efficiency

The Pptodisk command crunches and saves the bank numbered 'bank' into the file 'file$' using the PowerPacker algorithm.
The optional 'efficiency' parameter determines how good the bank is to be packed and must range between 0 (very fast, but less efficient) and 4 (best, but slow).

The powerpacker.library is requires for this and the two more commands. Sorry for the name 'Pptodisk' but 'Ppsave' has already been used by AMOS.

IMPLODER LOAD
instruction: load and decrunch a fileimploder file
Imploder Load file$,bank

This instruction tries to load and unpack a file-imploded file. The result will be stored in the bank numbered 'bank'.
If the specified file is not packed using file imploder, it will be loaded into memory normally. If the value of 'bank' is a negative number, the file will be unpacked into Chip ram.

Imploder Load does not need more memory to unpack the packed bank. Additionally, Imploder Load is extremely fast.

The File Imploder is part of the Turbo Imploder 4.0 package. Turbo Imploder 4.0 is a very fast packer for executables, written by Albert-Jan Brouwer and Peter Struijk.

The File Imploder can be found on the installation disk in the C: drawer (AMCAF_Install:C/Fimp) along with the manual AMCAF_Install:C/Fimp.man).

IMPLODER UNPACK
instruction: decrunch a fileimploder bank
Imploder Unpack sbank **To** tbank

This command unpacks the File Imploder packed file, which is stored in the memory bank 'sbank', into the target bank 'tbank'.

If the source bank does not contain a File Imploded file (which can be determined be the first Longword 'IMP!') a error message will be returned.
A negative 'tbank' value will force the bank to be unpacked into Chip ram.

Date and Time Functions

These functions are used to convert a disk object or system time to different formats.

Functions

CURRENT TIME
function: acquire the current time
time=**Current Time**

This function contains the current time. This is NOT a value in the standard DOS-format as this one would require two longwords. For everyone, who's interested: the time is created out of Wordswap(minutes)+ticks.

CT TIME$
function: create a complete time string
time$=**Ct Time$**(time)

This very handy function converts the normal parameter 'time' into a string in the format 'HH:MM:SS'.

CT HOUR
function: extract the hour from a time
hour=**Ct Hour**(time)

This function separates the hour from the packed time.

CT MINUTE
function: return the minute of a time stamp
minute=**Ct Minute**(time)

This function returns the number of minutes passed out from the Longword 'time'.

CT SECOND
function: calculate the second of a time
second=**Ct Second**(time)

The Ct Second function returns the number of seconds from the parameter 'time'.

CT TICK
function: extract the number of vertical blanks from the time
tick=**Ct Tick**(time)

Calculates the number of vertical blanks (=1/50 of a second) from the parameter 'time'.

CT STRING
function: evaluate a string into a time code
time=**Ct String**(time$)

This function will evaluate the valid time in the time$ string and then return the time as number or -1, if the string has been invalid.

The time string has to be in the format "HH:MM" or "HH:MM:SS".
This command only works on OS2.0 and higher.

CURRENT DATE
function: return the current date
date=**Current Date**

This command returns the current system date. This value counts the days passed since 1st January 1978.

To separate the number of years, months and day there certainly are many supporting functions.

CD DATE$
function: create a complete date string
date$=**Cd Date$**(date)

This simple function returns the date stamp 'date' as string in the format 'WWW DD-MMM-YY'.

CD YEAR
function: return the year from a date
year=**Cd Year**(date)

This function returns the year out from the DOS date stamp 'date'.

CD MONTH
function: return the month from a date
month=**Cd Month**(date)

Cd Month calculates the month of the argument 'date'. 'month' therefore lies between 1 and 12.

CD DAY
function: return the day from the date stamp
day=**Cd Day**(date)

This function returns the day in the month of the date stamp parameter 'date'. The result 'day' will be a value between 1 and 31.

CD WEEKDAY
function: return the weekday from the date stamp
weekday=**Cd Weekday**(date)

This function returns the weekday of the parameter 'date'. 'weekday' can range between 1 (monday) and 7 (sunday).

CD STRING
function: return a date stamp from a valid string
date=**Cd String**(date$)

This function will evaluate the valid date in the date$ string and then return the date stamp or -1, if the string has been invalid.
The string has to be in the format "DD-MMM-YY" or "DD-month-YY".

Strings like "Today" or "Tomorrow" are also allowed, weekday strings refer to the last occurrence of the week.
EG: "Monday" represents last monday and not next monday.

Note: This command only works on OS2.0 and higher.

Four Player Adapter

If you are writing games, it's a good thing to add a multi player option.
Given the case even four or more players may compete against each others, you should think about implementing the four player adapter.
Obviously, it's more comfortable to enjoy a game when using a joystick than a keyboard.

PJOY
function: acquire direction of a joystick
bitmap=**Pjoy**(j)

This command corresponds to the AMOS function Joy, with the difference, that one of the parallel port joysticks is checked instead of the normal joysticks.
'j' must be either 0 or 1.

```
The value 'bitmap' contains the following bits:

Bit 0=1: the joystick is currently moved up
Bit 1=1: the joystick is currently moved down
Bit 2=1: the joystick is pressed to the left
Bit 3=1: the joystick is pressed to the right
Bit 4=1: the fire button has been pressed
```

PJUP
function: check if the joystick is pressed up
flag=**Pjup**(j)

This command corresponds to the Jup(j) function for normal joysticks. Pjup tests, if the joystick in the parallel port is currently pressed up.
If this is the case, True (-1) will be returned otherwise False (0).

PJDOWN
function: check if joystick is pressed down
flag=**Pjdown**(j)

Corresponds to the Jdown(j) function for normal joysticks. Pjdown tests, if the joystick in the parallel port is currently pressed down.
If this is the case, True (-1) will be returned otherwise False (0).

PJLEFT
function: check if joystick is pressed left
flag=**Pjleft**(j)

Corresponds to the Jleft(j) function for normal joysticks. Pjleft tests, if the joystick in the parallel port is currently pressed to the left.
If this is the case, True (-1) will be returned otherwise False (0).

PJRIGHT
function: check if joystick is pressed right
flag=**Pjright**(j)

Corresponds to the Jright(j) function for normal joysticks. Pjright tests, if the joystick in the parallel port is currently pressed to the right.
If this is the case, True (-1) will be returned otherwise False (0).

PFIRE
function: check if fire button is pressed
flag=**Pfire**(j)

Corresponds to the Fire(j) function for normal joysticks. Fire(j) tests, if the firebutton for the joystick in the parallel port is currently pressed.
If this is the case, True (-1) will be returned otherwise False (0).

Commands for Other Gamepads

XFIRE
function: return the state of the fire buttons on a gamepad
flag=**Xfire**(port,button)

This function returns the state of a possibly existent second fire button on a joystick or joypad in the gameport numbered port.
If the lowlevel-library is available, all the other buttons can be checked as well. This library is distributed along with Kickstart 3.1 but even works with 2.0.

Possible values for the button parameter:

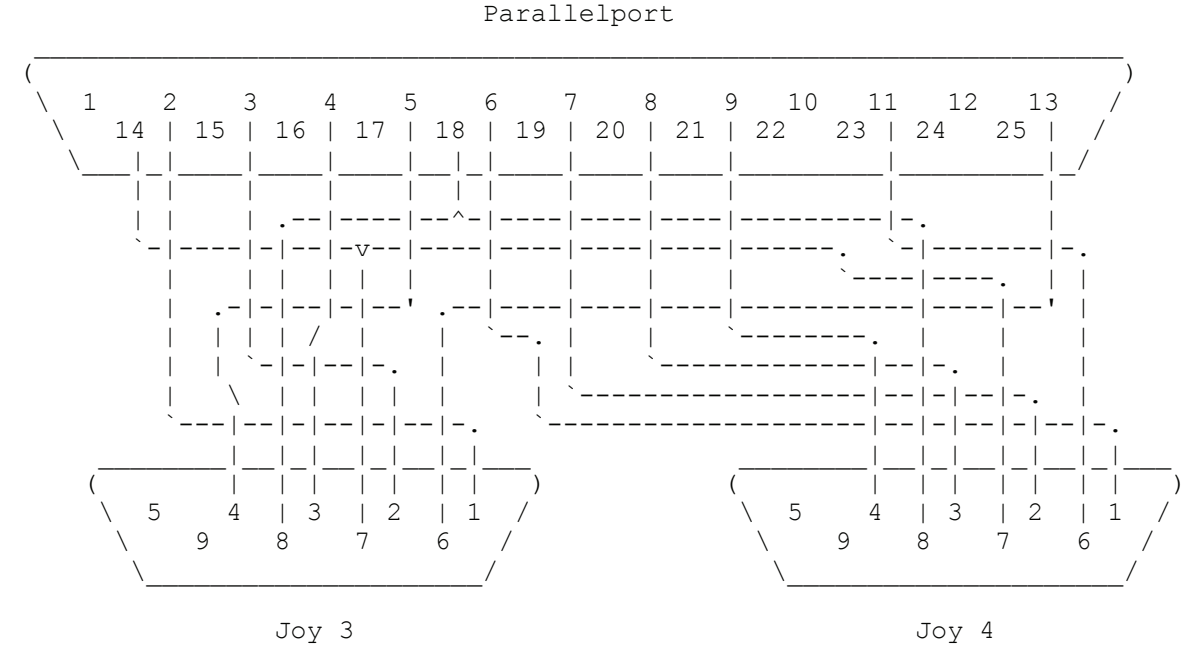
```
0: This is the normal fire button which exists on every joystick.
1: Second fire button on a few joysticks or the blue button on a gamepad.
2: Yellow fire button on a CD³² controller.
3: Green fire button
4: Reverse button on the gamepad
5: Forward button
6: Play/Pause button
```

Build a Game Adapter

Remark: I do NOT guarantee the correctness of this plan, nor I am responsible for any damage done to your computer if you're doing something wrong.
If you don't feel skilled enough to build the adapter yourself, you will find it in better electronic shops for a bargain.

```
Requirements: 1 25 pins male parallel socket.
              2 9 pins female joystick sockets.
              1 25 lines cable.
```

Blueprint of the parallel and joystick sockets.



For the ones that think this plan is to confusing:

Parallelport:		Joy 3:		Joy 4:	
P0	2			UP	1
P1	3			DOWN	2
P2	4			LEFT	3
P3	5			RIGHT	4
P4	6	UP	1		
P5	7	DOWN	2		
P6	8	LEFT	3		
P7	9	RIGHT	4		
BUSY	11	FIRE	6		
SEL	13			FIRE	6
+5 V	14	+5 V	7	+5 V	7
GND	18	GND	8	GND	8

As already said, there's no guarantee that it will work (although mine perfectly does!).

Mouse Commands

On request of some users, AMCAF now supports the reading of a second mouse in joystick port 1.
Therefore you can now realize your games which require a second mouse in a two player mode.

LIMIT SMOUSE
instruction: define the movement region of the mouse on the screen
Limit Smouse
Limit Smouse x1,y1 **To** x2,y2

This command defines the region in which the mouse can be moved on the screen.
If the parameters are omitted, the full size of the current screen will be used as default.

SMOUSE SPEED
instruction: set the speed of the mouse
Smouse Speed value

Smouse Speed sets the speed of the mouse. value therefore represents the factor by which power of 2 the mouse should be slowed down. 0 = max speed, 1 = normal AMOS mouse speed. Higher values than 4 are not sensible.

SMOUSE X
instruction: set the x coordinate of the mouse
Smouse X xpos

This command sets the x coordinate of the mouse to xpos.

SMOUSE Y
instruction: set the y coordinate of the mouse
Smouse Y ypos

This command sets the y coordinate of the mouse to ypos.

X SMOUSE
function: return the x coordinate of the 2nd mouse
xpos=**X Smouse**

This command returns the x coordinate of the second mouse.

Y SMOUSE
function: return the y coordinate of the 2nd mouse
ypos=**Y Smouse**

This command returns the y coordinate of the second mouse.

SMOUSE KEY
function: return the state of the mouse buttons
mk=**Smouse Key**

This command returns the state of the mouse buttons in port 1.

```
Bit
0 left mouse button
1 right mouse button
2 middle mouse button
```

Vector Rotate Commands

Note: this should not be a replacement for the 3d-extension

At least not at this time. Nevertheless you can create vector graphics without problems which is proved by the example programs.

VEC ROT POS
instruction: position the camera
Vec Rot Pos posx,posy,posz

Using Vec Rot Pos you can change the position of the camera. 'posx' and 'posy' hold the movement position of the camera in X and Y direction.
The value in 'posz' contains the distance to the camera.

There is no need to recalculate the matrix if you only change the position of the camera.

VEC ROT ANGLES
instruction: set the viewing angles
Vec Rot Angles angx,angy,angz

This command sets the three viewing angles, which are used for the vector rotation.
'angx' represents the angle, the coordinate is to be rotated along the X-axis.

Same with 'angy' and 'angz', which are related to the Y- and the Z-axis.
The angles are -like with Qsin and Qcos- neither values in degree nor radians format, but a revolution (360 degrees) equals the value 1024.

After you have given the angles and the new position, you can calculate the new vector matrix using the Vec Rot Precalc command.

VEC ROT PRECALC
instruction: calculate the precalc matrix
Vec Rot Precalc

This command must be called before every vector rotations, always after you have changed the viewing angle.
It calculates a internal matrix that holds the results of often needed calculations which are required for the rotation in three dimensions.

After you have called Vec Rot Precalc, you can use the Vec Rot X,Y and Z functions.

Functions

VEC ROT X
function: calculate the 2d x value
newx=**Vec Rot X**(x,y,z)
newx=**Vec Rot X**

This function calculates the X coordinate, which results in the vector rotation of a three dimensional point x,y,z around all three axis.
This coordinate is automatically projected from 3D to 2D by dividing it through the distance.

If you call the function with the parameters x,y,z all three new coordinates are calculated, i.e the y,z position too.
If you require the Y coordinate too, when simply write the Vec Rot Y command **without** the parameters. Same with Vec Rot Z.

This parameter system can be also used on Vec Rot Y in the same way.

VEC ROT Y
function: calculate the y value
newy=**Vec Rot Y**(x,y,z)
newy=**Vec Rot Y**

This function calculates the new Y coordinate after the rotation. For more detailed information see Vec Rot X.

VEC ROT Z
function: return the z coordinate
newz=**Vec Rot Z**(x,y,z)
newz=**Vec Rot Z**

This function returns the new Z coordinate after the rotation.

Please note that the current camera position has already been added to this value.
The Z coordinate can be used to sort objects by distance or to evaluate a brightness level.
More information on the syntax or the parameters see Vec Rot X.

Protracker Music Commands

As the AMOS Tracker commands are really crap and contain a lot of bugs, of course there is a replacement for them in the AMCAF extension.

Since version 1.1 there are new commands even to replay sound effects during the normal protracker music.
The AMOS Music Extension is rather obsolete now.

Here a brief list of all advances of the AMCAF commands:

- CIA timing or VBL timing.
- Volume control.
- Channel toggling.
- Vu meters.
- Possibility to receive signals from the module.
- Support of all Protracker effect commands.
- Playback of sound effects along with the music.

PT PLAY
instruction: replay a module
Pt Play bank
Pt Play bank,songpos

Pt Play stars a Protracker music module which has be situated in memory bank number 'bank'.
Optionally, you can specify a song position, which the music should be played from.

Before you start the music with Pt Play you should choose either Vertical Blank-Timing or CIA-Timing for the module using Pt Cia Speed.

With help of the Pt Instr Play instruction you can replay any instrument of the music track.
This method has the advantage that you can reuse a sample that occurs in the music for your sound effects.

If you want to use the instruments only, without replaying the music, you can call the Pt Bank command instead.
The music will be turned off again with a call to the Pt Stop instruction.

PT STOP
instruction: stop the current music module
Pt Stop

This command stops the current music module. You can restart the music with the Pt Continue command from the very position you stopped it.

PT CONTINUE
instruction: restart a previously stopped protracker music
Pt Continue

This command restarts the protracker music at that point it has previously been stopped using Pt Stop.
Do not attempt use Pt Continue without having started the music once with Pt Play.

PT VOLUME
instruction: set the volume of the protracker music
Pt Volume value

The Pt Volume instruction sets the volume of the music. The parameter 'value' may range from 0 (silent) to 64 (full volume).

PT VOICE
instruction: toggle the audio channels
Pt Voice bitmask

This command defines the channels which should be used for the music and which should be free for e.g sound effects.

The 'bitmask' parameter contains -similar to the AMOS Voice command- four bits, that are assigned to one channel each. If a bit is set, this channel can be heard. Pt Voice %1111 turns on all sound channels.

PT CIA SPEED
instruction: change the replay speed of a protracker module
Pt Cia Speed bpm

With the help of this command, you can set the speed for replaying the Protracker modules.
However, the music may change the speed on it's own if it has been composed for CIA-Timing.

The parameter 'bpm' set the number of beats per minute or if you specify a value of zero, the timing will be switched from CIA-Timing to Vertical Blank Timing.
Then the bpm rate is automatically set to exactly 125 and all CIA-Timing from the module are ignored if such appear in the music.

CIA-Timing causes some problems in conjunction with AMOS sprites, which show up as a flickering at regular intervals. If you work with sprites you should switch back to VBL-Timing where possible using Pt Cia Speed 0.

If a music module is not replaying correctly using VBL-Timing, i.e it is running to fast or slow, you must switch to Cia-Timing.
This can achieved with a Pt Cia Speed 125 command directly in front of a Pt Play instruction.
The default is CIA-Timing with 125 bpm.

PT RAW PLAY
instruction: play a chunk of memory as a sound sample
Pt Raw Play voice,address,length,freq

This command corresponds to the normal AMOS command Sam Raw. Pt Raw Play replays a piece of memory as an audio sample.

The 'voice' parameter sets the channels to use, 'address' contains the starting address of the memory block.
'length' is the length of the sample in bytes and 'freq' holds the replaying speed in Hertz.

PT BANK
instruction: set the bank to use with pt instr play
Pt Bank bank

This command is used if you want to play back instruments from a music module but the music bank has not yet been specified with Pt Play.

PT INSTR PLAY
instruction: play an instrument of a protracker module
Pt Instr Play instnr
Pt Instr Play voice,instnr
Pt Instr Play voice,instnr,freq

The Pt Instr Play instruction replays an instrument of the current Protracker music module.

The 'voice' parameter contains a bitmask that says on which channels the instrument number 'instnr' should be played. If it is omitted, the sound effect is output on all four channels. 'instnr' must range between 1 and 31.
The optional argument 'freq' holds the frequency which should be used for the sample.

If you want to replay a sample repeatedly, just specify a negative value for 'instnr'.
The volume of the instrument can be set with the Pt Sam Volume command.

Before you can replay an instrument of a module, you first have to specify the bank that holds the music.
This can be achieved either by a call to Pt Bank or Pt Play.

PT SAM BANK
instruction: set the bank to use with amos samples
Pt Sam Bank bank

If you want to replay standard AMOS samples with AMCAF, you have to inform AMCAF about the Sample-Bank, which contains the sound effects.
And that's exactly what Pt Sam Bank does. 'bank' holds the bank number of the AMOS sample bank.

Please remember, that you can replay the instruments of music modules directly.

PT SAM PLAY
instruction: replay a sample from an amos sample bank
Pt Sam Play samnr
Pt Sam Play voice,samnr
Pt Sam Play voice,samnr,freq

The Pt Sam Play command plays a sample out of an AMOS sample bank. The advantage to the normal Sam Play instruction is that the sounds 'interact' with the Protracker music.

The 'voice' parameter contains a bitmask, that describes, on which channels the sample number 'samnr' should be replayed. If it is omitted, the sound effect will be played on all four sound channels. The optional argument 'freq' holds the replaying speed, that shall be used.

If you want the sample to be looping, just give a negative value for 'samnr'. The volume of the sample can be changed using Pt Sam Volume.

PT SAM STOP
instruction: stop the sfx on specific audio channels
Pt Sam Stop voice

This command stops a sound effect on the channels, which are defined in the bitmap 'voice'. This is valid for samples started by Pt Sam Play, Pt Instr Play and Pt Raw Play.

PT SAM VOLUME
instruction: set the volume of a sound effect
Pt Sam Volume volume
Pt Sam Volume voice,volume

This instruction sets the volume for the following sample replays. The 'volume' parameter must range from 0 to 64.
If you specify the optional parameter 'voice', the command only has effect on the currently played sample, but not on the following samples.

PT SAM FREQ
instruction: change the replaying speed of a sample
Pt Sam Freq chan,freq

This command changes the frequency of the sample currently being played on channel chan to the new frequency in freq.
If no sample is currently played on this channel, it will have no effect nor will it have effect on the protracker music.

PT VU
function: return the current vumeter value
vol=**Pt Vu**(channel)

This command allows you to read the current volume of channel number 'channel'.
If a new note is played, 'vol' contains the volume level else 0.

PT SIGNAL
function: check for signals from the music
signal=**Pt Signal**

The various music trackers Soundtracker, Noisetracker, Startrekker or Protracker feature effect commands e.g.
To change the volume of a note, or for portamento. Example for a Protracker noteline:

```
00 C-1 1000 --- 0000 G-2 2000 --- 0000
   ^ ^ ^
   | | `-----------------------------------------------.
   | `---------------------.                           |
   `--------------.        |                           |
   Plays the note C in the 1st octave using instrument 1.

Additionally, you can use effects, e.g like in this line here:

00 C-1 1C20 --- 0F04 G-2 2E00 --- 0801
        ^^^      ^^^      ^^^      ^^^
Set vol.to $20  Speed  Filter Off  \|/
         (=32)  4 VBLs              Y
 .---------------------------------'
 |
```

Now this is an effect, which is unused in all trackers. It has been left free to synchronize various events to the music.
The value behind the '8' is directly transferred to Pt Signal, when the music reaches the specific point.
Warning: The value used in Protracker is entered as hex number.

With this technique you can wait for a special acoustic signal, or sync some kind of musician bobs to the music.

If there is no signal waiting, zero is returned in the variable 'signal'. After using Pt Signal, the old signal will be cleared.
Pt Signal can only keep one value at once, i.e if the last value hasn't been read out, the next signal will overwrite the last one and is therefore lost.

If the end of the music is reached an $FF signal is sent automatically.

PT CPATTERN
function: return the current song position
songpos=**Pt Cpattern**

This function returns the current pattern/song position.
This can be used to restart a song at a given position when you stopped the music with Pt Stop.

PT CPOS
function: return the current pattern line
row=**Pt Cpos**

This function returns the current 'row' of the songs current pattern. The returned value is a number between 0 and 63.

PT CINSTR
function: return the current instrument being played
num=**Pt Cinstr**(chan)

Pt Cinstr returns the instrument being played on music channel chan. num is therefore a value between 0 and 31.
A value of 0 tells you, that no sample has been trigged at that very moment.

PT CNOTE
function: return the frequency of the current instrument
num=**Pt Cnote**(chan)

Pt Cnote returns the frequency of an instrument being played on music channel chan at that very moment.

PT INSTR ADDRESS
function: return the address of an instrument
address=**Pt Instr Address**(instnr)

This command returns the starting address of the music instrument with the number 'instnr' of the current Protracker module.

PT INSTR LENGTH
function: return the length of an instrument
length=**Pt Instr Length**(instnr)

This command returns the length of the instrument 'instnr'.

PT DATA BASE
function: return the address of the pt-database
address=**Pt Data Base**

This command returns the starting address of the internal data table of the protracker replay routine. Please don't write into this area randomly.

String and Integer Functions

AMCAF provides a vast number of useful functions to assist with integer math and string manipulation.

BINEXP
function: return the result of two to the power n
v=**Binexp**(a)

This function calculates the result of the exponential function 2^a, but is much faster than the normal AMOS expression. The parameter a must lie between 0 and 31.

```
Examples: Binexp(1)=2, Binexp(3)=8, Binexp(16)=65536, Binexp(24)=16777216
```

BINLOG
function: return logarithmic function on a basis of two
a=**Binlog**(v)

Binlog is the reverse function to Binexp. It returns the logarithm to the value 'v' with basis 2.
'v' therefore must be a power of 2, otherwise you will get an error.

Binlog is handy to get the number of bitplanes out from the amount of colours of a screen (exception: HAM).

```
Examples: Binlog(2)=1, Binlog(8)=3, Binlog(65536)=16, Binlog(16777216)=24
```

LSL
function: return a value as a quick multiplication by a power of two
nv=**Lsl**(v,n)

Rotates the number 'v' to the left by 'n' bits. I.e that a number 'v', which is shifted one bit to the left contains the value v*2, with two bits v*4, with 3 bits v*8 etc.

This function is very quick and should be used instead of multiplications wherever possible.

LSR
function: return a value as a quick division by a power of two
nv=**Lsr**(v,n)

LSR shifts a number 'v' to the right by 'n' bits. This function does the same as a division by 2^n, but is much quicker.

WORDSWAP
function: return the swap of the upper and lower 16 bits of a value
newval=**Wordswap**(value)

This command swaps the upper and the lower Word of a value.

QARC
function: return an angle from the coordinates
angle=**Qarc**(deltax,deltay)

This function will return the angle to a point at the relative coordinates deltax,deltay.
Angle is in the same format used with Qsin and Qcos, so an angle of 360 degree is equivalent to 1024.
This function is **very** fast and rather accurate.
It's normally used for all kinds of 'aiming-at' routines, e.g to get the angle from one player to another to shoot a missile at ;-)

QSIN
function: return a sine of an angle
value=**Qsin**(angle,radius)

This is a function to replace the original sine function of AMOS.
It is faster and even allows you to multiple the value with the a parameter.

No math libraries are required.
The angle of the sine 'angle' is not a value in radians or degrees, but with Qsin, an angle of 360 degrees equals a value of 1024.

QCOS
function: return the cosine of an angle
value=**Qcos**(angle,radius)

This is a function to replace the original cosine function of AMOS.
It is faster and even allows you to multiple the value with the a parameter.

No math libraries are required.
The angle of the sine 'angle' is not value in radians or degrees, but with Qcos, an angle of 360 degrees equals a value of 1024.

QRND
function: return a random number
value=**Qrnd**(maximum number)

Qrnd is totally identical to the Rnd function, with the only difference, that this one is much faster.

QSQR
function: return the square root of a value
root=**Qsqr**(value)

This function calculates the square root from the value 'value'.
However, it only works with integer and is faster than the AMOS square root function =Sqr.

ASC.W
function: convert a word string into a number
word=**Asc.w**(word$)

Asc.w is used to convert a word string to a number value. Therefore the result will be between 0 and 65535.
If the length of 'word$' is less than two, the function is aborted and an error message is returned.

ASC.L
function: convert a long string into a number
long=**Asc.l**(long$)

The Asc.l-function converts a 4 bytes long string back into a number. This value can range between -2147483648 and +2147483647.
If 'long$' contains less than four characters, you will get an error message immediately.

VCLIP
function: restrict a value to a given range
newval=**Vclip**(val,lower **To** upper)

The Vclip function works like a Min(Max(val,lower),upper) expression, it restricts a value val into its lower and upper boundaries.

VIN
function: return a flag to test if a value is within a range
flag=**Vin**(val,lower **To** upper)

This function will check if a value lies between lower and upper. If this is the case, flag will return True (-1), otherwise False (0).

VMOD
function: return a value from a modulo operation on a value
newval=**Vmod**(val,upper)
newval=**Vmod**(val,lower **To** upper)

Vmod restricts a value val into the lower and upper boundaries. However, it does it in another way compared to Vclip.
If val exceeds upper by 1, it will be set to lower, if it exceeds upper by 2, it will be set to lower+1.
If it goes deeper than lower by 1, it will be set to upper and so on.

```
So this function is not the same as

Add val,delta,lower To upper

If lower is omitted, zero is taken as lower boundary.

Example:

Print Vmod(100,50 To 150) would return 100 (should be clear).

Print Vmod(151,50 To 150) would return 50.

Print Vmod(152,50 To 150) would return 51.

Print Vmod(49,50 To 150) would return 150.

Print Vmod(0,50 To 150) would return 101.
```

EVEN
function: return a flag if a number is even
flag=**Even**(value)

This function returns True (-1), if a number is even or False (0), if a number is odd.

ODD
function: return a flag if a number is odd
flag=**Odd**(value)

This function returns True (-1), if a number is odd or False (0), if a number is even.

PATTERN MATCH
function: match a string with a certain pattern
flag=**Pattern Match**(sourcestring$,pattern$)

Pattern Match checks, if the string 'sourcestring$' matches the pattern 'pattern$'.
If this is the case, then True (-1) will be returned otherwise False (0).
The pattern may contain any regular DOS jokers a asterisk (*) will be converted into '#?' automatically.

This command only works on OS2.0 and higher.

CHR.W$
function: return a two byte string from a number
word$=**Chr.w$**(word)

This function converts a number into a 2 bytes string.
The upper 16 bits of the value are ignored and therefore you should only use values from 0 to 65535.

CHR.L$
function: return a four byte string from a number
long$=**Chr.l$**(long)

The Chr.l$ function converts a number into a 4 bytes string. 'long' can be any number you like.
Using this technique, you can save numbers as normal strings.

LZSTR$
function: return a right adjusted number with leading zeros
s$=**Lzstr$**(v,n)

This function is nearly identical to Lsstr$ with the difference that leading zeros are not replaced by space characters.
The parameter 'n' sets the number of digits the string should contain and can range from 1 to 10.

LSSTR$
function: return a right adjusted number
s$=**Lsstr$**(v,n)

Similar to the AMOS function Str$, Lsstr$ creates a string out of a number.
However, with Lsstr$ the number will be created right justified with 'n' digits, leading zeros are replaced by spaces.
The sign of the number will not be printed. 'n' must lie within 1 to 10.

INSSTR$
function: insert the second string into the first string
newstr$=**Insstr$**(a$,b$,pos)

This function inserts the string b$ at the position pos into string a$.

```
Example:

Print Insstr$("Hello Ben!","dear ",6) creates 'Hello dear Ben!'.
```

CUTSTR$
function: cut out a piece of a string
newstr$=**Cutstr$**(s$,pos1 **To** pos2)

This function cuts out part of the string s$ from letter position pos1 to pos2.

```
Example:

Print Cutstr$("Hello dear Ben!",7 To 11) would generate 'Hello Ben!'.
```

REPLACESTR$
function: replace a string with another one
newstr$=**Replacestr$**(s$,search$ **To** replace$)

This function searches in the string s$ for occurrences of search$ and replaces them with the string replace$.

ITEMSTR$
function: returns an item contained in a string
item$=**Itemstr$**(s$,itemnum)
item$=**Itemstr$**(s$,itemnum,sep$)

This is a very handy function that helps you to avoid little string arrays where actually no are required.
s$ contains a number of so called items, which are numbered increasingly from zero onwards.

Normally, the items are separated with a '|' character, but you can give your own single character for separation in the optional string sep$.

Empty strings for s$ are not allowed and will create an error message, however, empty items can be used without hesitation.

Trying to access an item, that does not exist, will create an error as well.

```
Example:

Print Itemstr$("Ben|Semprini|Petri|Andy",1) would return 'Semprini'.

Print Itemstr$("The quick brown fox",2," ") would return 'brown'.

Print Itemstr$("zero|one|two||four|five,5) would return 'five'.

Print "The weather is "+Itemstr$("great|fine|nice|not bad|rainy|awful|apocalyptic",WEATHER)

would have a similar effect to:

Dim W$(6)
For A=0 To 6
  Read W$(A)
Next
Print "The weather is "+W$(WEATHER)
[...]
Data "great","fine","nice","not bad","rainy","awful","apocalyptic"
```

Misc Commands and Functions

EXCHANGE BOB
instruction: swap the two images in the sprite bank
Exchange Bob image 1,image 2

This command simply swaps the two images image 1 and image 2 in the current sprite bank. image 1 and image 2 must exist as a valid image, otherwise an error will be reported.

EXCHANGE ICON
instruction: swap the two images in the icon bank
Exchange Icon image 1,image 2

This command simply swaps the two images image 1 and image 2 in the current icon bank. image 1 and image 2 must exist as a valid image, otherwise an error will be reported.

AUDIO LOCK
instruction: reserve the audio device
Audio Lock

When you start AMOS, the audio.device will be not informed, that AMOS wants to have the audio channels.
Due to this flaw, other programs that are running in the background can replay a sound at any time, which will irritate the AMOS sound system.
To avoid this and to be more system friendly, you can use the command Audio Lock to reserve the sound channels.
As AMOS has a bug, you'll have to do it like this:

```
Extremove 1     : Rem removes music extension
Trap Audio Lock : Rem reserves the audio channels
ERR=Errtrap     : Rem tests, if audio lock was successful
Extreinit 1     : Rem initializes the music extension again
Extdefault 1    : Rem calls the default routine of the extension
```

This Routine should be called as soon as possible and BEFORE you start to play music or sounds.
If the audio channels are already in use by any other program, ERR holds an error code, otherwise zero.

To free the channels again, you have to call Audio Free.

AUDIO FREE
instruction: free the audio device
Audio Free

Using Audio Free you can free previously allocated sound channels again.

OPEN WORKBENCH
instruction: reopen a previously closed workbench
Open Workbench

This command tries to open the workbench again, if it has been closed previously.
The Workbench screen can be closed using the AMOS command Close Workbench to get more memory.

FLUSH LIBS
instruction: free as much memory as possible
Flush Libs

This command closes all libraries, fonts and devices, which are currently not in use and tries to get as much memory as possible. During this process the PowerPacker and the diskfont.library will be 'flushed', too.

WRITE CLI
instruction: write text into the cli window
Write Cli text$

The Write Cli instruction writes the string 'text$' into the CLI window, AMOS or your program has been start from.
If this window does not exist, no text will be output.

RESET COMPUTER
instruction: reset your computer
Reset Computer

This very dangerous instruction reboots your Amiga. Obviously, this command never returns to the program.

NOP
instruction: no effect
Nop

This 'command' has no effect et al. It's only use is for speed testing routines.

AGA DETECT
function: check if the computer has aga chipset
flag=**Aga Detect**

This system friendly function detects the AGA chipset, found in A1200, A4000 and CD³² machines.
It returns -1 (True), if the new chipset is mounted or 0 (False) if the ECS or OCS chipset are still in use.
Note that it does not check for the kickstart version only, so A2000 and A500 machines with kick 3.0 or higher will be detected as non-AGA correctly.
Therefore, if the AGA chipset had been disabled either using the boot menu or the "Setpatch NOAGA" command, it will return that no AGA is available.

SCANSTR$
function: return the name of a key
key$=**Scanstr$**(scancode)

This very handy function returns the name of a key according to the parameter 'scancode', which can be fetched using the AMOS Scancode function. If there is no key for the scancode, an empty string will be returned.

COMMAND NAME$
function: return the name of the program
fname$=**Command Name$**

Returns the file name of the program under which AMOS or the compiled program has been started.
This is required for example to read the own Tool Types.

TOOL TYPES$
function: read the tool types of an icon
tools$=**Tool Types$**(filename$)

With this function you can acquire the Tool Type of an icon. The supplied file 'filename$' must not have a '.info' appended!
The various Tool Types are separated by a line feed character (Chr$(10)). So they can be printed out easily using Print.
The name of your own program can be fetched using Command Name$.

AMOS CLI
function: return the cli process number of an amos.program
n=**Amos Cli**

This function gives back the number of the cli process out of which the program/AMOS has been started off or zero, if AMOS has been started from Workbench.
This gives you the choice to either interpret options from the command line or from the tool types of the appropriate icon.

SPEEK
function: return a signed byte from memory
value=**Speek**(address)

The Speek function reads a Byte from the memory address 'address', exactly the AMOS function Peek.
However, Bit 7 is used as sign bit so the result will be a value between -128 and 127.
You are advised to use this function instead of Peek if you have poked a negative value into memory and then want to read it again.

SDEEK
function: return a signed word from memory
value=**Sdeek**(address)

The Sdeek function reads a Word from the EVEN memory address 'address', exactly the AMOS function Deek.
However, Bit 15 is used as sign bit so the result will be a value between -32678 and 32677.
You are advised to use this function instead of Deek if you have doked a negative value into memory and then want to read it again.

CPU
function: return the number of the fitted cpu
chip=**Cpu**

The cpu function returns the identification number of the central processing unit currently installed your Amiga.
This number can currently lie between 68000 to 68060. Kickstart 1.3 knows only CPUs to the 68020.

FPU
function: return the id number of an coprocessor
chip=**Fpu**

This function returns the number of an installed mathematic coprocessor or 0, if none is fitted. As before, Kickstart 1.3 only knows the 68881.

On 68040/68060 machines, the cpu contains the fpu, so 68040 or 68060 will be returned instead.

NFN
function: no effect
dummy=**Nfn**

This 'function' returns nothing useful. It's only used, like Nop, in speed testing routines.

COP POS
function: return the current address of the copper list
address=**Cop Pos**

If you create your own copper list, you can use this function to remember the position of the next copper instruction.
Later you can then write to this address directly to change the values of a copper instruction.

Extension Commands

Again some commands and functions for advanced users.
Before using these, think about what you are doing, then again, and once more, and more important, save your program before starting it.

EXTDEFAULT
instruction: call the default routine of an extension
Extdefault extnb

This command has been written especially for advanced users.
With this one you can call the routine in the AMOS extension in slot 'extnb' which is normally executed after the start of a program or by the Default command.
Often this routine resets the interior database to the default values.

Extdefault should be called after re-linking an extension with Extreinit.

If no extension is loaded in slot number 'extnb', this instruction has no effect.

EXTREMOVE
instruction: remove an extension from memory
Extremove extnb

The Extremove command removes the extension in the slot 'extnb' from memory like when exiting AMOS.
After this command no further instructions requiring this extension may be called.
As AMOS tries to unlink the extension itself on quitting, this could cause some trouble.
So don't remove an extension too long, revoke it with Extreinit before exiting.

EXTREINIT
instruction: try to restart a previously removed extension
Extreinit extnb

This command causes a restart of the extension numbered 'extnb'.
Extreinit may only be called on a extension that has been removed using Extremove.
Otherwise, you can lose memory or even crash your computer.

AMOS TASK
function: return the address of the amos task structure
address=**Amos Task**

For advanced programmers. This function returns the address of the AMOS task structure.
Using this you can for example increase the task priority:

```
Areg(1)=Amos Task
Dreg(0)=3
dummy=Execall(-$12C)
```

This program sets the task priority to +3.

AMCAF VERSION$
function: return an amcaf version string
Print **Amcaf Version$**

This command returns the current AMCAF version number and a greetings list.

AMCAF BASE
function: return the base address of the amcaf database
address=**Amcaf Base**

This command returns the base address of the AMCAF database.

AMCAF LENGTH
function: return the length of the amcaf database
length=**Amcaf Length**

This command returns the length of the AMCAF database.

Additional Information

Bank Information

AMOS banks are a linear block of memory (there are two Exceptions). In these chunks various kinds of data is stored. AMOS uses them mostly for packed graphics, sounds, music, AMAL programs, menus, resources and other data.

Generally, there are four main types of banks:

- Banks, that are stored in Chip ram and remain there permanently.
- Banks, that are stored in Fast ram and remain there permanently.
- Banks, that are stored in Chip ram and remain there temporarily only.
- Banks, that are stored in Fast ram and remain there temporarily only.

Icons and Sprites Banks

These banks do NOT consist of an linear block of memory. Therefore you neither can move, copy, encode, pack them nor make a checksum from them.
In addition, icon and sprites banks must always be in Chip ram.

Permanent Banks

These banks are called permanent, because they survive calls to Default, Erase Temp and the start of a program. Moreover, they are saved along with the program. Normally, permanent banks have the name 'Datas'.

Temporary Banks

Temporary banks only exist during the execution of the program. They are erased on every start, testing or saving process or using the commands Default or Erase Temp.
Normally, temporary banks have the name 'Work'.

Chip Ram

On Amiga computers, chip ram is the area in memory, which is accessed by both custom chips and the CPU.
At the moment, the size of chip ram is limited to 2 MB (even on A1200 and A4000), and old A500 do only have 512 KB chip ram by default.
For that reason your program should not use more than this 512 KB of chip memory. However, this does NOT mean, that your program needs to run on 512 KB total memory!

As custom chips access the memory at the same time with the processor, the CPU is slowed down, if the program is held in chip ram.
This effect is even more severe if the screen uses more than 16 colours in lowres or more than 8 colours in hires (not A1200/A4000).

Chip ram is mainly required for screens, bobs and sprites, music and sound effects, floppy disk drives and copperlists.

Fast Ram

Fast ram is the memory area which the custom chips don't have access to. So the processor is not slowed down, if the program runs in fast ram.

BUT: You must not store any data for bitplanes, bobs and sprites, music or sound effects and then try to access it by the custom chips.
This has very unpleasant if not lethal results in most cases.

All Amigas excluding the A3000/A4000 do not have fast ram mounted by default.

Ranger Ram

Ranger ram is a special type of ram: it is neither chip nor fast ram and only exist on an Amiga A500 that has a trap door slot memory expansion.
The custom chips cannot access this ram, although the memory is not faster at all

There's a way to make ranger ram to chip ram. Just resolder jumper 2 on the main board and switch off the memory expansion. I am not responsible for any possible damage that may occur by doing this.

Byte

A bytes has got 8 bits, therefore you can display values from 0 up to 255. A byte normally has got the suffix '.b' (in assembler).
Two bytes together create a Word, four a Longword.

Word

A word consists of two Bytes so that is 16 bits. With these 16 bits you can store addresses or data up to 64KB (65536).
A word must be on a even memory address or machines with MC68000 CPU will crash with Guru number $80000003.
Since MC68020 this is not important, but you never should assume that there is no MC68000 in the computer and use odd addresses!

If you insist on using uneven addresses, check out the installed cpu using the Cpu function, if a 68020 or higher is fitted.

Longword

Four bytes result in a unit of 64 bits. These units are called longwords and are used to address up to four gigabyte of data. For that reason they are used for every absolute address in memory. As with Words, they must lie on a even address boundary.

Cyclic

If a number reaches the upper boundary, the number is set to the lower boundary value and vice versa.

```
Example:

UB=31 (upper boundary)
LB=1  (lower boundary)
N=3   (number)
Do
  Inc N
  If N>UB Then N=LB
  If N<LB Then N=UB
  Print N
Loop
```

AGA Amiga

The new Amigas A1200, A4000 and CD³² have the AGA-Chipset.
This new chipset makes is possible not only to display 6 Bitplanes but even 8 Bitplanes in nearly every resolution.

These Amigas have 12 additional colour bits to the normal 12 bits ($0RGB) and therefore can use a 24 bit value ($00RRGGBB).
Even if AMOS Pro V2.0 does not currently support AGA, nevertheless AMCAF contains some commands for the future implementation.

The most important advantages of AGA-Amigas:

- Locale library: Many programs can use different languages
- Kickstart 3.0 with improved graphic routines
- Many, new, high resolutions
- up to 256 colours in nearly every resolution
- new HAM8-Mode for over 262.000 colours
- 16 colour Dual Playfields possible
- MC68020+ for more processor power
- 2 MB Chip ram

My advice: If you still have an old Amiga you should think about buying a new Amiga, it's worth it!

Amiga HAM Mode

As the Amiga can only display 6 or 8 bitplanes and you need much more colours than 64 or 256 colours to achieve photo quality. The Amiga developers invented a new tricky display mode: The HAM Mode (abbreviation for "Hold And Modify").

HAM6 (without AGA chipset):
The first 16 colours can be set as normal, these are displayed on the screen as usual.

The colours 16 to 31 modify the blue part of the last colour to the left.
Same with the colours from 32 to 47, which alter the green value and the colours from 48-63 change the red value accordingly. So you can display all the 4096 colours using only 6 bitplanes.

HAM8 (AGA chipset required):
Like the HAM6 mode the first colours are displayed correctly, but in HAM8 the base palette has got 64 colours.
The rest of the colours from 64 to 255 are responsible for the modification of the previous colour like shown above.
Using this technique and a intelligent base palette you can display every colour in the 16777216 colours big palette.

Disk Object

A disk object can be:

- a file on a drive
- a directory
- an assign
- a volume

The Blitter Chip

The Blitter is a co-processor inside the Amiga which is mainly used to copy and combine data (therefore BLockImageTransferER). Additionally, it can fill polygons and draw lines.
The Blitter is rather fast at this and works with an unbelievable speed of up to 16 million pixels per second. All data that is accessed by the Blitter chip must be in Chip ram.
AMOS uses the blitter for Bobs, Icons, Screen Copy and many other commands. The MC68020 and higher is much faster if only need to copy data.
The Blitter works at word boundaries and this results in cutting down the X coordinates to the nearest multiple of 16.

To fill a polygon using the Blitter, the lines must be only one pixel thick. This is the reason why there are two different ways to draw lines.

Blitter Minterms

The Blitter chip knows 256 different copying and combining modes. These are determined in two steps:

1. Eight different boolean terms are used upon the three databits. Each of the terms returns true on a specific combination of A, B and C.
2. These eight results of the terms are combined together using a logical OR. This result is the target bit D.

Bit	Minterm	Input-Bit
0	$\overline{A}\overline{B}\overline{C}$	000
1	$\overline{A}\overline{B}C$	001
2	$\overline{A}B\overline{C}$	010
3	$\overline{A}BC$	011
4	$A\overline{B}\overline{C}$	100
5	$A\overline{B}C$	101
6	$AB\overline{C}$	110
7	ABC	111

Procedure:

1. At which of the eight combinations of ABC should D be true?
2. Now set the bits of the bitmask.
3. If not all of the three source data streams are required, every combination with the unused bits and the desired bits must be chosen.

Protection Flags

These flags contain information about the type of a certain Disk object.

```
The protection value consists of following bits:

Bit 0=0: File can be erased
Bit 1=0: File is executable
Bit 2=0: File can be overwritten
Bit 3=0: File can be read
Bit 4=1: File has not been changed after copying
Bit 5=1: Executable can be made resident.
Bit 6=1: File is an Amiga-DOS script
Bit 7=1: File is hidden (does not actually work)
```

You're advised to use the function Object Protection$ to convert this bitmap into a String in the format "hsparwed".

The TOME Extension

TOME is an extension for AMOS Creator and recently for AMOS Professional too, which is dedicated to tile and map programming. These tiles are used in many games e.g Jump'n'Runs or strategy games.

As the whole game map would be far to memory hungry when kept as standard bitmap, the main graphics are cut into small pieces, so that they can be used repeatedly.
The map therefore only consists of one byte per position which points to the corresponding tile.

The current version of TOME is TOME V4.30.

AMCAF Database Structure

```
Version: V1.15
Length : 1964 Bytes
Note   : Contrary to most other extensions, the database is not
         kept in the AMCAF.Lib file. So AMCAF is very compact (*only* 40 KB!?).

                rsreset         ;Stars
St_X            rs.w    1       ;0
St_Y            rs.w    1       ;2
St_DbX          rs.w    1       ;4
St_DbY          rs.w    1       ;6
St_Sx           rs.w    1       ;8
St_Sy           rs.w    1       ;10
St_SizeOf       rs.b    0       ;12

                rsreset         ;Splinters
Sp_X            rs.w    1       ;0
Sp_Y            rs.w    1       ;2
Sp_Pos          rs.l    1       ;4
Sp_DbPos        rs.l    1       ;8
Sp_Sx           rs.w    1       ;12
Sp_Sy           rs.w    1       ;14
Sp_Col          rs.b    1       ;16
Sp_BkCol        rs.b    1       ;17
Sp_DbBkCol      rs.b    1       ;18
Sp_First        rs.b    1       ;19
Sp_Fuel         rs.w    1       ;20
Sp_SizeOf       rs.b    0       ;22

                rsreset         ;Blitterqueue
Bn_Next         rs.l    1
Bn_Function     rs.l    1
Bn_Stat         rs.w    1
Bn_Dummy        rs.w    1
Bn_BeamPos      rs.w    1
Bn_CleanUp      rs.l    1
Bn_B40l         rs.l    1       ;BLTCON0&BLTCON1
Bn_B44l         rs.l    1       ;Masks
Bn_B48l         rs.l    1       ;Source Address C
Bn_B50l         rs.w    1
Bn_B52w         rs.w    1       ;Source Address A.w
Bn_B54l         rs.l    1       ;Target Address D
Bn_B58w         rs.w    1       ;BLTSIZE
Bn_B60w         rs.w    1       ;Modulo C
Bn_B62l         rs.w    1       ;Modulo B&A
Bn_B64w         rs.w    1       ;Modulo A
Bn_B66w         rs.w    1       ;Modulo D
Bn_B72w         rs.w    1       ;BLTBDAT
Bn_B74w         rs.w    1       ;BLTADAT
Bn_XPos         rs.w    1
Bn_SizeOf       rs.b    0

                rsreset         ;AMCAF Main Datazone
O_TempBuffer    rs.b    80
O_FileInfo      rs.b    260
O_Blit          rs.b    Bn_SizeOf
```

```
O_BobAdr        rs.l    1
O_BobMask       rs.l    1
O_BobWidth      rs.w    1
O_BobHeight     rs.w    1
O_BobX          rs.w    1
O_BobY          rs.w    1
O_StarBank      rs.l    1
O_StarLimits    rs.w    4
O_StarOrigin    rs.w    2
O_StarGravity   rs.w    2
O_StarAccel     rs.w    1
O_StarPlanes    rs.w    2
O_NumStars      rs.w    1
O_CoordsBank    rs.l    1
O_SpliBank      rs.l    1
O_SpliLimits    rs.w    4
O_SpliGravity   rs.w    2
O_SpliBkCol     rs.w    1
O_SpliPlanes    rs.w    1
O_SpliFuel      rs.w    1
O_NumSpli       rs.w    1
O_MaxSpli       rs.w    1
O_SBobMask      rs.w    1
O_SBobPlanes    rs.w    1
O_SBobWidth     rs.w    1
O_SBobImageMod  rs.w    1
O_SBobLsr       rs.w    1
O_SBobLsl       rs.w    1
O_SBobFirst     rs.b    1
O_SBobLast      rs.b    1
O_QRndSeed      rs.w    1
O_QRndLast      rs.w    1
O_PTCiaVbl      rs.w    1
O_PTCiaResource rs.l    1
O_PTCiaBase     rs.l    1
O_PTCiaTimer    rs.w    1
O_PTCiaOn       rs.w    1
O_PTInterrupt   rs.b    22
O_PTVblOn       rs.w    1
O_PTAddress     rs.l    1
O_PTBank        rs.l    1
O_PTSamBank     rs.l    1
O_PTTimerSpeed  rs.l    1
O_PTDataBase    rs.l    1
O_PTSamVolume   rs.w    1
O_AgaColor      rs.w    1
O_HamRed        rs.b    1
O_HamGreen      rs.b    1
O_HamBlue       rs.b    1
                rs.b    1       ;Pad
O_VecRotPosX    rs.w    1
O_VecRotPosY    rs.w    1
O_VecRotPosZ    rs.w    1
O_VecRotAngX    rs.w    1
O_VecRotAngY    rs.w    1
O_VecRotAngZ    rs.w    1
O_VecRotResX    rs.w    1
O_VecRotResY    rs.w    1
O_VecRotResZ    rs.w    1
O_VecCosSines   rs.w    6
O_VecConstants  rs.w    9
O_BlitTargetPln rs.l    1
O_BlitSourcePln rs.l    1
O_BlitTargetMod rs.w    1
O_BlitSourceMod rs.w    1
O_BlitX         rs.w    1
O_BlitY         rs.w    1
O_BlitWidth     rs.w    1
O_BlitHeight    rs.w    1
O_BlitMinTerm   rs.w    1
O_BlitSourceA   rs.l    1
O_BlitSourceB   rs.l    1
```

```
O_BlitSourceC   rs.l    1
O_BlitSourceAMd rs.w    1
O_BlitSourceBMd rs.w    1
O_BlitSourceCMd rs.w    1
O_BlitAX        rs.w    1
O_BlitAY        rs.w    1
O_BlitAWidth    rs.w    1
O_BlitAHeight   rs.w    1
O_PTileBank     rs.l    1
O_BufferAddress rs.l    1
O_BufferLength  rs.l    1
O_PowerPacker   rs.l    1
O_PPCrunchInfo  rs.l    1
O_DiskFontLib   rs.l    1
O_DirectoryLock rs.l    1
O_DateStamp     rs.l    3
O_OwnAreaInfo   rs.b    1
O_OwnTmpRas     rs.b    1
                rs.w    1       ;Pad
O_AreaInfo      rs.b    24
O_Coordsbuffer  rs.b    20*5
O_TmpRas        rs.b    8
O_FontTextAttr  rs.b    8
O_AudioPort     rs.b    32
O_AudioIO       rs.b    68
O_ChanMap       rs.w    1
O_AudioOpen     rs.w    1
O_AudioPortOpen rs.w    1
                rs.w    1       ;Pad
O_PaletteBufs   rs.w    32*8
O_ParseBuffer   rs.b    512
O_SizeOf        rs.l    0
```

Bitplanes

Let's go for a small tutorial, in which bitplanes and how they work is explained.

1. What is a bitplane?
On an Amiga computer the video picture is created by so called bitplanes. These are linear blocks in chip memory, of which every single bit represents one dot on the screen.

Using only one bitplane, you can only see two different colours (either bit set or bit clear). If more than one bitplane is placed on top of others, you will get 2^n colours for n bitplanes.

```
Example:
We have an 16 colours screen. So it has got 4 bitplanes. Normally, the
bitplanes are counted from 0 to n-1...

Binary 0 1 0 1 = Decimal 5
       | | | |    .---v---v---v---v--------------------
       | | | `----+ 1 |   |   |   | Bitplane 0
       | | |      |--v^--v^--v^--v^--v------------------
       | | `------+--+ 0 |   |   |   | Bitplane 1
       | |        |  |--v^--v^--v^--v^--v-----------------
       | `--------+--+--+ 1 |   |   |   | Bitplane 2
       |          |  |  |--v^--v^--v^--v^--v----------------
       `----------+--+--+--+ 0 |   |   |   | Bitplane 3
                  |  |  |  |---^---^---^---^---'
                  |  |  |  |
                  |  |  |
                  |  |
                  |
```

You must imagine the bitplanes to be 'overlapped over each other'. The colour index 5 is displayed in the bitplanes like that.
Which colour the dot on the screen finally has, is determined by the palette settings.

2. Overlapping and Transparency.
Imagine, you draw a figure A in bitplane 0 and a figure B in bitplane 1 only. This produces the following:

The area, on which none of the figures are displayed, has got the colour zero (both bitplanes are cleared: %00).
whereas on regions, where only figure A is appearing, the colour 2^0=1 is used (bit 0 is set, bit 1 is clear: %01).
Figure B alone sets the bit in bitplane 1 and therefore is displayed in colour 2^1=2 (=%10).
If both figures are overlapping, the bits in both bitplanes are set and this results in colour 2^0+2^1=3 (=%11).

Now we can define the palette. For instance:
Palette 0,$F00,$F0,$FF0

The background is black ($000), figure A is red ($F00) and figure B is green ($0F0), when overlapping yellow is generated ($FF0).

By just thinking about this correlation you can achieve nice effects.
Try to find out, how the two figures will look like, when using these palettes:

```
1. Palette 0,$FFF,$888,$FFF
2. Palette 0,0,0,$FFF
```

You find the solution below.

3. Glenz and Fade
When e.g a cube appears 'transparent', out of glass or electric, the effect is called 'Glenz'.
This is mostly used for vector effects. There are two major types of Glenz vectors:

a) Wire frame objects:
With Glenz wire frame objects the lines are drawn in a different bitplane per vertical plane keeping the two or three previous frames intact.
These old and new frames then overlap, and by choosing the right palette (additive colour values per bitplane) the points where the lines are overlapping look lighter and somehow glitter.

```
Example for an additive colour palette (eight coloured screen):

DARK=$333 : LIGHTER=$666 : BRIGHT=$FFF

'     %000 %001 %010   %011  %100   %101    %110   %111
Palette 0,DARK,DARK,LIGHTER,DARK,LIGHTER,LIGHTER,BRIGHT
```

Just look at the bit values and count the number of set bits to get the right colour value.

By changing the palette permanently you can also create a Motion Blur or Fade effect.
To achieve this, you must set every colour with the bit for the current bitplane, in which you are drawing at the moment, to the brightest, colour all colours with the bit of the previous bitplane and not with the bit of the current one to a middle colour and so on...

b) Solid objects:
Glenz on solid object is done like this: the polygons, that are facing away from the viewer (and therefore cannot actually be seen) are drawn on an other bitplane than the polygons that face the viewer.
By setting the colours according to the bitplanes, the object will seem to be transparent. The only thing to do is to mix the colours of the bitplanes, in which the different polygons are drawn.

4. Bitplane modes and their specialities.
The old ECS-Amigas can only display up to 6 bitplanes simultaneously. So 2^6=64 colours is the maximum (excluding the HAM mode). Though there are some special modes:

a) ExtraHalfBright (EHB).
As the OCS and ECS chipset has only 32 colour registers, the other 32 colours are displayed at half the value. When using EHB you can produce some neat shadow effects by writing into the 6th bitplane.
Note: EHB pictures cannot be faded out perfectly.

b) Hold And Modify (HAM).
HAM is a method, to decompress six bitplanes to twelve bitplanes by hardware. Therefore the colours from 16 to 63 are used reach the wanted colour by changing the red, green or blue value of the previous colour.
Very good for static pictures and pre calculated animation but useless for games and realtime graphics. The only sensible way to display moving objects on a HAM screen is to use sprites.

c) Dual Playfield.
The Amiga chipset can display all even bitplanes (0,2,4) separated from the odd ones (1,3,5).
It's better to say, he puts the one playfield on top of the other by using colour 0 as 'window' to the other playfield.
In this mode each playfield can have 2^3=8 colours.

The chipset has got separate control registers for even and odd bitplanes each, as each Playfield must be independent for Dual Playfield mode.

BitPLane CONtrol (BPLCON0) $100.
Here you determine the number of bitplanes and the resolution and can toggle the special modes.
Bit table:

```
15      HIRES           Toggle hires mode
14-12   BPUx            Number of bitplanes
11      HOMOD           Toggle HoldAndModify
10      DBPLF           Toggle Dual Playfield
 9      COLOR           Toggle colour burst output
 8      GAUD            Use audio input from a genlock
 7      8BPL            8 bitplanes (AGA)
 6      SHIRES          Superhires (ECS/AGA)
 3      LPEN            Activate lightpen
 2      LACE            Enable interlace mode
 1      ERSY            Switch to external synchronization
```

The scroll register (BPLCON1) $102.
By using this register, the screen can be scrolled to the left by up to 15 pixels. The bits 0-3 are used for the even bitplanes, the bits 4-7 for the odd ones.

Modulo registers (BPL1MOD/BPL2MOD) $108/$10A.
These registers set the amount of bytes to be added to the memory of the bitplanes after each rasterline. This is utilized by playfields that are bigger than the visible area.
When writing a negative value you can achieve vertical zoomers or mirror effects.

If you want to alter a register using Set Rain Colour, you can calculate the new 'colour' with the following formula:
(REGADR-$180)/2

Note: Enabling 5 to 6 bitplanes in low resolution or 3 to 4 bitplanes in high resolution will cost free processor time, even if you only display the screen.
This is true when the running program is placed in chip ram or must access chip ram.

5. How can I access all these effects?
Simple: Define a rainbow, call Set Rain Colour and enter the new values for the registers using Rain() instead of supporting the colours.
The only limitation: Due to the AMOS rainbow restrictions you can only manipulate ONE (!) register per rasterline.

Look carefully at the example programs. These demonstrate every single effect mentioned here.
By the way: You can get the pointer to the single bitplanes using Logbase(planenr) and Phybase(planenr).

Solution to the questions in 2:

1. Palette 0,$FFF,$888,$FFF
Figure A is white and is moving 'over' figure B, which is grey, because if they overlap the colour white is created.

2. Palette 0,0,0,$FFF
Fig A and Fig B are invisible if they dont overlap, otherwise colour 3 is created which was set to $FFF (white).

Command Index

Functions:

09.01.03 =Odd - Returns, if a number is odd
04.02.02 =Object Blocks - Returns the length of a file in blocks
04.02.02 =Object Comment$ - Gives back the filenote of an Object
04.02.02 =Object Date - Returns the date of creation of an Object
04.02.01 =Object Name$ - Returns the name of an Object
04.02.02 =Object Protection - Returns the Protection flags of an Object
04.03.01 =Object Protection$- Returns a Protection flags string
04.02.02 =Object Size - Gives back the length of a file
04.02.02 =Object Time - Returns the time of creation of an Object
04.02.02 =Object Type - Returns the type of an Object
03.04.02 =Pal Get - Reads a saved palette entry
04.03.01 =Path$ - Returns the directory of a full path
04.03.02 =Pattern Match - Compares a string with a certain pattern
06.01.02 =Pfire - Check if fire button is pressed
06.01.01 =Pjdown - Check if joystick is pressed down
06.01.01 =Pjleft - Check if joystick is pressed left
06.01.01 =Pjoy - Acquire direction of a joystick
06.01.01 =Pjright - Check if joystick is pressed right
06.01.01 =Pjup - Check if joystick is pressed up
08.01.05 =Pt Cinstr - Returns the current instrument being played
08.01.05 =Pt Cnote - Gets the frequency of the current instrument
08.01.04 =Pt Cpattern - Gets the current song position
08.01.04 =Pt Cpos - Returns the current pattern line
08.01.05 =Pt Data Base - Gets the address of the PT-Database
08.01.05 =Pt Instr Address - Returns the address of an instrument
08.01.05 =Pt Instr Length - Returns the length of an instrument
08.01.04 =Pt Signal - Checking for signals from the music
08.01.04 =Pt Vu - Returns the current Vumeter value
09.01.01 =Qarc - Fast arc function
09.01.02 =Qcos - Fast cosine function
09.01.02 =Qrnd - Fast replacement for Rnd
09.01.02 =Qsin - Fast sine function
09.01.02 =Qsqr - Fast replacement for Sqr
03.04.02 =Red Val - Calculates the red value of a colour
09.01.04 =Replacestr$ - Replaces a string with another one
03.04.03 =Rgb To Rrggbb - Converts a ECS-colour into AGA colour format
03.04.03 =Rrggbb To Rgb - Converts a AGA colour into a ECS-colour value
10.01.02 =Scanstr$ - Returns the name of a key
03.06.01 =Scrn Bitmap - Returns the screen bitmap address
03.06.01 =Scrn Layer - Returns the screen layer address
03.06.01 =Scrn Layerinfo - Returns the screen layerinfo address
03.06.01 =Scrn Rastport - Returns the screen rastport address
03.06.01 =Scrn Region - Returns the screen region address
10.01.03 =Sdeek - Deeking a signed word
06.02.01 =Smouse Key - Checks the mouse key
10.01.03 =Speek - Peeking a signed byte
03.02.07 =Splinters Active - Returns how many splinters are still moving
10.01.02 =Tool Types$ - Reads the Tool Types of an icon
03.01.03 =Turbo Point - Fast replacement for Point
09.01.02 =Vclip - Restricts a value to a given range
07.01.01 =Vec Rot X - Calculates the 2D X value
07.01.02 =Vec Rot Y - Determines Y value
07.01.02 =Vec Rot Z - Returns the Z coordinate
09.01.03 =Vin - Tests, if the value is within a range
09.01.03 =Vmod - Does a modulo operation on a value
09.01.01 =Wordswap - Swapping the upper and lower 16 bits
03.01.03 =X Raster - Gets the X position of the raster beam
06.02.01 =X Smouse - Reads the x coordinate
06.01.02 =Xfire - Reads out the fire buttons on a game pad
03.01.03 =Y Raster - Gets the Y position of the raster beam
06.02.01 =Y Smouse - Reads the y coordinate of the mouse

Commands:

03.04.02 Amcaf Aga Notation - Toggle AGA-Amiga colour format
10.01.01 Audio Free - Frees the audio device
10.01.01 Audio Lock - Reserves the audio device
02.01.02 Bank Code Add.y - Additional algorithm encoding
02.01.03 Bank Code Mix.y - Mix between Add and Xor
02.01.03 Bank Code Rol.y - Rotation to the left
02.01.03 Bank Code Ror.y - Rotation to the right
02.01.03 Bank Code Xor.y - Xor algorithm encoding

02.01.02 Bank Copy - Copies a bank
02.01.04 Bank Delta Decode - Removes the delta-encoding of a bank
02.01.03 Bank Delta Encode - Prepares a bank with delta-encoding
02.01.02 Bank Name - Changes the name of a bank
02.01.01 Bank Permanent - Makes a bank Permanent
02.01.02 Bank Stretch - Extends a bank after it has been reserved
02.01.01 Bank Temporary - Makes a bank Temporary
02.01.02 Bank To Chip - Moves a bank into Chip ram
02.01.01 Bank To Fast - Moves a bank into Fast ram
03.05.03 Bcircle - Drawing a circle to fill it using the blitter
03.05.02 Blitter Clear - Clearing a bitplane with the help of the blitter
03.05.01 Blitter Copy - Copying and modifying a bitplane
03.05.02 Blitter Copy Limit - Setting the Blitter Copy area
03.05.02 Blitter Fill - Filling polygons using the blitter
03.05.02 Blitter Wait - Waiting for the blitter has finished his task
03.01.01 Bzoom - Zooms a region
03.03.01 Change Bank Font - Setting the screen font using a font bank
03.03.01 Change Font - Loading a font directly from disk
03.03.02 Change Print Font - Changing the font that is used by Print.
03.01.01 Convert Grey - Creates a grey scale picture
03.02.05 Coords Bank - Reserves a bank to store the coordinates
03.02.05 Coords Read - Reading the coordinates into a bank
04.01.01 Dload - Loads a file Permanently
04.01.01 Dsave - Saves a file to disk
04.02.01 Examine Dir - Inits the reading of a drawer
04.02.01 Examine Object - Gets all information about an Object
04.02.01 Examine Stop - Stops the reading of a directory
10.01.01 Exchange Bob - Swaps the two images in the sprite bank
10.01.01 Exchange Icon - Swaps the two images in the icon bank
11.01.01 Extdefault - Calls the default routine of an extension
11.01.01 Extreinit - Tries to revoke a extension
11.01.01 Extremove - Removes a extension from memory
03.01.02 Fcircle - Draws a filled circle
03.01.02 Fellipse - Draws a filled ellipse
04.01.01 File Copy - Copies a file
10.01.02 Flush Libs - Frees as much as possible memory
03.01.01 Ham Fade Out - Fades out a ham picture
04.04.01 Imploder Load - Loads and decrunches a FileImploder file
04.04.02 Imploder Unpack - Decrunches a FileImploder bank
04.01.02 Launch - Starts a new process
06.02.01 Limit Smouse - Defines the movement region
03.03.01 Make Bank Font - Creating a font bank
03.02.02 Make Pix Mask - Picks up a mask for the shifting process
03.02.01 Mask Copy - Screen Copy with a mask
10.01.02 Nop - No effect
10.01.01 Open Workbench - Reopens the workbench again
03.04.01 Pal Get Screen - Saves the palette of a screen
03.04.02 Pal Set - Changes an entry of a saved palette
03.04.01 Pal Set Screen - Sets the palette of a screen
03.04.01 Pal Spread - Spreads from one colour entry to another
03.02.02 Pix Brighten - Increase colour indexes (not cyclic)
03.02.02 Pix Darken - Decrease colour indexes (not cyclic)
03.02.01 Pix Shift Down - Decrease colour indexes (cyclic)
03.02.01 Pix Shift Up - Increase colour indexes (cyclic)
04.04.01 Ppfromdisk - Loads and unpacks a powerpacked file
04.04.01 Pptodisk - Packs and saves a file as PP20
04.04.01 Ppunpack - Unpacks a powerpacked file
04.01.01 Protect Object - Modifies the Protection bits of an Object
08.01.02 Pt Bank - Sets the bank for the use with Pt Instr Play
08.01.02 Pt Cia Speed - Changing the replaying speed
08.01.01 Pt Continue - Restarts a previously stopped music
08.01.03 Pt Instr Play - Plays an instrument of a protracker module
08.01.01 Pt Play - Replays a module
08.01.02 Pt Raw Play - Plays a chunk of memory as sound sample
08.01.03 Pt Sam Bank - Sets the bank to use with AMOS samples
08.01.04 Pt Sam Freq - Changes the replaying speed of a sample
08.01.03 Pt Sam Play - Replays a sample from an AMOS Sam Bank
08.01.03 Pt Sam Stop - Stops the sfx on specific audio channels
08.01.03 Pt Sam Volume - Sets the volume of a sound effect
08.01.01 Pt Stop - Stops the current music
08.01.02 Pt Voice - Toggling the audio channels
08.01.02 Pt Volume - Setting the volume of the music

```
03.04.01 Rain Fade          - Fades a rainbow out or to another one
03.01.02 Raster Wait        - Waits for a specific raster position
10.01.02 Reset Computer     - Resets your computer
03.06.01 Set Ntsc           - Switches to the 60Hz NTSC mode
04.01.02 Set Object Comment - Sets the filenote of an Object
04.01.02 Set Object Date    - Sets the date of an Object
03.06.01 Set Pal            - Switches back to 50Hz PAL mode
03.04.01 Set Rain Colour    - Changes the affecting colour of a rainbow
03.06.01 Set Sprite Priority- Changes the sprite priority in Dual playfield mode
03.02.03 Shade Bob Down     - Places a Shade Bob that decreases the colours
03.02.02 Shade Bob Mask     - Determinate the image to use for the bobs
03.02.03 Shade Bob Planes   - Setting the number of bitplanes to use
03.02.03 Shade Bob Up       - Places a Shade Bob that increases the colours
03.02.02 Shade Pix          - Plots a shade pixel
06.02.01 Smouse Speed       - Sets the speed of the mouse
06.02.01 Smouse X           - Sets the x coordinate of the mouse
06.02.01 Smouse Y           - Sets the y coordinate
03.02.07 Splinters Back     - Gets the background pixels
03.02.05 Splinters Bank     - Reserves memory for the splinters
03.02.05 Splinters Colour   - Sets the to-use colours
03.02.06 Splinters Del      - Clears the splinters
03.02.06 Splinters Do       - Do a complete drawing process
03.02.07 Splinters Draw     - Draws the splinters to the screen
03.02.06 Splinters Fuel     - Sets the range of a splinter
03.02.06 Splinters Gravity  - Changes the gravity
03.02.06 Splinters Init     - Initialises the splinters bank
03.02.06 Splinters Limit    - Changes the limits of the splinters
03.02.06 Splinters Max      - Sets the maximum of new appearing splinters
03.02.07 Splinters Move     - Moves the splinters
03.02.04 Td Stars Accelerate- Toggles the acceleration
03.02.03 Td Stars Bank      - Reserves some memory for the stars
03.02.04 Td Stars Del       - Clears the stars from the screen
03.02.04 Td Stars Do        - Does a complete drawing process
03.02.05 Td Stars Draw      - Draws the stars
03.02.04 Td Stars Gravity   - Determines the gravity force
03.02.04 Td Stars Init      - Inits the stars
03.02.03 Td Stars Limit     - Sets the limits of the stars
03.02.05 Td Stars Move      - Moves the stars
03.02.04 Td Stars Origin    - Places the origin of the stars
03.02.03 Td Stars Planes    - Selects the planes to be used for the stars
03.01.02 Turbo Draw         - Fast replacement for Draw
03.01.02 Turbo Plot         - Fast replacement for Plot
07.01.01 Vec Rot Angles     - Sets the viewing angles
07.01.01 Vec Rot Pos        - Positions the camera
07.01.01 Vec Rot Precalc    - Calculates the precalc matrix
04.01.01 Wload              - Loads a file Temporarily
10.01.02 Write Cli          - Writes something into the cli window
04.01.01 Wsave              - Saves a file to disk
```