FBLITGUIDE

Stephen Brookes

COLLABORATORS			
	TITLE : FBLITGUIDE		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Stephen Brookes	January 20, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

FBLITGUIDE

1.1 FBlit

```
FBlit v3.66

 Legal  Stuff
 Distribution
  What It Is
 Requirements
 Installation
    Usage

   Problems
 Who Done It?
   History

   FBlitGUI
  Task Lists
 The  Patches
```

1.2 Legal Type Stuff

```
 FBlit, FBlitGUI and fblit.library are © Stephen Brookes 1997 - 2000

 The software contained in this archive is incomplete (beta),
experimental in nature and fundamentally dangerous, so don't use it. I
will not accept responsibility for any undesirable effects resulting
from the use or abuse of any software or information contained in this
archive.
```

1.3 Distribution

This archive is freely distributable.

The latest version of FBlit is currently available from...

<http://www.tpec.u-net.com>

Please make sure you have the latest version from that address before reporting bugs.

1.4 What It Is

What is it?

FBlit is a hack that should reduce the amount of Chip RAM used by OS based applications on Amigas with native (non-gfx card) graphics. It also may speed up some things, and help reduce colour flicker.

Yes, but what is it?

FBlit is a collection of patches that allow the OS to deal with graphics data outside of Chip memory.

This is not normally possible because the OS rendering functions use the blitter hardware, and for that to work the data must be within the addressable range of the blitter (ie. Chip memory). So FBlit replaces the OS rendering functions that may use the blitter, with equivalent functions which use the CPU instead, and hence can operate in any memory.

Why on earth would you want to do that?

Basically, 2MB is not enough Chip memory for a modern Amiga (no sniggering please).

The memory used for screens must be in Chip RAM, so that the video display hardware can get at it, but all other graphics data is only kept in Chip RAM so that the OS can use it's blitter assisted rendering functions. It is for this 'other' (non-screen) graphics data that the 2MB limit has become a real problem. A single JPEG picture, when decoded, may be well over 2MB in size, making it impossible to display, even in a window on an existing screen. Or, if you run NewIcons, it can be impossible to view the contents of a large directory, since you may run out of Chip RAM before all the necessary icons have been created. HTML browsers also tend to rapidly use up all available Chip memory etc. etc.

Hey, I'm ELITE! What is it really?

Ah... Hmmm..... Well, it consists of several parts.

The BltBitMap(), BitMapScale() and BltClear() patches are responsible for letting the OS use discrete, non-displayable bitmaps outside Chip RAM. This is enough to allow most bitmaps, intuition images, superbitmaps etc. to be stored in Fast RAM and is arguably the most useful part of FBlit.

In order for the OS to use rastport/bitmaps (and that means most of the actual graphics rendering functions like Text(), RectFill(), Draw() etc.) outside the Chip range, some more functions needed to be replaced ie. BltTemplate(), BltPattern(), SetRast() and Draw(). This is not quite enough to entirely stop the OS using the blitter for rendering. It will still be used by Area...() functions for instance, but in that case the blitter is only used on TmpRas, and not on any actual bitmaps. These functions (AreaEnd() and Flood()) are patched by FBlit, so that fast RAM TmpRas can be dealt with, but it is important to note that the blitter is in fact still used for these operations!

Being able to use rastport/bitmaps outside Chip memory is not really of any great benefit in itself, since usually a rastport's bitmap will simply be the screen's bitmap on which the rastport appears, and therefore must be in Chip memory. It became necessary as a result of the next part of FBlit. When all bitmaps are allocated with planes in Fast RAM, temporary bitmaps allocated for overlapped window regions will also go in Fast RAM, so parts of rastports can end up outside Chip memory under certain circumstances.

Having the OS able to use Fast RAM for non-displayable bitmaps is all very nice, and maybe useful for programmers who know about it and want to take advantage of it, but it makes no difference to existing software. Also, programmers who wanted to use this option would need to allocate their own custom bitmaps with planes in Fast RAM, which is not exactly 'OS friendly', so many would probably not want to do it. In order to force software to use bitmaps in Fast RAM, AllocBitMap() is patched, so that bitmaps allocated without the #BMF_DISPLAYABLE flag may be created in Fast memory. This behaviour can be adjusted on a per task basis, so that things which don't like it can still get bitmaps with Chip planes.

There are also several more patches to deal with a some specific issues/problems.

AddBob() and RemIBob() may be patched so that any BOB images in Fast RAM are copied back to Chip on the fly. This is designed for use with NewIcons RTG option, to allow dragging of the RTG icons.

QBSBlit() may be patched so that calls from DrawGList() are run on an emulator instead of on the blitter hardware. This is mainly aimed at OS3.5 icons.

OpenScreenTagList() can be patched so that attempts to open screens with bitmaps that contain non-Chip planes may be dealt with. This can happen when tasks that have been forced to use Fast RAM bitmaps try to open a screen using a bitmap that was not allocated with #BMF_DISPLAYABLE.

1.5 Requirements

Minimum requirements for running FBlit are:

 68020 processor
 v39/OS3.0
 Some Fast RAM

 In addition, FBlitGUI requires MUI.

 AGA graphics are recommended, but FBlit should also work on OCS/ECS chipsets.

 It is also recommended that you do not use FBlit if you have a graphics card, unless you are very clear about what you are doing! FBlit is, in some ways, rather similar to RTG software (CGX/P96), which makes attempting to run both FBlit, and a true RTG system, problematic at best.

1.6 Installation

Installation

 Copy 'FBlit' and 'FBlitGUI' to 'C:'.
 Copy 'fblit.library' to 'Libs:'.

 Alternatively, move the FBlit drawer to wherever you like.

 Add the following line to your 'S:startup-sequence' at some point after 'ENV:' has been assigned. The line just before 'BindDrivers' is usually a good bet.

 C:FBlit

 (or <path to FBlit drawer>FBlit)

 If you are replacing an older FBlit installation and want to use the latest default configuration you should delete 'ENVARC:fblit.cfg' before rebooting. (Note: this is currently the only way to restore the default configuration!)

Installation Related Problems

 It is most important that FBlit is not launched with 'run' (or any equivalent). If it is 'run', the patches will not be installed at the point in the startup-sequence where FBlit was launched, and the order relative to other patches cannot be maintained.

 FBlit should be launched before anything else that may patch graphics related functions eg. MCP, NewIcons, CGX etc. otherwise the effects created by these things will be cancelled out by FBlit's patches.

Boot picture programmes also tend to patch graphics functions, and it may be necessary to install 'PatchControl' (or equiv') if you want to have FBlit launched after your bootpic software.

Mixing the components of FBlit (GUI, library) from different archives is not recommended since they are often incompatible. This can also apply to the configuration file ('ENVARC:fblit.cfg'). Although all versions of FBlit are able to read and (more or less) understand .cfg files from any other version, the actual configuration described may not be compatible. This is only really dangerous when the configuration is newer than the FBlit (ie. V3 configuration on V2 FBlit), but problems can arrise going the other way eg. configurations for older versions may have FDraw set to discard fast data (correctly), while in newer versions it should be set to process.

Renaming parts of FBlit, including 'FBlit' itself, is not a good idea and can give confusing results.

Currently, FBlit will not generate any form of error output. If it fails to install, it will do so silently.

1.7 Usage

Safe(ish) Stuff

FBlit's default configuration is probably good enough for most people, and I don't advise you to try and change it unless you really understand exactly what FBlit is doing and the implications of that. However, there are a few changes you may want to make to the rest of your system.

NewIcons

Once FBlit is installed and running, you can safely switch NewIcons into RTG mode (see NewIcons prefs/docs). Please don't try this while FBlit is not actually running, or dragging icons will cause memory corruption!

OS3.5 colour icons

OS3.5 can also use RTG icons and these will work with FBlit, but a little more effort is required. You will need an up to date copy of Stephan Rupprecht's 'WBCtrl' programme (see Aminet).

Towards the end of your S:startup-sequence file, find the line that reads 'LoadWB'. Change this line too...

```
LoadWB SIMPLEGELS
```

...and on the line before it, enter the command...

```
WBCtrl IMT=ICONFAST
```

Note that with this configuration you will no longer get the transparent effect for dragged icons.

MCP

If you use MCP you should disable QuickDraw and enable QuickLayers. Neither of these steps is necessary, but this is currently the best configuration for operation with FBlit.

WorkBench/Multiview/DataTypes

While running FBlit, you should find that WorkBench/window backdrop images use no Chip memory, so you can go completely mad with these if you like (Fast RAM permitting). The same is true for Multiview, if you use it to display in a window rather than on a screen. In fact, anything that uses datatypes may benefit from this, but note that not all datatypes can use Fast RAM. Most commonly, ilbm.datatype (<v43) will always use Chip memory, so you may wish to change your ilbm.datatype, or use other image formats for backdrop pictures.

Screens/colours

You might also find that you can comfortably increase the number of colours on your WB screen by a notch or two. If you use PAL/NTSC (15kHz) screenmodes, 256 colours can be quite usable. If you are stuck with 30kHz modes (dblPAL etc.), things will still grind to a halt at about 64 colours.

Screens will use just as much Chip memory as before, so if you open a lot of screens you will still run out of Chip RAM pretty quick. But, since you can probably use a more colourful WB screen, it will be more practical to run some applications on there than to have everything opening their own screens.

FText

Rick Pratt's FText patch is available from Aminet. FText can significantly speed up text rendering by causing it to be generated in Fast RAM instead of Chip memory.

Not So Safe Stuff

FBlitGUI allows you to change FBlit's configuration. It also allows you to create random memory corruption, make your system unstable, destroy the contents of your hard drives.... So playing with it is not really recommended, but for the people who will play anyway, here are the most (perhaps, 'only') useful changes you might want to make.

Include/Exclude mode

By default FBlit runs in 'include' mode. You may want to add more tasks to the include list. Or with current versions of FBlit, it is often better to use 'exclude' mode instead.
(see Task Lists)

OS3.5 colour icons (again)

There is an alternative method for supporting OS3.5 RTG icons which allows you to keep the transparent drag effect at the cost of some speed. Using FBlitGUI...

install and activate 'QBSBlitPatch'
un-install both 'AddBobPatch' and 'RemIBobPatch'

Now you can safely remove the 'SIMPLEGELS' switch (if it exists) from the 'LoadWB' command in your S:startup-sequence.

The 'WBCtrl IMT=ICONFAST' command is still required to activate OS3.5 RTG icons. You may also try using 'WBCtrl IMT=FAST' instead, which will speed up icon dragging somewhat, but it is not yet clear wether this is entirely safe with current versions of FBlit.

Note that this configuration will also work with NewIcons RTG, but in that case offers no real benefit over the default config. It also may have some unwanted side effects, since the BOBs are actually rendered by the CPU on interrupt time. This has not caused any major problems as far as I know, but it is not very 'nice' and QBSBlitPatch may well go away again if I can come up with a neater solution based on the Add/RemIBobPatch approach.

If you decide to go back to the original configuration, remember to replace the 'SIMPLEGELS' switch, and make sure that WBCtrl is using 'IMT=ICONFAST' (not 'IMT=FAST'), otherwise various bad things are likely to happen (namely, a Chip RAM leak, and random Chip memory corruption).

Icons In General

To be clear (ha!), there are basically three possible FBlit configurations relevant to dragged icons/BOBs.

AddBobPatch/RemIBobPatch installed, QBSBlitPatch un-installed. This is the default. It allows RTG NewIcons, and RTG OS3.5 icons (provided the LoadWB SIMPLEGELS switch is used, and that only icon.library (and not workbench.library) are in RTG mode). It will also work perfectly well if no RTG icons are in use.

QBSBlitPatch installed, AddBobPatch/RemIBobPatch un-installed. This cfg again allows RTG NewIcons and RTG OS3.5 icons (probably without any conditions), but rendering will be slower. And again, this will work fine with no RTG icons.

AddBobPatch/RemIBobPatch and QBSBlitPatch un-installed. In this configuration RTG icons can not be dragged safely, but in the interests of having as few patches as possible, you might use this configuration if you don't have OS3.5 or NewIcons (or if you simply don't want to use RTG icons(?!)).

No other possible combination of these three patches is useful.

Speeed

By the time you get all the way down here, there actually isn't much you can do to speed things up any more. But, depending on your hardware, you may find that setting FBltTemplate and FBltPattern to process all Chip data will speed up some rendering operations.

1.8 Misc Problems

General
Most problems with FBlit are caused by incorrect installations, or interactions with other patches (ie. MCP, VisualPrefs). The effects caused by this are many and various, missing graphics, missing window gadgets, corruption of the WorkBench screen, total mayhem etc. etc.
(see Installation Problems)

Lines
Line related corruption is perhaps the next most common problem. This is simply caused by the incomplete nature of the Draw() patch and is safe, but not very nice.
(see FDraw 'Notes' for more on this one)

Birdie/Stack
FBlit's patches are likely to increase the stack requirements of some OS functions and this can cause trouble. The combination of FBlit and Birdie can be enough to push some software over the edge, and you may need to use 'StackAttack' (etc.) to deal with this.

Wordworth7
WW7 may fail to display text if it's bitmaps are promoted to Fast RAM. This can be cured by setting the tooltype 'PICASSO=TRUE' in the 'Wordworth' icon.

This and that still use Chip memory
Even if you run FBlit in exclude mode, some programmes will still use Chip memory unnecessarily. There are several reasons for this, and nothing much can really be done about it. Programmes may build their own bitmaps in Chip memory, or they may include graphics within Chip hunks of their exe files etc. etc. However, if the programme in question uses datatypes, it may be that you have a datatype (eg. v39 ilbm.datatype) which will always use Chip memory under FBlit.

Also remember that Chip RAM may be used for things other than graphics ie. audio buffers.

OS3.5 Chip memory leaks
If you try to use OS3.5's RTG icon option with FBlit's default configuration, you will get a Chip memory leak unless the 'SIMPLEGELS' switch is specified for LoadWB.
(see Usage)

Note that OS3.5 contains it's own unrelated Chip memory leaks, at least up to (and including) BoingBag#1.

128 Colour Screens
There seems to be something a little strange about 128 colour screens, at least on some 1200s. Graphics may become corrupt. I don't know what

causes this, and FBlit is not required to make it happen. It will happen
on a clean OS installation, so it appears to be either an OS bug, or a
hardware issue.

1.9 Who Done It?

Currently, all code, design and implementation, is by Stephen Brookes

Contact: sbrookes@tpec.u-net.com

My thanks to the following people...

Artur Chlebek for Polish documentation (assisted by Mikolaj Calusinski
and Przemyslaw Gruchala).

Evan Tuer for the original MWB icon.

Phil Vedovatti and Luca Longone for the NewIcons icons(?).

And yet more folk, for various testing, bug hunting, support,
encouragement etc.

Luca Longone
Rick Pratt
Ian Greenway
Przemyslaw Gruchala
Piotr Powlow
Marco De Vitis
Artur Chlebek
Jess Sosnoski
Matt Sealey
Evan Tuer
Gary Colville
James L Boyd
Colin Wenzel
Iain Barclay
Oliver Borrmann

And everybody else who has mailed me about FBlit...

1.10 History

Changes from v2.63

3.66

FAllocBitMap
- bitmap creation could fail when given corrupt parameters, possibly trashing random memory etc.

FBlit
- should now refuse to start up if OS <v39, or if there is no free fast RAM available, or the processor is <68020, instead of crashing (thanks to Mikolaj Calusinski)

3.64

FAreaEnd
- new(old) patch to deal with possible fast RAM TmpRas

FFlood
- same as FAreaEnd

FAllocBitMap
- has been extended, and now allocates bitmaps itself. Mostly this is just to allow the removal of FAllocMem, but it also should be more compatible with virtual memory systems.
- an option was added to select the 'type' of memory used for bitmap's planes
- a nice silly/dangerous/useless option was added to allow the creation of displayable bitmaps in fast memory.

FAllocMem
- has gone, since changes to FAllocBitMap make it redundant

FBltBitMap
- direct interleaved support is back again (by popular demand (of Albin))
- a stupid bug was found, this one's old, and would have caused serious corruption problems (rare/intermittant) for inverted copy operations.
- the non-aligned copy routine has been redesigned to be ~100% efficient re. memory accesses on 32bit systems (~33% fewer accesses)

FBlit.guide
- fixed broken link (thanks to Artur)

3.56

FBltBitMap
- is in the middle of a major over-haul...
- the 'non-pretty' copy routines have gone. All operations are now done by the 'pretty' copy routine (ie. it is in 'Always Pretty' mode all the time).
- the copy function has a new routine to deal specifically with small operations (<33bits).
- stack check option has gone

```
  - direct support for interleaved bitmaps has gone(!)

 FAreaEnd
  - has gone, due to being redundant (hopefully), but may be back in a
    partially functional form at some point...

 QBSBlitPatch
  - a new patch to support BOBs. This is an equivalent for AddBobPatch
    and RemIBobPatch that may be useful for OS3.5 RTG icons. It is also
    functionally identical to FDrawGList, and the old blitter emulator
    has returned to support it.



3.51

 FBltBitMap
  - a bug was found and killed. When in 'always pretty' mode, some small
    operations (<33bits) could be rendered one lw to the right of the
    correct position.
  - added a stack check option

 FAllocMem
  - changed so that it can (hopefully) be applied safely over
    muguardianangel(MGA). Note however that if you launch MGA after
    FBlit is installed, all bitmap promotion will be disabled! You can
    restart promotion by uninstalling and reinstalling the FAllocMem
    patch (this will require 'PatchControl' or equiv').

 FBltTemplate
  - ooops! Fixed a nasty bug. When set to process both chip and fast
    data, FBltTemplate would fail to lock layers.

 FBltPattern
  - is completed, finally. (this also cures the slight pattern alignment
    error under certain circumstances, in the old version)
  - as with FBltTemplate, if set to process both chip and fast data, it
    will not waste time classifying the data and stats will therefore be
    invalid for this configuration.



3.47

 FBlit
  - RAM classification method changed, same for fblit.lib

 FAllocBitMap
  - clears #BMF_STANDARD (again). Note: this was irrelevant since this
    flag is redundant in the bitmap structure. It is only a return
    value from GetBitMapAttr().

 FBltPattern
  - ignores fast mem rp_AreaPtrn (yet again)
  - a dangerous bug was killed (thanks to Luca)

 FDrawGList
```

```
  - gone

 AddBobPatch
  - new patch to move fast BOB images to chip

 RemIBobPatch
  - cleans up after AddBobPatch
```

3.40

```
 FBlit
  - fblit.library can now be stored in the same directory as FBlit

 FBlitGUI
  - removed a bug associated with large task lists (thanks to Luca)

 FAreaEnd
  - fixed bubble help ;)
  - parameters for discard have changed, now only calls with a fast
    ram TmpRas will be discarded (which should be none at all)

 OSTLPatch
  - a new patch (with catchy name) to address multiview/datatype
    problems
```

3.36

```
 FBlit
  - removed Enforcer hits generated when FBlit couldn't open libraries
    (thanks to Marco)

 FBlitGUI
  - removed Enforcer hits generated when FBlit was launched with patches
    uninstalled (also thanks to Marco)

 FBltTemplate
  - completely rewritten (now uses fblit.library instead of BlitEm)
  - if configured to process both chip and fast data, the patch no
    longer wastes time classifying the data. Stats will therefore be
    invalid in this configuration!

 FBltPattern
  - partly rewritten to use the lib... not completed yet

 FAreaEnd
  - was trashing the function's return value (thanks to Luca)

 FAllocMem
  - rewritten for no good reason. Stats are completely removed!

 FText
  - has been removed, and (probably) won't be back.
```

```
3.32

  - everything changes
  - the path bug has been fixed again. FBlit would still hang if
    launched from the root of a device, or from a device with a space
    ('Ram Disk:')

 FDraw
  - is partially operational. No patterned or complement lines yet
```

1.11 FBlitGUI

```
General

 FBlit's GUI can be launched from WorkBench by double clicking the FBlit
icon, or from CLI by calling FBlit a second time (ie. 'c:FBlit').
Invoking FBlitGUI directly may also work, but is not recommended. Note
that the GUI can only be used while FBlit itself is running.

 The GUI presented appears quite similar to many MUI based prefs editors
but there are several important differences!

 Nearly all changes made to the configuration are applied immediately,
rather than through a 'Test' gadget.

 The 'Quit' gadget refers to FBlit itself, not the GUI. It is no longer
possible to quit FBlit safely so you should never use this gadget (and I
should remove it).

 There is no menu, and no easy way to restore the default configuration.
Currently this can only be achieved by deleting 'ENVARC:fblit.cfg' and
re-booting.

 The other major gadgets function as you might expect...

 'Save' will store the cfg permanently. The current, and future
instances of FBlit will use this configuration.

 'Use' will not store the cfg. Only the current instance of FBlit will
use this configuration.

 'Cancel' will revert to the configuration present when the GUI was
launched.

 The window's close gadget is equivalent to 'Cancel'.

 Most gadgets will display 'help' bubbles, if the pointer rests over
them long enough, and you haven't disabled this feature. Wether they are
really helpful or not is another matter.
```

Danger!

I've said it often enough maybe, but here we go again...

Using the GUI, it is easily possible to configure FBlit so that random Chip memory corruption will occur. This is very dangerous, not least because it may not be obvious that it is happening. Such corruption will eventually bring down the system, but also has the potential to corrupt data files that are later written to disk, perhaps losing your hard work, and under certain conditions may corrupt the disk file system itself!!

This corruption will occur if the original OS functions that FBlit patches are presented with data structures outside Chip memory. That is why FBlit patches these functions in the first place.

So you should definitely avoid either un-installing or deactivating the patches, unless these docs say otherwise. Also, do not change the 'Fast Data Options' of the patches, these should always be set to 'Process' (or 'Discard', if no 'Process' option is available).

1.12 Task Lists

FBlit uses lists of task names to decide which programmes will be forced to use bitmaps in Fast memory. If you decide you want more programmes to use Fast RAM for bitmaps, there are basically two approaches.
(see FAllocBitMap for GUI related details)

Include Mode
This is the default mode, and will force only those tasks named on the include task list to use Fast RAM bitmaps. In this mode the exclude task list is irrelevant. The down-side of include mode is that you may end up with enormous task lists if you add your own tasks, and you are never likely to catch all the programmes that could be safely promoted.

Exclude Mode
In this mode, all tasks will be forced to use Fast RAM bitmaps except those named on the exclude task list. The include task list is irrelevant in this mode. This is inherently more dangerous than include mode, since software that cannot be promoted safely may bring down your system before you know about it!

Adding a Task
Adding tasks in include mode, or using exclude mode, is potentially a dangerous thing. Some programmes cannot safely use bitmaps outside Chip memory, and if you promote such software it may corrupt memory! Mainly, these are likely to be older, more or less OS unfriendly programmes that try to use the Amiga's hardware directly on bitmaps, but there are other

possible problems that may catch out newer, and OS friendly software (even assuming FBlit itself is bug free) eg. using memory allocated as a bitmap for other purposes (assuming it will be Chip memory), or placing some significance on the current amount of free Chip memory.

Whichever mode you choose to run in, the process of adding a task is the same.

Simply typing the names of programmes into the string gadget is not the best tactic for a couple of reasons. Firstly, the 'task names' which FBlit recognizes are the names used by actual task/process structures, and these are not necessarily the same as (or even at all similar to) the names of programmes themselves. Secondly, FBlit can only effect tasks which call AllocBitMap() requesting a non-displayable bitmap, and many tasks will not do that themselves.

To get around these problems, FAllocBitMap has a 'Task Logging' option. If this is enabled, the names of tasks which can usefully be added to the task lists will be stored in the task log when they call AllocBitMap(). You can then select them from the drop down menu on the lists pages of the GUI. Tasks which cannot be made to appear on the task log also cannot be promoted to Fast RAM, and having such tasks in the lists simply wastes time. Note however, that tasks which are already on the currently active task list will not appear in the log!

So, here is the process for adding a task to the lists... (it's going to sound complicated, but it isn't really, honest, would I lie to you?)

Enable FAllocBitMap's 'Task Logging' function. (if you have just enabled this option, remember to leave the GUI via 'Use' or 'Save', not 'Cancel' or the close window gadget)

Make sure FBlit's GUI is closed. (due to lazy programming, the GUI's copy of the task log is only updated when it is actually launched ATM!)

Now start up and use the software you are interested in. (if it is already running you may have to quit and restart, to persuade it to re-allocate it's bitmaps)

Launch FBlit's GUI again, and go to the relevant FAllocBitMap/Lists page (ie. 'Include List' for include mode, 'Exclude List' for exclude mode).

Select the drop down menu next to the string gadget and double click on the likeliest looking task name/s.

Some experimentation may be required if it's not obvious which task names belong to the software you are interested in.

Changes made to the task lists will not take effect until you exit the GUI via 'Use' or 'Save', and may not effect the specific tasks that were added/removed until those tasks have been closed and restarted.

Tasks to Exclude

These are task names which are known to have problems with FBlit, if they are promoted. You should add these to your exclude task list if you

wish to use exclude mode. And avoid adding them to your include list, if you use include mode.

'V'
- it appears that the problems with this task may have been solved as of version 3.1 of Voyager, but for anyone still using older versions...

  This is Voyager's main task, and unfortunately it is currently incompatible with FBlit. Promoting this task will cause problems when cached images are re-scaled. However, this is a little rare, and you may want to promote it anyway since promotion can reduce the number of other V related crashes. This may or may not apply to older versions of Voyager (<V^3).

'DPaint'
- for some versions of DPaint (at least version 5) this task should be excluded. It seems to use the blitter hardware directly. Other versions of DPaint may work correctly (or not).

In addition, while FDraw remains incomplete, you might want to exclude some other things which exhibit line related corruption (mostly gfx editing software, and analogue clocks) for aesthetic reasons.

1.13 The Patches

A general description of all the patches is available toward the end of What It Is .

FBlit's patches have a number of configuration options, but the majority of these are of no use on a normal installation. They are mainly for testing/development purposes, and playing with them is potentially dangerous.

Patch Installation
Chip/Fast Data Options
Info and Statistics

FBltBitMap
FBltClear
FBltTemplate
FBltPattern
FBitMapScale
FFlood
FAllocBitMap
FSetRast
FAreaEnd
FDraw
OSTLPatch
AddBobPatch

RemIBobPatch
QBSBlitPatch

1.14 Patch Installation

Patch Installation

Patch Installation options are common to all the patches and consist of two checkboxes:

Installed selects wether or not the patch is actually installed on the system. Unless you are running 'PatchControl' (or equiv.) it may be impossible to uninstall a patch that has been overwritten by another patch.

Most of FBlit's patches cannot be safely un-installed! Unless the documentation for a specific patch says otherwise, you should never attempt to un-install it.
(see FBlitGUI 'Danger!')

Activated selects wether a patch is active or not. This allows a patch to be disabled when it has become impossible to un-install, due to over-patching. Deactivating a patch is equivalent to un-installing it, and therefore the comments about the dangers of un-installing a patch apply to this too. Please don't do it, unless the documentation for that patch specifically says that it is safe, or you will risk corruption of your Chip memory!

1.15 Chip & Fast Data Options

Chip/Fast Data Options

All the patches that replace blitter functions have 'Chip Data Options' and 'Fast Data Options'. 'Chip Data' refers to data that is in Chip memory, and 'Fast Data' covers data anywhere else (not necessarily Fast RAM in the strict sense).

Chip Data Options effect operations where all relevant data is in Chip memory, and therefore can safely be processed by the original blitter functions. In the case of functions which deal with rastports, this applies only when all planes of all associated bitmaps are in Chip RAM. The following settings are available for patches with this option (additional settings exist for certain patches):

Pass On means that the original function will be used to do this operation.

Process or Process All , the patch's CPU routines will be used. This is mostly intended for testing, but in some cases it may be quicker to use the CPU routines than the blitter functions.

Fast Data Options effect operations when any data is outside Chip memory, and so can not be processed by the original blitter assisted functions. For rastports this applies if any plane of any associated bitmap is outside Chip RAM (wether or not the operation actually occurs in those planes). All patches with this option offer these possible settings:

Pass On passes the operation on too the original function. This is invariably a very bad thing! Do not use this setting, or you will almost certainly have corrupted Chip memory to deal with!

Process causes the operation to be processed by the CPU routines. This is the only useful setting for this option.

Discard means that these operations will simply not be done. This may or may not be dangerous, but it is certainly not very useful. Most likely you will just get blank/corrupt graphics.

1.16 Info & Stats

Info & Stats

The meaning of the stats is not documented, and the labels may be misleading. This is not very interesting stuff, but here are the meanings of the stats and gadgets that are common to all patches at the moment:

Version information is (usually) correct for all patches.

Original/Current/Patch addresses should also be accurate (but note that some 'PatchControl' type programmes may corrupt these values). The 'Original Addr' is the address of the original function which was patched (it is meaningless while the patch is not installed). 'Current Addr' is the current address of the function. 'Patch Code' is the address of the patch. Not particularly useful, but you can for instance tell if the patch has been overwritten by something else ('Current Addr' will not be the same as 'Patch Code'), or if the original function had already been patched ('Original Addr' is not in ROM).

Update/Reset gadgets. These refer to the statistics, if the patch offers any. Reset will set any stats counters back to zero. Update will update the display with the current stat values. The display is not updated in real time since several of the patches may be called when the GUI is refreshed, which would cause a feedback problem. Anyway, take into account that using Update may in itself effect the stats.

1.17 FBltBitMap

Patches:
 BltBitMap()

Purpose:
 Allows BltBitMap() to operate on bitmaps which have planes outside of Chip memory ie. outside the addressable range of the blitter hardware. This behaviour is inherited by several other functions eg. ClipBlit(), BltBitMapRastPort(), BltMaskBitMapRastPort()... and consequently, higher level things such as superbitmap sync operations, intuition.image rendering, icons....

How:
 FBltBitMap is a complete replacement for BltBitMap() which uses the CPU to do the operations instead of the blitter, though the blitter may still be used when it is possible/appropriate.

 Currently, 'simple' functions (copy, copy&invert, fill etc.) use custom 32bit routines. 'Complex' operations ('cookie cutter'...) are done on a simple 16bit three channel blitter function simulator.

Options:

  Patch Installation:
  You should never un-install or deactivate FBltBitMap!

  Chip Data Options:
  (default - Pass On Complex)

  Additional options:

   Pass On Complex will pass on operations which require performing logical functions using both source and destination data to the original BltBitMap(). For these operations the CPU routines currently have no real advantage over the blitter.

  Fast Data Options:
  (default - Process)

Notes:

 FBltBitMap has a few 'side effects'...

 For most operations it is faster than the original since (at least on AGA) the CPU can do 32bit accesses while the blitter is stuck at 16bit. So the CPU may effectively have only half as much work to do. Also the

CPU can better avoid unnecessary memory accesses. This is somewhat offset by the fact that the blitter might operate in parallel with the CPU, does logical operations for free and can hog the Chip bus.

On non-interleaved data the CPU routines will suffer less from colour flicker because all (data) planes are copied at once on a row by row basis, while the blitter will move one whole plane at a time. There may still be some flicker since non-data planes are handled separately from data planes. And, without double buffering, there will be the usual visual artifacts where the live raster passes the row currently being rendered.

1.18 FBltClear

Patches
BltClear()

Purpose
Allows BltClear() to function outside of Chip memory. This is required since BltClear() may be used on memory that is part of a bitmap.

How
FBltClear is a fully functional CPU only replacement for BltClear().

Options:

Patch Installation:
You should never un-install or deactivate FBltClear!

Chip Data Options:
(default - Pass On Asynch)

Additional options:

Pass On Asynch will only pass on asynchronous operations which may be done by the blitter while the CPU does something more interesting (theoretically).

Fast Data Options:
(default - Process)

Notes

FBltClear is usually faster than the original BltClear(), since the job appears smaller to the 32bit CPU than to the 16bit blitter, at least on a 32bit system. For asynchronous calls it is probably still better to use the blitter, when possible.

There is an interesting (perhaps not for you ;) side effect of FBltClear, reported by Rick Pratt. If set to process all Chip data, some corruption may occur (eg. on the standard WB analogue clock). This is obviously a bad thing, but it is apparently nothing to do with the code. It isn't really clear what causes this effect, or given the symptoms, why it doesn't seem to have much more wide ranging effects on the entire system. The effect is only known to appear on (some) 68030 accelerated systems, and the best guess is that it is some sort of hardware bug, related to the caches, or perhaps a bus timing issue. Data caches should not be active in Chip memory anyway, but disabling them did clear up the problem... Oh well.

1.19 FBltTemplate

Patches
BltTemplate()

Purpose
Allows BltTemplate() to function outside of Chip memory. This is needed to support rastport's bitmaps outside Chip RAM.

How
FBltTemplate is a fully functional CPU based equivalent of BltTemplate().

Options:

Patch Installation:
You should never un-install or deactivate FBltTemplate!

Chip Data Options:
(default - Pass On)

Fast Data Options:
(default - Process)

Notes

Almost the only thing BltTemplate() is used for is text rendering. Because the CPU routines are 32bit, and because of the nature of text (it usually forms a fairly wide rectangle), FBltTemplate is often faster than BltTemplate(), so you may want to set the 'Chip Data Option' to 'Process'. It does depend on hardware, and may not be a good idea on OCS/ECS, but you can easily test this one with any graphics based text speed test (SysSpeed etc.). CON: output speed tests however are not likely to benefit much.

1.20 FBltPattern

```
Patches
 BltPattern()


Purpose
 Allows BltPattern() to function outside of Chip memory. This is needed
to support rastport's bitmaps outside Chip RAM.


How
 FBltPattern is a fully functional, CPU based, replacement for
BltPattern().


Options:

  Patch Installation:
 You should never un-install or deactivate FBltPattern!

  Chip Data Options:
  (default - Pass On)

  Fast Data Options:
  (default - Process)



Notes

 BltPattern() is widely used by the OS for many (perhaps most) rastport
rendering calls. You may find that FBltPattern is faster than
BltPattern() for some operations (eg. RectFill()), but it may well be
slower for others, so I don't particularly advise you to set the 'Chip
Data Option' to 'Process', or not...
```

1.21 FBitMapScale

```
Patches
 BitMapScale()


Purpose
 Allows BitMapScale() to function outside of Chip memory. This is needed
to support discrete bitmaps with planes in Fast RAM.


How
 FBitMapScale is a completely CPU based replacement for BitMapScale().
It may also call BltBitMap() and so requires FBltBitMap.
```

```
Options:

  Patch Installation:
 You should never un-install or deactivate FBitMapScale!

  Chip Data Options:
  (default - Pass On)

  Fast Data Options:
  (default - Process)


Notes

 FBitMapScale is rarely used, and has not had any great effort put into
it. It is probably slower than the original in most circumstances.

 It is also worth noting that FBitMapScale is virtually guaranteed not
to produce output identical to that of the original BitMapScale()
function.
```

1.22 FFlood

```
Patches
 Flood()


Purpose
 Alows Flood() to operate when the provided TmpRas is not in Chip RAM.


How
 This patch checks the supplied TmpRas. If it is not in Chip memory,
another TmpRas will be allocated in Chip RAM, and Flood() is called.
The Chip TmpRas is freed again when Flood() returns.


Options:

  Patch Installation:
 FFlood should not be un-installed, or deactivated.


Notes

 This patch still requires that sufficient Chip memory is available to
perform the original function, and operations with fast RAM TmpRas will
in fact be slower than when a Chip TmpRas is provided initially.

 The only programme that is currently known to use TmpRas in Fast RAM
```

is PPaint, when in it's full RTG guise.

1.23 FAllocBitMap

Patches
 AllocBitMap()

Purpose
 This patch is responsible for forcing graphics data to be allocated in Fast memory.

How
 Depending on the configuration, FAllocBitMap will decide wether or not an AllocBitMap() call from a given task should be promoted. If a bitmap is to be allocated in Fast RAM, FAllocBitMap will create it internally, otherwise AllocBitMap() will be called.

Options:

  Patch Installation:
 FAllocBitMap may be fairly safely un-installed, or deactivated at any time. Doing so will stop FBlit from promoting graphics to Fast RAM.

 Task Logging:
 The names of tasks which call AllocBitMap() will be stored in the task log if this option is enabled. Tasks which cannot be made to appear in the log cannot be effected by FAllocBitMap and therefore should not be put in the task lists.

 It is important to remember that the copy of the task log used by FBlitGUI is not maintained in real time, it is only updated/valid at the point when the GUI is invoked. So, to get an up to date copy of the log you may have to shut down, and restart the GUI.

 There are a couple of other points about the task log. Tasks that are already named in the currently active 'list' will not appear in the log. Also, the log is case sensitive, which means that a task can appear several times with differing capitalization (ie. 'Multiview' and 'multiview'), while the actual task 'lists' are not.

 Task List Options:
 When forcing tasks to allocate bitmaps in Fast RAM, FAllocBitMap has two modes of operation.

  Include mode is the safer setting, and the default (currently). In this mode only those tasks named in the 'Include List' will be promoted to Fast memory.

Exclude mode is a bit more risky. In this mode all tasks will be forced to use Fast RAM, except those named in the 'Exclude List'.

~Note that only one of the two task lists is active at any given time. For 'include' mode, only the 'Include List' is used, the 'Exclude List' is irrelevant. And vice versa for 'exclude' mode.

Anonymous Tasks:
This option defines what happens to tasks which have no name, and so cannot be handled by the task name lists.

Pass On , anonymous tasks will use Chip RAM (default).

Promote , the tasks will use Fast RAM.

Displayable Bitmaps
Selects wether or not displayable bitmaps should be promoted to fast memory.

Pass On , displayable bitmaps will use Chip memory (default).

Promote , displayable bitmaps use fast memory. This is not a good idea! Please don't use this setting! This setting is only provided in case anybody wants to try writing a display driver for fast RAM screens. Using it without such a driver, you will just see garbage when a new screen is opened.

Promotion Memory
Defines the 'type' of memory to be used for bitmap's planes.

MEM_FAST , is the default, and means that planes will always use fast memory only! Even if you have no fast memory, and loads of free chip (or other non-fast) RAM, it will never be used for bitmap's planes.

MEM_ANY , means that any memory can be used for planes. The 'best' available memory will be used first, but other types of 'public' (ie. not virtual) memory may be used if required. This setting should work, fine (if not, it is due to bugs in FBlit), and should be the defualt, but it has not had significant testing, so it isn't (yet).

Lists
The 'Lists' page allows editing of the 'Include' and 'Exclude' task lists. Tasks may be added by entering their names in the string gadget, or by selecting them from the drop down task log. To remove an entry, select it in the list view and press the 'Remove' gadget.

Task list editing is the only part of the GUI which does not operate in real time. Changes made to the task lists will only take effect upon exiting the GUI (via 'Use' or 'Save').

Unlike the task log, the task lists are not case sensitive. So a single entry ('multiview') will cover any capitalization ('Multiview', 'MultiView').

Notes

 Bitmaps created by FABM are not guaranteed to be identical to those created by AllocBitMap().

 Currently, FABM will create bitmaps with a 16bit aligned width (32 would be better, but causes problems with some very popular software), displayable bitmaps are width aligned to 64bit.

 Minplanes is implemented, for what that's worth. Friend bitmaps are supported in that, when appropriate, a bitmap will be made interleaved to match the 'friend'. Interleaved bitmaps have a size limit of <1024 bytes per row, and <1024 rows. bm_Pad is set to zero, or the magic interleave value. bm_Flags is set to zero.

 FreeBitMap() is not patched. FABM relies on it operating in a certain manner, as it does in graphics.library v39/40 (and may not in older, or (unlikely) future revisions).

 Note that although AllocBitMap() is supposed to accept ULONG parameters, it appears that it ignores the upper WORD and this has allowed some software to function while passing corrupt values. To support such broken software, FABM now masks the input values to WORD for 'sizex'/'sizey', and BYTE for 'depth'.

1.24 FSetRast

Patches
 SetRast()

Purpose
 Allows SetRast() to work outside Chip memory.

How
 This patch calls either BltClear() or BltBitMapRastPort() to do the actual work, so both those functions must support operations outside Chip memory (ie. FBltClear and FBltBitMap are required).

Options:

  Patch Installation:
 You should never un-install or deactivate FSetRast!

  Chip Data Options:
  (default - Pass On)

```
  Fast Data Options:
  (default - Process)
```

1.25 FAreaEnd

```
Patches
 AreaEnd()


Purpose
 Alows AreaEnd() to deal with fast RAM TmpRas.


How
 This patch checks the supplied TmpRas. If it is not in Chip memory,
another TmpRas will be allocated in Chip RAM, and AreaEnd() is called.
The Chip TmpRas is freed again when AreaEnd() returns.


Options:

  Patch Installation:
 FAreaEnd should not be un-installed, or deactivated.


Notes

 This patch still requires that sufficient Chip memory is available to
perform the original function, and operations with fast RAM TmpRas will
in fact be slower than when a Chip TmpRas is provided initially.

 The only programme that is currently known to use TmpRas in Fast RAM
is PPaint, when in it's full RTG guise.
```

1.26 FDraw

```
Patches
 Draw()


Purpose
 Allows Draw() to work outside Chip memory.


How
 FDraw is a completely CPU based equivalent for Draw(). ..Or it will be,
if I ever get around to actually writing it.
```

```
Options:

  Patch Installation:
 You should never un-install or deactivate FDraw!

  Chip Data Options:
  (default - Process Hor)

  Additional options:

   Process Hor causes horizontal, non-patterned lines
to be rendered with the CPU routines. The CPU routines are far quicker
than Draw() for horizontal lines, but may be slower for all other
orientations.

  Fast Data Options:
  (default - Process)



Notes:

 FDraw is incomplete and this can cause some problems! Mostly these
problems take the form of lines that are not erased from the display.
This is mainly due to the lack of support for COMPLEMENT mode rendering
and is completely safe, but not very pretty. These effects are not
common in the default configuration, but will happen regularly if you
set FDraw to process all Chip data. They are also more common if
FAllocBitMap is set to 'exclude' mode since more operations will then be
processed by the CPU routines.

 FDraw currently does not produce results identical to Draw() for
diagonal lines. This can cause some artifacts where lines are rendered
next to each other, or bound a blitter rendered rectangle (clock hands).
```

1.27 OSTLPatch

```
Patches
 OpenScreenTagList()


Purpose
 This hack attempts to stop programmes from opening screens with non-
displayable bitmaps that may have been promoted to Fast RAM.


How
 If a bitmap is supplied to OpenScreen/TagList(), the patch will check
wether the planes are in Chip memory. If they are not, it will attempt
to allocate an identical bitmap with the #BMF_DISPLAYABLE flag set. If
this fails the patch returns failure. If it succeeds, the original image
is copied to the new bitmap with BltBitMap(), and (the rather unpleasant
```

bit) the contents of the bitmap structures are swapped. The newly
allocated bitmap (with the original bitmap definition) is then freed.
The original bitmap (with the new bitmap definition) is used for the
OpenScreenTagList() call.

Options:

  Patch Installation:
 So long as the rest of FBlit is doing it's job, it may be safe to un-
install OSTLPatch. The result of doing this is that screens may be
opened in Fast RAM, though this is very rare (the commonest offender ATM
is MultiView). Such screens will most likely be displayed as complete
garbage since the video hardware will be displaying some random regions
of Chip memory.

Notes

 As mentioned, if the rest of FBlit is working correctly, it may
actually be possible to run a screen in Fast RAM safely. You simply
can't see it, so it's not terribly helpful. However it is an interesting
possibility. Theoretically, it might be possible to keep screens in Fast
RAM and use a simple 'video driver' to actually display the current
frontmost screen in Chip RAM. Perhaps with double buffering? MMU
refresh? Maybe not... There would certainly be some problems... As ever,
it would be more appropriate to write a native driver for P96. Or wait
for the OS to deliver RTG.

1.28 AddBobPatch

Patches
 AddBob()

Purpose
 Along with RemIBobPatch this hack is required to allow icons
to be dragged, when running NewIcons in RTG mode.

How
 If the BOB's image is outside Chip RAM, it will be copied back to an
auto-expanding buffer in Chip memory. The BOB's image pointer is changed
to reflect this, and AddBob() is called.

Options:

  Patch Installation:
 This patch may be safely un-installed if you do not use NewIcons (or
OS3.5 icon.library) in RTG mode, or if you use QBSBlitPatch instead.
If you un-install AddBobPatch, you must also un-install RemIBobPatch!

Notes

 The Add/RemIBobPatch hack was designed specifically for NewIcons RTG,
FBlit does not officially support GELS outside Chip memory.

1.29 RemIBobPatch

Patches
 RemIBob()

Purpose
 Along with AddBobPatch this hack is required to allow icons
to be dragged, when running NewIcons in RTG mode.

How
 RemIBob() is called. Then, if the BOB's image is in the Chip buffer,
the BOB's image pointer is restored to the original Fast image. If this
was the last image in the Chip buffer, the buffer is freed.

Options:

  Patch Installation:
 This patch may be safely un-installed if you do not use NewIcons (or
OS3.5 icon.library) in RTG mode, or if you use QBSBlitPatch instead.
If you un-install RemIBobPatch, you must also un-install AddBobPatch!

Notes

 The Add/RemIBobPatch hack was designed specifically for NewIcons RTG,
FBlit does not officially support GELS outside Chip memory.

1.30 QBSBlitPatch

Patches
 QBSBlit()

Purpose
 Allows OS3.5 RTG icons to be dragged.

How
 QBSBlit() calls from DrawGList() are intercepted, and the
bltnode/function is modified to run on a blitter emulator. The original
QBSBlit() is then called.

Options:

  Patch Installation:
 Currently this patch is un-installed by default. If you install this
patch, you must un-install AddBobPatch and RemIBobPatch (and vice
versa).

Notes

 To some extent this patch offers almost complete support for GELS (at
least excluding 'SimpleSprites') outside Chip memory, but I still deny
that FBlit supports this.

 The blitter emulator used by QBSBlitPatch is not particularly fast. It
is marginally slower than the hardware blitter on an 060/50 even when
most of the data involved is in Fast RAM.

 There is a potential problem with this scheme. The blitter emulator
ends up running on an interrupt, and inevitably it will tie up that
interrupt for far longer than the normal bltnode/function would have
done. No side effects of this have been reported, but that doesn't mean
there aren't any.

1.31 Programming

 under contruction