1993 International Amiga Developers Conference Notes

Commodore

Table of Contents

1993 International Amiga Developers Conference Notes

Amiga Developers Conference Notes
Orlando, 1993

V39 and AA Compatibility

by Commodore Engineering and CATS

Introduction

The capabilities of the built-in Amiga graphics hardware did not change significantly between the introduction of the Amiga 1000 in 1985 and the release of the Amiga 3000 in 1990. The ECS chipset and the display enhancer added several new graphics modes and increased the functionality of existing modes -- yet the market demanded more.

This document explains some of the features of the latest generation Amiga graphics hardware known as the AA (double-A) chipset, and ways to ensure application compatibility with these and other V39 features and changes. Additionally, coding methods are outlined which will allow current software to automatically exploit some of the new features.

New Hardware Features

The most important capabilities of the new Amiga graphics hardware are summarized below.

- ❑ Enhanced Bandwidth. A 32-bit wide data bus allows doubling of Chip memory bandwidth (up to 2x the normal bandwidth) and supports the input of 32-bit wide bitplane and sprite data. Bandwidth may be doubled again (to 4x) by using Fast Page Mode RAM.
- ❑ More Bitplanes. The maximum number of bitplanes has increased to 8 in all resolution modes. This translates to a 256-entry color table for each available mode.
- ❑ Larger Palette. Each entry in the color table may now be 25 bits wide (8 bits each for Red, Blue, and Green data plus 1 bit for genlock information). This translates to a palette of 16,777,216 colors.
- ❑ HAM and HALFBRITE are available in all display resolutions and depths, allowing for greatly enhanced display modes such as 8-bitplane HIRES HAM.
- ❑ Enhanced Dual Playfield Support. Each playfield may now have up to 4 bitplanes. The bank of 16 colors in the 256-color table is independently selectable for each playfield.
- ❑ Enhanced hardware scrolling support. The resolution of bitplane scrolling has increased to 35ns.

- ❑ Enhanced Sprite Support. Sprite resolution can be set to Lores, Hires, or SuperHires, independent of screen resolution. Attached sprites are now available in all modes. However, some new higher bandwidth modes may only allow one sprite (making an attached sprite impossible). Odd and even sprites may use their own independent 16-color bank from the 256-color table. Old format sprites are still 16 bits wide, but new format sprites may be 32 or 64 bits wide. Sprites may now optionally appear in the border region. The horizontal positioning resolution of sprites has increased to 35ns (equivalent to SuperHires pixel widths.)
- ❑ Hardware scan-doubling support. 15 kHz bitplanes and sprites may now be scan-doubled for flicker-free display on 31 kHz monitors, and for enhanced display sharing with 31 kHz bitplanes.
- ❑ ECS compatibility. New chips will power-up in an ECS compatibility mode, which will allow many older self-booting programs to be run on new machines.

Ensuring Compatibility

This section covers programming techniques that will help ensure that your application will work as expected when running under V39 and on systems that use AA and future generation graphics hardware.

Support of Release 2 is integral to supporting new graphics chips. One of the main reasons is the display database. Starting with the ECS chipset, not all machines have all display modes available. This will be even more true in the future. The only way to know if a particular mode is available is to check the display database.

For a detailed explanation of how to properly open a screen using the display database information to check for available modes, refer to the *Amiga ROM Kernel Reference Manual: Libraries*, 3rd Edition. Also see the Amiga Mail articles entitled "An Introduction to V36 Screens and Windows" (page IV-3) and "Opening Screens and Windows on Any Amiga" (Page IV-17). Some other pertinent details can also be found in the Paris DevCon notes article entitled "Monitors, Modes, and the Display Database."

Once a screen or other display has been opened, all manipulations should also be handled using system calls. In particular, these manipulations must be handled by the system software:

- ❑ Allocation of graphics resources - use the graphics or intuition library
- ❑ Palette manipulation - use the graphics library
- ❑ Manipulating icons - use the icon library

- ❑ Manipulating mouse pointer sprite - use intuition.library to select or change imagery; input.device to move it
- ❑ Manipulating sprites other than the mouse pointer - use the graphics library
- ❑ Allocating and changing colormaps and colortables - use the graphics library
- ❑ Bitplane allocations - use AllocBitMap() under V39 and higher, and AllocRaster() under earlier releases, but please refer to the next section for important information about limitations of AllocRaster().

Furthermore, these manipulations should be handled by the system software whenever possible:

- ❑ Drawing operations - use the graphics library and Intuition library
- ❑ Blitter operations - if possible, use only use higher level blitter functions from graphics.library such as BltBitMap(), BltRastPort(), etc. If you must wait for a blit to complete, always use the system's WaitBlit() function, never your own. The system knows about and handles problems with the BLITTER_DONE signal that different revisions of the Agnus chip have. If you feel that must use the blitter hardware directly (sacrificing all hope for any RTG compatibility), be sure to properly OwnBlitter, then WaitBlit, then use blitter, then DisownBlitter.

Remember that system structures are subject to change or extension. For new graphics hardware, structures likely to be extended are those associated with ColorMaps, ViewPortExtras, Sprites, and many others. To deal with changing system structures, developers should use system calls to allocate, create, and manipulate these structures. Many new "Get" and "Set" calls have been added to graphics.library to provide an upwards-compatible mechanism for reading and setting structure members which were previously accessed by program code either directly or through macros which accessed them directly. For future compatibility, do not directly access or change any structure or value for which a "Get" and "Set" call is available. And do not declare or AllocMem any structure for which a special allocation call is provided.

Most of the new "Get" and "Set" graphics calls for V39 are listed below. Note that some of these functions are workalike replacements for old gfxmacros which poked or peeked a structure, and others provide tag-based capability for getting and/or setting more than one attribute (Attr) of a structure:

```
SetOutlinePen(rp,pen)(a0,d0)                        SetMaxPen(rp,maxpen)(a0,d0)
GetBitMapAttr(bm,attrnum)(a0,d1)                    SetWriteMask(rp,msk)(a0,d0)
SetRPAttrsA(rp,tags)(a0/a1)                         GetRPAttrsA(rp,tags)(a0/a1)
GetAPen(rp)(a0)                                     GetBPen(rp)(a0)
GetDrMd(rp)(a0)                                     GetOutlinePen(rp)(a0)
SetABPenDrMd(rp,apen,bpen,drawmode)(a1,d0/d1/d2)
```

Use the newest functions available. For compatibility reasons, it is sometimes not possible to change the behavior of an existing system function to provide enhanced capabilities. In such cases, new functions may be provided. For upwards compatibility, conditionally use the newest functions that are available. For example, when running under an OS earlier than V39, applications that wish to allocate a BitMap should use AllocRaster() and InitBitMap(). New or updated applications should use the new AllocBitMap() call instead when running under V39 or higher. See the V39 graphics library Autodocs for specifications of new functions such as AllocBitMap().

Hardware Compatibility

The new chips power up in an ECS compatible state that allows a large portion of older, self-booting (i.e., game) software to run even though these older titles often access hardware registers directly. However, directly accessing hardware should be avoided by any software that expects to run after the operating system has been started. Keep in mind also that it is difficult to maintain hardware register-level compatibility even on powerup as new and more enhanced chipsets are developed. Avoid directly programming the hardware whenever possible.

In cases where applications *must* access the hardware:

- ❑ Applications must not write spurious data to, or interpret data from, currently unused bits or addresses.
- ❑ Applications must set undefined bits to zero for writes and ignore them for reads.
- ❑ Mask out all the bits except the ones the application is actually interested in.
- ❑ Do not assume the initial state of any registers.
- ❑ Do not read write-only registers or write to read-only registers.
- ❑ Do not read or write *bytes* to the custom chip registers (they are*word* registers).

Graphics and Intuition Issues

The following notes detail some known coding practices that can cause software to function incorrectly on V39 and AA machines.

- ❑ BitMap plane allocations unsuitable for new modes. This problem surfaces because the new chips have an increased fetch bandwidth. The data for each bitplane scan line must be properly aligned in Chip memory for new display modes to work.

For instance, these allocation methods should all be avoided under V39 and higher:

```
/* WRONG for AA */
for(planenum=0; planenum<DEPTH; planenum++)
    bitmap.Planes[planenum]=(PLANEPTR)AllocMem(RASSIZE(width,height),MEMF_CHIP);

/* Also WRONG */
allplanes=AllocMem(DEPTH*RASSIZE(width,height),MEMF_CHIP);

/* Also WRONG */
for(planenum=0; planenum<DEPTH; planenum++)
    bitmap.Planes[planenum]=(PLANEPTR)AllocRaster(width,height);
```

The correct V39 method for allocating rasters is to use AllocBitMap() for graphics-level bitmaps and CUSTOMBITMAP screens, or let Intuition OpenScreen() or OpenScreenTagList() allocate your bitmap. These methods will allow you to get the greater bandwidth and more efficient displays available under V39 in a manner that will be upward-compatible with future enhancements. The new V39 graphics function AllocBitMap() properly handles all allocation and alignment issues. The Intuition OpenScreenTags() call uses AllocBitMap() under V39.

If for some reason your existing code design prevents you from conditionally calling AllocBitMap() or OpenScreen() when running under V39 and higher, you may be able to at least get compatibility with the currently available higher-bandwidth modes by using AllocRaster() or AllocMem() after rounding your width up to a multiple of 64 pixels (absolutely no guarantees here; future chips may require greater alignment; we suggest you conditionally code to use AllocBitMap() for future compatibility).

- ❑ Incorrect assumptions about BitMap->BytesPerRow. Applications that use system functions to initialize and allocate BitMaps and their elements should be aware that the value of BitMap->BytesPerRow may be different from the expected value, depending on the bandwidth mode chosen, the custom chip type, method of allocation, and the asked-for width of the allocated planes.

Here's the explanation. First, due to fetch-alignment restrictions in AA, some modes require a higher granularity (BytesPerRow must be a multiple of 4 or 8, instead of just 2 under ECS). Second, BytesPerRow actually means two different things, which have always been identical, but aren't any more when using interleaved bitmaps. BytesPerRow officially means "the number of bytes you have to add to a pointer to a byte of the BitMap to get to the same place one row down." It no longer can be depended on to mean "the number of bytes in this row."

To detect software compatibility problems related to BytesPerRow changes, it is extremely important to test on a machine with the AA chipset and Mode Promotion turned on in the IControl Preferences editor, and if the software supports various display sizes, to test with sizes which are not divisible by 64. Two typical symptoms of BytesPerRow problems are 1) skewing of the display, where the pixel data for every line is progressively further right or left, giving a diagonal appearance to the display, and 2) images saved with excess blank space at the right.

> *Note* - CATS intends to provide a developer tool, probably called "BumpBPR", which will force all Screen and AllocBitMap() BitMap widths to a multiple of 64 pixels to emulate the higher scanline alignment of higher bandwidth AA modes. This should allow developers to test for many BitMap->BytesPerRow problems on non-AA and no-promotion machines running V39.

- ❑ Non-matching BitMap->BytesPerRow. Some applications assume that two BitMaps of the same width, but allocated by different methods, will be swappable or block-copyable in various manners. This is often no longer the case. For example, a BitMap allocated by OpenScreen (which calls AllocBitMap) may have raster line storage widths rounded up to provide higher alignment for higher-bandwidth displays. But the raster lines of BitMap planes allocated with AllocMem() are generally rounded up only to a multiple of 16 with a hardcoded macro (RASSIZE), and for compatibility reasons, AllocRaster() currently still performs only the same rounding to a multiple of 16.
- ❑ Interleaved BitMaps. For interleaved bitmaps, BytesPerRow is quite a bit larger than the number of bytes in this row. Screen grabbers and screen printers seem to be the primary victims of this change. For compatibility, the Workbench screen is non-interleaved if it opens before IPrefs has run. If opened or reset after IPrefs has run, the Workbench screen will be interleaved, so under V39, the Workbench screen in a normally booted environment is likely to be interleaved. GetBitMapAttr(bm, BMA_WIDTH) can return the true width of a BitMap in pixels. However, do not use this width when saving a BitMap as an ILBM since this width may be rounded up for alignment reasons.
- ❑ Incorrect assumptions about the internal structure of BitMap plane data. New types of BitMaps may have a significantly different layout. For example, new V39 interleaved BitMaps are allocated as one contiguous block of Chip memory and are internally different from non-interleaved plane data.

Interleaved BitMaps consist of line 0 of plane 0, followed by line 0 of plane 1, line 0 of plane 2, on through to line 0 of plane n. Then the pattern repeats with the data for the next line. This BitMap layout reduces the color flashing effect which normally accompanies the blitting of individual planes. Interleaved BitMaps can be requested by applications under AA. Note however, that callers are not guaranteed to receive an interleaved BitMap whenever they ask for one. The BMF_INTERLEAVED flag to AllocBitMap() is considered a request, not a requirement. If no sufficiently large continuous block of chip memory is available, it may not be possible to allocate the BitMap interleaved. In that case, AllocBitMap() will attempt to allocate an ordinary BitMap instead. Applications should be written so as not to be sensitive to whether or not a BitMap is interleaved. In those rare cases when it might matter, GetBitMapAttr(BMA_FLAGS) can be used.

- ❑ Incorrect assumptions about the Display Database. Data about a display mode could change between one version of Kickstart and another, between different models of the Amiga and even between similar models. Treat the data in the display database as dynamic! Information that was available under V37 may not necessarily be available under V39. For instance, VGA and A2024 mode information is no longer in ROM and unless they are added by the user, there will be no information about these modes available.

Never cache information about the DEFAULT_MONITOR_ID modes because the default monitor can be changed by the user with the promotion feature.

Use the 2.1 asl.library screen mode requester to present your user with choices which are actually available (Note - 2.1 asl.library works under 2.0 Kickstart, and a free amendment to distribute 2.1 asl.library is available for 2.0 Workbench licensees).

- ❑ Direct handling of ViewPorts. Applications that use low-level graphics calls to create the View directly may have problems in V39. MakeVPort() and MrgCop() can now return an error. In addition, the gap required between ViewPorts may now need to be significantly larger than on pre-AA chips, where it was never more than 3 non-interlaced lines (6 interlaced lines). The V39 graphics function CalcIVG() (Calc Inter Viewport Gap) may be used to determine how many blank lines will be needed above a ViewPort. Note that Intuition currently uses at least 3 lines, or MAX(3,CalcIVG(v,vp)) (substitute 6 for 3 if interlaced). Also note that for old display modes with old depths and 4-bit-per-gun colors, the gap is unchanged from that under the ECS and original chipsets.

- ❑ Incorrect use of PAL or NTSC flags. The common method of determining whether a machine is running in PAL or NTSC mode has always been to check for the PAL bit in GfxBase->DisplayFlags:

```
BOOL IsPAL;

IsPAL=(GfxBase->DisplayFlags & PAL) ? TRUE : FALSE;
```

One classic reason for desiring such a PAL/NTSC determination is to decide what size or type display to open. The other common reason is to determine what clock constant to use in serial or audio period calculations (Amigas built as PAL machines have a slightly different system clock crystal frequency than Amigas built as NTSC machines, and serial/audio period calculations are dependent on this frequency).

Since V37 and V39 are more adaptable, this bit may no longer be counted on to mean the user's preferred display mode, nor to even signify the probable clock crystal in the user's system.

For accurate serial and audio period calculations, you need to determine whether the system has a system clock crystal designed for PAL or NTSC. There is no totally foolproof way of doing this. Under 1.x through 2.x, your best indicator of whether a system has a PAL system clock crystal is the GfxBase->DisplayFlags PAL bit. This bit should tell you the hardware default of the machine, and should only mislead you if the user has moved the PAL/NTSC jumper or has run a PD program to change the bootup mode between PAL and NTSC.

Under V39 (and probably above), new BootMenu options now allow the user to software-select PAL or NTSC as the default graphics environment. This can modify the GfxBase->DisplayFlags PAL bit, but obviously does not change the system clock crystal. Therefore, under V39 and higher, a new DisplayFlags bit called REALLY_PAL has been added to signify the whether the motherboard hardware jumper setting appears to be PAL. Therefore, under V39 and higher, the best bit available for determining NTSC or PAL hardware is the new REALLY_PAL bit.

If your software needs to determine the characteristics of default displays, prior to V37 look at DisplayFlags PAL bit. Under V37 or higher, instead use GetDisplayInfoData() to get information about the modeid LORES_KEY or HIRES_KEY, and then check the DisplayInfo.PropertyFlags for properties such as DIPF_IS_PAL (which will be true for both PAL and DblPAL).

If your software instead needs to determine the characteristics of the user's preferred display mode (as evidenced by the mode of her Workbench or other default public screen), you can GetDisplayInfoData() on the modeID of the default public screen. Be aware that this modeID *might not* be any of the PAL or NTSC modes, but rather VGA or some other mode.

```
#include <graphics/displayinfo.h>

Bool IsPAL;
struct Screen *screen;
ULONG modeID = LORES_KEY;
struct DisplayInfo displayinfo;

/* Open graphics.library, etc. */

/* Above, modeID is initialized to LORES_KEY.  You should inquire about LORES_KEY
 * or HIRES_KEY if you are interested in the display characteristics of default
 * displays (for example, what OpenScreen() will provide if you don't use tags to
ask for a display mode with an explicit monitor like PAL or NTSC or VGA).
 *
 * Do the following LockPubScreen part if you are instead interested in the display
 * characteristics of the modeID of the user's default public screen (usually
 * Workbench).
 */

if (screen = LockPubScreen(NULL))
    {
    modeID =GetVPModeID(&(screen->ViewPort));
    UnlockPubScreen(NULL,screen);
    }

/*
 * Now get information about the modeID
 */
if (GetDisplayInfoData(NULL,(UBYTE *)&displayinfo,sizeof(struct DisplayInfo),
                                       DTAG_DISP, modeID))
    {
        /* True if PAL or DblPAL */
    if (displayinfo.PropertyFlags & DIPF_IS_PAL)
        IsPAL = TRUE;
    else
        IsPAL = FALSE;
    }
```

- ❑ Incorrect assumptions about ColorMaps and ColorTables. The color system has undergone significant changes so new V39 graphic functions should be used to manage color whenever possible. ColorMaps should always be allocated using GetColorMap(); freed using FreeColorMap(); colors changed using LoadRGB4/32(), SetRGB4/32(), or SetRGB4/32CM(); and colors queried using GetRGB4/32(). Specifically, the value ColorMap->ColorTable and the structure it points to should never be poked or read directly.

The color functions with names containing "32" are new V39 functions for getting, setting, and loading color registers. These new functions treat color guns (R, G, B) each as 32-bit values for handling not only the 8-bit color available in AA but also any conceivable future needs. Use of these new 32-bit color functions is required to display the 16 million different color shades available under AA and V39. The old 4-bit color manipulation functions can only provide 4096 different colors. Use these new 32-bit color functions to ensure the future compatibility of your V39 applications.

When using 4 or 8-bit R/G/B values, scale your values to 32 bits. Scale 8 bit R/G/B values to 32 bits by duplicating the 8-bit value in all 4 bytes of the 32-bit value. Scale 4-bit values to 32 bits by duplicating the 4 bit value in each nibble of the 32-bit value.

4-bit red value $3 becomes $33333333
8-bit red value $1F becomes $1F1F1F1F
8-bit red value $03 becomes $03030303

The graphics.library VideoControl() function should be used to get, set, or clear the special features associated with ColorMaps.

- ❑ Poking Copper lists or copinit. The structure and order of Copper lists will change for all modes, so any program that relies on poking the Copper lists will likely break. Certain programs (especially games) make assumptions about copinit and poke it directly. These programs will break with the AA chips and V39 unless the AA machine is running in a non-AA old-chip emulation mode. Note again that copinit has changed, and will continue to change. Do Not Touch.
- ❑ Dangerous bits in the display hardware. Many bits in the custom chip registers that were previously undefined now have a function. Beware of poking the following bits.
- ❑ In BPLCON0: bits 0, 4, 5, 6, and 7
- ❑ In BPLCON2: bits 7, 8, 9
- ❑ In BPLCON3: bits 0,1,6,7, and bits 9 through 15
- ❑ Illegal use of the processor to write to bitplane and sprite data registers. Bitplane and sprite data registers should NOT be written to by the processor. The correct way for data to be fed to the chips is by setting up DMA that points to the data in Chip memory.
- ❑ Illegal re-use of sprites on the same line. Attempting to re-use sprites on the same horizontal line by redefining the sprite data is illegal and unsupported. Sprites may only be re-used vertically, and then only with at least a one line gap between re-uses. See the *Amiga Hardware Reference Manual* for an example of sprite re-use.
- ❑ Incorrect allocation of sprite image data. Because of the increased fetch bandwidth of the new chips, sprite image data must be properly aligned in Chip memory. Under 2.0, the best way to allocate this memory is to call the Exec library AllocMem() function. Use the new AllocSpriteData() call for allocation of new mode sprites under V39 and higher.
- ❑ Using an attached sprite for the Intuition pointer was quasi-supported under 2.0. It no longer works.
- ❑ Forgotten FreeSprite() call. Sprites may now have a number of different modes. However, under Intuition, these modes are global to all sprites. If a program gets a sprite in a particular mode, but then does not free it, Intuition (and all Intuition-based programs including Workbench) are forced to use sprites in the mode of the forgotten sprite. In general, it is good practice to not use sprites other than the pointer when coding software that is meant to be Intuition-compatible.

- ❑ Illegal use of Intuition pointer data. Intuition's pointer handling has become much more sophisticated. People playing tricky games with the pointer data may get into trouble if they make assumptions about the data format of the Intuition pointer which can change depending on the display mode.
- ❑ The old SetPointer() and ClearPointer() functions still work, giving compatible Intuition pointers. Likewise, the pointer imagery in devs:system-configuration will be used until a pointer.prefs file is received. However, due to growing complexity in the pointer subsystem, calling SetPointer() or ClearPointer() from within an input handler or inside Begin/EndRefresh() runs a risk of deadlocking. There are currently some patches in place to help prevent a deadlock, however we warn people to stick to simple rendering functions only when inside LockLayer(), LockLayerInfo(), BeginRefresh(), BeginUpdate(), etc. and not to call Intuition or other high-level system functions inside of LockIBase(). As well, great care should be taken inside an input handler (it's generally best for the handler to signal a high-priority task which actually does the work). Don't be the application that dies under the next release when an inappropriate function that happened to work now deadlocks because its handling has become more sophisticated. See the Autodocs for these functions for more information on what other system calls are OK to use with these functions.
- ❑ The pointer information returned by GetPrefs() is no longer kept up-to-date, since the pointer data can exceed the storage space available in struct Preferences. (The ROM default pointer will be returned in all cases). (Like V37, V39 ignores the pointer data in calls to SetPointer() after the first one, for reasons such as this).
- ❑ Screen.BitMap is becoming obsolete. The original Screen structure has an embedded instance of a BitMap structure, which can not grow to support current needs. When Intuition allocates a on-CUSTOMBITMAP screen's BitMap, it now uses AllocBitMap(), and just copies the old struct BitMap portion into the embedded `&screen->BitMap`. Intuition internally now references the true BitMap (obtainable as `screen->RastPort.BitMap`) instead of `&screen->BitMap`. We recommend that applications do the same. This is the direction we're headed anyway for RTG, and is needed for double-buffering. There are currently some patches in Intuition to handle people who are poking the `Screen.BitMap` depth or planes. Such poking is strongly discouraged. See the new V39 Intuition function ChangeScreenBuffer().

- ❑ OpenScreen failure if too deep. Since all modes and all depths are no longer available on every systems, OpenScreen will fail if you request a mode/depth combination that is not displayable. This can be an issue both for applications that may have been counting on allocating extra planes this way, or counting on having extra planes that would not be displayed. There is a new SA_ErrorCode value, OSERR_TOODEEP.
- ❑ Console compatibility Applications which draw graphics in the console portion of the window, but use console scroll commands may not work if they are drawing their text in other than pen 1 or pen 0 when only the bitplane for those colors is scrolled.
- ❑ Intuition and Graphics are involved in a scheme to maximize compatibility with older sprite-using applications. The AA chipset supports variable sprite resolution and width, but the setting affects all sprites. If an application requests a specific sprite width (through the old graphics.library/GetSprite() call or the new graphics.library calls (GetExtSpriteA() and ChangeExtSpriteA()), and Intuition's sprite is not compatible with that request, then graphics.library will blank Intuition's sprite and notify Intuition. Intuition will rebound by generating the most suitable pointer sprite which is compatible.
- ❑ Pens for Screens. The handling of defaults for the pen array is a bit involved because of compatibility issues. The only change for old applications should be those who pass an SA_Pens array of {~0}. They will get the new values for BARDETAILPEN, BARBLOCKPEN, and BARTRIMPEN. This will only affect the color of their screen title bar, and the color of the menus of any window on that screen which requests new-look menus (WFLG_NEWLOOKMENUS). Applications that specify no SA_Pens array or ones who specify an SA_Pens array with at least one explicit pen provided get V37-compatible defaults. Our options were limited because the request for default palette and default pens were too loosely coupled (until SA_LikeWorkbench).

Applications that use pen sharing on their own screens should allocate and set pens 3-7 if they plan on using those colors in console. The console device still uses pens 0-7 just as it always has.

Promotion-Related Issues

Under V39 and AA, when Mode Promotion is turned on, display modes which use the default monitor (e.g., display modes that are not explicitly PAL or NTSC) are promoted to scan-doubled or de-interlaced modes such as DblPAL or DblNTSC to provide a flicker-free display. Mode Promotion, on or off, is controlled system-wide by the IControl Preferences editor and the availability of the DblNTSC or DblPAL monitor.

Mode promotion can change the behavior of the system in a manner which may be incompatible with certain applications.

- ❑ The overscan limits of DblNTSC (DblPAL) are a little less than the overscan limits of NTSC (PAL). (For 3.01, the DblNTSC (DblPAL) limits have been extended and are now comparable to, but still less than, NTSC (PAL)).
- ❑ It may be harder to center a DblNTSC or DblPAL screen on certain multiscan monitors than it was to center a hardware de-interlaced NTSC or PAL screen. (3.01 provides additional centering flexibility which helps solve this problem).
- ❑ An interlaced screen is promoted to a non-interlaced screen, which has obvious implications on custom copper-lists and copper-list pokers.
- ❑ The higher resolutions/depths of the AA chipset require higher alignment restrictions on bitplanes. Fortunately, most applications either let Intuition allocate their screen's BitMap or else they have a custom BitMap whose width is a multiple of 64 pixels (the highest alignmentoverscan limits of NTSC (PAL). (For 3.01, the DblNTSC (DblPAL) limits have been extended and are now comparable to, but still less than, NTSC (PAL)).
- ❑ It may be harder to center a DblNTSC or DblPAL screen on certain multiscan monitors than it was to center a hardware de-interlaced NTSC or PAL screen. (3.01 provides additional centering flexibility which helps solve this problem).
- ❑ An interlaced screen is promoted to a non-interlaced screen, which has obvious implications on custom copper-lists and copper-list pokers.
- ❑ The higher resolutions/depths of the AA chipset require higher alignment restrictions on bitplanes. Fortunately, most applications either let Intuition allocate their screen's BitMap or else they have a custom BitMap whose width is a multiple of 64 pixels (the highest alignment currently required by AA). However, if the custom BitMap is an unusual width, it may not be sufficiently aligned for the hardware. Such a screen can come up skewed when promoted.
- ❑ Coercion of displays in back to match a promoted or explicit DblNTSC or DblPAL front display may result in a lower resolution mode for unsuitably aligned back displays.

"1x" modes require 16-pixel (word boundary) alignment of each scan-line. "2x" modes require 32-pixel (longword boundary) alignment, while "4x" modes require 64-pixel (double-longword boundary) alignment.

Here is a short mode reference:

- ❑ 140 ns pixels (lores in 15 kHz modes, extra-lores in 31 kHz modes)
  - 1-8 planes require 1X
- ❑ 70 ns pixels (hires in 15 kHz modes, lores in 31 kHz modes)
  - 1-4 planes require 1X
  - 5-8 planes require 2X
- ❑ 35 ns pixels (super-hires in 15 kHz modes, hires in 31 kHz modes)
  - 1-2 planes require 1X
  - 2-4 planes require 2X
  - 5-8 planes require 4X

As the graphics.library AllocBitMap() function takes care of allocating suitably-aligned BitMaps for you, you do not need to worry about alignment when using modern system calls.

- ❑ The AA hardware does not allow dual-playfield non-interlaced screens to be scan-doubled, so they will appear half as tall as their non-promoted counterparts.
- ❑ Like earlier chipsets, the AA chipset still supports eight sprites. In much the same way as ECS and original chipsets lose sprites when overscan is increased, many of the new modes have insufficient spare cycles to fetch data for these sprites. A promoted screen may have fewer sprites left than the corresponding 15 kHz mode, meaning that some sprites other than the pointer sprite may vanish.

Note - If the system is aware that additional sprites are needed, it will attempt to drop a display's bandwidth (if possible) to allow additional sprites. To cause this, either GetSprite() your sprites before opening your display, or RemakeDisplay() after doing your GetSprite() calls.

- ❑ There is currently no 31 kHz mode having 1280 pixels per line. That would require 17.5 ns pixel speeds, which is twice what the AA chipset is capable of. Therefore, SuperHires screens are promoted to 640 pixel-per-line screens, which generally can scroll. This is required for systems that have a VGA-only monitor, but otherwise might not be the desired way to display this mode. To prevent promotion of SuperHires to Hires, an application could explicitly ask for NTSC or PAL SuperHires. However, keep in mind that future chipsets may be capable of de-interlacing 1280-wide displays, and this would be defeated if NTSC or PAL is explicitly requested. Also remember that a user with a VGA-only monitor cannot display these modes. The V39BestModeIDA() function may be used to determine the best available mode.

- ❑ Applications that specifically request the NTSC or PAL monitor when opening a display will receive non-promoted PAL or NTSC, suitable for genlocking, etc. For interlace modes, this will not be a flicker-free display. Therefore, productivity and design applications that wish to offer an explicit PAL/NTSC choice should use the display database or 2.1 ASL screen mode requester to allow the user to choose a flicker-free display mode that is available on their system.

Other Compatibility Issues

- ❑ Library Vector Offsets and SetFunction(). In general, some new LVOs supersede old ones. While the old ones still work, software that calls SetFunction() based on the old LVOs may no longer catch what they were hoping to. An example from V37 was people who called SetFunction() on AutoRequest(). Most system requests go through EasyRequest() now. A V39 example would be SetWindowPointer() replacing SetPointer().
- ❑ Test for Memory by Poking Breaks. Apparently, some games test for RAM at $200000 by writing a pattern there and immediately reading it back. On the A1200 bus, such games will be fooled. Never test for RAM by poking/peeking. Always use Exec's memory allocation functions, or the Exec TypeOfMem() function which can tell the caller whether (and what type of) memory exists at any particular address.
- ❑ Changes to support for width-scaling of fonts Under V37, different widths of one YSize of a scalable font could be opened with a tagged OpenDiskFont() by first running a patch called setpatchwtam. Under V38, this patch program does nothing, and an incorrect already-opened width may be returned. Under V39, the correct requested width should again be returned, and these loaded width-scaled fonts are both hidden from AvailFonts() and should not be accidentally provided if application just does OpenFont() for the same YSize.
- ❑ Tablet driver writers should test that their tablets now work with console drag selection (known not to work in the 2.04 OS).
- ❑ SetMap is gone. The SetMap command (that set the keymap for the whole system) has been replaced by SetKeyBoard, that only sets the keymap for the current console. However, if you absolutely require it, the V37 SetMap command still works.
- ❑ Speech is gone. The narrator.device and translator.library are not part of V38 and V39. Related handlers and utilities are also gone. Therefore, new machines shipped with V38 or V39 do not have speech.

- ❑ The V39 Iffparse.library is not 1.3 compatible. Use the V37 library instead.
- ❑ Preferences file format changes The format of some Preferences files have changed of necessity to support new system capabilities, and these files may continue to change. The documentation which has been provided only shows some chunks which may be encountered in a system Preferences file. There is no guarantee that new chunks will not be added, or that the current chunks will continue to be used. Do not read or write system Preferences files. Instead use other provided system mechanisms for reading and setting such items (for example, use device-specific commands or structures for controlling device preferences, functions such as QueryOverscan() and LockPubScreen() for determining display characteristics, etc.)
- ❑ Self-modifying, copied, and directly loaded code Self-modifying code without proper cache control has been breaking since the 68020 was introduced. The larger caches of the 68030 and 68040 processors make it even more necessary to use the exec cache control functions such as CacheClearU() which have been available since V37. The cache control functions make it possible to flush the processor caches after modifying code in place, copying code to a different memory address, or placing code into memory via any mechanism that bypasses LoadSeg. Cache-control functions are also provided for DMA controllers which DMA data into memory.
- ❑ IFF code. Older IFF sample/example code contains many hardcoded limits and accesses some structures which may no longer be accessed directly. The older code (and even the code listed in the *Amiga ROM Kernel Reference Manual: Devices*, 3rd Edition) also would not load more than 32 color registers. The new IFF code, NewIFF39.lzh, attempts to use the latest system functions wherever possible in a conditional backward-compatible manner. The new code provides support for the full AA color palette, arbitrary display modes, the clipboard, overscan and autoscroll screens, and loading/saving of 24-bit ILBMs. See the ILBM Compatibility notes for additional information on ILBM compatibility and support of AA features.
- ❑ Pre/Post DMA Cache functions and 68040 For those of you with DMA devices that need to use the CachePreDMA() and CachePostDMA() calls: Be careful as incorrect use of these calls may look like they work fine in 68000/68020/68030 systems but may cause strange system behavior and even major system slowdown on 68040 systems.

The correct way to use these calls is:

```
original_length=length;
CachePreDMA(address,&length,0);

/* Optional, multiple ones here for breaking up DMA operations within the single
larger DMA block defined in the first call to CachePreDMA()
 */

CachePreDMA(....,....,DMA_Continue);

/*
 * Very important to call CachePostDMA() with the exact same values as CachePreDMA()
 */

CachePostDMA(address,&original_length,0)
```

Also, you must make sure they match up. If they do not, the system will not be able to figure out that the DMA that was starting has finished. Internally, the OS needs to track these things for the 68040 and higher processors and may need to track them even more in the future.

- ❑ An upcoming release of 3.0 will support the dos.library SetVBuf() function. From V36 to V39 this call did nothing. It will now properly change buffering modes, sizes, etc. For programs using Dos buffered I/O calls (FGetC, FPutC, FRead, etc.) this can make a major improvement in I/O performance if the buffer size is increased (and if the buffering mode is changed when writing to BUF_FULL).

For additional information on OS changes, take particular note of all BUGS and NOTES entries in the system autodocs.

Taking Advantage of New Features

By simply using system features and functions which were introduced in Release 2, you can write adaptable software that can automatically grow and support many new system software and hardware capabilities.

Add a bit of conditional code to treat color guns (R, G, and B) as at least 8-bits each, and to conditionally use the new 32-bit V39 color setting and getting functions, and your application can provide full AA palette support.

Here are some strategies for adapting to new system capabilities.

- ❑ Use the asl.library requesters if available. They are localized, they cut down on the amount of code you have to maintain, and they grow with the system. 2.0 Workbench licensees may get a free amendment to distribute the 2.1 asl.library which contains the screen mode requester.
- ❑ Adapt to previously impossible deeper modes. Properly written software can use the calling syntax of the current graphics and Intuition library functions to open the screen modes with larger numbers of bitplanes.

These modes are simply deeper bitplane versions of existing modes: Lores 6,7, and 8 bits; Hires 5,6,7, and 8 bits; SuperHires 3,4,5,6,7, and 8 bits; and VGA 3,4,5,6,7, and 8 bits.

Note that the display database must be checked to see if the user's machine supports the desired mode and number of bitplanes.

Intuition's screen functions can be used to open these previously impossible screens, but programs cannot rely on Intuition to throw out all "bad" possibilities. For example, V36 Intuition will open an 8-bitplane Lores screen, but this is a waste of memory if the new chipset is not present since the 3 extra bitplanes will simply go unused. Programs will therefore need to explicitly check the display database to find out what the chips support. In V39, Intuition will check the chips for you and fail to open a screen which is "too deep." There is a new SA_ErrorCode value, OSERR_TOODEEP.

Note on Mode IDs. The currently defined display database mode IDs have all been moved from <graphics/displayinfo.h> to the new include file <graphics/modeid.h>. Do not attempt to create your own display mode IDs by OR'ing together monitors and modes unless the include file explicitly allows it. Do not try to draw conclusions about display characteristics by interpreting bits in the display mode ID. Instead, pass the mode ID to the display database functions to discover the display characteristics (see the example below and other examples in Amiga Mail, DevCon Notes, *Amiga Rom Kernel Reference Manual: Libraries*, and the new screen mode requester example).

The complete example listed below will check the display database entry for a LORES screen and open a screen with the maximum allowable number of bitplanes (assuming there is enough Chip memory). On an Amiga with any chipset up to and including ECS, this code will open a 5-bitplane screen. On a machine with the AA chipset, the screen will be 8 bits deep. The code will then draw one vertical line for each available color using Intuition's preferred palette (which can be set by the user with Preferences). The example requires V36 or a later version of the OS.

```
/* maxdepthlores.c  */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/displayinfo.h>
#include <dos/dos.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/dos_protos.h>
```

```
#include <stdio.h>
#include <stdlib.h>

struct Library *IntuitionBase;
struct Library *GfxBase;

void Quit(char *whytext,LONG failcode)
{
if(*whytext)
    printf("%s",whytext);

if (IntuitionBase)
    CloseLibrary(IntuitionBase);

if (GfxBase)
    CloseLibrary(GfxBase);

exit(failcode);
}

void main(void)
{
ULONG modeID = LORES_KEY;
DisplayInfoHandle displayhandle;
struct DimensionInfo dimensioninfo;

UWORD maxdepth, maxcolors;
ULONG soerror = NULL,colornum;
struct Screen *screen;

if ((GfxBase = OpenLibrary("graphics.library",36))==NULL)
        Quit("graphics.library is too old <V36",RETURN_FAIL);

if ((IntuitionBase = OpenLibrary("intuition.library",36))==NULL)
        Quit("intuition.library is too old <V36",RETURN_FAIL);

if ((displayhandle=FindDisplayInfo(modeID))==NULL)
        Quit("modeID not found in display database",RETURN_FAIL);

if (GetDisplayInfoData(displayhandle,(UBYTE *)&dimensioninfo,
                            sizeof(struct DimensionInfo),DTAG_DIMS,NULL)==0)
        Quit("mode dimension info not available",RETURN_FAIL);

maxdepth=dimensioninfo.MaxDepth;
printf("dimensioninfo.MaxDepth=%d",(int) maxdepth);

if (screen=OpenScreenTags(NULL, SA_DisplayID ,modeID,
                                 SA_Depth, (UBYTE) maxdepth,
                                 SA_Title, "MaxDepth LORES",
                                 SA_ErrorCode, &soerror,
                                 SA_FullPalette, TRUE,
                                 TAG_END))

    {
    /* Zowee! we actually got the screen open! now let's try drawing into it. */
    maxcolors=1<<maxdepth;
    printf("maxcolors=%d",(int) maxcolors);

    for(colornum=0;colornum<maxcolors;++colornum)
        {
        SetAPen(&(screen->RastPort),colornum);
        Move(&(screen->RastPort),colornum,screen->BarHeight + 2);
        Draw(&(screen->RastPort),colornum,screen->Height - 1);
        }

    Delay(TICKS_PER_SECOND * 6);
    CloseScreen(screen);
    }
else
    {
    /* Hmmm.  Couldn't open the screen.  maybe not enough CHIP RAM?
     * Maybe not enough chips! ;-)
     */
    switch(soerror)
        {
        case OSERR_NOCHIPS:
            Quit("Bummer! You need new chips dude!",RETURN_FAIL);
            break;
        case OSERR_UNKNOWNMODE:
            Quit("Bummer! Unknown screen mode.",RETURN_FAIL);
            break;
        case OSERR_NOCHIPMEM:
            Quit("Not enough CHIP memory.",RETURN_FAIL);
            break;
        case OSERR_NOMEM:
            Quit("Not enough FAST memory.",RETURN_FAIL);
            break;
        default:
            printf("soerror=%d",soerror);
            Quit("Screen opening error.",RETURN_FAIL);
            break;
        }
    Quit("Couldn't open screen.",RETURN_FAIL);
    }

Quit("",RETURN_OK);  /* clean up and exit */
}
```

❑ Open on Workbench. One of the easiest ways to help a given program peacefully coexist with new hardware is to allow it to open on the Workbench screen (or on any public screen). Software written this way should be robust enough that it can find out from the system any information it might need about the display environment, and, if it understands the kind of screen that it's on, use the appropriate graphics.library calls to manipulate its windows.

Here's some example code that determines the depth of the default public screen and opens a window on it. If it were part of a real application, this code would tell how many colors the screen has.

Add some conditional V39 palette sharing code and you can even have your own color registers to use, if available, or closest matching register to share.

```
/* depthawarevisitor.c */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/displayinfo.h>
#include <dos/dos.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/dos_protos.h>

#include <stdio.h>
#include <stdlib.h>

struct Library *IntuitionBase;
struct Library *GfxBase;
struct Screen  *screen = NULL;

void Quit(char *whytext, LONG failcode)
{
```

```
if(*whytext)
     printf("%s",whytext);

if (screen)
    UnlockPubScreen(NULL, screen);

if (IntuitionBase)
    CloseLibrary(IntuitionBase);

if (GfxBase)
    CloseLibrary(GfxBase);

exit(failcode);
}

void main(void)
{
struct Screen *screen;
struct DrawInfo *drawinfo;
struct Window *window;
UWORD depth;

if ((GfxBase = OpenLibrary("graphics.library",36))==NULL)
    Quit("graphics.library is too old <V36",RETURN_FAIL);

if ((IntuitionBase = OpenLibrary("intuition.library",36))==NULL)
    Quit("intuition.library is too old <V36",RETURN_FAIL);

if (!(screen = LockPubScreen(NULL)))
    Quit("Can't lock default public screen",RETURN_FAIL);

/* Here's where we'll ask Intuition about the screen. */
if((drawinfo=GetScreenDrawInfo(screen)) == NULL)
    Quit("Can't get DrawInfo",RETURN_FAIL);
depth=drawinfo->dri_Depth;

/* Because Intuition allocates the DrawInfo structure, we have to tell it when          *
we're done, to get the memory back.
 */
FreeScreenDrawInfo(screen, drawinfo);
/* This next line takes advantage of the stack-based amiga.lib version of    *
OpenWindowTags().
 */
if (window = OpenWindowTags(NULL, WA_PubScreen ,screen,
                                  WA_Left, 0,
                                  WA_Width, screen->Width,
                                  WA_Top, screen->BarHeight,
                                  WA_Height, screen->Height - screen->BarHeight,
                                  WA_Flags, WINDOWDRAG|WINDOWDEPTH|WINDOWCLOSE|
                                            ACTIVATE|SIMPLE_REFRESH|NOCAREREFRESH,
                                  WA_Title, "Big Visitor",
                                  TAG_END))
    {

    printf("depth=%d",depth);

    /* All our window event handling might go here */

    Delay(TICKS_PER_SECOND * 10);

    /* Of course, some other program might come along and change the attributes of
     * the screen that we read from DrawInfo, but that's a mean thing to do to a
     * public screen, so let's hope it doesn't happen.
     */

    CloseWindow(window);
    }

Quit("",RETURN_OK);    /* clean up (close/unlock) and exit */
}
```

❑ If necessary, use tag-extended older structures with 1.3 functions. If you are not yet able to drop support for 1.3 systems, use tag-extended structures such as ExtNewScreen and ExtNewWindow to provide V37 and V39/AA enhanced capabilities with 1.3-compatible code. See the Intuition chapters of the *Amiga ROM Kernel Reference Manual: Libraries* for examples that use the tag-extended structures. An example of such coding is the NewIFF39 code.

New AA Display Modes (in addition to modes supported by ECS)

Display Mode	Planes	Colors	Bandwidth (see note 1)
LORES	6	64	1x
(320 x 200 NTSC	7	128	1x
or 320 x 256 PAL)	8	256	1x
	8 HAM	256,000+ (see note 2)	1x
HIRES	5	32	2x
(640 x 200 NTSC	6 EHB	64 (see note 3)	2x
or 640 x 256 PAL)	6 HAM	4096 (see note 4)	2x
	6	64	2x
	7	128	2x
	8	256	2x
	8 HAM	256,000+ (see note 2)	2x
SUPERHIRES	1	2 (see note 5)	1x
(1280 x200 NTSC	2	4 (see note 5)	1x
or 1280 x 256 PAL)	3	8	2x
	4	16	2x
	5	32	4x
	6 EHB	64 (see note 3)	4x
	6 HAM	4096 (see note 4)	4x
	6	64	4x
	7	128	4x
	8	256	4x
	8 HAM	256,000+ (see note 2)	4x
MULTISCAN	1	2 (see note 5)	1x
(160, 320, 640 x 480	2	4 (see note 5)	1x
non-interlaced)	3	8	2x
	4	16	2x
	5	32	4x
	6 EHB	64 (see note 3)	4x
	6 HAM	4096 (see note 4)	4x
	6	64	4x
	7	128	4x
	8	256	4x
	8 HAM	256,000+ (see note 2)	4x

New AA Display Modes (in addition to modes supported by ECS)

Display Mode	Planes	Colors		Bandwidth (see note 1)
SUPER72	1	2	(see note 5)	1x
(71 Hz refresh and	2	4	(see note 5)	1x
23.21 kHz scan rate;	3	8		2x
200, 400, 800 x 300	4	16		2x
non-interlaced or	5	32		4x
200, 400, 800 x 600	6 EHB	64	(see note 3)	4x
interlaced)	6 HAM	4096	(see note 4)	4x
	6	64		4x
	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x
EURO72	1	2	(see note 5)	1x
(69 Hz refresh and	2	4	(see note 5)	1x
29.32 kHz scan rate;	3	8		2x
160, 320, 640 x 480	4	16		2x
non-interlaced)	5	32		4x
	6 EHB	64	(see note 3)	4x
	6 HAM	4096	(see note 4)	4x
	6	64		4x
	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x
DOUBLENTSC	1	2	(see note 5)	1x
(58 Hz refresh and	2	4	(see note 5)	1x
27.66 kHz scan rate;	3	8		2x
160, 320, 640 x 280	4	16		2x
scan-doubled or	5	32		4x
160, 320, 640 x 480	6 EHB	64	(see note 3)	4x
non-interlaced or	6 HAM	4096	(see note 4)	4x
160, 320, 640 x 800	6	64		4x
interlaced)	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x

New AA Display Modes (in addition to modes supported by ECS)

Display Mode	Planes	Colors		Bandwidth (see note 1)
DOUBLEPAL	1	2	(see note 5)	1x
(58Hz refresh and	2	4	(see note 5)	1x
27.66 kHz scan rate;	3	8		2x
160, 320, 640 x 256	4	16		2x
scan-doubled or	5	32		4x
160, 320, 640 x 512	6 EHB	64	(see note 3)	4x
non-interlaced or	6 HAM	4096	(see note 4)	4x
160, 320, 640 x 1024	6	64		4x
interlaced)	7	128		4x
	8	256		4x
	8 HAM	256,000+	(see note 2)	4x

Notes

1 - The bandwidth number describes the relative amount of fetch bandwidth required by a particular screen mode. For example, a 5-bit deep MultiScan screen requires 4x fetch bandwidth while a 1-bit MultiScan screen requires only 1x. This means the hardware has to move data 4 times faster on the deeper screen. To be able to move data at these higher rates, the higher bandwidth modes require data to be properly aligned in Chip memory that is fast enough to support the bandwidth. Specifically, bandwidth factors of 1x require data to be on 16-bit boundaries, factors of 2x require 32-bit boundaries, and factors of 4x require 64-bit boundaries.

Restrictions like these are the best reason to use the system allocation functions whenever data is being prepared for the custom hardware. It is not guaranteed that all machines that have the new chip set will also have memory fast enough for the 4x modes. Therefore, the only way to know whether or not the machine will support the mode you want is to check the display database.

2 - New 8-bit HAM mode uses the upper 6 bits as an offset into a table of 64 24-bit base color registers and the lower 2 bits for modify mode control. This mode could conceivably allow simultaneous display of more than 256,000 colors (up to 16.8 million, presuming a monitor and screen mode with enough pixels). Please note that while the register planes and control planes are internally reversed in 8-bit HAM (the modify control bits are the two LSBs instead of the two MSBs), programs using graphics library and Intuition will not have to deal with this reversal, as it will be handled automatically for them.

3 - This is like the original EHB mode, but in new resolutions. It uses 5 bits as an offset into a table of 32 color registers plus a sixth bit to yield an additional 32 colors that are 1/2 as bright.

4 - This is like the original 6-bit HAM mode, but in new resolutions. It uses the lower 4 bits as an offset into a table of 16 color registers and the upper 2 bits for modify mode control. This mode allows simultaneous display of 4096 colors.

5 - These modes are similar to the VGA and SuperHires modes of the ECS chipset except that they are not restricted to a nonstandard 64-color palette.

◆

ARexx

by Michael Sinz, Christian Ludwig & Jerry Hartzler

SimpleRexx: A Simple ARexx Interface

by Michael Sinz

The ability to handle scripts is a powerful feature for any application. Whenever users have a repetitive, well-defined job to do with application software, script capabilities allow them to perform the job automatically without any user intervention. For example, many communication programs allow the user to set up a script which will automatically dial up a host system, login to it, and check for messages to download. To encourage the development of more applications which have this powerful feature, Commodore has added ARexx to V2.0 of the Amiga system software.

What Is ARexx?

ARexx is the Amiga implementation of the REXX language. Like BASIC, ARexx is an interpreted language. ARexx can be used alone for almost any programming job, but the real power of ARexx is unleashed when it is used with applications as a system-level scripting language. With the addition of its own ARexx port and a small amount of code, an application can gain all the power scripting capabilities offer. There are other benefits as well. The burden of writing code to handle scripts is reduced significantly. Also, ARexx helps provide a consistent interface to the user. Without it, every developer who implements scripts must invent their own scripting language, forcing the user to learn several different scripting languages rather than just one.

Perhaps best of all is that applications which support ARexx can ''talk'' to each other by sending commands and sharing data - even if they are from different vendors. For instance, a communications program with an ARexx port could be set up to automatically download data from a financial bulletin board and then pass the data to a spreadsheet application to create a graph or financial model. The ability of applications to talk to one another on a multitasking computer is known as Inter-process Communication (IPC).

Because it interprets scripts and passes strings, ARexx has been optimized for string processing. Almost all ARexx interactions involve passing a string; numbers are even passed as ASCII representations in most situations.

[illegible]
file shows the output of test.rexx.

In addition to the code examples listed here, you will need to install the rexxsyslib.library and rexxsupport.library in the libs: directory of the target machine. Don't forget to give the rexxmast command either in your startup-sequence or from the CLI in order to start ARexx.

Overview of Functions

The source to SimpleRexx is a single file, SimpleRexx.c. The header file, SimpleRexx.h, contains the type definitions and prototypes for the functions.

The SimpleRexx functions are used as follows:

```
rexx_context=InitARexx(AppBaseName, Extension)
```

```
worked = SetARexxLastError(rexx_content, rmsg,ErrorString)
```

This function uses the RVI (Rexx Variable Interface) to set a variable named <AppBaseName>.LASTERROR to the ErrorString. This is where the error message should go if there is an error. This function returns FALSE if this fails. You must call this routine after a GetARexxMsg() and before ReplyARexxMsg().

```
FreeARexx(rexx_context)
```

This closes a SimpleRexx context received from InitARexx().

The Future of ARexx

ARexx is much more than a system-level scripting language. Among other things, ARexx can be used like glue to put together a set of application modules. This frees the programmer from worrying about data sharing between modules so that more effort can be put into refining the code modules themselves. It also allows a user who understands ARexx to add their own refinements creating a truly custom environment. This customizability is also ideal for VARs or dealers who want to sell vertical market solutions.

```
#
# Makefile for SimpleRexx and SimpleRexxExample
#

CFLAGS= -b1 -cfist -d0 -ms0 -rr1 -v -w

HEAD= SimpleRexx.h

CODE= SimpleRexx.c SimpleRexxExample.c

OBJS= SimpleRexx.o SimpleRexxExample.o

LIBS= LIB:lcsr.lib rexxvars.o LIB:amiga.lib

.c.o:
  $LC $(CFLAGS) $*

SimpleRexxExample: $(OBJS) $(LIBS)
  BLink FROM LIB:c.o $(OBJS) TO SimpleRexxExample LIB $(LIBS) SD SC ND

SimpleRexx.o: SimpleRexx.c SimpleRexx.h

SimpleRexxExample.o: SimpleRexxExample.c SimpleRexx.h


/*
 * file: simplerexx.h
 *
 * Simple ARexx interface...
 *
 * This is a very "Simple" interface...
 */

#ifndef SIMPLE_REXX_H
#define SIMPLE_REXX_H

#include  <exec/types.h>
#include  <exec/nodes.h>
#include  <exec/lists.h>
#include  <exec/ports.h>

#include  <rexx/storage.h>
#include  <rexx/rxslib.h>

/*
 * This is the handle that SimpleRexx will give you
 * when you initialize an ARexx port...
 *
 * The conditional below is used to skip this if we have
 * defined it earlier...
 */
#ifndef AREXXCONTEXT

typedef void *AREXXCONTEXT;

#endif  /* AREXXCONTEXT */

/*
 * The value of RexxMsg (from GetARexxMsg) if there was an error returned
 */
#define REXX_RETURN_ERROR ((struct RexxMsg *)-1L)

/*
 * This function closes down the ARexx context that was opened
 * with InitARexx...
 */
void FreeARexx(AREXXCONTEXT);

/*
 * This routine initializes an ARexx port for your process
 * This should only be done once per process.  You must call it
 * with a valid application name and you must use the handle it
 * returns in all other calls...
 *
 * NOTE:  The AppName should not have spaces in it...
 *        Example AppNames:  "MyWord" or "FastCalc" etc...
 *        The name *MUST* be less that 16 characters...
 *        If it is not, it will be trimmed...
 *        The name will also be UPPER-CASED...
 *
 * NOTE:  The Default file name extension, if NULL will be
 *        "rexx"  (the "." is automatic)
 */
AREXXCONTEXT InitARexx(char *,char *);

/*
 * This function returns the port name of your ARexx port.
 * It will return NULL if there is no ARexx port...
 *
 * This string is *READ ONLY*  You *MUST NOT* modify it...
 */
char *ARexxName(AREXXCONTEXT);

/*
 * This function returns the signal mask that the Rexx port is
 * using.  It returns NULL if there is no signal...
 *
 * Use this signal bit in your Wait() loop...
 */
ULONG ARexxSignal(AREXXCONTEXT);

/*
 * This function returns a structure that contains the commands sent from
 * ARexx...  You will need to parse it and return the structure back
 * so that the memory can be freed...
 *
 * This returns NULL if there was no message...
 */
struct RexxMsg *GetARexxMsg(AREXXCONTEXT);

/*
 * Use this to return a ARexx message...
 *
 * If you wish to return something, it must be in the RString.
 * If you wish to return an Error, it must be in the Error.
 */
void ReplyARexxMsg(AREXXCONTEXT,struct RexxMsg *,char *,LONG);

/*
 * This function will send a string to ARexx...
 *
 * The default host port will be that of your task...
 *
 * If you set StringFile to TRUE, it will set that bit for the message...
 *
 * Returns TRUE if it send the message, FALSE if it did not...
 */
short SendARexxMsg(AREXXCONTEXT,char *,short);

/*
 * This function will set an error string for the ARexx
 * application in the variable defined as <appname>.LASTERROR
 *
 * Note that this can only happen if there is an ARexx message...
 *
 * This returns TRUE if it worked, FALSE if it did not...
 */
short SetARexxLastError(AREXXCONTEXT,struct RexxMsg *,char *);

#endif  /* SIMPLE_REXX_H */


/*
 * File: SimpleRexx.c
 *
 * Simple ARexx interface...
 * This is a very "simple" interface to the world of ARexx...
 * For more complex interfaces into ARexx, it is best that you
 * understand the functions that are provided by ARexx.
 * In many cases they are more powerful than what is presented
 * here.
 *
 * This code is fully re-entrant and self-contained other than
 * the use of SysBase/AbsExecBase and the ARexx RVI support
 * library which is also self-contained...
 */

#include  <exec/types.h>
#include  <exec/nodes.h>
#include  <exec/lists.h>
#include  <exec/ports.h>
#include  <exec/memory.h>

#include  <proto/exec.h>

#include  <rexx/storage.h>
#include  <rexx/rxslib.h>

#include  <string.h>
#include  <ctype.h>

/*
 * The prototypes for the few ARexx functions we will call...
 */
struct RexxMsg *CreateRexxMsg(struct MsgPort *,char *,char *);
void *CreateArgstring(char *,long);
void DeleteRexxMsg(struct RexxMsg *);
void DeleteArgstring(char *);
BOOL IsRexxMsg(struct Message *);

/*
 * Pragmas for the above functions...  (To make this all self-contained...)
 * If you use RexxGlue.o, this is not needed...
 *
 * These are for Lattice C 5.x  (Note the use of RexxContext->RexxSysBase)
 */
#pragma libcall RexxContext->RexxSysBase CreateRexxMsg 90 09803
#pragma libcall RexxContext->RexxSysBase CreateArgstring 7E 0802
#pragma libcall RexxContext->RexxSysBase DeleteRexxMsg 96 801
#pragma libcall RexxContext->RexxSysBase DeleteArgstring 84 801
#pragma libcall RexxContext->RexxSysBase IsRexxMsg A8 801

/*
 * Prototypes for the RVI ARexx calls...  (link with RexxVars.o)
 */
__stdargs long CheckRexxMsg(struct RexxMsg *);
__stdargs long GetRexxVar(struct RexxMsg *,char *,char **);
__stdargs long SetRexxVar(struct RexxMsg *,char *,char *,long);

/*
 * Now, we have made the pragmas needed, let's get to work...
 */

/*
 * A structure for the ARexx handler context
 * This is *VERY* *PRIVATE* and should not be touched...
 */
struct  ARexxContext
{
struct  MsgPort *ARexxPort; /* The port messages come in at... */
struct  Library *RexxSysBase; /* We will hide the library pointer here... */
  long  Outstanding;  /* The count of outstanding ARexx messages... */
  char  PortName[24]; /* The port name goes here... */
  char  ErrorName[28];  /* The name of the <base>.LASTERROR... */
  char  Extension[8]; /* Default file name extension... */
};

#define AREXXCONTEXT  struct ARexxContext *
```

```
#include  "SimpleRexx.h"

/*
 * This function returns the port name of your ARexx port.
 * It will return NULL if there is no ARexx port...
 *
 * This string is *READ ONLY*  You *MUST NOT* modify it...
 */
char *ARexxName(AREXXCONTEXT RexxContext)
{
register  char  *tmp=NULL;

  if (RexxContext) tmp=RexxContext->PortName;
  return(tmp);
}

/*
 * This function returns the signal mask that the Rexx port is
 * using.  It returns NULL if there is no signal...
 *
 * Use this signal bit in your Wait() loop...
 */
ULONG ARexxSignal(AREXXCONTEXT RexxContext)
{
register  ULONG tmp=NULL;

  if (RexxContext) tmp=1L << (RexxContext->ARexxPort->mp_SigBit);
  return(tmp);
}

/*
 * This function returns a structure that contains the commands sent from
 * ARexx...  You will need to parse it and return the structure back
 *     * so that the memory can be freed...
 *
 * This returns NULL if there was no message...
 */
struct RexxMsg *GetARexxMsg(AREXXCONTEXT RexxContext)
{
register  struct  RexxMsg *tmp=NULL;
register    short flag;

  if (RexxContext)
      if (tmp=(struct RexxMsg *)GetMsg(RexxContext->ARexxPort))
  {
    if (tmp->rm_Node.mn_Node.ln_Type==NT_REPLYMSG)
    {
      /*
       * If we had sent a command, it would come this way...
       *
       * Since we don't in this simple example, we just throw
       * away anything that looks "strange"
       */
      flag=FALSE;
      if (tmp->rm_Result1) flag=TRUE;

      /*
       * Free the arguments and the message...
       */
      DeleteArgstring(tmp->rm_Args[0]);
      DeleteRexxMsg(tmp);
      RexxContext->Outstanding-=1;

      /*
       * Return the error if there was one...
       */
      tmp=flag ? REXX_RETURN_ERROR : NULL;
    }
  }
  return(tmp);
}

/*
 * Use this to return a ARexx message...
 *
 * If you wish to return something, it must be in the RString.
 * If you wish to return an Error, it must be in the Error.
 * If there is an error, the RString is ignored.
 */
void ReplyARexxMsg(AREXXCONTEXT RexxContext,struct RexxMsg *rmsg,
      char *RString,LONG Error)
{
  if (RexxContext) if (rmsg) if (rmsg!=REXX_RETURN_ERROR)
  {
    rmsg->rm_Result2=0;
    if (!(rmsg->rm_Result1=Error))
    {
      /*
       * if you did not have an error we return the string
       */
      if (rmsg->rm_Action & (1L << RXFB_RESULT)) if (RString)
      {
        rmsg->rm_Result2=(LONG)CreateArgstring(RString,
                (LONG)strlen(RString));
      }
    }

    /*
     * Reply the message to ARexx...
     */
    ReplyMsg((struct Message *)rmsg);
  }
}

/*
 * This function will set an error string for the ARexx
 * application in the variable defined as <appname>.LASTERROR
 *
 * Note that this can only happen if there is an ARexx message...
 *
 * This returns TRUE if it worked, FALSE if it did not...
 */
short SetARexxLastError(AREXXCONTEXT RexxContext,struct RexxMsg *rmsg,
      char *ErrorString)
{
register  short OkFlag=FALSE;

  if (RexxContext) if (rmsg) if (CheckRexxMsg(rmsg))
  {
    /*
     * Note that SetRexxVar() has more than just a TRUE/FALSE
     * return code, but for this "basic" case, we just care if
     * it works or not.
     */
    if (!SetRexxVar(rmsg,RexxContext->ErrorName,ErrorString,
          (long)strlen(ErrorString)))
    {
      OkFlag=TRUE;
    }
  }
  return(OkFlag);
}

/*
 * This function will send a string to ARexx...
 *
 * The default host port will be that of your task...
 *
 * If you set StringFile to TRUE, it will set that bit for the message...
 *
 * Returns TRUE if it send the message, FALSE if it did not...
 */
short SendARexxMsg(AREXXCONTEXT RexxContext,char *RString,
      short StringFile)
{
register  struct  MsgPort *RexxPort;
register  struct  RexxMsg *rmsg;
register    short flag=FALSE;
```

```
  if (RexxContext) if (RString)
  {
    if (rmsg=CreateRexxMsg(RexxContext->ARexxPort,
          RexxContext->Extension,
          RexxContext->PortName))
    {
      rmsg->rm_Action=RXCOMM | (StringFile ?
            (1L << RXFB_STRING):0);
      if (rmsg->rm_Args[0]=CreateArgstring(RString,
            (LONG)strlen(RString)))
      {
        /*
         * We need to find the RexxPort and this needs
         * to be done in a Forbid()
         */
        Forbid();
        if (RexxPort=FindPort(RXSDIR))
        {
          /*
           * We found the port, so put the
           * message to ARexx...
           */
          PutMsg(RexxPort,(struct Message *)rmsg);
          RexxContext->Outstanding+=1;
          flag=TRUE;
        }
        else
        {
          /*
           * No port, so clean up...
           */
          DeleteArgstring(rmsg->rm_Args[0]);
          DeleteRexxMsg(rmsg);
        }
        Permit();
      }
      else DeleteRexxMsg(rmsg);
    }
  }
  return(flag);
}

/*
 * This function closes down the ARexx context that was opened
 * with InitARexx...
 */
void FreeARexx(AREXXCONTEXT RexxContext)
{
register  struct  RexxMsg *rmsg;

  if (RexxContext)
  {
    /*
     * Clear port name so it can't be found...
     */
    RexxContext->PortName[0]='0';

    /*
     * Clean out any outstanding messages we had sent out...
     */
    while (RexxContext->Outstanding)
    {
      WaitPort(RexxContext->ARexxPort);
      while (rmsg=GetARexxMsg(RexxContext))
      {
        if (rmsg!=REXX_RETURN_ERROR)
        {
          /*
           * Any messages that come now are blown
           * away...
           */
          SetARexxLastError(RexxContext,rmsg,
                "99: Port Closed!");
				ReplyARexxMsg(RexxContext,rmsg,
					NULL,100);
			}
		}
	}

	/*
	 * Clean up the port and delete it...
	 */
	if (RexxContext->ARexxPort)
	{
		while (rmsg=GetARexxMsg(RexxContext))
		{
			/*
			 * Any messages that still are coming in are
			 * "dead"  We just set the LASTERROR and
			 * reply an error of 100...
			 */
			SetARexxLastError(RexxContext,rmsg,
				"99: Port Closed!");
			ReplyARexxMsg(RexxContext,rmsg,NULL,100);
		}
		DeletePort(RexxContext->ARexxPort);
	}

	/*
	 * Make sure we close the library...
	 */
	if (RexxContext->RexxSysBase)
	{
		CloseLibrary(RexxContext->RexxSysBase);
	}

	/*
	 * Free the memory of the RexxContext
	 */
	FreeMem(RexxContext,sizeof(struct ARexxContext));
  }
}

/*
 * This routine initializes an ARexx port for your process
 * This should only be done once per process.  You must call it
 * with a valid application name and you must use the handle it
 * returns in all other calls...
 *
 * NOTE:  The AppName should not have spaces in it...
 *        Example AppNames:  "MyWord" or "FastCalc" etc...
 *        The name *MUST* be less that 16 characters...
 *        If it is not, it will be trimmed...
 *        The name will also be UPPER-CASED...
 *
 * NOTE:  The Default file name extension, if NULL will be
 *        "rexx"  (the "." is automatic)
 */
AREXXCONTEXT InitARexx(char *AppName,char *Extension)
{
register  AREXXCONTEXT  RexxContext=NULL;
register  short   loop;
register  short   count;
register  char    *tmp;

  if (RexxContext=AllocMem(sizeof(struct ARexxContext),
          MEMF_PUBLIC|MEMF_CLEAR))
  {
    if (RexxContext->RexxSysBase=OpenLibrary("rexxsyslib.library",
                    NULL))
    {
      /*
       * Set up the extension...
       */
      if (!Extension) Extension="rexx";
      tmp=RexxContext->Extension;
      for (loop=0;(loop<7)&&(Extension[loop]);loop++)
      {
        *tmp++=Extension[loop];
      }
      *tmp='0';

      /*
       * Set up a port name...
       */
      tmp=RexxContext->PortName;
      for (loop=0;(loop<16)&&(AppName[loop]);loop++)
      {
        *tmp++=toupper(AppName[loop]);
      }
      *tmp='0';

      /*
       * Set up the last error RVI name...
       *
       * This is <appname>.LASTERROR
       */
      strcpy(RexxContext->ErrorName,RexxContext->PortName);
      strcat(RexxContext->ErrorName,".LASTERROR");

      /* We need to make a unique port name... */
      Forbid();
      for (count=1,RexxContext->ARexxPort=(VOID *)1;
            RexxContext->ARexxPort;count++)
      {
        stci_d(tmp,count);
        RexxContext->ARexxPort=
            FindPort(RexxContext->PortName);
      }

      RexxContext->ARexxPort=CreatePort(
            RexxContext->PortName,NULL);
      Permit();
    }

    if (  (!(RexxContext->RexxSysBase))
        || (!(RexxContext->ARexxPort))  )
    {
      FreeARexx(RexxContext);
      RexxContext=NULL;
    }
  }
  return(RexxContext);
}


/*
 * File: SimpleRexxExample.c
 *
 * This is an example of how to use the SimpleRexx code.
 */

#include  <exec/types.h>
#include  <libraries/dos.h>
#include  <libraries/dosextens.h>
#include  <intuition/intuition.h>

#include  <proto/exec.h>
#include  <proto/intuition.h>

#include  <rexx/storage.h>
#include  <rexx/rxslib.h>

#include  <stdio.h>
#include  <string.h>

#include  "SimpleRexx.h"

/*
```

```
 * Lattice control-c stop...
 */
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */

/*
 * The strings in this program
 */
char *strings[]=
{
  "50: Window already open",     /* STR_ID_WINDOW_OPEN */
  "101: Window did not open",    /* STR_ID_WINDOW_ERROR */
  "50: No Window",       /* STR_ID_WINDOW_NONE */
  "80: Argument error to WINDOW command", /* STR_ID_WINDOW_ARG */
  "100: Unknown command",      /* STR_ID_COMMAND_ERROR */
  "ARexx port name: %s",     /* STR_ID_PORT_NAME */
  "No ARexx on this system.",    /* STR_ID_NO_AREXX */
  "SimpleRexxExample Window"     /* STR_ID_WINDOW_TITLE */
};

#define STR_ID_WINDOW_OPEN  0
#define STR_ID_WINDOW_ERROR 1
#define STR_ID_WINDOW_NONE  2
#define STR_ID_WINDOW_ARG 3
#define STR_ID_COMMAND_ERROR  4
#define STR_ID_PORT_NAME  5
#define STR_ID_NO_AREXX   6
#define STR_ID_WINDOW_TITLE 7

/*
 * NewWindow structure...
 */
static struct NewWindow nw=
{
  97,47,299,44,-1,-1,
  CLOSEWINDOW,
  WINDOWSIZING|WINDOWDRAG|WINDOWDEPTH|WINDOWCLOSE|
 SIMPLE_REFRESH|NOCAREREFRESH,
  NULL,NULL,
  NULL,
  NULL,NULL,290,40,-1,-1,WBENCHSCREEN
};

/*
 * A *VERY* simple and simple-minded example of using the SimpleRexx.c code.
 *
 * This program, when run, will print out the name of the ARexx port it
 * opens.  Use that port to tell it to SHOW the window.  You can also
 * use the ARexx port to HIDE the window, to READTITLE the window's
 * titlebar, and QUIT the program.  You can also quit the program by
 * pressing the close gadget in the window while the window is up.
 *
 * Note: You will want to RUN this program or have another shell available such
 *       that you can still have access to ARexx...
 */
void main(int argc,char *argv[])
{
short loopflag=TRUE;
AREXXCONTEXT  RexxStuff;
struct  Window  *win=NULL;
ULONG signals;

  if (IntuitionBase=(struct IntuitionBase *)
          OpenLibrary("intuition.library",NULL))
  {
    /*
     * Note that SimpleRexx is set up such that you do not
     * need to check for an error to initialize your REXX port
     * This is so your application could run without REXX...
     */
    RexxStuff=InitARexx("Example","test");

    if (argc)
    {
```

```
    if (RexxStuff) printf(strings[STR_ID_PORT_NAME],
          ARexxName(RexxStuff));
    else printf(strings[STR_ID_NO_AREXX]);
  }

  while (loopflag)
  {
    signals=ARexxSignal(RexxStuff);
    if (win) signals|=(1L << (win->UserPort->mp_SigBit));

    if (signals)
    {
    struct  RexxMsg   *rmsg;
    struct  IntuiMessage  *msg;

      signals=Wait(signals);

      /*
       * Process the ARexx messages...
       */
      while (rmsg=GetARexxMsg(RexxStuff))
      {
      char  cBuf[24];
      char  *nextchar;
      char  *error=NULL;
      char  *result=NULL;
      long  errlevel=0;

        nextchar=stptok(ARG0(rmsg),
              cBuf,24," ,");
        if (*nextchar) nextchar++;

        if (!stricmp("WINDOW",cBuf))
        {
          if (!stricmp("OPEN",nextchar))
          {
            if (win)
            {
              error=strings[STR_ID_WINDOW_OPEN];
              errlevel=5;
            }
            else
            {
              nw.Title=strings[STR_ID_WINDOW_TITLE];
              if (!(win=OpenWindow(&nw)))
              {
                error=strings[STR_ID_WINDOW_ERROR];
                errlevel=30;
              }
            }
          }
          else if (!stricmp("CLOSE",nextchar))
          {
            if (win)
            {
              CloseWindow(win);
              win=NULL;
            }
            else
            {
              error=strings[STR_ID_WINDOW_NONE];
              errlevel=5;
            }
          }
          else
          {
            error=strings[STR_ID_WINDOW_ARG];
            errlevel=20;
          }
        }
        else if (!stricmp("READTITLE",cBuf))
        {
          if (win)
          {
              result=win->Title;
            }
            else
            {
              error=strings(STR_ID_WINDOW_NONE);
              errlevel=5;
            }
          }
          else if (!stricmp("QUIT",cBuf))
          {
            loopflag=FALSE;
          }
          else
          {
            error=strings(STR_ID_COMMAND_ERROR);
            errlevel=20;
          }

          if (error)
          {
            SetARexxLastError(RexxStuff,rmsg,error);
          }
          ReplyARexxMsg(RexxStuff,rmsg,result,errlevel);
        }

        /*
             * If we have a window, process those messages
         */
        if (win) while (msg=(struct IntuiMessage *)
             GetMsg(win->UserPort))
        {
          if (msg->Class==CLOSEWINDOW)
          {
            /*
             * Quit if the close gadget...
             */
            loopflag=FALSE;
          }
          ReplyMsg((struct Message *)msg);
        }
      }
      else loopflag=FALSE;
    }
    if (win) CloseWindow(win);

    FreeARexx(RexxStuff);
    CloseLibrary((struct Library *)IntuitionBase);
  }
}


/*
 * File: test.rexx
 *
 * SimpleRexx test...
 *
 * You need to run the SimpleRexxExample first...
 */

Options FailAt 100

Options Results

/*
 * Try to read the window title bar
 */
Address EXAMPLE1 ReadTitle

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window title is 'Result

/*
 * Bad WINDOW command...
 */
Address EXAMPLE1 "Window Display"

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window is now open'

/*
 * Open the window
 */
Address EXAMPLE1 "Window Open"

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window is now open'

/*
 * Open the window
 */
Address EXAMPLE1 "Window Open"

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window is now open'

/*
 * Try to read the window title bar
 */
Address EXAMPLE1 ReadTitle

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window title is 'Result

/*
 * Hide the window
 */
Address EXAMPLE1 "Window Close"

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window is now closed'

/*
 * Try to hide the window again
 */
Address EXAMPLE1 "Window Close"

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'Window is now closed'

/*
 * Send a command that does not exist
 */
Address EXAMPLE1 Junk

if rc > 0 then say 'Error was 'EXAMPLE.LASTERROR
else say 'The command worked!!!'

/*
 * Quit the program...
 */
Address EXAMPLE1 Quit



File: test.results

Error was 50: No Window
Error was 80: Argument error to WINDOW command
Window is now open
Error was 50: Window already open
Window title is SimpleRexxExample Window
Window is now closed
Error was 50: No Window
Error was 100: Unknown command
```

Using ASCII2AG.ttx

by Jerry Hartzler

When I started work for the CATS Department, my title was CD Developer. My main job responsibility was to put together release 2.0 of the CATS Developer CD. One new feature on this release is the inclusion of all of the Amiga Mail Volume II technical articles in AmigaGuide format (AmigaGuide being a HyperText tool by the CATS Department). Formatting all 44 ASCII articles by hand would have been a nightmare, so I decided to write an ARexx macro, ASCII2AG.ttx, to do most of the work for me.

Basically, ASCII2AG.ttx will convert a specially formatted ASCII document into a simple AmigaGuide database. All of document's sections and sub-sections are converted into AmigaGuide NODEs and Link Points. The final result will be a document with a master table of contents showing all of the sections as 'buttons'. The user can then click a button to view the associated section. There may also be buttons at the end of a section for that sections's sub-sections. At the end of a sub-section may be buttons for that sub-section's sub-sub-sections and so on. Therefore, a User can read a document in whatever order desired and will no longer need to read an entire document to find what he/she is looking for.

The user does not need to know how to make an AmigaGuide database in order to run this ARexx macro, but it would certainly help. For more information see *AmigaGuide* by David Junod in the 1991 Devcon notes, and *AmigaGuide 101* in an upcoming issue of Amiga Mail.

Most of the commands in ASCII2AG.ttx are TurboText ARexx extension commands and therefore the script must be run under TurboText. TurboText is a text editor developed by Martin Taillefer and published by Oxxi, Inc. Every feature of TurboText can be controlled via ARexx commands, allowing the user the flexibility to customize and enhance the program.

Prepping the ASCII File

Before ASCII2AG.ttx is run, the ASCII file that is to become the AmigaGuide database must be prepared. The macro is not smart enough to know where the sections and sub-sections are in the document, so you must tell it. This is done by adding a 'Section Tag' in front of each section and sub-section heading.

Section Tag template:

```
<Section Number>@<NODE Name> <Section Header>
```

<Section Number>
The Section number is the priority number for that section. For example, all sections in a document are section number one (1@), all sub-sections are section number two (2@), all sub-sub-sections are number three (3@), and so on. Sections are limited to ten deep (0@ - 9@), section number zero being the main table of contents. The order of sections must be in the order that they will be read.

```
(0@) Table of Contents
(1@) Section
     • • •
(2@) Sub-Section
     • • •
(3@) Sub-Sub-Section
     • • •
(1@) Section
     • • •
(2@) Sub-Section
     • • •
(2@) Sub-Section
     • • •
etc...
```

Section tags must start in the first column of the section header line in order for ASCII2AG.ttx to recognize it.

A document should start with a 0@ section tag. For those familiar with AmigaGuide this will become the MAIN node. The 0@ section tag can be either the document's table of contents or introduction section. The ASCII2AG.ttx macro will add one automatically if it cannot find one and if the user requests it. Only one 0@ section tag can be in a document.

<NODE Name>
This is an optional parameter that allows the user to specify the NODE name. If a NODE name is not specified, ASCII2AG.ttx will supply one. NODE names must only be a single word and must immediately be after the @ (at) sign. Remember, so as not to confuse AmigaGuide, each NODE must be assigned a different name.

If a NODE name is assigned to section 0@ it will be ignored, and replaced by "MAIN". The NODE name will become the first word in the Section Header.

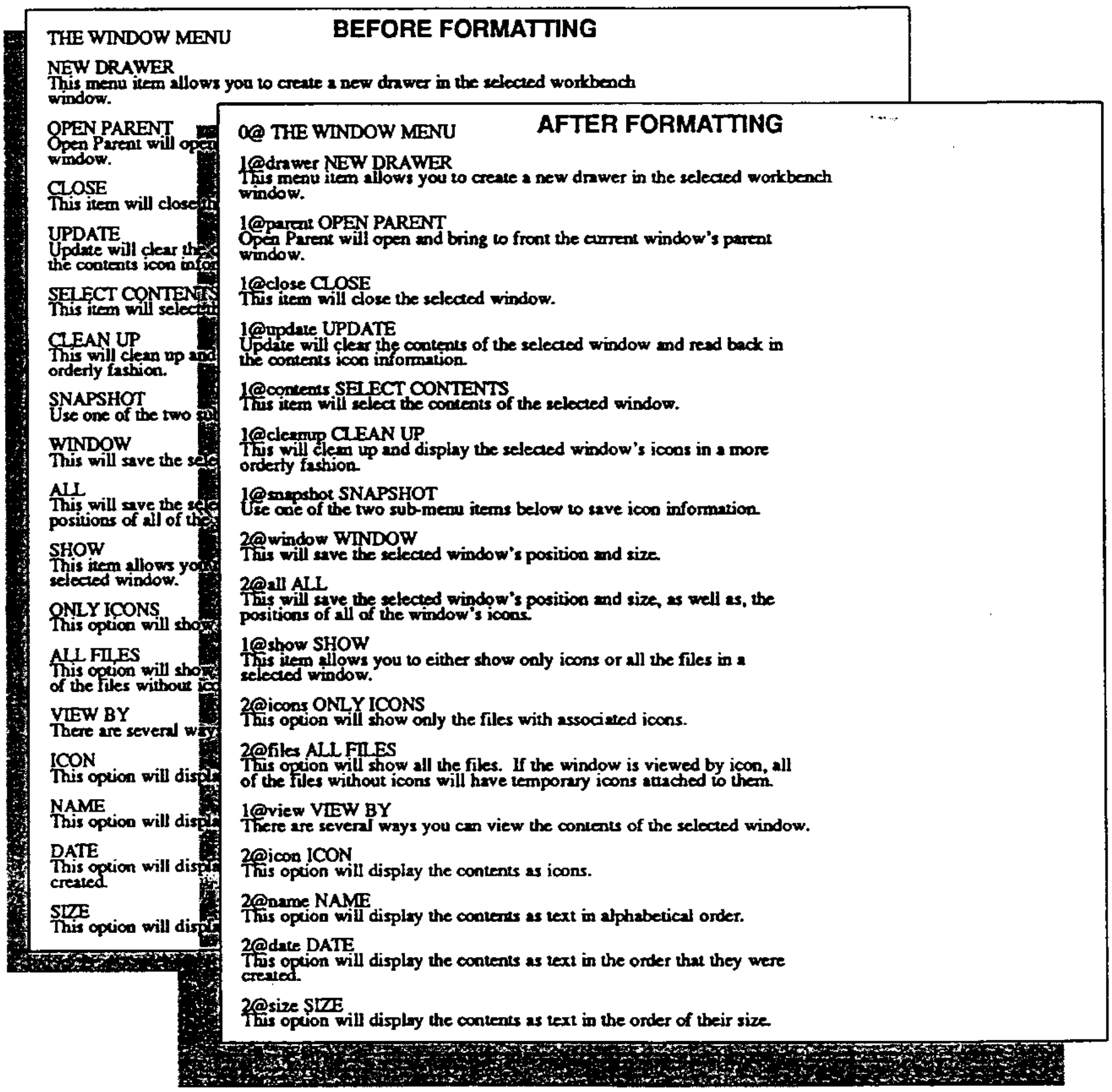
<Section Header>
The section header is the name or title of the section. ASCII2AG.ttx will make it the title parameter of a NODE, which is the text that is displayed in the title bar of the AmigaGuide window during the display of the NODE. It will also be the Label parameter, or the text within the button, of that NODE's Link Point.

A space must separate it from the NODE Name or the @ (at) sign.

If a Section Header is not supplied AmigaGuide will display the NODE Name in its window's title bar instead.

Example Script

As an example, here is an article on the Window menu option under Workbench 2.0.



BEFORE FORMATTING

```
THE WINDOW MENU

NEW DRAWER
This menu item allows you to create a new drawer in the selected workbench
window.

OPEN PARENT
Open Parent will ope
window.

CLOSE
This item will close

UPDATE
Update will clear th
the contents icon info

SELECT CONTENTS
This item will select

CLEAN UP
This will clean up and
orderly fashion.

SNAPSHOT
Use one of the two su

WINDOW
This will save the sele

ALL
This will save the sele
positions of all of the

SHOW
This item allows yo
selected window.

ONLY ICONS
This option will sho

ALL FILES
This option will show
of the files without ic

VIEW BY
There are several wa

ICON
This option will displa

NAME
This option will displa

DATE
This option will displa
created.

SIZE
This option will displ
```

AFTER FORMATTING

```
0@ THE WINDOW MENU

1@drawer NEW DRAWER
This menu item allows you to create a new drawer in the selected workbench
window.

1@parent OPEN PARENT
Open Parent will open and bring to front the current window's parent
window.

1@close CLOSE
This item will close the selected window.

1@update UPDATE
Update will clear the contents of the selected window and read back in
the contents icon information.

1@contents SELECT CONTENTS
This item will select the contents of the selected window.

1@cleanup CLEAN UP
This will clean up and display the selected window's icons in a more
orderly fashion.

1@snapshot SNAPSHOT
Use one of the two sub-menu items below to save icon information.

2@window WINDOW
This will save the selected window's position and size.

2@all ALL
This will save the selected window's position and size, as well as, the
positions of all of the window's icons.

1@show SHOW
This item allows you to either show only icons or all the files in a
selected window.

2@icons ONLY ICONS
This option will show only the files with associated icons.

2@files ALL FILES
This option will show all the files. If the window is viewed by icon, all
of the files without icons will have temporary icons attached to them.

1@view VIEW BY
There are several ways you can view the contents of the selected window.

2@icon ICON
This option will display the contents as icons.

2@name NAME
This option will display the contents as text in alphabetical order.

2@date DATE
This option will display the contents as text in the order that they were
created.

2@size SIZE
This option will display the contents as text in the order of their size.
```

Running ASCII2AG.ttx

Once the section tags have been added and the document has been saved, load it into TurboText - if it isn't already - and select "Exec ARexx..." from the "Macros" Menu. A load requester will come up. Select the ASCII2AG.ttx Macro and "_Exec" in the requester.

The Macro will first diagnose the document to find if section tags and NODE names are in order. As previously mentioned, if the macro cannot find the first section tag 0@ it will give the user the option of adding one.

When the macro is finished, the document can now be displayed in AmigaGuide.

```
/*********************************************************

        (c) Copyright 1992-93 Commodore-Amiga, Inc.
                   All rights reserved.

This software is provided as-is and is subject to change;
no warranties are made.  All use is at your own risk.  No
liability or responsibility is assumed.


                $VER: ASCII2AG.ttx 1.00 (07.01.93)
             ------------------------------------
        Make an ASCII file into an AmigaGuide database.
          File must specify where nodes start by:

              <node#>@<node name> <node title>

                 Written by Jerry Hartzler

*********************************************************/

options results

setinputlock on
getport                       /* Get some file information */
workport=result
getfilepath
file=result
getfileinfo
parse var result . . filename
filename=compress(filename,'" ')

getprefs findbackward              /* Save User prefs and */
fbw=result                         /* setup program prefs */
getprefs findignorecase
fic=result
getprefs findstring
fs=result
getprefs findwholewords
fww=result
setprefs findbackward off
setprefs findignorecase on
setprefs findwholewords off

/*********************************************************
       Check work file for correct node formatting
*********************************************************/

err=0
SetStatusBar Temporary 'Error Checking...'
movesof
insertline
moveup
lchar=0

movesof                       /* Check if there are any nodes */
find '?@'
if rc~=0 then do
    message='"No nodes found!"'
    call error1
    end


else do                      /* Check if 1st node is MAIN (0@) */
    getchar
    if result~=0 then do
        message='"First section tag is not 0@"'
        call error1
        end

    find '0@'       /* Check if more MAIN nodes are found */
    if rc=0 then do
        message='File must have only one 0@'
```

```
            call error2
            end
    end

do forever      /* Check if Nodes are in sequential order */
    find '?@'
    if rc~=0 then break
    call position
    llin=lin
    if col=1 then do
        getchar
        char=result
        test=datatype(char)
        if test='NUM' then do
            if char<lchar then lchar=char
            if lchar~=char & lchar+1~=char then do
                message='Section numbers not in sequentual order'
                call error2
                end
            lchar=char
            end

        moveright 2                /* Check if two nodes   */
        getchar                    /* have identical names */
        nchar=C2D(result)
        if nchar~=32 & nchar~=10 then do
            setprefs findignorecase off
            setprefs findwholewords on
            setbookmark 1
            getword
            fword='?'substr(result,2)
            do forever
                find fword
                if rc~=0 then break
                call position
                if col=1 then do
                    message='Same NODE names: lines' llin-1 '&' lin-1
                    call error2
                    end
                end
            setprefs findignorecase on
            setprefs findwholewords off
            movebookmark 1
            end
        end
    end

/*********************************************************
         Format work file as AmigaGuide database
*********************************************************/

movesof

text '@DATABASE'        /* Make work file an AG database */

if atnode=1 then do               /* Insert MAIN (0@) node */
    insertline                    /* if one does not exist */
    text '0@'
    movesof
    end

nodenum=0
endnum=0
lchar=1
do forever
    node=''
    find '?@'
    if rc~=0 then leave      /* exit when ?@ is not found */

    getcursorpos          /* does ?@ start on 1st column? */
    parse var result . col .

    if col=1 then do
        getchar
        char=result

        if char=0 then do         /* 0@ is the MAIN node */
            node='MAIN'
            end

        else do                  /* every NODE must have */
            text '@ENDNODE'      /* an ENDNODE           */
            insertline
            end

        delete
        getline 2
        line=result             /* Call to parse ?@ line */
        call nodename           /* for any NODE info     */
        moveright 1
        deleteeol
        text 'NODE' node name                /* Make NODE */
        movesol
        teest=datatype(char)

        if char~=0 & test='NUM' then do
            moveup                      /* Make Link Point */
            setbookmark endnum          /* for NODE        */
            movebookmark char-1
            if strip(name,'b','"')='' then name=node
            text '@{'name 'link' node'}' /* ^if no title */
            insertline                 /* use name of NODE */
            setbookmark char-1
            movebookmark endnum
            movedown
            endnum=char
            end
        end
    end

moveeof
text '@ENDNODE'     /* Insert ENDNODE at end of database */
insertline
call quit

/*********************************************************
                         Calls
*********************************************************/


nodename:                 /* Parse ?@ line for NODE info */
    name=''
    if length(line)>0 then do
        if C2D(left(line,1))~=32 & node='' then do
            node='"'subword(line,1,1)'"' /* Get NODE name */
            line=subword(line,2)
            end
        if length(line)>0 then name='"'strip(line,'b','" ')'"'
        else name=''                     /* ^Get NODE title */
        end
    if node='' then do              /* If NODE has no name */
        nodenum=nodenum+1           /* make one up         */
        node='"'filename'_'nodenum'"'
        end
    return


position:                        /* Find column position */
    getcursorpos
    parse var result lin col .
    return


error1:                /* Does user want a MAIN (?@) node */
    requestbool TITLE message PROMPT '"Make One?  OK=Yes CANCEL=No"'
    if result='NO' then call error2
    stnode=1
```

```
    return

error2:             /* Change work file to the way it was */
    err=1
    setbookmark 1
    moveeof
    delete
    movebookmark 1
    beepscreen
    SetStatusBar message
    call quit


quit:                                          /* Bye Bye */
    address value workport      /* Set back to user Prefs */
    setprefs findbackward fbw
    setprefs findignorecase fic
    setprefs findstring fs
    setprefs findwholewords fww
    setinputlock off
    if err=0 then do
        moveeof
        SetStatusBar 'Operation Complete'
        end
    exit

u
```

Overview of V39 Graphics

by Spencer Shanson

The Advanced Amiga Chip set (AA) redefines graphics performance on the Amiga by adding many new display modes, new features and more color to the Amiga platform. This article presents an overview of how the new V39 graphics library has implemented theses new features in the system software.

In V39, the graphics library has changed in the following areas:

- ❑ Support for new AA display modes
- ❑ Double buffering
- ❑ Retargettable sprite and screen (ViewPort) functions
- ❑ Palette sharing
- ❑ Bitmap functions
- ❑ Interleaved BitMaps
- ❑ Display mode promotion, coercion and selection
- ❑ RTG compatible RastPort functions.

The original Amiga graphics library exposed many device-dependent details to the application programmer. Because of this, introduction of new graphics devices has been slowed, and application support of new features has been delayed. To reverse this trend, no features have been added to the new graphics system which cannot be kept for the future. Thus, when newer graphics systems are introduced, system software will need fewer changes, and applications will be ready to automatically use new capabilities. Thus, parameters such as number of bits per pixel, resolution, color palette size, etc., are either variable or have been made very large.

Compatibility

The AA chips are register level compatible with the old and ECS chips at boot up time. However, when new AA modes are enabled and displayed, and a game then takes over the screen display without informing the OS of it, some registers may be in incompatible settings.

Two approaches will reduce this:

1) Old Programs which boot with their own custom boot block (games) will have AA features disabled unless they are specifically asked for. This should ensure transparent compatibility for all bootable games.

2) It will be possible to disable AA features for non-compatible programs. This will be done via the "BootMenu" which is available at system boot time. New options in this menu will allow disabling of AA, disabling of ECS, and switching of PAL and NTSC.

For bootable games that want AA features, use the V39 graphics function SetChipRev(). This lets you upgrade the Amiga to be any chip revision you need, and handles the extra house-keeping that graphics needs to operate the selected chipset (such as updating the graphics database). The only restriction is that it is not possible to downgrade the setting from AA down to ECS or A.

Get/Set Functions

This release of graphics.library will attempt to ease the (future) transition to Retargetable Graphics. New functions are provided to do some operations in a more device-independent manner. This will help when we have to support foreign graphics devices, more than 8 bits per pixel, chunky pixels, and true-color displays.

These new "Get/Set" functions will allow device independent access to fields in the RastPort structure which were only previously manipulable by direct structure access.

`VOID SetAPen(rp, pen)`	Sets the current APen value
`ULONG GetAPen(rp)`	Return current APen value
`VOID SetBPen(rp, pen)`	Sets the current BPen value
`ULONG GetBPen(rp)`	Return current BPen value
`VOID SetDrMd(rp, mode)`	Sets the current DrawMode
`ULONG GetDrMd(rp)`	Return current DrawMode
`VOID SetOPen(rp,c)`	Was a macro. Now use the SetOutlinePen() function, or the SafeSetOutlinePen() macro.
`ULONG GetOPen(rp)`	Return current area outline pen
`VOID SetWrMsk(rp,m)`	Was a macro. Now use the SetWriteMask() function, or the SafeSetWriteMask() macro.
`ULONG GetWrMsk(rp)`	Description of this function is left as an exercise for the reader.
`VOID SetABPenDrMd(rp,apen,bpen,drmd)`	Sets the APen, BPen, and DrawMode for a RastPort. Faster than separate calls.

```
VOID SetRPAttr(rp,taglist)
```

You asked for it so here it is. Lets you set many of the RastPort attributes with one tag-based call. Here are the attributes currently supported:

RPTAG_Font	Font for Text() RPTAG_SoftStyle style for text (seegraphics/text.h)
RPTAG_APen	Primary rendering pen
RPTAG_BPen	Secondary rendering pen
RPTAG_DrMd	Drawing mode (see graphics/rastport.h)
RPTAG_OutLinePen	Area Outline pen
RPTAG_WriteMask	Bit Mask for writing.
RPTAG_MaxPen	Maximum pen to render (see SetMaxPen())

```
VOID GetRPAttr(rp, taglist)
```

No prizes for guessing what this does. It supports the extra tag RPTAG_DrawBounds. This tag is passed with a pointer to a rectangle structure. The returned rectangle structure will contain the bounds of the clipping regions inside the RastPort. This can be used to optimize drawing and refresh.

Color Map Functions

The color palette in AA is different in a lot of ways from the ECS one:

- It has 24 bits per entry, plus one bit to select transparency.
- There are 256 entries which is enough for many programs running on the same screen to share the palette.

All colors are now specified as 32 bit left-justified fractions, and are truncated based upon the number of bits that the hardware is capable of displaying.

There are now ...RGB32() functions to replace the ...RGB4() functions. These all work in 32-bits per gun, irrespective of the device the colors are intended for. Devices that cannot handle the color resolutions will truncate the colors to the most significant n bits. That is why it is important to duplicate the most significant n bits throughout the 32-bit resolution. For example, pure white should be treated as:

```
R = 0xffffffff, G = 0xffffffff, B = 0xffffffff
```

and not

```
R = 0xf0000000, G = 0xf0000000, B = 0xf0000000
```

The new color palette functions allow for multiple applications to coordinate their access to the palette. This allows applications to, for instance, dynamically remap pictures to match the color palette of the Workbench screen, display animations in windows, etc.

back to 1x mode, if 1x mode is capable of displaying the screen.

AllocBitMap() allocates an entire bitmap structure, and the display memory for it. AllocBitMap() allows you to use more than 8 planes, and also allows you to specify another bitmap pointer, thus telling the system to allocate the bitmap to be "like" another bitmap. A bitmap allocated in such a matter may be able to blit to this bitmap faster. Such a bitmap may be stored in a foreign device's local memory. Do not assume anything about the structure of a bitmap allocated in this manner. The size of a bitmap structure is subject to change in future graphics releases. Thus, you should use AllocBitMap()/FreeBitMap() for your raster allocation.

Sprite Functions

Graphics sprite functions (MoveSprite()) have been extended to understand large sprites, selectable sprite pixel resolution, and movement of scan-doubled sprites. Sprite positioning is no longer rounded down to lo-res pixel resolution. Applications will no longer have to "know" about the hardware-dependent format of sprite data.

The new sprite functions work with an ExtSprite structure, which is obtained with the tag-based AllocSpriteData() function. This function allows you to specify a BitMap for the image of the sprite, and tags to specify scaling factors to apply to the BitMap. The returned ExtSprite structure is then passed to GetExtSprite(), which is the V39 equivalent of the old GetSprite() function. There is no equivalent FreeExtSprite(0 function; FreeSprite() does the trick. The ExtSprite allocated with AllocSpriteData() is freed with FreeSpriteData().

The AA chip set imposes some limitations on sprites:

- All sprites in a ViewPort will be of the same resolution and width. There is no individual sprite resolution/width control. If the sprite you allocate is of a different width than Intuition's pointer, then Intuition is notified, and takes the appropriate steps to maintain the pointer imagery.
- In the programmable beam modes (namely Multiscan, Euro72, DblNTSC, DblPAL, and Super72), only sprite 0 is likely to be available; the other sprites are lost to bitplane DMA. MakeVPort() has code that detects which sprites have been reserved (with GetSprite() or GetExtSprite()), and if possible, will drop the display bandwidth. This will make available more DMA to the lower numbered sprites, at the expense of more bitplane DMA access. MakeVPort() will not drop the bandwidth if doing so would cause the loss of bitplanes in the display, so this is not always guaranteed to work for you.

All the AA sprite features are available through new tags for VideoControl():

VTAG_SPEVEN_BASE_SET/GET
: This sets the base color number of the even numbered sprites. The AA chip set allows odd numbered and even numbered sprites to use the colors of different 16-color banks, as opposed to the previous chip sets where all sprites used colours 16-31. Legal values to set the base are 0-255, but will be rounded down to the nearest multiple of 16.

VTAG_SPODD_BASE_SET/GET
: As VTAG_SPEVEN_BASE_SET/GET, only for the odd numbered sprites. For example, if the following tag list is passed to VideoControl():

```
struct TagItem[] = {
     {VTAG_SPEVEN_BASE_SET, 32},
     {VTAG_SPODD_BASE_SET, 144},
     {TAG_DONE}, };
```

then the sprites use the following colors:

Sprite	Colors
0	32 (transparent), 33,34,35
2	36 (transparent), 37,38,39
4	40 (transparent), 41,42,43
6	44 (transparent), 45,46,47
1	144 (transparent), 145,146,147
3	148 (transparent), 149,150,151
5	152 (transparent), 153,154,155
7	156 (transparent), 157,158,159

[Attached sprites use the palette settings of the odd numbered sprites.]

Normally, there are only (2 << depth) colours in a ColorMap (with the exception that Intuition screens always have a minimum of 32 colours). If you wish to set the sprite's colors to use pens that are outside of this range, then you should use the VTAG_FULLPALETTE_SET tag, which specifies that the ColorMap should contain entries for all possible colors (256). Therefore, the colors of these entries can be set with the usual ...RGB32() functions. Not setting VTAG_FULLPALETTE_SET in this case may cause unpredictable colors and enforcer hits.

VTAG_SPRITERESN_SET/GET

This allows you to set all the sprites in the ViewPort to one of 5 resolutions.

SPRITERESN_140NS:	All sprites have 140ns pixels.
SPRITERESN_70NS:	All sprites have 70ns pixels.
SPRITERESN_35NS:	All sprites have 35ns pixels.
SPRITERESN_DEFAULT:	All sprites have the Intuition default resolution.
SPRITERESN_ECS:	Compatible with ECS resolutions;140ns, except in 35ns display pixel modes (SuperHires), where the sprite pixels are 70ns.

VTAG_DEFSPRITERESN_SET/GET

For setting the default sprite resolution, as used by SPRITERESN_DEFAULT.

VTAG_BORDERSPRITE_SET/GET/CLR:

This sets the bordersprite option in the AA chipset for the ViewPort, which allows sprites to appear in the borders outside of the normal display window (DisplayClip), i.e., the area that is normally color 0. However, this does not apply to the horizontal blanking area.

VTAG_PF1_TO_SPRITEPRI_SET/GET:

All revisions of the Amiga chips have allowed the priorities of sprites to playfield to be set. Usually, the priority is such that the sprites always appear in front of the playfield. There is an entry in the ViewPort structure for altering this priority, but V39 provides this tag in preference to writing to the ViewPort structure.

VTAG_PF2_TO_SPRITEPRI_SET/GET:
As VTAG_PF1_TO_SPRITEPRI_SET/GET, only for playfield 2. On the Amiga, playfield 2 is the default playfield, and playfield 1 is only used in DualPlayfield modes!

Display Mode IDs and the Graphics Database

The graphics database has been extended where necessary for AA information, and these changes are limited to the DisplayInfo structure. The DisplayInfo->PaletteRange is only one WORD long, which is insufficient for the AA 24-bit colour resolution, and so has been superceded by three new entries called RedBits, BlueBits and GreenBits. These are one BYTE each, and show how many bits of precision is available for each colour gun. 255 bits each for red, green and blue should be enough for the foreseeable future (pun intended).

Some new DIPF_IS flags were added to the database, providing more information about each display mode ID.

DIPF_IS_SPRITES_ATT shows the mode supports attached sprites. The 35ns display modes on ECS could not support them.

DIPF_IS_SPRITES_CHNG_RES shows that the mode supports sprites that can change resolution, and so the VTAG_SPRITERESN_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_SPRITES_BORDER shows that this mode supports border sprites, so the VTAG_BORDERSPRITE_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_SCANDBL shows that this mode is scandoubled (each display line is repeated once), so that half as many lines take up the same physical area on the monitor. There should be no need for you to look at this bit.

DIPF_IS_SPRITES_CHNG_BASE shows that this mode supports sprite base colour offset changing, so that the VTAG_SPODD/SPEVEN_BASE_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_SPRITES_CHNG_PRI shows that this mode supports changing the playfield to sprite priority, so that the VTAG_PF2/PF1_TO_SPRITEPRI_SET VideoControl() tag will work on ViewPorts in this mode.

DIPF_IS_DBUFFER shows that this mode will work with the double-buffering ChangeVPBitMap() function. You should check this flag before using the double-buffering functions on the ViewPort.

DIPF_IS_PROGBEAM shows that this mode is a programmed beam-sync mode.

DIPF_IS_FOREIGN shows that this mode is not native to the Amiga chip set. Currently, no mode IDs will have this flag set, but under RTG many 3rd party display devices will. You may want to start checking for this flag now.

With the plethora of new display modes that are available under V38 and V39, it is now becoming harder and harder for both the application writer and the end user to know which display mode ID they need to use. This will be especially true under RTG when the application writer will have no way of knowing all the possible display mode IDs that are available on third party display devices. A function was added for V39 to alleviate this problem, and provide a means by which the system can calculate the best mode ID to use given a number of requirements, called BestModeID(). It takes the following tags:

BIDTAG_DIPFMustHave
The DIPF_ flags that this display mode ID must have set. Default is NULL, so there is no preference.

BIDTAG_DIPFMustNotHave
The DIPF_ flags that this display mode ID must not have set. For example, you may wish to ensure that only native Amiga modes are considered, so use BIDTAG_DIPFMustNotHave with DIPF_IS_FOREIGN. The default value is defined as SPECIAL_FLAGS, which is (DIPF_IS_DUALPF | DIPF_IS_PF2PRI DIPF_IS_HAM | DIPF_IS_EXTRAHALFBRITE) so you may need to OR your particular requirements with SPECIAL_FLAGS.

BIDTAG_ViewPort
An initializes ViewPort for which a mode ID is sought. For example, to find an interlaced version of a ViewPort:

```
ID = BestModeID( BIDTAG_ViewPort, ThisViewPort,
                 BIDTAG_MustHave, DIPF_IS_LACE,
                 TAG_END);
```

BIDTAG_NominalWidth
BIDTAG_NominalHeight
Together make up the aspect ratio of the desired mode ID. If specified, will override the Width and Height of the ViewPort passed in BIDTAG_ViewPort, or the NominalDimensionInfo of the ID passed in BIDTAG_SourceID. Defaults to 640x200.

BIDTAG_DesiredWidth
BIDTAG_DesiredHeight Nominal width and height of the returned mode ID.

BIDTAG_Depth
Mode ID must support at least this many bitplanes. Defaults to vp->RasInfo->BitMap->Depth of the ViewPort passed in BIDTAG_ViewPort, or 1.

BIDTAG_MonitorID

The returned mode ID must belong to this monitor family.

BIDTAG_SourceID

BestModeID() will use characteristics of this mode ID, and override some of the characteristics with the other values in the taglist.

BIDTAG_RedBits
BIDTAG_BlueBits
BIDTAG_GreenBits

The mode ID must support at least these many bits for each color gun. Defaults to 4 bits each, so A2024 modes will not be considered.

As an example of its use, here is a portion of the code for the V39 CoerceMode() function (which is used by Intuition when coercing screens):

```
ULONG CoerceMode(struct ViewPort *vp, ULONG MonitorID, ULONG Flags)
{
...
     /* Coerce the ViewPort vp to the best fit ModeID in the monitor
      * MonitorID.
      */
     must = NULL;
     mustnot = SPECIAL_FLAGS;
     tag[t].ti_Tag = BIDTAG_ViewPort;
     tag[t++].ti_Data = vp;
     tag[t].ti_Tag = BIDTAG_MonitorID;
     tag[t++].ti_Data = MonitorID;

     if (GetDisplayInfoData(NULL, (APTR)&dinfo,
                            sizeof(struct DisplayInfo), DTAG_DISP, ID))
     {
          must = (dinfo.PropertyFlags & SPECIAL_FLAGS);
          mustnot = (SPECIAL_FLAGS & ~must);
          if ((Flags & AVOID_FLICKER) && (!(dinfo.PropertyFlags & DIPF_IS_LACE)))
          {
               /* we don't want lace if AVOID_FLICKER is set, and this
                * ViewPort is not naturally laced.
                */
               mustnot |= DIPF_IS_LACE;
          }
          tag[t].ti_Tag = BIDTAG_RedBits;
          tag[t++].ti_Data = dinfo.RedBits;
          tag[t].ti_Tag = BIDTAG_GreenBits;
          tag[t++].ti_Data = dinfo.GreenBits;
          tag[t].ti_Tag = BIDTAG_BlueBits;
          tag[t++].ti_Data = dinfo.BlueBits;
     }

     tag[t].ti_Tag = BIDTAG_DIPFMustNotHave;
     tag[t++].ti_Data = mustnot;
     tag[t].ti_Tag = BIDTAG_DIPFMustHave;
     tag[t++].ti_Data = must;
     tag[t].ti_Tag = TAG_DONE;

     return(BestModeIDA(tag));
}
```

As another example, consider an IFF display program with a HAM image, to be displayed in the same monitor type as the Workbench ViewPort.

```
ID = BestModeID(BIDTAG_NominalWidth, IFFImage->Width,
                BIDTAG_NominalHeight, IFFImage->Height,
                BIDTAG_Depth, IFFImage->Depth,
                BIDTAG_DIPFMustHave, DIPF_IS_AM,
                BIDTAG_MonitorID, (GetVPModeID(WbVP) & MONITOR_ID_MASK),
                TAG_END);
```

The definitions of the display mode ID keys have been moved from <graphics/displayinfo.h> to a new <graphics/modeid.h> file.

The include file <graphics/modeid.h> specifically says that the individual bits of the mode IDs should not be checked for any meaning, but that the database should be read to glean information about the mode ID. In order to maintain compatibility with old software, and make the application writer's job somewhat easier, I am willing to guarantee that any mode ID with the bit 0x00000800 set is a HAM mode, and any mode ID with the bit 0x00000080 set is an ExtraHalfBrite mode. These are the only bits that are guaranteed to mean anything in the mode ID itself. These bits correspond of course to the HIRES and EXTRA_HALFBRITE definitions in <graphics/view.h>.

Display Mode Promotion

Promotion is a software solution for the lack of hardware deinterlacing circuitry on the AA machines. With the promotion feature enabled in Icontrol, the default monitor (NTSC on NTSC machines, PAL on PAL machines), becomes DblNTSC on NTSC machine and DblPAL on PAL machines. There are a number of advantages and gotchas with this approach.

One advantage is that the graphics database is always truthful. Any enquiries about a default monitor mode ID will yield information relevant to whatever the default monitor happens to be at the time. This should not be a problem for any code written for V37 onwards, as the default monitor has always been either NTSC or PAL; now, there are just more possibilities. Another advantage is that V39-aware software that requires NTSC or PAL modes (e.g., video titling software), can now get such modes using specific NTSC or PAL mode IDs.

Here is a list of "gotchas" (that is, things to watch out for if you want display mode promotion to work correctly).

1) The default monitor is dynamic, and can change at any time. Therefore, do not cache any information about the default mode IDs, but read them from the database as you need them.

2) There is no equivalent in the Dbl... monitors to the SuperHires modes. The database LVOs sniff out SuperHires mode IDs for database enquiries, and map SuperHires mode IDs to the equivalent Dbl... Hires mode IDs if promotion is enabled.

3) Programs that rely on copper timings for UserCopperlists, such as SHAM displayers, may no longer work when promoted, because each line is shorter in time than the NTSC/PAL line. Therefore, there are less coppercycles per line.

4) Promoted ViewPorts have less sprites available than non-promoted ViewPorts.

5) For compatibility, we do not promote 1.3 style custom ViewPorts and Views (we check for the presence of a ViewExtra).

Miscellaneous

Interleaved screens have been added. These use a different layout of the graphics data in order to speed rendering operations and eliminate the annoying "color-flash" problem which is the hallmark of planar (as opposed to "chunky") display architectures. Set the BMF_INTERLEAVED flag when calling AllocBitMap().

Double buffering functions have been added. These allow applications to display very smooth, synchronized animation in fully an efficient "Intuition-friendly" manner. Call AllocDBufInfo() to allocate and initialize the DBufInfo structure, which is then passed to ChangeVPBitMap(). The double-buffering functions return up to two different types of messages. The first (dbi_SafeMessage) tells your program when it is safe to write to the old BitMap. The second (dbi_DispMessage) is sent when it is safe to call ChangeVPBitMap() again and be certain the new bitmap has been seen for at least 1 field. The autodocs for AllocDBufInfo() has example code showing how to safely double buffer with ChangeVPBitMap().

The DBufInfo structure should be freed with the FreeDBufInfo() function.

Due to the extra colours that need to be loaded in deeper AA screens, the gap between screens can now be greater than the three lines that Intuition traditionally kept. In fact, the number of lines between screens is variable, depending on the screen type and depth. A new function CalcIVG() (CalcInterViewPortGap) has been added to calculate the number of lines required by the copper to execute all the copper instructions before the display window is opened. Note however that CalcIVG() returns the true number of lines (rounded up to the next whole line) needed in ViewPort resolution units, but Intuition still maintains a gap of at least three lines between Screens. Therefore, when calling CalcIVG() to position screens in an Intuition environment, use the result of MAX((laced ? 6 : 3), CalcIVG(v, vp)).

There is one other caveat with respect to CalcIVG(). This function works by counting the number of copper instructions in the ViewPort->DspIns list, which is set up by MakeVPort(). If an Intuition screen is opened behind, then MakeVPort() is not called on that screen's ViewPort until it first becomes visible, so calling CalcIVG() with that screen's ViewPort will yield a result of 0.

Some operations have been sped up: RectFill() has been rewritten, WritePixel() uses the CPU (3x speedup) ScrollVPort is 10 times faster, and other optimizations.

~~Bugs~~ Anomalies

There is a big bug in the V37-V39 graphics hash-table code, which is used to associate a ViewExtra with a View; namely, only 8 Views can have ViewExtras attached to them. This has been fixed in the latest SetPatch and Kickstart.

ScrollVPort() and ChangeVPBitMap() have problems with 8-bit HAM mode. This has been SetPatch'ed.

Many other outstanding bugs have been fixed for V39.

Plans for 3.01

Some features that are planned for Release 3.01:

❑ An LVO for the games writers to handle color fades with user copperlists, and to allow independent scrolling of parts of the ViewPort for parallax scrolling games.

❑ Frame rate control for ChangeVPBitMap().

❑ Add tags to BestModeID() for overscan considerations.

❑ Still faster ScrollVPort (next beta).

◆

Intuition 3.0

by Peter Cherna

Graphics and AA-Chipset Issues

Support for New Modes

Intuition now has direct support for the AA display modes, through extensions to old mechanisms and through new tags. This includes the ability to select higher resolutions and to set colors in better than 4 bits-per-gun. The graphics database and the ASL screen-mode requester are the definitive places to get information about what modes and depths are available or desired. You can specify higher depths using SA_Depth, and new display modes with SA_DisplayID. SA_Colors32 supplants SA_Colors for specifying colors with a higher precision than 4 bits-per-gun. For full details, see the section on new screen features.

Mode Promotion

The AA chipset provides some flicker-free display modes that are roughly equivalent to NTSC or PAL when output through a display-enhancer or flicker-fixing product. While the AA chipset provides these modes without the significant extra expense of a display-enhancer, some software help is required. Conversely, the display-enhancer lives on the video output, and is completely transparent to software.

To use promotion, the user needs to have a multiscan monitor. He needs to install the DblNTSC or DblPAL monitor into his devs:monitors drawer, and ensure that the ''Mode Promotion'' option in IControl Prefs is on (starting with 3.01, this setting will be on by default, eliminating this last step). When this is done, the graphics database entries for the ''default'' monitor will map to the most appropriate modes from the DblNTSC (or DblPAL) monitor, in place of the NTSC (or PAL) monitor.

Interlaced screens that are promoted are displayed using double-height non-interlaced modes. For example, 640x400 NTSC interlaced appears as 640x400 double-NTSC non-interlaced. Non-interlaced 15 kHz screens are promoted using a chipset feature called ''scan-doubling,'' in which each scan-line is output twice, because the 200 or 256 lines of the original screen only fill half the screen when in 31 kHz modes.

Most applications which use the default monitor (either by explicitly using mode-IDs such as HIRES_KEY or by using V34-style 16-bit mode descriptions) and which go through Intuition and have relatively ordinary display requirements will be successfully and transparently promoted, and their screen will be displayed flicker-free. Applications which refer to modes of explicit monitors (e.g., SA_DisplayID of NTSC_MONITOR_ID|HIRES_KEY) are never promoted. The preferred method for an application is to use the ASL screen-mode requester, which normally presents all explicit modes (including the modes of DblNTSC and DblPAL) and omits the modes of the "default" monitor.

Promotion and Compatibility

De-interlacing in hardware is expensive but transparent, since the Amiga chipset's operation is unchanged by the presence of such hardware. However, promotion can change the behavior of the system in a manner which may be incompatible with certain applications. These are:

- o The overscan limits of DblNTSC (DblPAL) are a little less than the overscan limits of NTSC (PAL). (For 3.01, the DblNTSC (DblPAL) limits have been extended and are now comparable to NTSC (PAL)).
- o It may be harder to center a DblNTSC (DblPAL) screen on certain multiscan monitors than it was to center a de-interlaced NTSC (PAL) screen. (3.01 provides additional centering flexibility which basically solves this problem).
- o An interlaced screen is promoted to a non-interlaced screen, which has obvious implications on custom copper-lists.
- o The higher resolutions/depths of the AA chipset require higher alignment restrictions on bitplanes. Fortunately, most applications either let Intuition allocate their screen's BitMap or else they have a custom BitMap whose width is a multiple of 64 pixels (the highest alignment currently required by AA). However, if the custom BitMap is an unusual width, it may not be sufficiently aligned for the hardware. Such a screen can come up skewed when promoted.

"1x" modes require 16-pixel (word boundary) alignment of each scan-line. "2x" modes require 32-pixel (longword boundary) alignment, while "4x" modes require 64-pixel (double-longword boundary) alignment. Here is a short reference:

- o 140 ns pixels (lores in 15 kHz modes, extra-lores in 31 kHz modes)
  1-8 planes require 1X

o 70 ns pixels (hires in 15 kHz modes, lores in 31 kHz modes)
  1-4 planes require 1X
  5-8 planes require 2X

o 35 ns pixels (super-hires in 15 kHz modes, hires in 31 kHz modes)
  1-2 planes require 1X
  2-4 planes require 2X
  5-8 planes require 4X

As the graphics.library AllocBitMap() function takes care of allocating suitably-aligned BitMaps for you, you do not need to worry about alignment when using modern system calls.

- o The AA hardware does not allow dual-playfield non-interlaced screens to be scan-doubled, so they will appear half as tall as their non-promoted counterparts.
- o Like earlier chipsets, the AA chipset still supports eight sprites. In much the same way as ECS and original chipsets lose sprites when overscan is increased, many of the new modes have insufficient spare cycles to fetch data for these sprites. A promoted screen may have fewer sprites left than the corresponding 15 kHz mode, meaning that some sprites other than the pointer sprite may vanish.
- o There is no 31 kHz mode having 1280 pixels per line. That would require 17.5 ns pixel speeds, which is twice what the AA chipset is capable of. Therefore, SuperHires screens are promoted to 640 pixel-per-line screens, which generally can scroll.
- o Custom ViewPorts are not promoted, but a graphics-database aware application could open a ViewPort in DblNTSC or DblPAL.

Pointer Sprite Features

New Pointer Features

Intuition's handling of the pointer sprite has undergone significant rework for V39, partly to add support for AA sprites (35/70/140 ns sprite pixels, 16/32/64 pixels per sprite, scan-doubling of sprites), and partly to make some general improvement to Intuition.

The Intuition pointer now supports the various new sprite modes of the AA chipset. This includes 16, 32, and 64-bit wide sprites, as well as the sprite-pixel resolution control.

Intuition automatically positions the pointer sprite on screen pixel resolution boundaries, even if the pointer is in a lower resolution, with the exception that graphics.library only allows positioning the pointer on every second line of an interlaced screen.

For applications, there is a new boopsi class called "pointerclass," which is used to create Intuition pointer objects. The new <intuition/pointerclass.h|i> include file contains definitions for the attributes that pointerclass supports, including:

- o POINTERA_BitMap - BitMap to use for sprite imagery
- o POINTERA_XOffset, POINTERA_YOffset - sprite hot-spot
- o POINTERA_WordWidth - intended width in words of this pointer
- o POINTERA_XResolution, POINTERA_YResolution - intended resolution of this pointer

The resolution can be any of the hardware resolutions (ECS-compatible, 140 ns, 70 ns, or 35 ns), but Intuition also adds software-managed choices for

- o sprite resolution to match screen pixel resolution
- o sprite resolution to be "always lores" (~320 pixels per line)
- o sprite resolution to be "always hires" (~640 pixels per line)

See <intuition/pointerclass.h|i> for full details.

There is a new pointer-control function called SetWindowPointerA(), which takes a window and a taglist. The tag values are as follows:

- o WA_Pointer (APTR) - used to specify an application custom pointer, ti_Data points to an instance of "pointerclass" you typically obtain using NewObject(). If NULL, you are requesting the Preferences default pointer.
- o WA_BusyPointer (BOOL) - if ti_Data is TRUE, this tag requests the Preferences default busy pointer.
- o WA_PointerDelay (BOOL) - if ti_Data is TRUE, this tag requests that the change of pointer imagery be deferred for a short duration. This is very useful for an application which is about to be busy for an unknown but possibly very short duration. Such an application should request both the Preferences default busy pointer and the pointer-delay feature. If the application clears the pointer or sets another pointer before the delay expires, the pending pointer change is cancelled. This reduces

short flashes of the busy pointer caused by the application having brief intervals of busy-ness.

The same three tags are now recognized by OpenWindowTagList(), so you can now arrange for a window to open with a custom pointer (or standard busy pointer) already in place, and never see a brief flash of the default pointer.

The user can can now specify 32 pixel wide pointers in hires or lores using Pointer Preferences. Pointer Preferences supports a user-defined default pointer image as well as a user-defined busy pointer image.

Intuition blanks the pointer around size or imagery changes, to reduce ugly flashing that might otherwise result.

On an upbeat note, the notorious off-by-one error in sprite hot-spot position was eliminated from all new Intuition and Graphics calls. When using the new mechanisms, always specify the correct hot-spot offset. (The truth is, we tried to leave the error in, but couldn't agree whether the error should be one lores pixel or one sprite-resolution pixel.)

Sprite Compatibility

The old SetPointer() and ClearPointer() functions still work, giving compatible Intuition pointers. Likewise, the pointer imagery in devs:system-configuration will be used until a pointer.prefs file is received. However, due to growing complexity in the pointer subsystem, calling SetPointer() or ClearPointer() from within an input handler or inside Begin/EndRefresh() runs a risk of deadlocking. This has been kludged around, however:

> ***This is why we warn people to stick to simple rendering functions only!*** While inside LockLayerInfo(), LockLayer(), BeginRefresh(), BeginUpdate(), etc., and to not call Intuition or other high-level system functions inside of LockIBase(). As well, great care should be taken inside an input handler (it's generally best for the handler to signal a high-priority task which actually does the work). Don't be the application that dies under the next release when an inappropriate function that happened to work now deadlocks because its handling has become more sophisticated.

See the Autodocs for these functions for more information on what other system calls are okay to use with these functions.

Intuition and Graphics are involved in a scheme to maximize compatibility with older sprite-using applications. The AA chipset supports variable sprite resolution and width, but the setting affects all sprites. If an application requests a specific sprite width (through the old graphics.library/GetSprite() call or the new graphics.library calls (GetExtSpriteA() and ChangeExtSpriteA()), and Intuition's sprite is not compatible with that request, then graphics.library will blank Intuition's sprite and notify Intuition. Intuition will rebound by generating the most suitable pointer sprite which is compatible.

Using an attached sprite for the Intuition pointer was quasi-supported under 2.0. It no longer works.

The pointer information returned by GetPrefs() is no longer kept up-to-date, since the pointer data can exceed the storage space available in struct Preferences. (The ROM default pointer will be returned in all cases). (Like V37, V39 ignores the pointer data in calls to SetPointer() after the first one, for reasons such as this).

Pen-Sharing Support

Under V39, graphics.library has functions that let multiple applications share the pens in a palette (ObtainBestPen(), etc.). Palette sharing allows an application to gain exclusive access to some palette entries, which that application may then use or change as it sees fit. As well, an application may access a pen as shareable, meaning that other applications in need of a similar color may also be granted that pen value. Because these pens are shared among several clients, applications may not alter their color.

Intuition now uses and supports this pen-sharing scheme. For all types of screen, the sprite pens and pens found in the DrawInfo->dri_Pens are obtained as shareable. By default, all other pens are allocated as exclusive on behalf of the screen opener (this provides compatibility when visitor windows aware of the pen-sharing functions open on unaware public screens). Exclusive pens are for the screen owner's use only, and may be changed at the owner's will.

Screens that are aware of pen-sharing issues should set the {SA_SharePens,TRUE} tag, which instructs Intuition to leave all other pens unallocated. The Workbench screen is so marked, but pens 0 to 3 and ~0 to ~3 are also made shareable, for compatibility. Screens with SA_SharePens set to TRUE will have the new PENSHARED bit set in the screen->Flags field.

The application may then allocate pens as needed. A paint package, for example, would allocate all colors it uses as exclusive. Other applications might allocate several colors as shared or exclusive. Since Intuition opens all public screens in private state, the application has a chance to allocate its colors before making the screen available to visitors (see SetPubScreenModes()).

Preferences now listens to only 8 colors for the BitMap, which Intuition will set as the first four and the last four colors of the Workbench or any "Workbench-like" screen (those having the SA_FullPalette or SA_LikeWorkbench attributes). When the SA_Pens pen-array is evaluated, pens are masked to the number of available colors. As well, special definitions of pen-number (PEN_C3, PEN_C2, PEN_C1, and PEN_C0) mean the complementary pen numbers of pens 0 to 3, regardless of depth.

The way the DrawInfo pens are determined is Intuition picks a default pen-array. Then, any pens you supply with SA_Pens override the defaults, up until the ~0 in your array. If the screen is monochrome or old-look, the default will be the standard two-color pens. If the screen is two or more planes deep, the default will be the standard four-color pens. If any explicit pens are specified, the default colors for NewLook menus and the titlebar match the V37 colors. If the SA_Pens tag points at ~0, the NewLook menu colors will be used.

If the screen has the SA_LikeWorkbench property, the default will be the user's preferred pen-array, now changeable through Preferences. There is a preferred pen-array for four colors, and one for eight or more colors.

Miscellaneous Graphics-Level Changes

Intuition places a blanking gap between sliding screens. The time spent in this gap is used to load the hardware color registers, display-mode registers, and bitplane-pointers in a clean way. Under ECS and prior, three non-interlaced lines were sufficient, and this amount was hard-coded. Under AA, this amount may increase, so logic was added to graphics.library (CalcIVG()) to determine this amount. Intuition now bases its inter-screen gap on the result of this call.

Under 2.0, when a screen was coerced, Intuition determined what display mode to use based on tables built into Intuition. This was too limiting, so graphics.library now has a CoerceMode() function to fulfill this responsibility. Intuition now uses this function. (Interested persons should see graphics.library/BestModeIDA(), which has wider application than CoerceMode()).

Enhancements to Screens

Attached Screens

It is becoming increasingly popular for an application to have multiple screens open simultaneously. A typical use is an application that has a full-sized screen (perhaps in HAM mode) as a canvas, and a short screen as a control panel or palette. Since it is desirable that these screens slide and depth-arrange together, Intuition now provides the ability to attach screens together. OpenScreenTagList() now supports the SA_Parent tag to attach a new child to an existing parent. The SA_FrontChild and SA_BackChild tags can be used to attach an existing child in front of or behind a new parent. When opening a parent screen you can specify multiple SA_xxxChild tags, in order to attach multiple children. Draggable child screens can be dragged independently of each other and their parent, except that they can never go above their parent. Pulling down the parent below its natural top causes the child screens to move as well.

Attached screens always remain adjacent to each other in the screen depth-ordering. It is not possible to interpose some other screen between screens of the same family. User depth-arrangement (via Amiga-M/N or the screen depth-gadget), as well as old programmatic depth-arrangement (ScreenToFront() and ScreenToBack()), depth-arrange a screen's family as a single unit, moving them to the front or to the back of the list of screens without altering the ordering of screens within the family. The new ScreenDepth() function has an SDEPTH_INFAMILY option which allows the programmer to depth-arrange screens within their family.

There are times when it would be useful for a parent and child screen to masquerade as a single screen. This can allow independent setting of screen mode, depth, resolution, etc. Set the new SA_Draggable tag to FALSE to get a child screen which is non-draggable with respect to its parent. Trying to drag a child screen (through MoveScreen(), its drag-bar, or mouse-screen-drag) is equivalent to dragging the parent. The new ScreenPosition() function has an SPOS_FORCEDRAG option which allows the application to independently move such a child screen.

To complement attached screens, a feature called ''menu lending'' has been implemented. This allows menu button presses in one window to activate the menus of a different window, and have the menus appear in the screen of that other window. The idea is to allow unification of the menu strips of attached screens. The LendMenus() function is used for this.

If OpenScreen() is unable to attach screens (due to illegal hierarchies, etc.), the screen will fail to open, with a secondary error of OSERR_ATTACHFAIL. A parent screen may not itself have a parent, nor may a child screen have a child. Also, a child screen may not be the child of more than one screen. One parent may legally have several child screens.

Aside from the SA_Parent, SA_FrontChild, or SA_BackChild tags, and the drag and depth arrangement behavior described, attached screens are just like other screens, that is to say they live on the regular screen list (IntuitionBase->FirstScreen-> ...), and child screens don't inherit any properties from their parent. In particular this means you can run the same code under Release 2 and all you will lose are the depth arrangement and dragging relationships. That would yield the best situation under those versions, and no conditional code is required on your part.

Various combinations of DClips and over-sized scrolling screens are supported by attached screens, and they work in much the manner you would expect. Note that there are problems when non-draggable child screens are attached to parent screens and their DClips or dimensions are not equivalent. These problems are fixed in 3.01.

The attachdemo.c example on the DevCon disks shows usage of attached screens.

Double-Buffering Support for Screens

Intuition now supports double (or multiple) buffering inside an Intuition screen, with full support for menus, and support for certain kinds of gadget.

The AllocScreenBuffer() call allows you to create other BitMap buffers for your screen. the SB_SCREEN_BITMAP flag is used to get a buffer handle for the BitMap currently in use in the screen. Subsequent buffers are allocated with the SB_COPY_BITMAP flag instead, which instructs Intuition to copy the current imagery (e.g., the screen title-bar and any of your rendering) into the alternate BitMap. Normally you let Intuition allocate these alternate BitMaps, but if your screen is CUSTOMBITMAP, you should allocate the alternate BitMaps yourself.

To swap buffers, call the ChangeScreenBuffer() function, which attempts to install the new buffer. ChangeScreenBuffer() will fail if Intuition is temporarily unable to make the change (say while gadgets or menus are active). Intuition builds these functions on top of the new graphics.library ChangeVPBitMap() function, so the signalling information that graphics provides is also available to the Intuition user. To clean up, call FreeScreenBuffer() for each screen buffer. It is not necessary to restore the original buffer before freeing things. Consult the autodocs for full details.

When the user accesses the screen's menus, buffer-swapping will stop. The ChangeScreenBuffer() call will return failure during this time. When the user finishes his menu selection, buffer-swapping will be possible again. Only a small subset of gadgets are supportable in double-buffered screens. These gadgets are those whose imagery returns to the initial state when you release them (e.g., action buttons or the screen's depth gadget). To use other kinds of gadgets (such as sliders or string gadgets) you need to put them on a separate screen, which can be an attached screen.

Windows with borders are not supportable on double-buffered screens. Double-buffered screens are expected to consist nearly entirely of custom rendering.

An example program illustrating double-buffering under Intuition, with menu-lending and an attached screen to hold two slider gadgets is provided.

Miscellaneous Screen Features

The new {SA_Interleaved,TRUE} tag allows applications to request that their custom or public screen have an "interleaved" BitMap. Interleaved BitMaps are built out of a single allocation, instead of one per bitplane. Further, the data is laid out as follows:

```
bitplane 0, scan-line 0
bitplane 1, scan-line 0
...
bitplane n, scan-line 0
bitplane 0, scan-line 1
...
```

(Contrast this to regular BitMaps, where each bitplane is contiguous.) The primary advantage of interleaved BitMaps is that blitting between them is cleaner, because color artifacting is eliminated. As well, multiple small blits are replaced by fewer large blits, saving on blitter setup time and improving blitter/processor overlap.

The primary disadvantage of interleaved BitMaps is that BitMap->BytesPerRow no longer means "how many bytes are in one row of one bitplane." This field continues to mean "how many bytes must be added to the current address to arrive at the same pixel one row down." See the notes for the compatibility talk for further details. (Screen grabbers and screen printers seem to be the primary victims of this change.)

For compatibility, the Workbench screen is non-interleaved if it opens before IPrefs has run. If opened or reset after IPrefs has run, the Workbench screen will be interleaved.

The new SA_LikeWorkbench tag gives you a screen having the same attributes as the Workbench screen, including depth, colors, pen-array, screen mode, etc. Individual attributes can be overridden by using tags. (SA_Workbench itself overrides things specified in the NewScreen structure). Attention should be paid to hidden assumptions when doing this. For example, setting the depth to two makes assumptions about the pen values in the DrawInfo pens. Note that this tag requests that Intuition *attempt* to open the screen to match the Workbench. There are fallbacks in case that fails, so it is not correct to make enquiries about the Workbench screen then make strong assumptions about what you're going to get.

OpenScreen() now allocates an appropriate-sized ColorMap for new modes. You can override this with the SA_ColorMapEntries tag to let the application increase the ColorMap size if needed.

The SA_Draggable and SA_Exclusive tags are designed to help implement "game screens" that coexist with Intuition. {SA_Draggable,FALSE} allows the caller to make a screen non-draggable. {SA_Exclusive,TRUE} allows the caller to make a screen exclusive, meaning it will never share the display with another screen. Dragging down a screen that's in front of an exclusive screen won't reveal the exclusive screen. Although exclusive screens can autoscroll, but they can't be pulled down below their natural top (since nothing should be visible behind). Starting with 3.01, you can attach one or more exclusive screens with the SA_Parent and SA_xxxChild tags already described. Such screens form an exclusive family, and only coexist on the display with each other, not with other screens (exclusive or not).

Intuition and graphics.library now support over-sized scrollable A2024/Moniterm screens. While such screens can autoscroll and be dragged around, they can't be pulled down, since other screens would not be visible behind them (similar to the SA_Exclusive property, but dictated by the graphics database MonitorInfo.Compatibility field).

The new ScreenPosition() function extends the functionality of MoveScreen() to also include absolute screen positioning and optional movement of {SA_Draggable,FALSE} screens. The hope is that only the screen's opener (and not any commodity-type programs) would forcibly move a non-draggable screen. As well, ScreenPosition() allows you to specify a rectangle in screen coordinates which you wish to be made visible. Over-sized screens will be scrolled such that the rectangle you supply will be on the visible part of the display.

The new ScreenDepth() function unifies ScreenToFront() and ScreenToBack() while adding a flag allowing optional in-family depth-arrangement of attached screens. ScreenToFront(), ScreenToBack(), Amiga-M/N, screen depth-gadget action, and EasyRequest screen-popping all go through the ScreenDepth() LVO. Due to a bug, WBenchToFront() and WBenchToBack() do not yet do so, but this is expected to be fixed for 3.01. OpenScreen() and the routines which open the Workbench now call OpenScreenTagList() through the LVO, which will allow some useful SetFunction()ing.

OpenScreen() now supports the new SA_BackFill tag, to install a LayerInfo backfill hook for the screen.

Intuition now ensures that the Workbench screen is at least 640x200. This is needed in order to safely allow lores (and other odd resolution) Workbench screens.

The original Screen structure has an embedded instance of a BitMap structure, which is unfortunate. When Intuition allocates a screen's BitMap (i.e., a non-CUSTOMBITMAP screen), it now uses AllocBitMap(), and copies the struct BitMap into &screen->BitMap. This is the direction to head for RTG, and it is needed now for double-buffering. Intuition internally now references the real BitMap (obtainable as screen->RastPort.BitMap) instead of &screen->BitMap. It is recommended that applications do the same.

The new SA_Colors32 tag can be used to provide 32-bit color information at OpenScreen() time. ti_Data points to a longword-array that Intuition will pass to LoadRGB32(). See the autodoc for LoadRGB32() for details on how to format the data.

The new SA_VideoControl tag allows an application to provide a taglist which Intuition will pass to VideoControl() after opening the screen. This can be useful to turn on border-sprites, for example.

The requested screen depth is now validated. Making a request for a screen too deep causes failure with a secondary error of OSERR_TOODEEP.

Enhancements to Windows

The window depth-gadget now determines whether a click should send the window to front or to back based on whether the window is obscured or not, rather than on whether it's the top layer.

The new ScrollWindowRaster() function implements ScrollRasterBF() at the Intuition level. You will receive an IDCMP_REFRESHWINDOW event if there is damage to your window. Damage will appear if obscured parts of a simple refresh window are scrolled into view. *Note: this area is not cleared!*

When you supply the "alternate-size" rectangle for zooming using the WA_Zoom tag, you can now specify (-1,-1) for the upper-left corner. This instructs Intuition to perform size-only zooming. Wherever the window is placed, zooming will toggle size but not affect position (unless moving the window would be required to keep it on-screen). Using (-1,-1) under V37 is safe, and equivalent to using (0,0).

The Window->WindowPort is now allocated at OpenWindow() time, even if IDCMPFlags of zero are requested. This simplifies ModifyIDCMP() failure handling because failure can no longer happen when you set the window->UserPort to your shared port, then call ModifyIDCMP() to turn on messaging. ModifyIDCMP() can now only fail if you ask it to create the UserPort.

The new OpenWindow() tag WA_NotifyDepth allows a window to request IDCMP_CHANGEWINDOW messages when that window is depth-arranged. These messages arrive with an IntuiMessage->Code value of CWCODE_DEPTH to distinguish them from V37-style IDCMP_CHANGEWINDOW messages (sent in response to window movement or resizing), which have a Code value of CW_MOVESIZE.

When inactive, window borders are filled with BACKGROUNDPEN instead of pen zero. There are presumably a few more places when pen zero is being used incorrectly, but they are hard to track down (ever try to search 1.5 megabytes of source code for all references to "0"?)

The WA_HelpGroup and WA_HelpGroupWindow tags allow the programmer to identify multiple windows of the same application, for purposes of gadget help processing. See the section on gadget help for details.

Enhancements to Gadgets and Imagery

Extended Gadget Structure

V39 introduces the ExtGadget structure as a compatible substitute that can be used wherever old Gadget structures can be found. An arbitrary gadget can be identified as an ExtGadget if the GFLG_EXTENDED bit in its Flags field is set. Never attempt to read any of the extended fields of a gadget if this flag is not set. Starting with V39, all instances of gadgetclass or its subclasses are ExtGadgets. The extended fields include a new longword worth of flags, and a bounding box.

Gadget Bounding Box and GM_LAYOUT

Until now, the only area that was defined for gadgets was the select box area. However, the imagery of a gadget often extends outside its select box. For example, the border of a string gadget is often outside, as is its label. ExtGadgets have four new fields that describe the "bounding box." The bounding box can be used to allow relative size or position gadgets to work even if they have imagery outside the select box. The BoundsLeftEdge, BoundsTopEdge, BoundsWidth, and BoundsHeight fields of an ExtGadget are assumed valid if the GMORE_BOUNDS flag in the ExtGadget->MoreFlags field is set. Where Intuition wants to use a bounding box, but the gadget is not extended or does not have GMORE_BOUNDS set, the gadget select box will be used instead, which matches the V37 behavior.

The routine that manages erasing and redrawing GRELxxx gadgets during window resizing now bases its work on the gadget bounding box (if one is specified), instead of the gadget select box. This means that you can finally have GREL gadgets which have imagery (e.g., a gadget label) extending outside of the select box, and Intuition will correctly move or resize such a gadget.

As well, there is a new boopsi method for gadgets called GM_LAYOUT. If your gadget has any of the GREL properties, it will receive a GM_LAYOUT message when the gadget is first added (or the window containing the gadget is first opened), as well as whenever the window size changes. At GM_LAYOUT time, the gadget can change its gadget select box and/or bounding box. It can re-allocate or change its image dimensions if it likes. If it is a group-gadget, it can move its members around. To round this all out, there is a new "special relativity" property, GFLG_RELSPECIAL or GA_RelSpecial. Unlike the older GREL properties, this property doesn't affect the interpretation of the gadget box. It does allow your gadget to receive GM_LAYOUT messages, hence have arbitrary layout power.

The DevCon disks include a sample program called relspecial.c that implements a boopsi gadget whose size is kept at half the current size of the window, and which is centered in that window. The example makes use of GFLG_RELSPECIAL and GM_LAYOUT.

Gadget Help

Intuition now supports "gadget help". If a window enables this feature and the user passes the mouse over the bounding box of a gadget which has the GMORE_GADGETHELP property, then an IDCMP_GADGETHELP event will be sent. There is a corresponding boopsi GM_HELPTEST method which boopsi gadgets can use to refine their help-sensitivity areas or to delegate help-testing to member gadgets. Boopsi gadgets can also return values for the IntuiMessage Code field of the IDCMP_GADGETHELP message.

The gadget help feature may be turned on or off through the HelpControl() function.

The gadget help checking is optimized for performance. If the mouse is moving quickly, Intuition skips the check for gadget help. If Intuition discovers that the mouse is still over the same gadget as the last one that sent gadget help, no new IntuiMessage is sent unless the gadget wants to report a different IntuiMessage->Code value.

When the mouse is over a GMORE_GADGETHELP gadget, the IDCMP_GADGETHELP message has an IntuiMessage->IAddress which points to the gadget. When the mouse is over the window but not over any help-aware gadget, the IAddress points to the window itself.

When the mouse is not over the window, the IntuiMessage IAddress will be zero. Intuition will look "through" gadgets that do not have the GMORE_GADGETHELP property to see if some other gadget lies underneath.

Ordinarily, gadget help only applies to the active window. However, a multi-window application can mark all its windows as being in a group (using the WA_HelpGroup or WA_HelpGroupWindow tags), which makes Intuition test gadget help in all windows of the group when any one of them is the active one. There is a new utility.library function called GetUniqueID() which must be used to provide an ID for WA_HelpGroup. If you have only one window, there is no need to pass WA_HelpGroup. HelpControl() sets the state of gadget help for all windows of a group, and Intuition ensures that all windows of the same group are consistently set.

```
helpgroup = GetUniqueID();

for ( each window )
    {
    win[x] = OpenWindowTags(...,
                                        WA_HelpGroup, helpgroup, TAG_DONE);
    }
```

Inactive windows whose WA_HelpGroup matches the active window's are also subject to gadget help testing. IDCMP_GADGETHELP messages are sent to the window the mouse is over. The IDCMP_GADGETHELP message with an IAddress of zero means the mouse is not over the active window or any other window of the same group. This particular message is always sent to the active window (which is not necessarily the window in your group that last got a message).

All system gadgets (e.g., close, drag, size, depth) have GMORE_GADGETHELP set, so GadgetHelp-aware applications can (almost must) provide help on them too. You can check the gadget->GadgetType & 0xF0 field for GTYP_CLOSE, etc. Later include files define GTYP_SYSTYPEMASK to be this value (0xF0).

The gadgethelp.c example on the DevCon disks illustrates the correct handling of IDCMP_GADGETHELP messages.

Gadget Support for ScrollRaster() Damage Handling

Intuition now notices and repairs damage when boopsi gadgets use ScrollRaster(). (Such damage occurs when the gadget is in a simple-refresh window and part of the scrolled area is obscured). Such gadgets must set GMORE_SCROLLRASTER in order to benefit from this magic repair feature. Note that ScrollWindowRaster() is for applications. Boopsi gadgets must not use ScrollWindowRaster(), but rather use ScrollRaster() or ScrollRasterBF().

Miscellaneous Gadget Features

There is an important new Boopsi function in Intuition called DoGadgetMethodA(), that invokes the specified method, but includes a valid GadgetInfo structure if possible. SetGadgetAttrsA() has been re-implemented to go through DoGadgetMethodA(). Two SetGadgetAttrsA() bugs were fixed in the process. First, if a requester is off-window, it has no layer. SetGadgetAttrsA() of a gadget in such a requester wasn't being sent to the gadget, but now OM_SET is sent with a GadgetInfo of NULL. Second, there was no locking around the call. DoGadgetMethodA() is preferable to a direct Boopsi invocation (namely DoMethod()) because it offers both a valid GadgetInfo structure and arbitration around other gadget activity.

The default string edit hook now ignores Return or Enter keystrokes that have the repeat-qualifier set.

PROPNEWLOOK proportional gadgets in the borders of an active window have their knobs rendered in FILLPEN instead of SHINEPEN. When you click on the knob, they become SHINEPEN as before. This looks a lot better, is more consistent with inactive windows, and finally restores that all-important feedback to these prop gadgets.

Imagery Features

Under V37 and prior, Intuition handled ghosting a gadget by blasting a pattern of dots over its select area. This did not allow boopsi images to manage their own disabled rendering. For V39, Intuition has defined a new read-only attribute for image classes, called IA_SupportsDisable. If an image class returns TRUE in response to an OM_GET request of this attribute, it is asking to take responsibility for performing ghosting based on image state. When a gadget is first added to a window, Intuition will check its image to determine if it has the IA_SupportsDisable attribute. When such a gadget is disabled, Intuition skips its own disabled rendering, and draws the image using DrawImageState() using the IDS_DISABLED or the new IDS_SELECTEDDISABLED state.

The ROM boopsi image class for rendering frames, ''frameiclass'', now supports the standard frame types used by GadTools and recommended by the *Amiga User Interface Style Guide*, including the standard GadTools-style bevelled box, the GadTools string-gadget ridge, and the AppWindow icon drop-box specified by the Style Guide.
The imagery for the standard system arrows has been improved.

The GadTools checkbox and GadTools radio-button images are now marked as scalable to allow GadTools to support scaled checkboxes and radio-buttons.

The basic gadget types (classic prop, bool, string) now support GFLG_LABELIMAGE.

Enhancements to Menus

NewLook Title Bar and Menus

On aware screens, the title-bar has a nicer appearance.

For V39, Intuition defines three additional pens in the DrawInfo, which are used to control the rendering of the screen title bar and menus. These pens are:

- o BARDETAILPEN - pen used for details like text in the title bar and text or graphics in menus.
- o BARBLOCKPEN - pen used to fill the solid areas of the title bar and menus.
- o BARTRIMPEN - pen to use for the trim-line under the screen title bar.

It is intended that BARDETAILPEN and BARTRIMPEN should be black or dark, and that BARBLOCKPEN should be white or light-colored.

The handling of defaults is a bit involved because of compatibility issues. Applications that specify no SA_Pens array or ones who specify an SA_Pens array with at least one explicit pen provided get compatible defaults. Applications whose SA_Pens array consists solely of {~0} will get the new colors. Finally, screens which specify SA_LikeWorkbench will get the user's preferred pen-array, which allows the user to control the menu and title bar colors.

Under V37 and earlier, the menus themselves are drawn using the window's DetailPen and BlockPen, while the colors of the MenuItems are determined by the imagery (IntuiTexts or Images) chosen. For compatibility, old applications will have their menus rendered in V37-compatible colors. Applications will want to take advantage of NewLook menus when present, but they must request them through the {WA_NewLookMenus,TRUE} tag. This instructs Intuition to use BARDETAILPEN and BARBLOCKPEN for rendering the elements of your menus, instead of relying on the window's DetailPen and BlockPen. Note that the application (or any menu-building library) should ensure that the colors of struct IntuiText or struct Images used by MenuItems use matching pens. You can instruct GadTools to use the new colors by passing {GTMN_NewLookMenus,TRUE} to LayoutMenusA().

Intuition also ensures that the Amiga-key and checkmark symbols are colored to match the menu and are scaled to match the screen's font. If you are using a font other than the screen's font for the menus, you must create custom Amiga-key and checkmark symbols using ''sysiclass''. This image class recognizes a new tag, SYSIA_ReferenceFont, which you can use to set the size of a checkmark or Amiga-key symbol appropriately for your chosen font.

You can then use the (V36) WA_CheckMark tag or the new WA_AmigaKey tag to override the imagery Intuition will use in the menus.

Per screen, the default Amiga-key and checkmark images used will be appropriately colored and scaled to the screen's font. (You can find pointers to their imagery in the DrawInfo structure for that screen).

HIGHCOMP menu items in NewLook menus complement in such a way that pixels colored in BARDETAILPEN highlight into BARBLOCKPEN, and vice-versa.

The flag which indicates whether a window is using NewLook menus is publicly readable (WFLG_NEWLOOKMENUS).

Other Enhancements

Drawing Tablet Support

As an input device, a drawing tablet poses special problems for Intuition and for the tablet driver writer. Some of the problems include getting auxiliary information such as pressure through the IntuiMessage channel and providing suitable scaling. As well, the absolute nature of tablet devices poses some interesting problems for features like autoscroll. V37 added a simple tablet input event, but this was not enough. There now is a new subclass of IECLASS_NEWPOINTERPOS called IESUBCLASS_NEWTABLET. This subclass solves tablet handling quite nicely. The tablet driver fills out a few tablet-oriented properties (like the current value and range in X and Y), and then submits the InputEvent to input.device. Later, Intuition establishes the active screen and the rectangle to which the tablet should scale itself, then calls back the driver through a hook. This allows the tablet driver to handle screen resolution changes, oversized scrolling screens, pulled down screens, and attached screens. The tablet driver can scale according to some tablet preferences settings it manages (for example, preserve aspect, center, best fit horizontal, best fit vertical). Intuition supplies reasonable default scaling for simple tablet drivers that leave the hook NULL. See struct IENewTablet in <devices/inputevent.h>.

Windows that request the new WA_TabletMessages property receive extended IntuiMessages, which include a pointer to a TabletData structure. If this IntuiMessage originated from a tablet event, the TabletData pointer will be non-NULL, and the structure will have some information such as sub-pixel position. In addition, there is a pointer to a tag-list, and there are definitions for standard tags such as pressure, tilt, additional buttons, and a Z-coordinate. The definition and important comments on the TabletData structure can be found in <intuition/intuition.h>.

As an example, pressure is passed in with the TABLETA_Pressure tag. the pressure reading of the stylus. The ti_Data member is the pressure, which should be normalized to fill a signed long integer. Typical devices would not generate negative pressure, but the possibility is not precluded. The Preferences program shipped with a tablet driver for a pressure-sensitive device might offer two pressure sensitivity settings. The "contact threshold" would be the pressure below which no contact should be reported by the driver. This is the "zero point" for reported pressure. The "click threshold" would the pressure at which a button transition should be reported (by setting the InputEvent ie_Code to indicate a downpress of the select button). The tablet would send position/pressure events even when the pressure was below the click threshold (but above the contact threshold, of course).

When a tablet driver sends a new tablet event and the active window is tablet-aware, IDCMP_MOUSEMOVE events are sent to that window even if the pixel-level mouse-position is unchanged. This is to allow applications to hear changes in sub-pixel position or other parameters such as pressure. This means that tablet drivers must be careful to only send events when something actually changed.

If an input event is a tablet event, boopsi gadgets can get a pointer to the TabletData structure in the gpInput structure they receive through the GM_GOACTIVE and GM_HANDLEINPUT messages.

Miscellaneous Features

The new TimedDisplayAlert() function allows for alerts that time-out without user-intervention. Exec uses this new function to aid unattended operation of the Amiga, particularly in kiosk and video applications.

In struct Preferences, the unused WorkName[] field is now split into PrtDevName[], DefaultSerUnit, and DefaultPrtUnit, for multi-serial preferences and more flexible printer-preferences.

OpenScreenTagList(), OpenScreen(), OpenWindowTagList(), OpenWindow(), and the internal BorderPatrol() routine now go through stack-swapping. This protects applications from increased stack usage in these calls.
The Preferences LaceWB field (and whether the pretend screen mode from GetScreenData() is lace or not) now solely depends on the height of the text overscan rectangle of the true mode of the Workbench. This helps older applications opening on modes such as double-NTSC 640x400.

Intuition now handles MakeVPort() failure. Intuition will blank any failed ViewPort, and attempt to remake it at each opportunity. MakeScreen(), RethinkDisplay(), and RemakeDisplay() now have return codes that reflect MakeVPort() failure.

Rendering Optimizations

Several important rendering optimizations make Intuition appear snappier and cleaner. Here is a partial list:

- o When a WFLG_ACTIVATE window is opened, Intuition now activates it synchronously. The big benefit is that the window's border is no longer drawn inactive then activate.
- o EasyRequests and AutoRequests used to consist of a window with a requester inside, which meant two layers. This consumed memory and slowed down requester and other window operations. Now, the gadgets and imagery are brought up directly in the window, saving a layer.
- o The window sizing/dragging rubber-band box is now much faster.
- o The bar-layer of each screen no longer does any backfill processing, since Intuition fully re-renders the screen bar anyway.
- o Intuition now avoids spurious border and gadget refresh and sending a spurious IDCMP_REFRESHWINDOW event. Formerly, if an application had not cleared its window's damage when another damage-causing operation (e.g., menus, window sizing/movement) occurred on the same screen, another round of refresh was performed. Intuition now can tell if the window's layer had been damaged since the last IDCMP_REFRESHWINDOW message went out.
- o Menus are brought on-screen somewhat faster than before, and are removed very much faster than before. (3.01 and up only)
- o A lot of work has been done to reduce window border and gadget flashing during window resize operations. (3.01 and up only)

Bug Fixes

- o NextPubScreen() could write a zero-byte one byte past the end of the buffer the caller supplies. It no longer does this.
- o Clicking in the no-window area of a screen now makes that screen the active screen (for purposes of autoscrolling).
- o Fixed a long-standing bug where REQCLEAR messages weren't being sent when a requester having no layer is taken down while other requesters are still up in the window.

- o Setting a negative minimum width or height with WindowLimits() no longer allows you to crash the computer by turning a window inside out.
- o SetWindowTitles() now properly erases remnants of the previous title, even when odd extenders happen.
- o A ghosted string gadget no longer causes patterning in string gadgets that precede it in the list and have a non-zero container background-pen.
- o NewModifyProp() of a disabled prop gadget no longer clears away part of the ghosting.
- o SetGadgetAttrs() to a proportional gadget no longer can cause the window's installed clip-region to be lost.
- o The mouse-pointer no longer blanks in the first gap when three interlaced screens up.
- o Several causes of sprite-pointer jumping have been fixed.
- o The ROM default sprite pointer now is the 2.0/3.0 one, not the 1.3 one.
- o DClips of coerced ViewPorts are finally scaled as correctly as possible.

If you used a boopsi string gadget as an integer gadget, with Intuition supplying the buffer, and you specified a STRINGA_MaxChars of > 15, you would get a mismatched FreeMem() when the gadget is disposed. This is now fixed.

There is a bug in 3.0 (not in 2.0x and fixed in 3.01) where the autoscroll boundary was inadvertently switched to be the DClip of the active screen, where it used to be the "hull" of the DClips of all the screens. If there are two screens in the system with different DClips, the mouse can be way outside the DClip of the smaller screen. If that screen is active, it will AutoScroll at a ridiculous rate. For example, if the mouse is seventeen pixels below its DClip, moving it down one pixel causes the screen to autoscroll by eighteen, instead of one. This is now fixed.

Starting with 3.01, Intuition now updates its internal time values based on (nearly) *any* InputEvent it receives, instead of just IECLASS_TIMER ones. The problem was that outgoing IntuiMessages get their time from this internal time, which meant that IntuiMessage time was the time-stamp of the most recent timer tick, instead of the time-stamp of the event that actually triggered this IntuiMessage. This problem completely precludes correlating an IntuiMessage with the InputEvent that caused it, which is important for tablet people, for example. A SetPatch for earlier ROM versions is being considered.

In 3.00, there is a bug where the part of a window obscuring the title-bar area of a SCREENQUIET screen wasn't erased when the window was closed or moved away. Effectively, Intuition was relying on a layers side-effect that was optimized out for V39. Intuition fixes this for 3.01.

Starting with 3.01, when Intuition splits a single InputEvent into button and movement components, the button event is now sent first. This fixes some inconsistencies with extended input information like pressure, as well as odd behavior of the qualifiers, in particular IEQUALIFIER_MIDBUTTON.

◆

DataTypes

by David N. Junod

Introduction

DataTypes provides an object oriented approach for determining data types and handling those data types.

There are many advantages to using DataTypes:

- ❑ An application can detect what data type a file is and handle it accordingly. An example of this would be a BBS that examines incoming files and labels them by file type so that the appropriate integrity checks can be applied or the appropriate contents viewer invoked for that archive type. Or a directory utility that invokes that appropriate editor for the data type of the file that the user selects.
- ❑ A developer can easily embedded a DataType viewer within her application without worrying about writing a lot of code. For example, if an application needs to display a picture, or word-wrapped text in a proportional font, it can do it easily using the functions of DataTypes.
- ❑ A developer can easily add clipboard support to her application using the functions of DataTypes. Each root DataType class implements clipboard support in the native Amiga format for that type of data. All the application needs to do is submit its data to a DataType object and invoke the copy method.
- ❑ Applications can handle multiple formats of a data type transparently. For example a Paint package can handle ILBM, GIF, Windows BMP or any other type of picture data as long as there is a DataTypes class to handle it.
- ❑ A data viewer can transparently handle any type of data. An example of this is the MultiView utility that comes with 3.0 or the ClipView utility on the examples disk.
- ❑ It is easy to write a sub-class for most of the data types, because all the class implementor needs to do is convert the data to the internal Amiga format for that data type. For example, the picture class handles the color remapping of a 256 color picture to the screen depth of the destination Amiga.

- ❑ The DataType objects have a consistent interface. There is no difference between displaying a picture, text, or animation.
- ❑ A DataType object can be queried to see what methods and commands they support.

Determining Data Type

One of the main features of the DataTypes system is its ability to determine the data type of a block of data. This data block can reside in a file or the clipboard.

The following functions are used to determine the DataType of a data block:

ObtainDataTypeA()
Obtain the DataType descriptor for a data block.
ReleaseDataType()
Release the DataType descriptor for a data block.

The data type detection functions use the DataType structure.

```
struct DataType  {
    struct Node             dtn_Node1;
    struct Node             dtn_Node2;
    struct DataTypeHeader   *dtn_Header;
    struct List             dtn_ToolList;
    STRPTR                  dtn_FunctionName;
    struct TagItem          *dtn_AttrList;
    ULONG                   dtn_Length;
};
```

The DataType structure is read-only. The only pertinent field is the dtn_Header field, which points to a DataTypeHeader structure.

```
struct DataTypeHeader {
    STRPTR    dth_Name;
    STRPTR    dth_BaseName;
    STRPTR    dth_Pattern;
    WORD      *dth_Mask;
    ULONG     dth_GroupID;
    ULONG     dth_ID;
    WORD      dth_MaskLen;
    WORD      dth_Pad;
    UWORD     dth_Flags;
    WORD      dth_Priority;
};
```

The DataTypeHeader structure fields are as follows:

dth_Name
Descriptive name of the data type. For example, the description for an ILBM data type could possibly be "Amiga BitMap Picture".

dth_BaseName
This is the base name for the data type and is used to obtain the class that handles this data type.

dth_GroupID
This indicates the main type data that the object contains. Following are the possible values:

GID_SYSTEM	Fonts, Executables, Libraries, Devices, etc.
GID_TEXT	Formatted or unformatted text.
GID_DOCUMENT	Formatted text with embedded DataTypes (such as pictures).
GID_SOUND	Audio samples.
GID_INSTRUMENT	Audio samples used for playing music.
GID_MUSIC	Musical scores.
GID_PICTURE	Graphic picture or brush.
GID_ANIMATION	Moving picture or cartoon.
GID_MOVIE	Moving picture or cartoon with sound.

dth_ID
This is an individual identifier for the DataType. For IFF files it is the same as the FORM type, for example ILBM for an Amiga BitMap picture. For non-IFF files, it is the first four characters of dth_Name.

dth_Flags
The flags field contains, among other information, the coarse type of data. The type can be obtained by ANDing DTF_TYPE_MASK with this field.

DTF_IFF	Interchange File Format
DTF_BINARY	Non-readable characters
DTF_ASCII	Readable characters
DTF_MISC	Disks and drawers

dth_Pattern dth_Mask dth_MaskLen dth_Priority
These fields are used by the detection code in datatypes.library for determining the data type. See the ''Defining a DataType Descriptor'' section below for more information.

Following is a code fragment that shows how to determine the datatype of a file. This fragment uses functions from datatypes.library, dos.library, and iffparse.library.

```
STRPTR name = "somefilename";
BPTR lock;

struct DataTypeHeader *dth;
struct DataType *dtn;
UBYTE idesc[5];
STRPTR tdesc;
STRPTR gdesc;
UWORD ttype;

/* Obtain a lock on the file that we want information on */
if (lock = Lock (name, ACCESS_READ))
    {
    /* Get a pointer to the appropriate DataType structure */
    if (dtn = ObtainDataTypeA (DTST_FILE, (APTR)lock, NULL))
        {
        /* Get a pointer to the DataTypeHeader structure */
        dth = dtn->dtn_Header;

        /* Get the coarse type */
        ttype = dth->dth_Flags & DTF_TYPE_MASK;

        /* Get a pointer to the text strings */
        tdesc = GetDTString (ttype + DTMSG_TYPE_OFFSET);
        gdesc = GetDTString (dth->dth_GroupID);

        /* Convert the ID to a string. */
        IDtoStr (dth->dth_ID, idesc);

        /* Display the information */
        printf ("   Description: %s", dth->dth_Name);
        printf ("     Base Name: %s", dth->dth_BaseName);
        printf ("          Type: %d - %s", ttype, tdesc);
        printf ("         Group: %s", gdesc);
        printf ("            ID: %s", idesc);

        /* Release the DataType structure now that we are done with it */
        ReleaseDataType (dtn);
        }

    /* Release the DOS lock on the file */
    UnLock (lock);
    }
```

Embeddding DataTypes

Since DataType objects are a sub-class of the Intuition Gadget class, DataType objects can be attached to an Intuition window in a similar way that gadgets can be added to a window. DataTypes use a parallel set of functions because it requires additional information that the Intuition functions weren't able to provide.

Currently the handling of the data is limited to reading, writing, printing, viewing (audio or visual), and clipboard access. The MultiView utility is an example of an application that embeds DataType objects.

The following functions are used to access DataType objects.

NewDTObjectA()
Obtain a handle on a DataType object.

DisposeDTObject()
Release the handle on a DataType object.

SetDTAttrsA()
Set the attributes of a DataType object.

GetDTAttrsA()
Get attributes of a DataType object

AddDTObject()
Add a DataType object to a window.

RefreshDTObjectA()
Refresh the rendering of a DataType object.

RemoveDTObject()
Remove a DataType object from a window.

GetDTMethods()
Get a list of the methods that a DataTypes object supports. Write, Copy, and Select are examples of methods that an object may support.

GetDTTriggerMethods()
Get a list of trigger methods that a DataType object supports. Actions like Play, Pause, and Resume are examples of trigger methods that an object may support.

DoDTMethod()
Invoke a DataTypes method.

PrintDTObject()
Asynchronously print a DataType object.

GetDTString()
Get the localized text string for a DataTypes text id. Useful for obtaining localized error messages.

Creating a DataType Object

The DataTypes function NewDTObjectA() must be used to create a new DataType object.

```
dto = (Object *) NewDTObjectA (APTR name, struct TagItem *attrs)
```

The pointer that NewDTObjectA() returns is a pointer to a BOOPSI object. Like other BOOPSI objects, DataType objects are "black boxes" and are not to be peeked and poked without using the provided interface.

To create a DataType object, NewDTObjectA() needs to know the where to obtain the data used to create the object. By default the name is treated as a filename.

The attrs tag list is a list of tag/value pairs, each of which contains an initial value for an attribute. There are a number of attributes defined in <datatypes/datatypesclass.h> that can be used at creation time.

DTA_SourceType
: Specifies the type of the source data. The default is DTST_FILE. The types are:

  DTST_RAM
  : Source data is in RAM.

  DTST_FILE
  : Source data is a file. Name is the name of the file.

  DTST_CLIPBOARD
  : Source data is in the clipboard. Name is the unit number. For example, (APTR)0, for clipboard unit zero.

  DTST_HOTLINK
  : Reserved for future use.

DTA_Handle
: Can be used instead of the name field. If the source type is DTST_FILE then handle must be a valid BPTR file handle. If the source type is DTST_CLIPBOARD then handle must be a valid IFFHandle.

DTA_DataType
: Can be used to specify the class for handling the data. Data must be a pointer to a valid DataType. This should only be used when attempting to create a new object that doesn't have source data, or could be handled by multiple classes (for example, could be used to force an object to be handled by the AmigaGuide class).

DTA_GroupID
: If this tag is present, then the data must be of the specified type, or the object creation will fail with ERROR_OBJECT_WRONG_TYPE. This can be used by a Sound editor to ensure that only sounds can be loaded, for example.

DTA_TextAttr
: Specify the font to use for any text rendered by this object.

There are additional attributes that can specified at creation time, but are dependant on the data type of the object being created. See the header files <datatypes/#?class.h> for more attributes.

The following attributes, that are defined in <intuition/gadgetclass.h>, are also valid.

GA_Left
Specify the left edge of the gadget.

GA_RelRight
Specify the left edge of the gadget being relative to the right edge of the containing window.

GA_Top
Specify the top edge of the gadget.

GA_RelBottom
Specify the top edge of the gadget being relative to the bottom edge of the containing window.

GA_Width
Specify the width of the gadget.

GA_RelWidth
Specify the width of the gadget being relative to the width of the containing window.

GA_Height
Specify the height of the gadget.

GA_RelHeight
Specify the height of the gadget being relative to the height of the containing window.

GA_ID
Specify a ID associated with the gadget.

GA_UserData
Attach application data to the gadget.

GA_Immediate
Indicate that the application should be notified of gadgetdown events for this gadget.

GA_RelVerify
Indicate that the application should be notified of gadgetup events for this gadget.

GA_Previous
For adding the gadget to a list of gadgets.

GA_DrawInfo
A pointer to a struct DrawInfo.

In order for the application to receive information from the DataType object, it must set up a target for the notification attributes that the object sends out.

ICA_TARGET
: Specify a target for the notification attributes that the DataType object sends out.

ICA_MAP
: Specify an attribute mapping for the notification attributes that the DataType object sends out.

The usual method to obtain notification is to set up an ICA_TARGET of ICTARGET_IDCMP so that the application will receive the attributes via the IDCMP_IDCMPUPDATE Intuition message class. But it is also possible to set up another BOOPSI object as the receiver.

If NewDTObjectA() is successful, it returns a pointer to a DataType object. Otherwise it returns NULL and the reason for failure can be obtained using IoErr(). See the Autodocs for datatypes.library for more information.

Disposing of a DataType Object

When the application is done with the DataType object it has to dispose of the object. To dispose of a DataType object, you must use the DataTypes function DisposeDTObject():

```
VOID DisposeDTObject (Object *dto)
```

where dto is a pointer to the DataType object to be disposed. This will abort any PLAY trigger method, such as playing sounds or animations, but will wait for a print or save to complete.

Obtaining Environment Information for a DataType

In order to embed a DataType object in a window, it is necessary to ask the object what its minimum environment is. For example, since the remap code doesn't handle remapping HAM pictures, they must be shown on a HAM screen, and therefore can't be added to a window that is on a non-HAM screen.

```
ULONG modeid = INVALID_ID;
LONG nomwidth, nomheight;
BOOL useScreen = FALSE;
struct dtFrameBox dtf;
struct FrameInfo fri;

/* Get the attributes that we are interested in */
```

```
GetDTAttrs (dto,
                PDTA_ModeID,           &modeid, /* Get the mode ID */
                /* Get the desired size */
                DTA_NominalHoriz,      &nomwidth,
                DTA_NominalVert,       &nomheight,
                TAG_DONE);

/* Clear the structures */
memset (&dtf, 0, sizeof (struct dtFrameBox));
memset (&fri, 0, sizeof (struct FrameInfo));

/* Fill in the message */
dtf.MethodID = DTM_FRAMEBOX;
dtf.dtf_FrameInfo = &fri;
dtf.dtf_ContentsInfo = &fri;
dtf.dtf_SizeFrameInfo = sizeof (struct FrameInfo);

/* Perform the frame method */
if (DoDTMethodA (dto, NULL, NULL, (Msg) &dtf)){
    /* Check to see if the object requires a HAM screen */
    if (fri.fri_PropertyFlags & DIPF_IS_HAM) {
        printf ("HAM");
        useScreen = TRUE;
        }
        /* Check to see if the object requires an ExtraHalfBrite screen */
    else if (fri.fri_PropertyFlags & DIPF_IS_EXTRAHALFBRITE){
            printf ("ExtraHalfBrite");
            useScreen = TRUE;
            }
            /* A safety check to see if a screen is required */
            else if ((fri.fri_PropertyFlags == 0) && (modeid & 0x800)
                                                 && (modeid != INVALID_ID)) {
                    printf ("ModeID=0x%08lx", modeid);
                    useScreen = TRUE;
                    }
        }
else
    {
    /* No special environment required, can be attached to any screen mode */
    }
```

Adding a DataType Object to a Window

A DataType object must be added to a window using the AddDTObject() function of DataTypes.

```
LONG AddDTObject (struct Window *w, struct Requester *r, Object *dto,
                  LONG pos)
```

This function will add a DataTypes object to the existing gadget list for the specified window. The recommended value for pos is -1 which will cause the DataType object to be added to the end of the list.

DataType objects should not be added using the WA_Gadgets attribute to OpenWindowTagList() or by using the AddGList() function. There is special information that DataTypes requires that will not obtained if any method other than AddDTObject() is used.

When the DataType object is added to the window, the layout method for the object will be invoked. It is possible that the layout will take a while to perform, in that case the object will spawn a process to handle the layout asynchronously. In order to refresh the object's visual information, it is necessary to obtain IDCMP_IDCMPUPDATE messages from the object and refresh the object when a DTA_Sync attribute is received.

The following code fragment illustrates adding a DataType object to a window.

```
Object *dto;

struct IntuiMessage *imsg;
struct Window *win;
ULONG sigr;

struct TagItem *tstate, *tags;
ULONG tidata;
ULONG errnum;

BOOL going = TRUE;

/* Set the pertinent attributes of the DataType object */
SetDTAttrs (dto, NULL, NULL,

            /* Set the dimensions of the object */
            GA_Left,    win->BorderLeft,
            GA_Top,     win->BorderTop,
            GA_Width,   win->Width - win->BorderLeft - win->BorderRight,
            GA_Height,  win->Height - win->BorderTop - win->BorderBottom,

          /* Make sure we receive IDCMP_IDCMPUPDATE messages from the object */
          ICA_TARGET, ICTARGET_IDCMP,
          TAG_DONE);

/* Add the object to the window */
AddDTObject (win, NULL, dto, -1);

/* Refresh the DataType object */
RefreshDTObjects (dto, win, NULL, NULL);

/* Keep going until we're told to stop */
while (going)
{
    /* Wait for an event */
    sigr = Wait ((1L << win->UserPort->mp_SigBit) | SIGBREAKF_CTRL_C);

    /* Did we get a break signal */
    if (sigr & SIGBREAKF_CTRL_C)
        going = FALSE;

    /* Pull Intuition messages */
    while (imsg = (struct IntuiMessage *) GetMsg (win->UserPort))
    {
        /* Handle each message */
        switch (imsg->Class)
        {
            case IDCMP_IDCMPUPDATE:
                /* Get a pointer to the attribute list */
                tstate = tags = (struct TagItem *) imsg->IAddress;

                /* Step through the attribute list */
                while (tag = NextTagItem (&tstate))
                {
                        tidata = tag->ti_Data;
                        switch (tag->ti_Tag)
                        {
                            /* Change in busy state */
                            case DTA_Busy:
                                if (tidata)
                                    SetWindowPointer (win, WA_BusyPointer, TRUE,
                                                      TAG_DONE);
                                else
                                    SetWindowPointer (win, WA_Pointer, NULL,
                                                      TAG_DONE);
                                break;

                            /* Error message */
                            case DTA_ErrorLevel:
                                if (tidata)
                                {
                                    errnum = GetTagData (DTA_ErrorNumber, NULL,
                                                      tags);
                                    PrintErrorMsg (errnum,
                                                      (STRPTR)options[OPT_NAME]);
                                }
                                break;

                            /* Time to refresh */
                            case DTA_Sync:
                                /* Refresh the DataType object */
                                RefreshDTObjects (dto, win, NULL, NULL);
                                break;
                        }
                    }
                    break;
            }

            /* Done with the message, so reply to it */
            ReplyMsg ((struct Message *) imsg);
        }
    }
```

Removing a DataType Object from a Window

A DataType object must be removed from the window using the RemoveDTObject() function.

```
LONG RemoveDTObject (struct Window *w, Object *dto)
```

This function removes the DataType object from the window's gadget list.

This is the only way that a DataType object should be removed from the window list. Using RemoveGList() is not supported, nor is removing the object manually.

Setting an Existing DataType Object's Attributes

An objects attributes are not necessarily static. An application can ask an object to set certain attributes using the SetDTAttrs() function.

```
ULONG SetDTAttrsA (Object *dto, struct Window *w,
                   struct Requester *r, struct TagItem *attrs)
```

The return value is DataType object specific, but generally a non-zero value means that the object needs to be visually refreshed.

The following fragment illustrates how to set the current top values for a DataType object, using the VarArgs version of SetDTAttrsA().

```
SetDTAttrs (dto, window, NULL,
          DTA_TopVert, 0,
          DTA_TopHoriz, 0,
          TAG_DONE);
```

This will cause the DataType object to update its vertical and horizontal top values. If the object has been added to a window, then the display will be updated accordingly. Note that it is not okay to call SetGadgetAttrs() or SetAttrs() on a DataType object.

Getting a DataType Object's Attributes

The DataTypes function GetDTAttrsA() is used to obtain the values for a list of attributes from a DataType object.

```
ULONG GetDTAttrsA (Object *dto, struct TagItem *attrs)
```

Where dto is a pointer to a DataType object returned by NewDTObjectA().

And attrs is a TAG_DONE terminated array of attributes. Where the data element of each pair contains the address of the storage variable for that attribute.

This function will return a number that indicates that number of attributes that it was able to obtain. For example, if four attributes asked for and GetDTAttrs returns a four, then all the attributes were obtained.

The following code fragment illustrates how to get the current top values for a DataTypes object using the VarArgs form of GetDTAttrsA().

```
LONG topv, toph;

if (GetDTAttrs (dto, DTA_TopVert, &topv, DTA_TopHoriz, &toph, TAG_DONE) == 2)
   {
   printf ("Top: vertical=%ld, horizontal=%ld", topv, toph);
   }
else
   {
   ("couldn't obtain the top values");
   }
```

Writing A Sub-Class

For each of the DataType categories, there is a class that handles the data. Handlers for explicit data are sub-classes under the appropriate class. For example, a class that handles ILBM pictures would be a sub-class of the Picture class.

In order to fully understand the class concept used by DataTypes it helps to have a basic understanding of BOOPSI.

A sub-class must provide an OM_NEW method that converts the source data into the data format required by the super-class. The sub-class must optionally have a OM_DISPOSE method that discards any of the data constructed in the OM_NEW method. All other methods must be passed to the super-class.

Following is a listing of a example class dispatcher for a sub-class of the picture class:

```
ULONG ASM Dispatch (REG (a0) Class *cl, REG (a2) Object *o, REG (a1) Msg msg)
{
    struct ClassBase *cb = (struct ClassBase *) cl->cl_UserData;
    ULONG retval;

    switch (msg->MethodID)
    {
        case OM_NEW:
            if (retval = DoSuperMethodA (cl, o, msg))
            {
                /* Convert the source data to required data format */
                if (!GetObjectData (cb, cl, (Object *)retval,
                             ((struct opSet *) msg)->ops_AttrList))
                {
                    /* Force disposal of the object */
                    CoerceMethod (cl, (Object *) retval, OM_DISPOSE);
                    retval = NULL;
    }
            }
            break;

        /* Let the superclass handle everything else */
        default:
            retval = (ULONG) DoSuperMethodA (cl, o, msg);
            break;
    }

    return (retval);
}
```

Obtaining a Handle to the Source Data

The first step that a class has to perform in order to convert the source data, is to get a handle on the data. This is obtained by passing the DTA_Handle tag to the super-class using the OM_GET method. For example:

```
Object *o;
BPTR fh;

GetDTAttrs (o, DTA_Handle, &fh, TAG_DONE);
```

If the source Type is DTF_IFF, then DTA_Handle points to a struct IFFHandle that is already initialized and opened for reading, otherwise the handle points to a BPTR file handle.

Handling Errors

Whenever an error occurs and the class is unable to continue converting the data, then it must set an appropriate error using the DOS SetIoErr() function and the <dos/dos.h> error codes or the error codes defined in <datatypes/datatypes.h>.

For example, if the class is unable to allocate memory:

```
if (buf = AllocVec (size, MEMF_CLEAR))
    {
    ... continue with conversion ...
    }
else
    {
    SetIoErr (ERROR_NO_FREE_STORE);
    }
```

Data Format

The following sections outline the required data format for each of the main object types.

Picture Class

The picture class is the super-class for any static graphic classes. The structures and tags used by this class are defined in <datatypes/pictureclass.h>.

A picture sub-class must fill out a BitMapHeader structure as well as provide a Mode ID, a BitMap and a ColorMap during the OM_NEW method of the class. There is no need to provide any other methods, as the remainder is handled by the picture class itself.

The picture sub-class needs to fill in any fields of the BitMapHeader structure that the class has data for. This will ensure that the picture data can then be saved to a file or copied to the clipboard.

```
struct BitMapHeader *bmh;

/* Obtain a pointer to the BitMapHeader structure from the picture class */
```

```
if (GetDTAttrs (dto, PDTA_BitMapHeader, &bmh, TAG_DONE) && bmh)
   {
   /* Fill in some fields */
   bmh->bmh_Width  = 640;
   bmh->bmh_Height = 200;
   bmh->bmh_Depth  = 2;
   }
```

Before manipulating any color information, the picture sub-class must first tell the picture class how many colors it has so that the information required for color remapping can be established.

```
SetDTAttrs (dto, PDTA_NumColors, ncolors, TAG_DONE);
```

After the number of colors have been established, the palette information can be filled in for the picture. The following fragment illustrates filling in the palette information.

```
struct ColorRegister *cmap;
WORD n, ncolors;
LONG *cregs;

/* Get a pointer to the color registers that need to be filled in */
GetDTAttrs (dto,
             PDTA_ColorRegisters, &cmap,
             PDTA_CRegs,          &cregs,
             TAG_DONE);

/* Set the color information */
for (n = 0; n < ncolors; n++)
      {
      if (Read (fh, &rgb, QSIZE) == QSIZE)
         {
         /* Set the master color table */
         cmap->red   = rgb.rgbRed;
         cmap->green = rgb.rgbGreen;
         cmap->blue  = rgb.rgbBlue;
         cmap++;

         /* Set the color table used for remapping */
         cregs[n * 3 + 0] = rgb.rgbRed   << 24;
         cregs[n * 3 + 1] = rgb.rgbGreen << 24;
         cregs[n * 3 + 2] = rgb.rgbBlue  << 24;
         }
         else
            {
            /* Indicate that we encountered an error.  DOS Read
             * will have already filled in the IoErr() value */
            return (FALSE);
            }
      }
```

The picture class must get the actual picture information in standard Amiga bitmap format. If the bitmap is allocated using the graphics AllocBitMap() function, then the picture class can dispose of the bitmap at OM_DISPOSE time.

```
struct BitMap *bm;
if (bm = AllocBitMap (bmh->bmh_Width, bmh->bmh_Height, bmh->bmh_Depth, BMF_CLEAR,
                                                                        NULL))
    {
    /* Tell the picture class about the picture data */
    SetDTAttrsA (dto, PDTA_BitMap, bm, TAG_DONE);
    }
else
    {
    /* Indicate the error and that we encountered an error */
    SetIoErr (ERROR_NO_FREE_STORE);
    return (FALSE);
    }
```

A picture sub-class needs to provide the following fields to the super-class, during the OM_NEW method:

DTA_NominalHoriz (LONG)
Set to the width of the picture.

DTA_NominalVert (LONG)
Set to the height of the picture.

PDTA_ModeID (LONG)
A valid mode ID for the BitMap.

DTA_ObjName (STRPTR)
The name, or title, of the picture

DTA_ObjAuthor (STRPTR)
The author of the picture.

DTA_ObjAnnotation (STRPTR)
Notes on the picture.

DTA_ObjCopyright (STRPTR)
Copyright notice for the picture.

DTA_ObjVersion (STRPTR)
Version of the picture.

If a picture sub-class uses something other than AllocBitMap() to allocate the bitmap, it must free the bitmap itself by implementing an OM_DISPOSE method for the sub-class.

```
ULONG ASM Dispatch (REG (a0) Class *cl, REG (a2) Object *o, REG (a1) Msg msg)
{
    struct ClassBase *cb = (struct ClassBase *) cl->cl_UserData;
    struct localData *lod;
    ULONG retval;
    switch (msg->MethodID)
    {
        case OM_NEW:
            /* ... */
            break;
        case OM_DISPOSE:
            /* Get a pointer to our object data */
            lod = INST_DATA (cl, o);
            /* Tell the picture class that it doesn't have a
             * bitmap to free any more */
            SetDTAttrs (o, PDTA_BitMap, NULL, TAG_DONE);

            /* Free the bitmap ourself */
            myfreebitmap (lod->lod_BitMap);

        /* Let the superclass handle everything else */
        default:
            retval = (ULONG) DoSuperMethodA (cl, o, msg);
            break;
    }

    return (retval);
}
```

Sound Class

The sound class is the super-class for any sampled audio classes. The structures and tags used by this class are defined in <datatypes/soundclass.h>.

A sound sub-class needs to provide the following fields to the super-class, during the OM_NEW method:

SDTA_Sample (UBYTE *)
8-bit sound data. The sound class will FreeVec() the sample data.
SDTA_SampleLength (ULONG)
Number of 8-bit bytes in the sound data.
SDTA_Volume (UWORD)
Number ranging from 0 being the quietest to 64 being the loudest.
SDTA_Period (UWORD)
Amount of time to play the sound.
SDTA_Cycles (UWORD)
Number of times to play the sound. 0 being an infinite loop.
DTA_ObjName (STRPTR)
The name, or title, of the sound
DTA_ObjAuthor (STRPTR)
The author of the sound.
DTA_ObjAnnotation (STRPTR)
Notes on the sound.
DTA_ObjCopyright (STRPTR)
Copyright notice for the sound.
DTA_ObjVersion (STRPTR)
Version of the sound.

No methods other than OM_NEW are needed as the remainder is handled by the sound class itself.

The sound sub-class also needs to set any fields of the VoiceHeader structure that the class has data for. This will ensure that the sound data can be saved to a file or copied to the clipboard.

```
struct VoiceHeader *vh;

/* Obtain a pointer to the VoiceHeader structure from the sound class */
if (GetDTAttrs (dto, SDTA_VoiceHeader, &vh, TAG_DONE) && vh)
{
    /* Fill in some fields */
    vh->vh_Octaves     = 1;
    vh->vh_Compression = 0;
    vh->vh_Volume      = 63;
}
```

If a sound sub-class uses something other than AllocVec() to allocate the sound data, then it must free the sample itself. This is done by implementing an OM_DISPOSE method for the sub-class.

```
ULONG ASM Dispatch (REG (a0) Class *cl, REG (a2) Object *o, REG (a1) Msg msg)
{
    struct ClassBase *cb = (struct ClassBase *) cl->cl_UserData;
    struct localData *lod;
    ULONG retval;

    switch (msg->MethodID)
    {
        case OM_NEW:
            /* ... */
            break;

        case OM_DISPOSE:
            /* Get a pointer to our object data */
            lod = INST_DATA (cl, o);

            /* Tell the sound class that it doesn't have a
             * sample to free any more */
            SetDTAttrs (o, SDTA_Sample, NULL, TAG_DONE);

            /* Free the sample ourself */
            FreeMem (lod->lod_Sample, lod->lod_SampleLength);

        /* Let the superclass handle everything else */
        default:
            retval = (ULONG) DoSuperMethodA (cl, o, msg);
            break;
    }

    return (retval);
}
```

Text Class

The text class is the super-class for any formatted or non-formatted text classes. The structures and tags used by this class are defined in <datatypes/textclass.h>.

The text class provides the sub-class with a buffer containing the text data. The sub-class must then provide the text class with a list of Line segments during the layout method. The Line list should only be created at gpl_Initial time, unless the TDTA_WordWrap attribute is TRUE.

```
struct Line {
   struct MinNode    ln_Link;
   STRPTR      ln_Text;
   ULONG       ln_TextLen;
   UWORD       ln_XOffset;
   UWORD       ln_YOffset;
   UWORD       ln_Width;
   UWORD       ln_Height;
   UWORD       ln_Flags;
   BYTE        ln_FgPen;
   BYTE        ln_BgPen;
   ULONG       ln_Style;
   APTR        ln_Data;
};
```

The Line structure fields are as follows:

ln_Link
   MinNode used to link to the line list.

ln_Text
   Pointer to the text for this line segment.

ln_TextLen
   Number of bytes of text in this line segment.

ln_XOffset
   Left pixel offset from the left edge of the object for this line segment.

ln_YOffset
   Top pixel offset from the top edge of the object for this line segment.

ln_Width
   Width of the line segment in pixels.

ln_Height
   Height of the line segment in pixels.

ln_Flags
   Control flags for this line segment.

LNF_LF
   Used to indicate that this segment is the end of a line.

ln_FgPen
   Pen to use for the foreground (the text color) for this line segment.

ln_BgPen
   Pen to use for the background color for this line segment.

ln_Style
   Text attribute soft style to use for this line segment.

As each Line segment is allocated it must be added to the Line list.

```
struct List *linelist;
struct Line *line;

/* Get a pointer to the line list */
if (GetDTAttrs (o, TDTA_LineList, (ULONG) &linelist, TAG_DONE) && linelist)
{
    /* Create a Line segment */
    if (line = AllocVec (sizeof (struct Line), MEMF_CLEAR))
    {
        /* Add it to the list */
        AddTail (linelist, (struct Node *)&line->ln_Link);
    }
}
```

Currently, this is the hardest class to sub-class due to the number of methods that must be implemented. Below is the shell for a dispatcher and the layout method for a text sub-class.

Shell for a dispatcher and the layout method for a text sub-class

```
struct localData
{
  VOID  *lod_Pool;
  ULONG  lod_Flags;
};

ULONG ASM Dispatch (REG (a0) Class * cl, REG (a2) Object * o, REG
(a1) Msg msg)
{
  struct ClassBase *cb = (struct ClassBase *) cl->cl_UserData;
  struct localData *lod;
  struct List *linelist;
  ULONG retval = 0L;

  switch (msg->MethodID)
  {
  case OM_NEW:
    if (retval = DoSuperMethodA (cl, o, msg))
    {
      ULONG len, estlines, poolsize;
      BOOL success = FALSE;
      STRPTR buffer;

      /* Get a pointer to the object data */
      lod = INST_DATA (cl, (Object *) retval);

      /* Get the attributes that we need to determine
       * memory pool size */
      GetDTAttrs ((Object *) retval,
            TDTA_Buffer,     (ULONG)&buffer,
            TDTA_BufferLen,    (ULONG)&len,
            TAG_DONE);

      /* Make sure we have a text buffer */
      if (buffer && len)
      {
        /* Estimate the pool size that we will need */
        estlines = (len / 80) + 1;
        estlines = (estlines > 200) ? 200 : estlines;
        poolsize = sizeof (struct Line) * estlines;

        /* Create a memory pool for the line list */
        if (lod->lod_Pool = CreatePool (MEMF_CLEAR | MEMF_PUBLIC,
                         poolsize, poolsize))
          success = TRUE;
        else
          SetIoErr (ERROR_NO_FREE_STORE);
      }
      else
      {
        /* Indicate that something was missing that we
         * needed */
        SetIoErr (ERROR_REQUIRED_ARG_MISSING);
      }

      if (!success)
      {
        CoerceMethod (cl, (Object *) retval, OM_DISPOSE);
        retval = NULL;
      }
    }
    break;
```

```
case OM_UPDATE:
case OM_SET:
  /* Pass the attributes to the text class and force a refresh
   * if we need it */
  if ((retval = DoSuperMethodA (cl, o, msg)) && (OCLASS (o) == cl))
  {
    struct RastPort *rp;

    /* Get a pointer to the rastport */
    if (rp = ObtainGIRPort (((struct opSet *) msg)->ops_GInfo))
    {
      struct gpRender gpr;

      /* Force a redraw */
      gpr.MethodID  = GM_RENDER;
      gpr.gpr_GInfo = ((struct opSet *) msg)->ops_GInfo;
      gpr.gpr_RPort = rp;
      gpr.gpr_Redraw = GREDRAW_UPDATE;
      DoMethodA (o, &gpr);

      /* Release the temporary rastport */
      ReleaseGIRPort (rp);
    }
    retval = 0;
  }
  break;

case GM_LAYOUT:
  /* Tell everyone that we are busy doing things */
  notifyAttrChanges (o, ((struct gpLayout *) msg)->gpl_GInfo, NULL,
             GA_ID,     G(o)->GadgetID,
             DTA_Busy,  TRUE,
             TAG_DONE);

  /* Let the super-class partake */
  retval = (ULONG) DoSuperMethodA (cl, o, msg);

  /* We need to do this one asynchronously */
  retval += DoAsyncLayout (o, (struct gpLayout *) msg);
  break;

case DTM_PROCLAYOUT:
  /* Tell everyone that we are busy doing things */
  notifyAttrChanges (o, ((struct gpLayout *) msg)->gpl_GInfo, NULL,
             GA_ID,    G(o)->GadgetID,
             DTA_Busy,  TRUE,
             TAG_DONE);
 /* Let the super-class partake and then fall through to our
  * layout method */
  retval = (ULONG) DoSuperMethodA (cl, o, msg);

case DTM_ASYNCLAYOUT:
  /* Layout the text */
  retval = layoutMethod (cb, cl, o, (struct gpLayout *) msg);
  break;

case OM_DISPOSE:
  /* Get a pointer to our object data */
  lod = INST_DATA (cl, o);

  /* Don't let the super class free the line list */
  if (GetDTAttrs (o, TDTA_LineList, (ULONG) &linelist, TAG_DONE)
&& linelist)
```

```
    NewList (linelist);

   /* Delete the line pool */
   DeletePool (lod->lod_Pool);

 /* Let the superclass handle everything else */
 default:
   retval = (ULONG) DoSuperMethodA (cl, o, msg);
   break;
 }

 return (retval);
}

ULONG layoutMethod (struct ClassBase *cb, Class * cl, Object * o,
struct gpLayout * gpl)
{
  struct DTSpecialInfo *si = (struct DTSpecialInfo *) G
(o)->SpecialInfo;
  struct localData *lod = INST_DATA (cl, o);
  ULONG visible = 0, total = 0;
  struct RastPort trp;
  ULONG hunit = 1;
  ULONG bsig = 0;

  /* Switches */
  BOOL linefeed = FALSE;
  BOOL newseg = FALSE;
  BOOL abort = FALSE;

  /* Attributes obtained from super-class */
  struct TextAttr *tattr;
  struct TextFont *font;
  struct List *linelist;
  struct IBox *domain;
  ULONG wrap = FALSE;
  ULONG bufferlen;
  STRPTR buffer;
  STRPTR title;

  /* Line information */
  ULONG num, offset, swidth;
  ULONG anchor, newanchor;
  ULONG style = FS_NORMAL;
  struct Line *line;
  ULONG yoffset = 0;
  UBYTE fgpen = 1;
  UBYTE bgpen = 0;
  ULONG tabspace;
  ULONG numtabs;
  ULONG i, j;
  ULONG nomwidth, nomheight;

  /* Get all the attributes that we are going to need for a
successful layout */
  if (GetDTAttrs (o,
              DTA_TextAttr,  (ULONG) &tattr,
              DTA_TextFont,  (ULONG) &font,
              DTA_Domain,    (ULONG) &domain,
              DTA_ObjName,   (ULONG) &title,
              TDTA_Buffer,   (ULONG) &buffer,
              TDTA_BufferLen, (ULONG) &bufferlen,
              TDTA_LineList,  (ULONG) &linelist,
```

```
              TDTA_WordWrap,  (ULONG) &wrap,
              TAG_DONE) == 8)
 {
  /* Lock global object data so nobody else can manipulate it */
  ObtainSemaphore (&(si->si_Lock));

  /* Make sure we have a buffer */
  if (buffer)
  {
    /* Initialize the temporary RastPort */
    InitRastPort (&trp);
    SetFont (&trp, font);

    /* Calculate the nominal size */
    nomheight = (ULONG) (24 * font->tf_YSize);
    nomwidth = (ULONG) (80 * font->tf_XSize);

    /* Calculate the tab space */
    tabspace = font->tf_XSize * 8;

    /* We only need to perform layout if we are doing word wrap,
     * or this is the initial layout call */
    if (wrap || gpl->gpl_Initial)
    {
      /* Delete the old line list */
      while (line = (struct Line *) RemHead (linelist))
        FreePooled (lod->lod_Pool, line, sizeof (struct Line));

      /* Step through the text buffer */
      for (i = offset = num = numtabs = 0;
         (i <= bufferlen) && (bsig == 0) && !abort;
         i++)
      {
        /* Check for end of line */
        if (buffer[i]==13 && buffer[i+1]==10)
        {
          newseg = linefeed = TRUE;
          newanchor = i + 2;
          i++;
        }
        /* Check for end of page */
        else if (buffer[i] == 12)
        {
          newseg = linefeed = TRUE;
          newanchor = i + 1;
        }
        /* Check for tab */
        else if (buffer[i] == 9)
        {
          /* See if we need to terminate a line segment */
          if ((numtabs == 0) && num)
            newseg = TRUE;
          numtabs++;
        }
        else
        {
          /* See if we have any TABs that we need to finish out */
          if (numtabs)
          {
           offset += (((offset / tabspace) + 1) * tabspace) - offset;
           num = numtabs = 0;
           anchor = i;
          }
        /* Compute the width of the line. */
        swidth = TextLength (&trp, &buffer[anchor], num+1);
        if (offset + swidth > domain->Width)
        {
          /* Search for a whitespace character */
          for (j = i; (j >= anchor) && !newseg; j--)
          {
            if (buffer[j] == ' ')
            {
              num -= (i - j);
              newseg = TRUE;
              i = j + 1;
            }
          }

          newseg = linefeed = TRUE;
          newanchor = i;
          i--;
        }
        else
        {
          num++;
        }
      }

      /* Time for a new text segment yet? */
      if (newseg)
      {
        /* Allocate a new line segment from our memory pool */
         if (line = AllocPooled (lod->lod_Pool,
                                 sizeof (struct Line)))
        {
          swidth = TextLength (&trp, &buffer[anchor], num);
          line->ln_Text    = &buffer[anchor];
          line->ln_TextLen = num;
          line->ln_XOffset = offset;
          line->ln_YOffset = yoffset + font->tf_Baseline;
          line->ln_Width   = swidth;
          line->ln_Height  = font->tf_YSize;
          line->ln_Flags   = (linefeed) ? LNF_LF : NULL;
          line->ln_FgPen   = fgpen;
          line->ln_BgPen   = bgpen;
          line->ln_Style   = style;
          line->ln_Data    = NULL;

          /* Add the line to the list */
          AddTail (linelist, (struct Node *) line);

          /* Increment the line count */
          if (linefeed)
          {
            yoffset += font->tf_YSize;
            offset = 0;
            total++;
          }
          else
          {
            /* Increment the offset */
            offset += swidth;
          }
        }
          else
          {
            abort = TRUE;
          }

          /* Clear the variables */
          newseg = linefeed = FALSE;
          anchor = newanchor;
          num = 0;

          /* Check to see if layout has been aborted */
          bsig = CheckSignal (SIGBREAKF_CTRL_C);
        }
      }
    }
    else
    {
      /* No layout to perform */
      total = si->si_TotVert;
    }
  }

  /* Compute the lines and columns type information */
  si->si_VertUnit = font->tf_YSize;
  si->si_VisVert  = visible = domain->Height / si->si_VertUnit;
  si->si_TotVert  = total;

  si->si_HorizUnit = hunit = 1;
  si->si_VisHoriz = (LONG) domain->Width / hunit;
  si->si_TotHoriz = domain->Width;

  /* Release the global data lock */
  ReleaseSemaphore (&si->si_Lock);

  /* Were we aborted? */
  if (bsig == 0)
  {
    /* Not aborted, so tell the world of our newest attributes */
    notifyAttrChanges (o, gpl->gpl_GInfo, NULL,
              GA_ID,            G(o)->GadgetID,

              DTA_VisibleVert,  visible,
              DTA_TotalVert,    total,
              DTA_NominalVert,  nomheight,
              DTA_VertUnit,     font->tf_YSize,

              DTA_VisibleHoriz, (ULONG) (domain->Width / hunit),
              DTA_TotalHoriz,   domain->Width,
              DTA_NominalHoriz, nomwidth,
              DTA_HorizUnit,    hunit,

              DTA_Title,        title,
              DTA_Busy,         FALSE,
              DTA_Sync,         TRUE,
              TAG_DONE);
  }
 }
 return (total);
}
```

Defining a DataType Descriptor

DataTypes uses a simple descriptor to determine what type of data a file contains and what class, if any, is used to handle that data.

The DTDesc utility is used to define a DataType descriptor. The following steps describe how to use this utility to define a descriptor.

1. Load several sample files of the type being defined. This can be done by dropping their icons into the DTDesc window or by using the "Extras/Load Samples..." menu item.
2. The view area in the bottom right side of the DTDesc window will show the first 64 characters of the files. This area is used to define the Mask. The characters that don't match will be blotted out and only the similar characters will be shown. If more characters are shown as similar than really are, then they can easily be blotted out by rubbing over them.
3. DataTypes can be divided into several different categories. Use the Group menu to select the category the DataType descriptor belongs in.
4. The remaining fields must be filled out. Following is a table describing the fields and the information they require.

Name	Description
File Type	User description of the DataType.
Base Name	The base name of the DataType. The class library name is derived from this.
Name Pattern	The name of the file can be used to indicate the DataType. AmigaDOS wildcards can be used to specifythe file name.
Function	A function can be used to further define a data type. This function must be a stand-alone executable that uses the DataType Descriptor Function Interface.
Case Sensitive?	The Mask can be either case sensitive or not.
Priority	Descriptors are sorted by Type, Function, NamePattern, and Mask. The priority field allows a DataType descriptor to be assigned a different priority than a similar DataType descriptor.
Type	This is a read-only field and is used to indicate what basic type a DataType is. Types include IFF, Binary and ASCII.

5. Once a DataType descriptor has been defined, it must be saved. Select the "Project/SaveAs..." menu item for the file requester used to save a DataType descriptor. The default name for a DataType descriptor is the text entered into File Type field.
6. In order for a new DataType descriptor to be loaded, the datatypes.library must be flushed from the system. This can either be done with the FlushLibs command or by using Avail Flush several times. Another way that the new DataType descriptor can be loaded is by using the AddDataTypes command with the REFRESH option.

DataType Descriptor Function Interface

Sometimes the fields within the DTDesc utility are not enough to define a DataType descriptor. In this case it is necessary to use a Function to further narrow down a DataType.

A Function is a stand-alone executable that expects the following arguments.

```
retval = Function (dthc);
d0                 a0

BOOL Function (struct DTHookContext *);
```

The function must return TRUE if the data matches, FALSE if it doesn't.

The DTHookContext structure contains the fields that are necessary for narrowing down the DataType. Following is a listing of the DTHookContext structure.

```
struct DTHookContext {
struct Library          *dthc_SysBase;
struct Library          *dthc_DOSBase;
struct Library          *dthc_IFFParseBase;
struct Library          *dthc_UtilityBase;

/* File context */
BPTR                     dthc_Lock;         /* Lock on the file */
struct FileInfoBlock    *dthc_FIB;          /* Pointer to a FileInfoBlock */
BPTR                    dthc_FileHandle; /* Pointer to the file handle (may be NULL) */
struct IFFHandle        *dthc_IFF;          /* Pointer to an IFFHandle (may be NULL) */
STRPTR                  dthc_Buffer;        /* Buffer */
ULONG                   dthc_BufferLength;  /* Length of the buffer */
};
```

The DTHookContext structure fields are as follows:

dthc_SysBase dthc_DOSBase dthc_IFFParseBase dthc_UtilityBase
These are library bases that your function can utilitize. It will need to open any other libraries that it needs.

dthc_Lock
If the source data is a DOS file, then this is a lock on the file.

dthc_FIB
If the source data is a DOS file, then this is a filled in FileInfoBlock for the file.

dthc_FileHandle
If the source data is a DOS file, then this is the file handle for the file otherwise the handle will be NULL. The file is guaranteed to be at the beginning.

dthc_IFF
If the source data is IFF, then this is the IFFHandle for accessing the data, otherwise the handle will be NULL. The position is guaranteed to be at the beginning of the data. The DOS file fields must not be accessed if the data is IFF.

dthc_Buffer
This buffer contains the first dthc_BufferLength bytes of data.

dthc_BufferLength
Indicates the number of bytes in dthc_Buffer, up to 64.

The following example shows how to write a simple DataTypes Descriptor Function.

```
/* This example is to be compiled with no stack checking.  It must not be linked *
with any startup code so that DTHook is the entry point for the executable.
 */
#include <exec/types.h>
#include <dos/dos.h>
#include <dos/dosextens.h>
#include <datatypes/datatypes.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/utility_protos.h>

#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>
#include <pragmas/utility_pragmas.h>

/****************************************************************************/
#define SysBase         dthc->dthc_SysBase
#define DOSBase         dthc->dthc_DOSBase
#define UtilityBase     dthc->dthc_UtilityBase
/****************************************************************************/

BOOL __asm DTHook (register __a0 struct DTHookContext * dthc)
{
    BOOL retval = FALSE;
    register ULONG i;
    UBYTE ch;

    /* Make sure we have a buffer */
    if (dthc->dthc_Buffer)
    {
        for (i = 0; (i < dthc->dthc_BufferLength) && !retval; i++)
        {
            ch = dthc->dthc_Buffer[i];

            /* Look at the data... */
        }
    }
    return retval;
}
```

The next example shows how to write a DataTypes Descriptor that looks into an IFF file for needed chunk information.

```
/* This example is to be compiled with no stack checking.  It must not be linked
 * with any startup code so that DTHook is the entry point for the executable.
 */
#include <exec/types.h>
#include <dos/dos.h>
#include <dos/dosextens.h>
#include <datatypes/datatypes.h>
#include <datatypes/pictureclass.h>
#include <libraries/iffparse.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/iffparse_protos.h>
#include <clib/utility_protos.h>

#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>1
#include <pragmas/iffparse_pragmas.h>
#include <pragmas/utility_pragmas.h>

/*****************************************************************/
#define SysBase          dthc->dthc_SysBase
#define DOSBase          dthc->dthc_DOSBase
#define IFFParseBase     dthc->dthc_IFFParseBase
#define UtilityBase      dthc->dthc_UtilityBase
/*****************************************************************/
#define BMH_SIZE (sizeof (struct BitMapHeader))
/*****************************************************************/

BOOL __asm DTHook (register __a0 struct DTHookContext * dthc)
{
    struct BitMapHeader bmh;
    struct ContextNode *cn;
    struct IFFHandle *iff;
    BOOL retval = FALSE;

    /* Make sure that this is an IFF data type */
    if (iff = dthc->dthc_IFF)

        /* Stop on the BitMapHeader (type, id) */
        if (StopChunk (iff, ID_ILBM, ID_BMHD) == 0)

            /* Scan through the IFF handle */
            if (ParseIFF (iff, IFFPARSE_SCAN) == 0L)

                /* Make sure we have a current chunk */
                if (cn = CurrentChunk (iff))

                    /* Make sure the current chunk is ILBM BMHD */
                    if ((cn->cn_Type == ID_ILBM) && (cn->cn_ID == ID_BMHD))

                        /* Read the chunk data */
                        if (ReadChunkBytes (iff, &bmh, BMH_SIZE) == BMH_SIZE)

                            /* See if the depth is set to 24 */
                            if (bmh.bmh_Depth == 24)
                                retval = TRUE;
    return (retval);
}
```

◆

Changes in the V39 OS

by Martin Taillefer

Many changes were made to the system software for V39. The major differences are in the graphics subsystem and in Intuition. There were also hundreds of changes made to other areas of the system software. This document presents the majority of these changes, and explains the possible impact of these new features on the developer.

V39 requires both a new disk set, and a new ROM. V38 is a disk-based upgrade requiring only a new disk set. For a complete list of changes in V38, refer to "Release 2.1 Overview" in the *V39 Release Notes*, available from CATS (part number: V3.0).

The main changes to the contents of the disks in V39 are detailed in Appendix A and in Appendix B. Here are the highlights of changes to system disk organization:

❑ The V38/V39 system software is shipped on FFS floppies. This is mainly to increase available storage.

❑ Under V38, a Storage drawer was created on the Extras disk. Under V39, a separate Storage disk is provided. Storage contains the same five drawers as in the Devs drawer. This is where things that are not currently used are kept. To activate a monitor, the user is expected to drag the desired monitor icon from Storage/Monitors to Devs/Monitors. Same applies for printer drivers, keymaps, DOS drivers, and datatypes

❑ The Fonts disk is now called FONTS instead of AmigaFonts. This enables the disk to be used directly whenever fonts are needed by the system, instead of requiring the user to assign FONTS: to the disk.

❑ Note that there are different specific disk sets shipped with different machines and packages. For example, low-end V38 systems do not get a Locale disk, and instead have a Locale directory on the Workbench disk.

❑ Deleted Files. The following system files present under V37 are no longer included with the system software.

Devs/narrator.device	This device is deleted as part of the removal of speech support.
Libs/translator.library	This library is deleted as part of the removal of speech support.

Tools/Colors	This tool has been removed in V39 because its functionality conflicted with pen sharing, and was generally not very nice to applications.
Utilities/Display	This program is replaced under V39 by the more flexible and powerful MultiView utility.
Utilities/More.info	This icon has been removed because we want to encourage users to use MultiView instead of More when running from Workbench. The More program itself remains on the disk since it is referenced by many read me files, and the like.

- ❑ Speech Support. The various components supporting synthesized speech are no longer included with the operating system. This can be a serious liability for applications depending on synthesized speech for correct operation. The V37 components supporting speech still function correctly under V38 and V39. The V38/V39 installation procedures do not remove the V37 files when updating a system.

Further details of how the system disks have changed can be found in Appendix A and B. V39 ROM changes are listed in Appendix C and D.

V38/V39 API Changes

This section discusses what changes in the programming model can affect developers.

Finding Version Information

An important point to mention is how to determine if a system is running V38 instead of V37. The recommended approach is to open version.library, and check its version. For example:

```
struct Library *VersionBase;
if (VersionBase = OpenLibrary("version.library",0))
{
        if (VersionBase->lib_Version >= 38)
        {           /* user is running at least V38 Workbench */       }
        else
        {           /* user is running at most V37 Workbench */       }
}
else
{      /* can't tell what the user is running, assume the minimum version
        * that your application supports        */
}
```

The above technique lets you determine which general version is in use for the disk-based software. *Never* assume this is a reflection of every other library in the system. For example, if you need features that are only present in V38 asl.library, you must explicitly check the version of asl.library before using it. The same is true for all other system libraries.

To determine the general version of the ROM, use SysBase->LibNode.lib_Version.

GadTools Library

Many enhancements were made to GadTools for V39. Below you will find short descriptions of some of the new features. A complete list of all the new features is provided in Appendix A, and more details can be had in gadtools.doc.

Menus. GadTools fully supports Intuition's NewLook menus. NewLook menus can be added to an application by providing the {WA_NewLookMenus, TRUE} tag to OpenWindowTags(), and providing the {GTMN_NewLookMenus, TRUE} tag to LayoutMenus(). These two tags will give your application NewLook menus. The pens used to render these menus are controllable by the user through the V39 Palette prefs editor.

You can now put an arbitrary command string in the right-hand side of a menu, where the Amiga-key equivalent normally goes. To do this, point the NewMenu.nm_CommKey field at the string (e.g., "Shift-Alt-F1), and set the new NM_COMMANDSTRING flag in NewMenu.nm_Flags.

GT_GetGadgetAttrs(). GadTools now has a GT_GetGadgetAttrsA() function which retrieves attributes of the specified gadget, according to the attributes chosen in the tag list. For each entry in the tag list, ti_Tag identifies the attribute, and ti_Data is a pointer to the long variable where you wish the result to be stored.
Here is an example:

```
LONG top = 0;
LONG selected = 0;
LONG result;
result = GT_GetGadgetAttrs(listview_gad, win, NULL,
                           GTLV_Top, &top,
                           GTLV_Selected, &selected,
                           TAG_DONE);
if (result != 2)
{
    printf( "Something's wrong!" );
}
```

Palettes. GadTools palette gadgets got a major overhaul for V39. There are a few new features, some reduction in memory usage, and an increase in performance.

Palette gadgets no longer display a box filled with the selected color. The selected color is instead denoted by a box drawn around the color square in the main palette area. This change slightly impacts the size of palette gadgets. Depending on the conditions, V39 PALETTE_KIND gadgets can be slightly smaller than the V37 ones.

Another change which can affect the size of the palette gadgets is the smarter layout of the color squares. An attempt is now made to keep the color squares as square as possible, based on the aspect ratio information obtained from the gfx database. As many color squares as possible are put on the screen, until things get too small in which case the upper colors are thrown away.

The GTPA_ColorTable tag was added in support of sparce color tables. It lets you define arbitrary pens to be used to fill-in the color squares of the palette gadget.

The GTPA_NumColors tag lets you define the exact number of colors to display in the palette gadget. Under V37, the GTPA_Depth tag played the same role. The problem with GTPA_Depth is that only multiples of 2 in number of colors can be specified. GTPA_NumColors allows any number of colors.

Listviews. GadTools listviews got a major overhaul for V39. There are a few new features, some reduction in memory usage, and an increase in performance.

The biggest difference that is readily noticeable is the disappearance of the display box at the bottom of the scrolling list area. Instead of showing the selected item in the display box, the selected item remains highlighted within the scrolling list. This was done to prepare listviews for multi-select support. Multi-selection requires a highlighting scheme as a display box would not work. The removal of the display box has a slight impact on the size of the listview. Listviews that had a display box under V37 got slightly smaller vertically under V39.

The other major addition to GadTools listviews is GTLV_CallBack. This tag allows a callback hook to be provided to gadtools for listview handling. Currently, the hook only gets called to render an individual item. Every time GadTools wishes to render an item in a listview, it invokes the call back function, which can do custom rendering. This allows GadTools listviews to be used to scroll complex items such as graphics and images.

Finally, listviews can now be disabled using {GA_Disabled, TRUE}.

The ASL Library

ASL got rewritten for V38. Its file requester is now extremely fast and offers more features to the user, and to the programmer. Its font requester now comes up about twice as fast the first time it is displayed, and instantaneously for subsequent uses. ASL also now includes a screen mode requester, which is very useful in the context of AA, and will be invaluable for AAA and RTG.

New Names. V38 *<libraries/asl.h>* contains new names for most of the structures, tags, and flag bits used by ASL. The names are now consistent, and more descriptive. We encourage the use of the new names as much as possible.

To ensure your code only uses the new names, just define the symbol ASL_V38_NAMES_ ONLY before including *<libraries/asl.h>*:

```
#define ASL_V38_NAMES_ONLY
#include <libraries/asl.h>
```

The symbol definition will prevent the old-style names from being defined. This is the same method used to disable the pre-V37 names in Intuition header files.

Don't Trust Me. The many filtering tags that you can supply to the ASL requesters are only advisory in nature. For example, even if you request that only non-proportional fonts be allowed in the font requester, the user can still type in the name of a proportional font. The user can also enter out-of-bounds values for any numeric parameter. You must check all results for sanity prior to using them.

Common ASL Tags. The tags for the different requesters are now distinct from one another instead of being all grouped together. All options that could be set via flag bits now have individual tags to control them. For example, there is now an ASLFR_DoSaveMode tag that does the same as the FRF_DOSAVEMODE flag bit.

Some basic tags are supported by all requester types:

ASLXX_Window. This indicates the parent window of the requester. You provide a window pointer, and if you don't specify an ASLXX_Screen tag, then the ASL requester will open on the same screen as this window.

ASLXX_PubScreenName. You provide this tag with the name of a public screen to open the ASL requester on.

ASLXX_Screen. You provide this tag with a pointer to a Screen to open on. If none of the above three tags are specified, then the ASL requester will open on the current default public screen. This will most likely by Workbench.

ASLXX_PrivateIDCMP. By default, ASL requesters use the same IDCMP port as their parent window (specified using ASLXX_Window). You can request that a private IDCMP port be used by setting this tag to TRUE. If you don't provide a parent window, then a private port is automatically used and this tag need not be provided.

ASLXX_IntuiMsgFunc. You provide this tag a pointer to a Hook structure, which indicates a routine to run whenever an IntuiMessage arrives at the ASL requester's IDCMP port that is not meant for the requester's window. This happens whenever the ASL requester is sharing the parent's window IDCMP port, and a message for the parent window arrives at the port. The function being called can process the message for the application.

ASLXX_SleepWindow. If you set this tag to TRUE, ASL will put the parent window (specified using ASLXX_Window) to sleep until the ASL requester is satisfied. This involves blocking all input in the parent window, and displaying a busy pointer in it.

ASLXX_TextAttr. You can provide a pointer to a standard TextAttr structure which defines the font to use in the ASL requester. If this tag is not supplied, the default behavior is to use the font of the screen on which the requester opens. Note that just like for Intuition objects, the font indicated by the TextAttr structure must already be in memory before using the requester. There is no guarantee that ASL will actually use the font you asked for. If that font is too large, ASL will use a smaller one. Also, as of V39, the file requester's file list requires a monospace font. So if you provide a proportional font, the file requester will use the system default font instead for its list of files.

ASLXX_Locale. You pass this tag a pointer to a Locale structure, as obtained from the locale.library/OpenLocale() function. The locale is used to determine in which language, and using which country information, the various requesters should be displayed. If this tag is not provided, or its value is NULL, then the default system Locale is used. This corresponds to what the user has currently picked in the Locale prefs editor. As of V39, certain items such as the dates in the file requester always use the system default Locale instead of the Locale provided with this tag.

ASLXX_TitleText. Provide this tag a string which will be used for the title of the requester. If this tag is not provided, the requester has no title.

ASLXX_PositiveText. Provide this tag a string which will be used for the label of the positive choice gadget in the requester. If this string is not provided, a localized default is used (English default is "OK"). We recommend using this tag to make the action of the various requesters more clear to the user. For example, when a file requester is being displayed to load a file, it is clearer if the gadget says "Open" instead of simply "OK".

ASLXX_NegativeText. Provide this tag a string which will be used for the label of the negative choice gadget in the requester. If this string is not provided, a localized default is used (English default is "Cancel").

ASLXX_InitialLeftEdge, ASLXX_InitialTopEdge, ASLXX_InitialWidth,
ASLXX_InitialHeight. These four tags let you specify the position and size of the ASL requester window. These are only requests that ASL may decide to ignore. When an ASL requester is closed, it is easy to determine the position and size of the requester when the user closed it. It is a good idea for an application to remember these values and use them in subsequent invocations of the requester.

The File Requester. The file requester supports a number of tags. Some of the tags introduced in V38 are discussed here.

Note that the future holds many changes in the ASL file requester that will substantially increase its flexibility to both the programmer and the user. If you use the ASL file requester in your applications today, you will automatically get the majority of these new features when they become available. Rolling your own file requester means a lot of extra work, and means that you will not automatically benefit from the new interface when it becomes available.

ASLFR_DoSaveMode. Setting this tag to TRUE indicates the file requester is being used for saving information. When in save mode, the file requester changes slightly. The most obvious change is the use of a different set of colors in the file list. This is a direct queue to the user that something is being selected for writing, not reading. Second change is that if the user enters a directory name which doesn't exist, ASL will prompt the user to see if the user wishes to create the directory. This lets the user create new directories to save files into. There are likely to be more differences between normal and save mode in the future. Please use this tag when appropriate.

ASLFR_RejectIcons. When set to TRUE, this tag prevents icons (.info files) from being displayed in the file requester. If this tag is not specified, icons will be displayed by the file requester. Please use this tag in all your software. Workbench users should never have to see .info files. The default behavior of the file requester is to display .info files, which is incorrect. Unfortunately, this default behavior cannot be changed due to compatibility.

ASLFR_DoPatterns. Setting this tag to TRUE causes a Pattern gadget to be displayed in the file requester, which allows the user to enter AmigaDOS patterns to filter out files. The default is to have no pattern gadget.

ASLFR_DrawersOnly. Setting this tag to TRUE causes the file requester to have no File gadget, and to only display directory names in its file list. This is a useful option if you wish to have the user select a destination directory for a particular task.

ASLFR_RejectPattern. Provide an AmigaDOS pattern to this tag, and any files matching this pattern will not be displayed in the file requester. This pattern can never be edited by the user. Note that the pattern you provide here must have already been parsed by dos.library/ParsePatternNoCase().

ASLFR_AcceptPattern. Provide an AmigaDOS pattern to this tag, and only files matching this pattern will be displayed in the file requester. This pattern can never be edited by the user. Note that the pattern you provide here must have already been parsed by dos.library/ParsePatternNoCase().

ASLFR_FilterDrawers. Setting this tag to TRUE causes the ASLFR_RejectPattern, ASLFR_AcceptPattern, and the Pattern text gadget to also apply to filter drawer names. Drawers are normally never affected by pattern filtering.

The Screen Mode Requester. The screen mode requester allows the user to select amongst the wide range of display modes available on the Amiga. Future chip sets, and the RTG effort will yield a very large additional number of display modes. Using the ASL requester is a good way to ensure that you benefit from the maximum upwards compatibility possible. In many cases, by using the screen mode requester, your applications will automatically be able to open displays in new modes that are introduced in future OS revisions.

An important point to note is that the interface presented to the user by the screen mode requester is likely to change dramatically in the future. This is because the increased number of available modes will make the current interface difficult to use and understand at best. A new scheme is being developed that should allow the user to more easily and accurately pick display modes. If you use the ASL screen mode requester in your applications instead of rolling your own version, your application will automatically get this new interface when it becomes available.

The ultimate goal of the screen mode requester is to return a graphics mode id. The mode ids are used since V37 to distinguish between the different display modes in the system. The screen mode requester can also return additional information including requested display sizes and depth.

As all other ASL requesters, you get the result of a request by reading the contents of the requester structure after the AslRequest() calls return success. There are two fields worth mentioning in the ScreenModeRequester structure: sm_BitMapWidth and sm_BitMapHeight. These values define the values to pass to AllocRaster() and InitBitMap() when hand-building a BitMap structure to display the requested width, height, and depth. This is useful when running on a V37 ROM. For V39 use, simply use the values in sm_Width and sm_Height, and pass those to AllocBitMap() directly.

Here are the most important screen mode requester tags.

ASLSM_InitialDisplayID. This tag sets the initial mode that is selected in the mode listview. You provide it a graphics mode id, and that mode will be initially selected in the file requester.

ASLSM_InitialDisplayWidth, ASLSM_InitialDisplayHeight, ASLSM_InitialDisplayDepth. These tags determine the initial setting of three gadgets that can optionally be displayed in the requester. The default if these tags are not provided are a width of 640, a height of 200, and a depth of 2.

ASLSM_InitialOverscanType. This tag determines the initial setting of the Overscan Type cycle gadget. This optional gadget lets the user pick which overscan setting to use for the given mode. The values you can provide to this tag are: OSCAN_TEXT, OSCAN_STANDARD, OSCAN_MAXIMUM, and OSCAN_VIDEO. OSCAN_VIDEO is only available starting with V39, it is not in V38. The four OSCAN_XX constants are defined in *<intuition/screens.h>*.

ASLSM_InitialAutoScroll. This tag can be set to TRUE or FALSE and determines what the initial state of the option AutoScroll checkbox gadget should be.

ASLSM_DoWidth, ASLSM_DoHeight, ASLSM_DoDepth, ASLSM_DoOverscanType, ASLSM_DoAutoScroll. These tags control which of the optional gadgets is displayed in the screen mode requester. The default is to have none of these present. Setting any of these tags to TRUE will make the gadget appear.

ASLSM_MinWidth, ASLSM_MaxWidth, ASLSM_MinHeight, ASLSM_MaxHeight, ASLSM_MinDepth, ASLSM_MaxDepth. These tags let you specify limits to the values the user is allowed to enter in the requester.

ASLSM_PropertyFlags, ASLSM_PropertyMask. These two tags cooperate to allow a flexible means of filtering out unwanted display modes. Each display mode in the graphics database has a given set of properties associated with it. These tags allow you to determine which modes to display in the mode list of the requester based on the properties of the modes.

The mode properties are defined as a ULONG of flag bits. The definitions for all supported properties are in *<graphics/displayinfo.h>*. Examples include DIPF_IS_WB and DIPF_IS_LACE.

ASLSM_PropertyMask defines which properties you are concerned with. Those are the only bits for which the specific setting matters to you. The setting of all other property bits doesn't matter to you.

ASLSM_PropertyFlags defines exactly in which state the property flags you are have shown interest in via the ASLSM_PropertyMask tag should be.

The way ASLSM_PropertyMask and ASLSM_PropertyFlags interact is indentical to how the two flags parameters interact in exec.library/SetSignal(). This is how ASL uses the tag values internally:

```
if ((displayInfo.PropertyFlags & propertyMask) ==
    (propertyFlags & propertyMask))
{
    /* Mode accepted */
}
else
{
    /* Mode rejected */
}
```

An example can help understand the relationship between the two tags. If you want to display only screen modes that can be used for the Workbench screen, and that are not interlaced, you would set ASLSM_PropertyMask to (DIPF_IS_WB| DIPF_IS_LACE), and ASLSM_PropertyFlags to (DIPF_IS_WB). The mask value means you are interested in the state of the DIPF_IS_WB and DIPF_IS_LACE bits. The flags value says you want the DIPF_IS_WB to be set, and the DIPF_IS_LACE bit to be clear.

The Locale Library

The locale.library provides the core of system localization. It is explained in detail in the *V39 Release Notes* available from Commodore's CATS department (part number: V3.0). Also refer to locale.doc for more information.

The Bullet Library

The bullet.library contains the Compugraphic outline font rendering engine. It is explained in the *V39 Release Notes* available from Commodore's CATS department (part number: V3.0). Also refer to bullet.doc for more information.

Color Wheel and Gradient Slider

The color wheel and gradient slider are two new BOOPSI classes introduced in V39. Together, they offer a very nice interface to let the user select or adjust colors. The V39 Palette prefs uses both classes in its interface.

Using either of these classes requires you to first open them as libraries, and then create new BOOPSI objects using the NewObject() function:

```
if (ColorWheelBase = OpenLibrary("gadgets/colorwheel.gadget",0))
{
    cw = NewObject(...);
}
```

For an introduction to programming the color wheel and gradient slider refer to the *Amiga Mail* newsletter, Sept/Oct 1992 issue, page IV-91 also published in *V39 Release Notes* available from Commodore's CATS department (part number: V3.0).

The colorwheel.gadget. The color wheel class provides the ability to create gadgets enabling the user to control the hue and saturation components of an HSB (Hue-Saturation-Brightness) color space. The companion gradient slider class enables control of the brightness component of the color space.

The color wheel can operate on screens of any depth, and adapts its rendering to the number of colors available. The system's pen sharing system is used in order to maximize the number of colors used by the wheel. A color wheel gadget is (normally) responsible for choosing it's own color pens to draw in (using graphics.library/ObtainBestPen()). However, the creator of the gadget can "donate" some of its pens to the gadget, using the WHEEL_Donation tag.

The reason that the color wheel picks its own colors is because it has the ability to display several different layouts depending on the number and variety of colors available. For example, when opening on a screen of low depth or when opening on a screen where all the pens have already been allocated exclusively, the gadget will display a "monochrome" version of the color wheel, where instead of colored segments, the letters "R" (for red), "G" (for green), "B" (for blue), "Y" (for yellow), "C" (for cyan) and "M" (for magenta) will be used as labels.

You can talk to the color wheel using HSB or RGB, even though the color wheel only really deals with HSB in its user-interface. All communications with applications are performed with full 32-bit color component values. Internally everything is currently kept and processed in 16-bit space, although this might change in the future.

Here are the main tags supported by the color wheel class:

WHEEL_Hue. Lets you control the Hue component of the color wheel. This is effectively the angle around the wheel where the desired color lies. A hue value of 0 is all red, and nothing but red. Increasing the value moves the color towards all green at $55555555, full blue at $AAAAAAAA, and back to red at $FFFFFFFF.

WHEEL_Saturation. Lets you control the Saturation component of the color wheel. This is effectively the distance from the center of the wheel where the desired color lies. A saturation value of 0 puts the knob at the center of the wheel and always yields white. Increasing the value progressively moves the knob farther away from the center. until the value $FFFFFFFF is reached in which case the knob is as far as it can go.

WHEEL_Brightness. Lets you control the Brightness component of the color wheel. The color wheel does not itself have any means of displaying or editing brightness, but it does maintain this value internally. Used with the WHEEL_GradientSlider tag, you can control the value of a gradientslider object by passing WHEEL_Brightness to a color wheel. A brightness value of 0 means all black. Increasing the value progressively brightens the current color, until the value $FFFFFFFF is reached in which case the color is as bright as it gets.

WHEEL_Red, WHEEL_Green, WHEEL_Blue. Let you specify new RGB values for the color wheel. The values provided are converted into HSB values are then used.

WHEEL_Screen. This is a required tag that lets you specify the screen on which the color wheel is to be displayed. Note that once a color wheel is created, the screen should not be closed until the color wheel object is discarded using DisposeObject().

WHEEL_BevelBox. If you set this tag to TRUE, a bevel box will be drawn around the color wheel object.

WHEEL_GradientSlider. This tag lets you attach a gradient slider object to the color wheel. You give this tag a pointer to a gradient slider object obtained previously from NewObject(). Once this is done, anytime the various tags that can affect the brightness component of the current color is sent to the color wheel, the color wheel automatically changes the value of the attached gradient slider to match that new brightness value. Reading the brightness value from the color wheel returns the current value indicated by the gradient slider.

Using this tag effectively allows you to treat the color wheel and gradient sliders as a single gadget. Once things are set up, all communications occur through the wheel object, and the gradient slider can pretty much be ignored.

The gradientslider.gadget. The gradient slider gadget class is a type of non-proportional slider. The primary feature of the gradient slider is it's appearance. Unlike normal sliders, a gradient slider can display a "spread of colors" or "color gradient" in the slider container box. The "knob" or "thumb" of the slider appears to slide on top of this color gradient.

The color gradient effect is built-up using a combination of multiple pens and half-tone dithering. The application must tell the slider exactly which pens to use in creating the gradient effect, and in what order to use them. Essentially, it does this by passing an array of pens (terminated by ~0, just like a PenSpec) to the slider. The first pen in the array is used as the color at the top of the slider (or left, if it is horizontal), and the last color in the array is used at the bottom (or right). The other pens will be used at evenly spaced intervals in between. Dithering is used to smoothly fade between the pens, allowing the illusion of a continuous change in color.

Here are the main tags supported by the gradient slider class:

GRAD_MaxVal. Lets you set the maximum value supported by the slider. This must be in the range $0..$FFFF.

GRAD_CurVal. Lets you set the current value of the slider. This should be in the range 0..GRAD_MaxVal.

GRAD_PenArray. Lets you specify an array of pens that the slider should use to create its gradient background. The array can contain any number of pens, and is terminated with a pen value of ~0. These pens can be allocated as shared, since their RGB value is not altered by the slider. The first pen is used on the top or left of the slider, and the last pen is used on the bottom or right. All other pens are evenly spaced out and used in between. Dithering is used between the pens to enhance the smoothness of the gradient transition.

A NULL pen array causes the background of the slider to be rendered in the screen's background color. A pen array containing only a single pen causes the background to be rendered using that pen.

New File Systems

The V39 ROM file system supports 6 different disk formats:

❑ DOS\0	This is the traditional Amiga file system. It has 488 bytes of data per block. It offers a high level of redundancy and validation which makes it very reliable. It is also fairly slow.
❑ DOS\1	This is the original Fast File System introduced in 1.3. It uses a disk layout quite similar to DOS\0, except it forgoes some redundancy in the name of speed. It stores 512 bytes per block, and runs substantially faster than DOS\0.
❑ DOS\2	This is an international flavor of DOS\0. The only difference between this format and DOS\0 is the hashing algorithm used to locate the files and directories on the disk. DOS\0 has a bug in dealing with making international characters case-sensitive. DOS\2 corrects this bug, and is otherwise identical to DOS\0.
❑ DOS\3	This is an international flavor of DOS\1. The only difference between this format and DOS\1 is the hashing algorithm used to locate the files and directories on the disk. DOS\1 has a bug in dealing with making international characters case-sensitive. DOS\3 corrects this bug, and is otherwise identical to DOS\1.
❑ DOS\4	This is a directory-caching version of DOS\2. It has the same general disk layout, with the addition of special directory cache blocks. These cache blocks store the contents of each directory. The advantage of this is that getting a directory listing can be an order of magnitude faster than on normal DOS\2 disks. The directory caches require a slight amount of disk space, and maintaining them slightly slows down file creation, deletion, and update. This file system is mostly meant for floppies, as it doesn't generate a very noticeable performance increase on hard drives.
❑ DOS\5	This is a directory-caching version of DOS\3. It has the same general disk layout, with the addition of special directory cache blocks. These cache blocks store the contents of each directory. The advantage of this is that getting a directory listing can be an order of magnitude faster than on normal DOS\3 disks. The directory caches require a slight amount of disk space, and maintaining them slightly slows down file creation, deletion, and update. This file system is mostly meant for floppies, as it doesn't generate a very noticeable performance increase on hard drives.

Disk Block Format. Here are some notes concerning changes to the disk block format required by the new SetOwner() call, and more importantly the changes present in DOS\4 and DOS\5 to support directory caching.

The format of root blocks is unchanged under V39.

The format of file header blocks is slightly different in order to store owner information. One of the reserved fields is now being used:

```
struct FileHeaderBlock
{
    ULONG  fhb_Type;          // T.SHORT
    ULONG  fhb_OwnKey;          // key of this block
    ULONG  fhb_HighSeq;       // total blocks in file
    ULONG  fhb_DataSize;      // number of data block slots used
    ULONG  fhb_FirstBlock;    // first block in this file
    ULONG  fhb_Checksum;      // checksum of this block

    ULONG  fhb_HashTable[];   // variable number of hash table entries

    ULONG  fhb_Reserved       // always 0
    ULONG  fhb_OwnerXID       // owner UID/GID
    ULONG  fhb_Protect        // protection bits
    LONG   fhb_ByteSize       // total size of file in bytes
    char   fhb_Comment[92];   // comment as a BCPL string
    ULONG  fhb_Days;          // creation date and time
    ULONG  fhb_Mins;
    ULOGN  fhb_Ticks;
    char   fhb_FileName[36];  // filename as BCPL string
    ULONG  fhb_Link;          // pointer to object header is linked to
    ULONG  fhb_BackLink;      // pointer to previous object in link chain
    ULONG  fhb_HashChain;     // next entry with same hash value
    ULONG  fhb_Parent;        // pointer back to parent directory
    ULONG  fhb_Extension;     // 0 or pointer to first extension block
    ULONG  fhb_SecType;       // ST.FILE
);
```

User directory blocks were modified to add the same owner information as for file header blocks. Under DOS\4 and DOS\5, the user directory blocks also point to the new directory cache blocks.

```
struct UserDirectoryBlock
{
    ULONG  udb_Type;          // T.SHORT
    ULONG  udb_OwnKey;        // key of this block
    ULONG  udb_SeqNum;        // highest seq (always 0)
    ULONG  udb_DataSize;      // always 0
    ULONG  udb_NextBlock;     // always 0
    ULONG  udb_Checksum;      // checksum of this block

    ULONG  udb_HashTable[];   // variable number of hash table entries

    ULONG  udb_Reserved;      // always 0
    ULONG  udb_OwnerXID;      // owner UID/GID
    ULONG  udb_Protect;       // protection bits
    ULONG  udb_Junk;          // not used (always 0)
    char   udb_Comment[92];   // comment as a BCPL string
    ULONG  udb_Days;          // creation date and time
```

entries vary in size based on the length of the file name and comment for the entry. A cache entry looks like:

```
struct ListEntry
{
    ULONG fle_Key;        // Key (fhb block) of file
    ULONG fle_Size;       // size in bytes
    ULONG fle_Protection; // protection bits
    ULONG fle_OwnerXID;   // owner info
    UWORD fle_Days;       // date (allows 179 years)
    UWORD fle_Min;        // 0..(60*24 - 1)
    UWORD fle_Tick;       // 0..3599
    UBYTE fle_Type;       // ST.XXXXX (all fit in a byte)
    char  fle_FileName[]; // variable length BCPL string
    char  fle_Comment[];  // variable length BCPL string
};
```

These are the values for the various constants referred to above:

```
// block type constants
#define T.DELETED 1
#define T.SHORT   2
#define T.LONG    4
#define T.DATA    8
#define T.LIST    16

DCFS_REV      1
DIRLIST_KEY   32
```

[redacted]

ReadArgs(). Commodities now use ReadArgs() instead of the custom command-line parsing routines in amiga.lib. This makes them more consistent with the rest of the system. We encourage third party commodity writers to also switch to ReadArgs().

Keyword Synonyms. ParseIX() now understands many synonyms for keys and qualifiers. The intent was to provide more consistent naming, and to decrease the likelihood the user would make a mistake while entering a key sequence. For a complete list of all the keywords and synonyms refer to the *V39 Release Notes* available from Commodore's CATS department (part number: V3.0)

Input Events. One bug found in a few applications during V39 development was incorrect initialization of InputEvents inserted into the input.device's input stream. These events can work most of the time, and suddenly not work with specific applications. Of specific interest, the inputEvent->ie_SubClass was found to often contain garbage. This field should generally be set to 0. Another field that is often incorrectly initialized is the ie_TimeStamp field. Make sure this field has something meaningful in it. Note that using the IND_WRITEEVENT input.device command automatically initializes the ie_TimeStamp field.

Bugs In commodities.library. Under V37, input events output by translator objects were inserted into the input stream after the commodities.library input handler. This was contrary to the documentation. Starting with V38, events generated are inserted immediately following the translator object. This means commodity objects coming after the translator now get to see the translated events, while in V37, they wouldn't.

Another bug with translator objects is that they insert their attached list of input events in reverse order. This cannot be fixed due to compatibility problems. It is easily worked around by arranging the input events in the reverse order of how you want them to be inserted. The InvertString() function in amiga.lib generates an inverted list of input events, which is just fine for translator objects.

Starting with V39, input events generated by translator objects are stamped with the same time as the input event that activated the translator. This helps applications that look at the time of input events.

In V38, the ParseIX() function has a bug which prevents the following keys on the numeric keypad from being used in commodities: . 9 ( ) / * +. This bug is fixed in V39.

Until V39, input events of type IECLASS_NEWPOINTERPOS never got their extended tag list data copied when it needed to be. This can cause general confusion or crashes if a sender object is used on an event of that type. The only solution is to refrain from sending messages of type IECLASS_NEWPOINTER.

Until V39, the InvertKeyMap() function never initialized the ie_SubClass and ie_TimeStamp fields of the input event it created.

The workaround to this bug is simple:

```
inputEvent->ie_SubClass          = 0;
inputEvent->ie_TimeStamp.tv_secs  = 0;
inputEvent->ie_TimeStamp.tv_micro = 0;
if (InvertKeyMap(ansiCode,inputEvent,NULL))     ...
```

Mount Lists

The Mount procedure got a major overhaul in V38. The major change from the user's perspective is that the handlers that get mounted in the system can now be controlled exclusively through Workbench by dragging icons around. This is especially important in light of CrossDOS.

New File Format. Although the traditional DEVS:MountList file is still fully supported, a new method of storing mount information is now recommended. The new format involves simply splitting up the various entries that used to be in a mountlist, into separate files. Each file controls a unique handler. The format for the information in the mount files is the same as what is used in a traditional MountList with three exceptions. For example:

```
/* Aux-Handler mount entry */
Handler   = L:Aux-Handler
Stacksize = 1000
Priority  = 5
```

The differences with old style MountLists are as follows:

- ❑ Only a single device can be defined per file.
- ❑ The name of the device is not specified in the file and is instead the same as the name of the file.
- ❑ The # at the end of the entry can now be omitted.

Every parameter that can be specified in a mount file can also be specified in the tooltypes for the icon of the mount file. The parameters in the toolteyps override any equivalent parameter setting present in the mount file. The format for the tooltypes is straightforward. For example:

```
LOWCYL=0
HIGHCYL=79
```

This lets the Workbench user easily control certain attributes associated with any given mount entry. For example, the WB user can easily set the size of RAD from tooltypes.

If the user wishes to only use your handler once in awhile, he can drag the icon for the handler in the Storage drawer. This prevents the handler from being mounted on every boot. If the user wants to use the handler for a given session, he can then simply double-click on the mount entry icon to get it mounted.

It is fairly simple to create an installation procedure that deals with both old style and new style mount files:

```
if (V38)
{
    /* Create a mount entry file.  If the user indicates he wants to use the
     * handler permanently, put it into DEVS:DOSDrivers.  Otherwise, just put
     * it into SYS:Storage/DOSDrivers.  Don't forget to provide an icon for
     * the mount entry.
     *
     * If the mount entry is in DEVS:DOSDrivers, it will get mounted
     * automatically when the system is rebooted.  The user can also mount
     * it for the current session by double-clicking on it.
     */
}
else /* V37 or before */
{
    /* Create an old-style mount entry and append it to DEVS:MountList.
     * Add a Mount statement in the user-startup of the system.
     */
}
```

File System Resource List. Mount now has the ability to fish out file systems from the FileSystem.resource list. When mounting a file system, Mount first look in the FileSystem.resource to see if the system already contains a file system of the correct DOS type. If there is such a file system, its seglist is used for the new handler.

For example, the CrossDOS file system registers itself in the FileSystem.resource the first time it is used. Any subsequent attempts to mount a handler having the same DOS type as the CrossDOS file system results in that handler using the same seglist as the original CrossDOS handler.

If a file system seglist is reentrant, it should be added to the FileSystem.resource list of file systems, which will enable it to be used in the manner described above. Be sure the seglist is totally reentrant before doing this (no global variables). Consult the Rom Kernal manual to learn how to add a file system to the FileSystem.resource list.

The new FORCELOAD keyword can be specified in mount entries. When its value is set to 1, it tells mount to always load the file system for this handler from disk, even if it is present in the FileSystem.resource list. This can be quite handy during the testing phase of a handler.

Unit Keyword. Stream handlers have always had the advantage of being able to let the user provide startup arguments to the handler via the Startup keyword. For example:

```
Startup = "this is a bunch of options"
```

The value of the parameter is available to the handler as a BSTR in the dol_Startup field of the handler DosList structure. This only works for stream handlers though, file system handlers make use of the dol_Startup field as a BPTR to the FileSysStartupMsg created for them by the Mount command.

The Unit keyword can now be used to passed textual options to a file system handler. If a string is specified in the mount entry, then the fssm_Unit field of the file system's FileSysStartupMsg will be a C pointer to the NULL-terminated argument string, instead of being the traditional unit number.

New Values For GlobVec. The GlobVec keyword in a mount list can now be set to -2 and -3 in addition to the other values already supported. Here is a table of possible values:

GlobVec Value	**Meaning**
0	Default global vector, used for handlers written in BCPL. Startup packet comes in a register.
-1	No global vector, used for C and ASM handlers. Process has to wait for startup packet.
-2	Same as -3, except for handlers written in BCPL.
-3	No global vector, used for C and ASM handlers. The difference with -1 is that the handler can do DOS IO prior to returning its startup packet. On the flip side, the handler is responsible for making sure there is only one instance of itself loaded by providingcorrect arbitration around the poking of the dol_Taskfield of the DosList structure. With a GlobVec of -1, this is taken care of by DOS.

Even though the values -2 and -3 are only allowed starting with the V39 Mount command, the V37 dos.library handles those values correctly. It's just that prior to V37, the Mount command would refuse to process these values.

Version Command

There has been some confusion about what the format of version strings are. The exact format is detailled in the *Amiga User Interface Style Guide* on page 110. The format is:

```
$VER: <name> <version>.<revision> (dd.mm.yy)
```

The string starts with $VER: followed by a space, followed by <name>. <name> is the name of the program. <name> can NOT contain any spaces. You can use underscores to achieve a similar effect.

After <name> comes a single space, followed by <version>. <version> is the major version number for the program. There should be no leading zeros.

After <version> comes a single space, followed by <revision>. <revision> is the minor version number for the program. There should be no leading zeros.

Following the revision number comes a space, and then the date. The date is specified in numeric form only, with no leading zeros on any of the components. First comes the day of the month, then the month, then the two digits year. In the future, a four digit year format will also be supported, but not yet.

Embedding a version string in the exact format described above will let the C/Version command find the version string. In the future, there will be a system call to enable applications, and other system tools, to obtain the version information from files easily.

ROM Tags. The version command not only looks for the $VER: version strings, it also search executable files for standard ROM tags. ROM tags only contain a version number, they have no revision or date fields. The recommended approach is to store a pointer to a version string in the RT_IDSTRING field of the ROM tag. This string should be in the same format as the normal version string, except without the "$VER: " prefix.

An executable file containing a ROM tag with a properly initialized RT_IDSTRING field does not need any other version information in it, such as a separate $VER-style string.

Leading Zeros. Leading zeros are not supported as part of the version and revision numbers. So "2.04" is not a valid version/revision pair as far as the version command is concerned. The reason this is so is because of the VERSION and REVISION command-line options of the Version command. These allow the version and revision of a file to be checked by a script such as:

```
Version asl.library VERSION 38 REVISION 4
```

The above command will return 0 if asl.library is greater or equal to 38.4, and 5 if it is not. Specifying:

```
Version asl.library VERSION 38 REVISION 04
```

Does the same thing. This means that from the standpoint of the Version command, 2.04 is the same as 2.4, and thus 2.04 evaluates as being greater than 2.1, which is not the desired effect. So, the version and revision numbers must not be treated as a floating-point number, but as two separate and distinct integers.

This brings up the concept of user versions and internal versions. V37 is known to the public as Release 2.04, and V39 is known as 3.0. All the version strings in the system software use version strings starting with V39. The recommended approach is as follows:

- ❑ Assign your products user-space numbers of the form 2.04 and the like.
- ❑ Assign the various components of your distribution the same version numbers as for your product. For example, "2".
- ❑ Assign each individual components of your distribution their own unique, monotonically increasing revision numbers. These numbers have nothing to do with the user-space number of your product as a whole.

An example is in order. Assume a word processor called "SliceAndDiceWord", which is at version 4.03. All the files composing the distribution would have a version number of "4", and a file-specific revision number:

```
Package number : SliceAndDiceWord 4.03
Main executable: SADW 4.1423
Support library: SADW.library 4.4129
README file    : SADW.README 4.6
```

When SliceAndDiceWord 5.0 comes out, all the version numbers of the files included with the distribution get bumped to 5, and the revision numbers start at 0 again.

Miscellaneous

This section presents miscellaneous tidbits of information worth noting when developing applications.

Overscan Names. Starting with V38, the two user-settable overscan regions are known as "Text Size" and "Graphics Size". Please use the following names in your user documentation:

```
OSCAN_TEXT      --> "Text Size"
OSCAN_STANDARD  --> "Graphics Size"
OSCAN_MAXIMUM   --> "Extreme Size"
OSCAN_VIDEO     --> "Maximum Size"
```

Audio Beep. Starting with V38, the system provides standard support for audio beeps, and complete sampled sounds, as a replacement for DisplayBeep(). You may wish to rely exclusively on this feature instead of providing your own application-specific control over an equivalent feature.

ToolType Comments. We strongly encourage you to follow the convention adopted in V38 with respect to specifying all tooltypes supported by an application within the icon. Any tooltypes not in use can be commented out by putting a set of ( ) around the tooltype. This causes Workbench to ignore the tooltype, yet still leaves it there for the user to see and uncomment.

Keymap selection. If you wish to offer application-specific control over the keymap being used, you have to load the keymaps you wish to use from disk yourself. Some applications were using the SetMap command to load keymaps into memory for them. SetMap is no longer part of the system, so this technique will no longer work. The DevCon disks contain an example showing how to load keymaps from disk.

Note that if you wish to offer a list of keymaps to the user, you should be looking in the KEYMAPS: assign list for the available keymaps. To work under V37, If this assign list doesn't exist, you can look in DEVS:Keymaps. The example code shows how this can be done.

Appendix A: V39 Disk Changes

Following is a description of most of the important changes made to the disk-based system software between V38 and the initial release of V39. This doesn't cover changes made to modules discussed in other talks, such as the addition of datatypes and AmigaGuide.

C/Install

- o Doing "INSTALL DF0: CHECK" on a disk formatted with dir cache now reports the disk as either "DC-OFS" or "DC-FFS" as appropriate.

C/IPrefs

- o Requires Kickstart V39 and handles the new palette, pointer, and wbpattern preferences.
- o Is smarter about when it needs to reset the WB screen. Now sends an initial IP_SCREENMODE message to Intuition to force the initial Workbench screen to be interleaved even if there is no screenmode.prefs file.
- o Now uses utility.library 32-bit math routines and uses datatypes.library to load sound samples instead of custom 8SVX code.
- o Causes much less screen flashing when changing fonts. No longer has the QUIT/S command-line option. (Removing this option allows a few K of RAM to be saved in every machine. Gives better error messages when it can't find a font or a picture.

C/Mount

- o Added support for GlobVec of -2 and -3.
- o No longer considers empty strings an error. This allows something like:

```
Device = ""
```

to work. This fix will be helpful to the Envoy software. Note that this bug is in V38, so there will soon be many many Mount commands out there that don't support empty strings.

C/RequestChoice

This is a new C: program that lets AmigaDOS and ARexx scripts use the Intuition EasyRequest() feature. The template is:

```
TITLE/A,BODY/A,GADGETS/M,PUBSCREEN/K
```

TITLE
Specifies the title of the requester.

BODY
Specifies the text of the requester. Linefeeds can be embedded using *n.

GADGETS
Specifies the labels for the different gadgets at the bottom of the requester.

PUBSCREEN
Lets you specify the name of a public screen to open the requester on.

The number of the selected gadget is printed at the console on exit, and is returned as the secondary return value (seen as the Result2 variable in the Shell).

C/RequestFile

This is a new C: program that lets AmigaDOS and ARexx scripts use the ASL file requester. The template is:

```
DRAWER,FILE/K,PATTERN/K,TITLE/K,POSITIVE/K,NEGATIVE/K,ACCEPTPATTERN/K,
REJECTPATTERN/K,SAVEMODE/S,MULTISELECT/S,DRAWERSONLY/S,NOICONS/S,
PUBSCREEN/K
```

DRAWER
Specifies the initial contents of the Drawer gadget.

FILE
Specifies the initial contents of the File gadget.

PATTERN
Standard AmigaDOS pattern. Specifies the initial contents of the Pattern gadget. Not providing this option causes the file requester not to have any Pattern gadget.

TITLE
Specifies the title of the file requester.

POSITIVE
Specifies the text to appear in the positive (left) choice in the file requester.

NEGATIVE
Specifies the text to appear in the negative (right) choice in the file requester.

ACCEPTPATTERN
Specifies a standard AmigaDOS pattern. Only files matching this pattern will be displayed in the file requester.

REJECTPATTERN
Specifies a standard AmigaDOS pattern. Files matching this pattern will not be displayed in the file requester.

SAVEMODE
Specifying this option causes the requester to operate in save mode.

MULTISELECT
Specifying this option causes the requester to allow multiple files to be selected at once.

DRAWERSONLY
Specifying this option causes the requester to not have a File gadget. This effectively turns the file requester into a directory requester.

NOICONS
Specifying this option causes the requester never to display icons (.info) files.

PUBSCREEN
Lets you provide the name of a public screen to open the file requester on.

The selected files are printed on the console at exit, enclosed in double quotes and separated with spaces. The command generates a return code of 0 if the user selected a file, or 5 if the user cancelled the requester.

C/SetFont

- o Now sends ESC-C to the console handler only after doing SetFont() in the console window's rastport. The only reason SetFont ever worked was by luck, since the escape sequences were buffered by DOS and flushed only after the program exited.
- o No longer crashes when passed a font name with more than 80 characters. Now returns with ERROR_BAD_NUMBER if asked for a font < 4 points. No longer opens console.device.

C/SetPatch

o Activates AA chips.

o On some systems, NMI interrupts could be caused due td hardware bugs. These NMIs can cause software/hardware to perform incorrectly and has caused major performance problems in some systems. In the V37 ROM, the NMI vector pointed at a generalized routine that saved all registers, does some work which ends up doing nothing, restores all registers, and exits. This patch will, under V37 ROMs, move that vector to point directly at an RTE.

o SetPatch no longer will read the longword after the end of each of its hunks.

o Now checks for the CPU type before trying to open the 68040.library This is so that the library does not even get loaded if you don't need it. (The 68040.library also does the check.)

o Does not turn on the caches if the intruction cache is not already on. This is such that the bootmenu option will not be "undone" by SetPatch.

C/Version

o Now does versions of gadget and datatypes classes.

o Significant change in behavior required when not using the standard version string format:

Junk following the date parameter is printed whenever the FULL option is provided on the command line. Incorrectly formatted components of a version string are printed whenever the FULL option is provided.

Note that as always, specifying incorrectly formatted version strings will cause the VERSION and REVISION command line options of the Version command not to work as expected. Basically, using these keywords, will result in output like "2.1 < 2.04", which is not what is usually expected.

The incorrect version string formats are not supported by the Installer. It will behave much like the VERSION and REVISION cmd-line options of the Version command.

Classes/Gadgets/colorwheel.gadget

The colorwheel.gadget is a new BOOPSI class to create color wheels to allow color selections by the user. See colorwheel.doc for more info.

Classes/Gadgets/gradientslider.gadget

The gradientslider.gadget is a new BOOPSI class to create sliders having a gradient background. This is to be used in combination with the colorwheel.gadget. See gradientslider.doc for more info.

Devs/DataTypes

This new directory in V39 is the place where a user puts datatypes.library descriptor files.

Devs/printer.device

o Added small amount of code to support AA 8-bit HAM mode printing.

o Also now uses GetBitMapAttr() if running under V39 to obtain dimensions of source BitMap structure.

L/FastFileSystem

o Equivalent to the V39 ROM file system, including DOS\5 support. See ROM release notes for more info.

Libs/68040.library

This is a support library for 68040 systems. It addresses a number of issues with that CPU. There are no publically callable functions in this library.

Libs/amigaguide.library

The amigaguide.library is a new system module providing a means whereby developers can easily add on-line help to their applications. See amigaguide.doc for more info.

The amigaguide.library was previously available for use in commercial applications. This is the first release of the library as a part of the core operating system. Changes since the previous release include:

o All new V39-only version uses datatypes.library for object handling.

o Now has an AppWindow, plus an "Open..." menu item. To open a new document, AmigaGuide performs a LINK to it, so that you can RETRACE back through the documents that you have opened.

o Now supports word wrapping and proportional fonts.

o Now supports font attributes.

o Now supports embedded objects. This means that you can LINK to any type of object that the user has a DataTypes descriptor and class for---such as pictures and animations.

o Modified how files are located, to search for a localized help file. Now searches in the following order. For each preferred language

```
PROGDIR:Help/<language>/<name>
HELP:<language>/<name>

  PROGDIR:<name>
  <name>
```

This matches how locale.library searches for catalog files. For compatibility, each directory from the AmigaGuide/Path environment variable are also searched.

o Renamed S:help.guide to HELP:<language>/Sys/amigaguide.guide

o All sub-databases are now opened localized.

Libs/asl.library

o Added support for the new WA_HelpGroupWindow Intuition tag. ASL requesters now inherit the group id of their parent window.

o Removed V37 code. The library now requires V39.

o Added support for OSCAN_VIDEO in the screen mode requester.

o Fixed rendering bug with selected items in the file requester. Depending on the size of the requester, sometimes the rendering didn't occur correctly. Fixed font requester layout bug with checkmark labels using certain fonts.

o Now monitors the Size numeric gadget more closely in the font requester.

o ASL now uses scaled checkmark imagery in the font and screenmode requesters. Also now uses ROM busy pointer instead of custom one.

Libs/commodities.library

o Fixed ParseIX() bug introduced in V38 preventing the following keys on the numeric keypad from being used in commodities: 9 ( ) / * + .

Libs/datatypes.library

datatypes.library is a new system module providing transparent data handling abilities to applications. Application developers can register their data format with datatypes.library and provide a class library for handling their data within other applications. See datatypes.doc for more information.

Libs/diskfont.library

o Marks DUPLICATE fonts using one of many free bits in the TextFont Extension. These are fonts opened by diskfont, but match some other font already on the public font list by name, Y size, relevant style, and relevant flags. The font is only marked as a duplicate if a NON-DUPLICATE occurence of this font already exists.

  DUPLICATE fonts are ignored by OpenFont() if the caller is not tag aware even if there is a font on the list which has a better implied DPI.

o Tightened up requirements for source font when scaling. Specifically, an already scaled RAM font of an exact Y size match but differing by DPI will not be used as the source for a bitmap scaled font (adding scale error, to scale error).

o The code which generates the outline fonts now does comparison for OT_DotSize when it calls OpenFont(); essentially honoring the caller's request for an OT_DotSize other than the default that diskfont will provide when creating outlined fonts. This makes it possible to have multiple outline fonts which differ only by OT_DotSize.

o Underlined outline fonts are now supported if the engine can manage it when rendering the glyphs. New tags in diskfonttag.(hi) were added for this purpose. V39 diskfont will ask for underlining if requested in the TextAttr, or TTextAttr, but if not supported by the engine, then it will simply try to open, or create a non-underlined version. So there is no change in behavior for bullet fonts in that it already opens a non-underlined version. The difference is by asking OpenFont() for a non-underlined version, it does not decide to create another copy because of the result returned from WeighTAMatch().

  Of interest, our own Text() function when used with SetSoftStyle() does a broken underline (just one pixel, but its definitely not solid), so diskfont.library V39 first asks for broken underlining from the engine, and then solid. If neither form of underlining is supported, then the above is true.

o Tag for algorithmic strike-through added to diskfonttag.(hi); this is another feature supported in Final Copy, and could be requested if wrapped in a bullet interface.

o Underlining off, double solid, and double broken underlining also defined for the new OT_UnderLined tag.

  Underlining is not an intrinsic style for outline fonts. This does not actually absolutely have to be the case, but we lack a tag to indicate intrinsic underlined style now, so this is not a new behavior, or limitation. The assumption being that underlining is one of those things that is probably best done algorithmically by the application, or for the purposes of diskfont, by the engine. Therefore this also leaves open the possibility of placing the underlining code within diskfont for engines that do not support it should we wish to do so in the future.

Libs/iffparse.library

o Fixed three bugs in the OpenClipboard() function. First bug was that if the clipboard.device couldn't open, two calls to FreeSignal() were made with uninitialized values as parameters. The result of this was a corrupt signal mask in the Task field. Second bug what the OpenDevice() was called with an IO request that didn't have a valid MsgPort pointer in it. Finally, the two message ports allocated by the function were not being initialized correctly and would cause a system crash if a message ever got to either of them.

Prefs/#?

o Now restore task->tc_UserData on exit of prefs editor.

Prefs/Font

o Default for WB icon text mode when there is no .prefs file is now JAM2 instead of JAM1. This matches the ROM.

Prefs/IControl

o Added Mode Promotion gadget when running on AA machines

Prefs/Input

o "Show Double-Click" now operates asynchronously. That is, while the double-click sample is displayed, it is still possible to click on other gadgets.

Prefs/Overscan

o Now bounds check values read from prefs file against what the gfx database says. This avoids problems with V38 overscan prefs files, and with running VGAOnly or not.

Prefs/Palette

o Brand new interface using the colorwheel and gradientslider gadget classes. Also allows pen spec editing.

Prefs/Pointer

o Now opens on the Workbench screen if it can obtain three exclusive pens.

o Allows editing of the Normal and Busy pointers and supports Low-Res and High-Res pointer resolutions.

o Uses datatypes.library for clipboard support.

o Semaphore around save during Test. This is to prevent confusing error requesters if the user presses the Test button several times while waiting for the pointer to appear.

o Added "Load Image..." menu item in the "Edit" menu.

o Uses {SA_LikeWorkbench, TRUE} when using a custom screen.

o Increased width of window by 20 pixels to make room for longer strings.

o Added NOREMAP tooltype and command-line option. This causes paste and import image to not remap.

Prefs/ScreenMode

o Now skips the default monitor when scanning the mode list instead of skipping PAL or NTSC. This means that the prefs file will never contain "default monitor" as a mode, it will always contain an explicit mode.

Prefs/Sound

o Now uses datatypes.library for sound loading and playing. Also uses a DataTypes filter in the load sample requester so that only sounds will be shown.

Prefs/WBPattern

  o Added support for picture backdrops.

  o Added support for Workbench screen backdrop.

  o Defined a more efficient chunk for the preference data.

  o Fully supports the First4/Last4 Workbench color scheme.

  o Uses datatypes.library for clipboard support.

  o All internal bitmaps now max out at a depth of 3 (less memory usage).

  o Rearranged the gadgets.

  o Semaphore around save during Test. This is to prevent confusing error requesters if the user presses the Test button several times while waiting for the pattern to appear on the Workbench.

  o Added "Load Image..." menu item in the "Edit" menu.

  o Added NOREMAP tooltype and command-line option. This causes paste and import image to not remap.

S/Startup-Sequence

  o Removed "Echo" statement. That means the initial Shell window will no longer be displayed on bootup on a standard system. This means a noticeable decrease in bootup time on slower machines (it takes about a second to open and close the shell window). It means even more savings in time when the WB has more bitplanes. Finally, it also looks a heck of a lot more professional.

  o Added C: in front of every pertinent command. This speeds up the S-S noticeably on floppy systems, since normally the current directory is scanned first for a command, and then C:. By prefixing C:, it no longer checks the current directory making things a whole lot faster.

  o Added deferred assign for HELP:.

  o Added multi-assign of LIBS: to SYS:Classes in support of the new disk-based BOOPSI classes.

  o Now runs VGAOnly monitor file if it is installed.

Storage/Monitors/#?

  o Added DblNTSC and DblPAL in support of AA scan doubling.

  o Added VGAOnly monitor.

  o Added AA support throughout.

System/DiskCopy

  o Enabled code to correctly report read errors. Under V38, read errors are reported as write-errors.

System/Format

  o Requires V39.

  o Added directory caching support in the form of an extra gadget in the main Format window, and some new command-line options. Command-line template is now:

```
DEVICE=DRIVE/K/A,NAME/K/A,OFS/S,FFS/S,INTL=INTERNATIONAL/S,
NOINTL=NOINTERNATIONAL/S,DIRCACHE/S,NODIRCACHE/S,NOICONS/S,QUICK/S
```

  o Uses GTLV_MakeVisible tag in its device listview.

  o Fixed bug where long volume names would run off the right edge of the main window by using the new GTTX_Clipped gadtools tag.

System/Intellifont

  o Renamed from Fountain to Intellifont.

  o Bare minimum needed to fix visual bug evident when running under V39 gadtools; no longer uses GTLV_Selected tag, though still does not use V39 style selection via GTLV_ShowSelected tag.

Tools/Commodities/MouseBlanker

This is a mouse blanking commodity. It blanks the mouse when you start typing and unblanks it when the mouse is moved or any of the mouse buttons are hit.

Tools/HDToolBox

  o Brand new GadTools interface.

  o Now determines the version number of a file system automatically instead of requiring it to be entered manually.

  o Fixed a great number of bugs.

  o When CD_ROM drive was hooked on the SCSI bus and the disk was inserted, HDToolBox hanged up beacause the program assumed only 512 bytes blocks. It Now checks the size with the "READ_CAPACITY" command, and if it fails, sends "INQUIRY" command and gets device type. If it's a CD-ROM device, sets block size to 2048.

Tools/IconEdit

  o Implements the First4/Last4 Workbench color scheme.

  o Uses datatypes.library for clipboard support and picture loading/saving. Therefore, you can now load any picture type that you have a class for. When loading or pasting a picture, the colors are mapped to the Workbench GUI colors (either 2, 4 or 8 colors). 8 color icons will work with any depth Workbench screen.

  o Default clipboard viewer is now "Multiview CLIPBOARD".

  o Checks to see if icon already exists when saving a picture or source file and won't overwrite it.

  o File requester's initial position now matches the preference editors.

  o Now draws the coordinates in the window title bar using the pen spec.

  o Was reloading the icon image after save, even if save failed.

  o Will now properly undo after loading a new icon image.

  o Added NOREMAP tooltype. This causes paste and import image to not remap.

  o Fixes Enforcer hit caused when dropping Drawer icon (without a drawer) into the AppWindow.

Tools/ShowConfig

  o Added AA chipset support.

Utilities/MultiView

This is a generic view-anything utility that uses datatypes.library for its object handling.

3.0 Installation Procedure

- o It is no longer necessary to boot from the Install disk.
- o Install Languages no longer exists. This is now merged into the main installation script.
- o Install Printers no longer exists. The user can easily drag a printer driver icon from the Storage disk to DEVS:Printers
- o The disk now includes the 68040.library, which gets copied to the HD when performing an install
- o The various Prod_prep scripts were enhanced and work better.
- o There are many little utilities in the C: directory which aid the main install script to do a better job.

Changes to the install script include:

- o Requires V39 ROMs
- o Now integrates the old "Install Languages" script. When starting the main install script, you are now asked whether to perform a complete installation, or only update the languages.
- o It now virtually always makes the right decision as to where the WB files should be copied. As a result, if the user picks Novice mode, he will not be asked where the files should go.
- o Now preserves the tooltypes of many icons. This will avoid zapping the stuff put there by users, making the update process much more transparent.
- o Now preserves the locations of most files. For example, if the user moved Blanker in WBStartup, it will be correctly updated in WBStartup, and left there. In V38, the copy in WBStartup would be deleted and the new version installed in Tools/Commodities.
- o Now preserves left out icons. The UpdateWBFiles program ensures that the contents of the .backdrop file are up to date which removes the need to delete the file.
- o Now updates V37 font prefs files to V38/V39 format.
- o Now deals with the "Storage3.0" disk
- o Now asks which keymaps to install.
- o Installs the 68040.library.
- o Knows about systems with MapROM, and installs a special startup-sequence.
- o On an NTSC system, always copies the NTSC monitor to DEVS:Monitors, and copies PAL to DEVS:Monitors on PAL systems.
- o When installing languages, deletes languages the user did not choose. This is to remove old V38 files that might be hanging around.
- o Does better sniffing to determine whether an A2090 is present, and displays extra info to inform the owner that some files need to be copied.
- o Will optionally reboot the system for the user at the end of the installation procedure (after asking permission first, of course).
- o Asks all its questions up front. Once the questions are answered, it starts the actual installation.
- o Now deletes all C= printer drivers and keymaps before starting the installation process. This was previously done right before re-installing them. This is to increase the likelihood that things will fit on the HD by deleting obsolete stuff ahead of time.
- o Does slightly better icon positioning to avoid things moving "by themselves".
- o Now copies the A2232 port-handler and A2232 aux-handler when needed.

Changes to the HDSetup script include:

- o Will now partition the HD and format it automatically instead of requiring a reboot.
- o The partitions created are now called Workbench (HD0) and Work (HD1). HD0: is created as 8M.
- o DOS\3 is now used by default, and the DMA mask is set to 0xfffffffe instead of 0xfffffffc
- o The version number for the file system being installed on a machine is now extracted from the file system load file instead of being hardcoded in the prep scripts
- o After completing the reselection on, reselection off, or update superkickstart option, the system is rebooted automatically (after asking the user of course).
- o Updating an A3000 super kickstart no longer requires the user to hit RETURN. Errors are also detected now.

Appendix B: Release 3.01 Disk Changes

Following is a description of most of the important changes made to the disk-based system software between Release 3 and Release 3.01. This doesn't cover changes made to modules not discussed in this talk. It also doesn't cover any changes after December 1992. Since 3.01 is still under development as of this writing, more changes are likely to be made to the software prior to release.

C/ConClip

- Now consumes over 3100 bytes less RAM when running. That means over 3K more RAM in every system we ship.
- Localized
- Now keeps iffparse.library and clipboard.device closed whenever they are not in use.
- Added CLIPUNIT synonym to existing UNIT command-line option. This is to make it comply with the style guide and be more consistent with other programs such as WBPattern. The template is now:

```
    CLIPUNIT=UNIT/N,OFF/S
```

C/Copy

- Now correctly handles failure of SetProtection(), SetComment(), and SetFileDate(). Because SetComment() fails on NFS and on CrossDOS, SetFileDate() never was done on files, making the CLONE option not fully work. All three functions are considered failures only if IoErr() reports something different than ERROR_ACTION_NOT_KNOWN.
- In case of error after a call to DupLock(), an error code was being set always to ERROR_NO_FREE_STORE instead of using the result of IoErr().
- The default size for the buffers used was always equivalent to (BUF=0) which caused the buffers to be the size of the files being copied. This was contrary to the docs, and caused problems when copying large files through Envoy, as it could easily eat up all the memory in the system, not leaving enough for the memory needed by Envoy. The default size is now BUF=128 which gives a 64K buffer.

C/IPrefs

- Added needed support for the new display position control now offered by Overscan prefs.
- If there is no icontrol.prefs file, IPrefs sets the default for Mode Promotion to ON. That means machines will have mode promotion on by default.
- Now ensures that requested width, height, and depth for Workbench, fall within the allowed range as defined in the graphics database.

C/Mount

- Only change is the version number. Versions 39.1 and 39.2 of this program were accidentally numbered 38.1 and 38.2.

C/Protect

- Fixed error reporting. When used with wildcards, it would generate errors of the form "Can't set protection for #?" instead of giving a descriptive file name, and cause of error.

C/SetPatch

- Due to a bug fix late in the game, the filesystem broke with on DCFS file systems if the lock passed in was not Examine()e This patch fixes this by forcing an Examine() on a lock befor the ExAll() is called. It only does this on systems with V39 dos.library since only that ROM has the problem with DCFS ExA
- ChangeVPBitMap() was not properly doing the bitplane swizzle 8-bit HAM mode. The reason that this didn't show up earlier if the relationship between the bitplane pointers was the sam in the original and new bitmaps, the bug would correct itself So, you would only see this bug if your memory was fragged, o some other allocations got in between the separate allocation bitplanes.
- ScrollVPort() had the same bug as ChangeVPBitMap().

  The patch locks the ActiViewCprSemaphore, swizzles the bitmap calls the old entry point, and unswizzles the bitmap, when ca on a HAM-8 viewport.
- The patches for graphics.library/ScrollVPort() and ChangeVPBi now bump the graphics revision number to 90 if it is 89. Thi is so that installing the setpatch on an A1200 or A4000 will break people using work-arounds for the bug.
- When calling BltBitMap() with both source and destination int and a mask of -1, the low byte of d7 would be changed to the on exit. Patch saves d7, calls old, and restores it.
- Added the graphics monitor/view association hash patch.
- BltMaskBitMapRastPort() used to interpret the mask data incor if both the source and destination bitmaps were interleaved.
- Now includes the CP2024 Conner patch from the V37 SetPatch.

C/Version

- Obtaining the version of printer.device while it is loaded in trash memory. This is because printer.device doesn't initial lib_IdString. Although Version was checking for NULL, the la string was causing problems in the output routines.
- Fixed the FILE option which was causing the version numbers r displayed.

Classes/Gadget/colorwheel.gadget

- Querying the wheel for an explicit red/green/blue value woulc most up to date brightness value from the gradient slider cau to get slightly out of sync.
- Fixed bug where trying to open the class library under 1.3/2. cause a system crash instead of simply failing.

Classes/Gadget/gradientslider.gadget

- Fixed bug where trying to open the class library under 1.3/2. cause a system crash instead of simply failing.

Devs/mfm.device

- Fixed small problem of the wrong return results when opening with an invalid unit number.
- Changed the format function to update the physical track and in-memory buffer,

[redacted]

More important is that the dimensions of Multiscan, Euro72, Super72, DblNTSC and DblPAL are now greater than they used to be. This is especially important for DblNTSC and DblPAL, which are meant to resemble NTSC and PAL as closely as possible. The Dbl.... monitors are now 720 pixels wide for MaxOScan, compared to 724 for NTSC/PAL, and the 676 they used to be!

Also, if not running with VGAOnly, the "dead" part of the display on the left hand side of the screen is now useable as VideoOScan.

o Defined ScanDoubled versions of the Multiscan, Euro72 and Super72 monitors for better coercion on AA machines.

o All these monitors should now work under ECS, and give even greater dimensions.

o No AA modes should be in RAM on non-AA machines.

o Changed the names of the monitors from DoubleNTSC/PAL.monitor to DblNTSC/PAL.monitor to be consistent with the ModeID name strings.

o All the monitors now work only with graphics.library V39.

o With Kickstart 39.106 and VGAOnly, DblNTSC/PAL screens can only be 704 pixels wide max, whilst under the current kickstart they are 720 pixels wide. The DblNTSC/PAL monitor now checks for this.

o Cleared up an enforcer hit with adding the A2024 after iprefs was run, and promotion was enabled.

o Sprites were not being clipped properly in PAL A2024 modes when they crossed the panel boundaries. This was causing screen trashing when the pointer was at certain positions because the pointer would end up in the A2024's special control line.

Prefs/IControl

o Removed "Preserve Colors" gadget. The "Avoid Flicker" gadget has been moved into the "Miscellaneous" gadget group instead of being by itself in the "Coercion" group.

o If there is no prefs file, mode promotion is now turned on by default.

Prefs/Input

o Now uses GTLV_MakeVisible tag to ensure the currently selected keymap is visible in the listview.

Prefs/Overscan

o On the edit screen, changed the label of the left gadget from "Use" to "OK".

o Added code to support the new display positioning control. When you enter the overscan editing screen, if the mode supports it, there are four arrows that cause the entire visible portion of the display to move around. This is a wonderful new feature for DblNTSC, allowing sync-dependant centering of display modes. It is also possible to move the display by using the cursor keys.

o Changed the background color on the edit screen from black to dark grey. This makes it much easier to detect when the display is being pushed too far off the right edge of the display, as the color of the display turns quite dim.

o Made the edit screen SA_Exclusive in order to avoid folks dragging it down and revealing their WB. Since the program now makes dynamic changes to hsync and vsync, the WB can look bad when the edit screen is pulled down.

o Made the edit screen SA_Interleaved.

Prefs/Palette

o When incapable of displaying a colored color wheel, the program now puts up a gadget labelled "Color Wheel..." in place of the actual color wheel. Clicking on this gadget causes the color wheel to come scrolling up from the bottom of the screen until it fills the whole display. This color wheel screen lets the user edit the current color using the wheel. There is a pair of "OK/Cancel" gadgets to accept or reject the new color selection. Clicking on either gadgets causes the color wheel to scroll off the bottom of the display. Kinda neat :-)

o The sample section of the program now shows a sample screen title bar.

o When opening on a custom screen, no longer has a window border and title bar.

o When switching modes from 4 to multicolor and back, will no longer present an empty window if it is not capable of creating its new set of gadgets. The program will exit instead.

o Selecting a color > 4 in the main palette, and switching to 4 Color Settings would not correctly redraw the gradient slider to use a color within the available selection.

o Now uses color conversion routines from colorwheel.gadget instead of having inline duplicates.

o Fixed incorrect flags set in some menu items so that click select of these items worked incorrectly.

o Was letting you select between 4 and Multicolor Settings, even when running on a 4 color screen incapable of displaying the multicolor settings.

o When running on an A2024 display, it no longer displays a color wheel, nor lets you display one.

o Clicking in the sample section of the program causes the pen associated with the item clicked on to become selected in the pen list. That is, clicking on the sample window title bar will automatically select the "Active Window Title Bars" pen within the pen listview.

Prefs/Printer

o Now uses GTLV_MakeVisible tag to ensure the currently selected printer is visible in the listview.

Prefs/PrinterPS

o The default margins for graphics printing are now 1 inch on the left and right, instead of 1 inch on the left and 2 on the right (this was due to a typo).

o When in the Graphics Scaling page, selecting "Last Saved" from the menu would not cause the sample graphics to be redisplayed with the new values.

Prefs/ScreenMode

o Fixed formatting string so that horizontal scan rates with a decimal value less than .1 now come out right. That is, 29.02 was coming out as 29.2.

System/Format

o From within the device selection listview, it was possible to double-click on two different devices, and have Format accept the selection. A check is now done so that both clicks of a double-click occur on the same device.

o When formatting from Workbench, Format now ignores any trailing colons in the volume name. It would get all confused if someone tried to name a disk "Hello:" instead of simply "Hello". This fixes the most common error automatically.

Tools/Calculator

o Now gets the window title string from the current catalog instead of having it hardcoded to "Calculator".

Tools/Commoditites/ClickToFront

o Now also filters mouse clicks that come through as IECLASS_POINTERPOS and IECLASS_NEWPOINTERPOS, in addition to IECLASS_RAWMOUSE. This fixes the bug where ClickToFront did not work with events generated by tablet drivers.

Tools/Commodities/CrossDOS

o Fixed bug where the window would not open when there was no L:FileSystem_Trans directory, or no file within it.

o Fixed incorrect cleanup path in case of errors while scanning directories. The bug would cause a crash the next time the window was opened.

o Now only notifies CrossDOS handler tasks of changes in settings when new settings differ from the old ones.

o No longer has the CrossDOS semaphore locked when it locks the DOS device list. This could potentially cause deadlocks.

o Fixed bug where translation files were opened and read, but never closed.

o Fixed bug that would cause a crash under V37 or harmless Enforcer hits under V39. To reproduce, start the CrossDOS commodity and copy a large file to PC0 (devs:Kickstart for example). While the copy is in progress, pick PC0 from the CrossDOS commodity and change its text filtering setting. Now click the close gadget of the commodity program. The window stays open until the large write operation completes. As soon as this happens, the window closes, and a crash occurs under 2.0.

Tools/Commodities/FKey

o When there was a key sequence in FKey with a command such as "Insert Text" with a string associated with it, switching the command to something else, and coming back to "Insert Text" would cause the current string to be "forgotten". To get it back, you had to click in the text gadget and press RETURN. This is now fixed.

o Selecting a command such as "Insert Text", then typing in a string as argument would cause that string to be bound to the key sequence forever. That is, even if the command was switched to something like "Cycle Windows", the string argument would be saved out to disk and reloaded in memory the next time the program ran. This was incorrect as the command didn't have anything to do with the string. String arguments are now discarded for those commands that don't support them.

o The active key sequence is now correctly highlighted in the listview.

Tools/HDToolBox

o Now supports Host ID in "Partitioning" screen. Now multiple machines can share a single disk using different partitions.

o Added CHECKBOX_KIND gadget for allowing a block size other than 512 bytes in "File System Maintenance". To change file system block size in "File System Characteristics", select this gadget first.

o Changed from BUTTON_KIND gadget to CYCLE_KIND gadget for "File System Type". They get labelled with their dostype as found in the file system resource list.

o Added CHECKBOX_KIND gadgets for "Fast File System", "International Mode" and "Directory Cache".

o Added CYCLE_KIND gadget for file system block size.

o Recognizes the "Help" key now.

HDSetup/HDSetup

o Localized the name of the "Work" partition following request from Germany.

Install/Install

o No longer copies the A2232 port-handler from the install disk when an A2232 card is detected. The 39.1 port-handler handles the A2232 automatically

Appendix C: V39 ROM Changes

Following is a description of most of the important changes made to the ROM-based system software between V37 and the initial release of V39. This doesn't cover changes made to modules discussed in other talks, such as graphics and Intuition.

BootMenu

o Brand new interface featuring 4 different displays:

Main Page: Lets you select between "Boot Options...", "Display Options..." and "Expansion Board Diagnostic...". Clicking any one of these brings up the corresponding page. The "Boot" gadget resumes the boot operation using the options selected in the other pages, and the "Boot With No Startup-Sequence" gadget does the same, except it doesn't execute the startup-sequence.

Boot Page: Lets you control boot-related options. The listview on the left lets you pick which device to boot from. The one on the right lets you enable/disable devices in the system. There is also a "Disable CPU Caches" gadget. It turns off the CPU caches for the current boot, which saves a lot of games that break on 68040 processors because of the big caches. Use or Cancel bring you back to the main page.

Gfx Page: Lets you pick what mode to boot in, and what chip set to emulate. Use or Cancel bring you back to the main page.

Diag Page: Lists all the boards in the system and shows their state (Working or Defective). This page is automatically brought up during system bootup if a board is defective.

o Hitting any key toggles the display between NTSC and PAL. A message on the main page indicates this fact.

cia.resource

o Changed the priority of the interrupt servers to +120 such that they don't miss interrupts.

con-handler

o Fixed a low-memory trashing problem were CON: would signal a NULL task.

o It no longer will use proportional fonts (or rather fail at trying to use them) when opened on a public screen.

o It is no longer possible to size the window such that the entire interior goes away.

o Fixes the title bars (the parsing routine wasn't skipping over delimiters). This also fixes the "funny" results you would get with malformed CON: lines (like con:0a/0/640/200/title having a title of "200").

o Fixes negative sizes in the window spec. Height < 0 gives you max displayable height.

o Always opens window big enough for one line of text.

console.device

o CMD_CLEAR fixed, was broken in V37.

o Uses screen dimensions (rounded off to nearest byte width) to calculate maximum character map width instead of using bm_BytesPerRow. Likewise uses screen height instead of bm_Rows.

o First pass at the scrolling optimizations. Recalcs scroll mask at reset time (set to defaults), at full clear screen time (FF, or HOME/CLS), and whenever you set new pen/cell/background colors via the standard SGR sequences.

o Rework how conceal mode works - no longer sets rp_Mask to 0 which also disabled ALL output, including scrolling, clearing, etc... not good for character mapped consoles in particular since it causes the display & map to lose sync.

o Scroll DOWN no longer tries to fill in vacated portions of the window with text in the off-screen buffer (if any). This was useless as a scrollback feature, could crash if you scrolled down more than one window's worth of text (ouch), didn't perform a window refresh (thereby leaving hidden characters in the visible map until you resized, or revealed), and was inconsistent with SMART REFRESH consoles - the application using scroll down would reasonably expect that the vacated portion is entirely empty.

o First pass at breaking up large CMD_WRITEs. Now unlocks layers approx every 256 characters (this is simple, low-overhead code which does not try to be exact). Recalcs everything when layers are relocked.

Helps quite a bit. No more locks for the entire file when doing COPY foo *. Makes console much more friendly; you can now resize windows (only minor delay) during long writes, click in other windows, etc.

Note this simple code won't work for a long stream of text with no control characters; this however is extremely rare. Even text files have LF's, however in any case the problem is no worse than it was before.

o The above feature makes it possible to cheaply break a CMD_WRITE, hence making it possible to easily fix the DisplayBeep() deadlock bug. DisplayBeep() is now postponed a few CPU instructions until after layers are unlocked.

o First pass at modifying mouse tracking code (drag selection) to use input events instead of PeekQualifier().

This code change was added so tablet drivers, and commodities can be used in character mapped consoles with selection capability. PeekQualifier() returns only what input.device thinks the qualifiers are; not what's seen by applications which watch/use the input stream.

o Now supports a new private sequence ("ESC[s") which sets the current SGR settings as your defaults. This affects ESC[0m (reset all SGR settings), ESC[39m (reset default primary pen), and ESC[49m (reset default secondary pen (cell color)).

Text style info, and reverse mode (on/off) are also saved, and hence restored when ESC[0m is sent.

This is intended as a user sequence for use in your shell-startup; it allows you to use other colors/settings, and not have these constantly reset by programs like MORE, LS, etc. I'm recommending it not be used by applications; only users for their shells. An application which can deal with this problem of SGR settings should continue to do whatever it is doing now. ESCc (reset console) does however reset the default SGR settings to their true defaults.

o Downcoded pack.c. Is many times faster (if the maps are not disorganized; the maps become disorganized as text is scrolled off screen, so in these cases an initial resize can still take a moment - didn't want to touch that code though). For an organized map, resizing even very large windows (e.g., Moniterm size) with 8x8 or smaller fonts (so we have a really large map) is virtually instantaneous on a 3000, and nearly so even on 68000 machines. It still takes time to redraw the text (limited primarly by the Text() function), but the time needed to pack, and unpack the map is a fraction of what it was.

o Borderfill code added so ESC[>#m fills to borders if an explicit line length and/or page length have not been set. No change to cu_XRExtant or cu_YRExtant in public portion of console unit structure.

Border refers to the area outside of the normal console rendering area up to the window right/bottom borders. The size of this area is 0-N pixels where N is a maximum of the font width-1 or height-1.

o OpenDevice() now fails if trying to open a character mapped console, but memory can not be allocated for the map. In V37, OpenDevice() returned success for this case which left you with a half functioning console window - clearly confusing for the user, and virtually worthless because of the lack of refresh info needed to fix up damage.

OpenDevice() still works the same if you have a SIMPLE_REFRESH window, but did not ask for a character map (uncommon, but doable).

o Also fixed a bug which you may never see now that the above code was added; clearing a simple refresh window which lacked a character map cleared a garbage rectangle; layers prevents this from being a crash, and code elsewhere inhibited negative rectangles. The bug exists in 1.3 also, and was partially fixed for 2.0; the bug use to be apparent in SUPER_BITMAP windows, and because the case of SIMPLE_REFRESH without a map is rare for console.device, the bug has probably never been noticed (found during memoration testing).

o Removed code which checks to see if the application had drawn over the cursor in a console.device opened by the application. The kludge did a ReadPixel() of every pixel where the cursor was drawn, and if any bits did not match the expected color (also modified by pattern), cursor drawing was turned off for that console for as long as that console window was opened.

Applications (few) which draw over the console cursor, but do not explicitly turn it off will now have a patterned rectfill the size of the cursor (generally 8x8) in the upper left hand corner of the window if the window is deactivated. This is a minor visual problem, though not one which should cause anyone to crash, or not run. The problem will also never be noticed if the console window is not deactivated.

dos.library

o Fixes a bug in Open() where if the path was more than 127 characters long, a random byte of memory would be trashed.

o Has support for fib_OwnerXXX for the networking people. ExAll() supports ED_OWNER.

o Added ExAllEnd().

o Added SetOwner().

o Fixed overrun error in FGets() (if no newline or EOF, it reads one byte too many into your buffer - workaround for V36/37 - allocate buffer 1 byte larger than passed in.

o HUNK_RELOC32SHORT now works at the right value (1020). Also added a 32-bit PC-relative reference mode, mainly for >= '020-only executables.

o Added GVF_SAVE_VAR. For SaveVar(), it will now do the same actions for ENVARC:whatever, as well as ENV:.

o Fixed character classes in ParsePatternNoCase(). The classes weren't being promoted to upper case (i.e. [a-z] should have become [A-Z]). Note that only ParsePatternNoCase() was affected by this bug, not MatchFirst()/MatchNext()

o Fixed FreeDosObject(xxx,DOS_CLI). It wasn't freeing the strings associated with the CLI.

o FindCliProc(0) now returns NULL, even though it's an invalid CLI number.

o For people passing in an RDArgs structure but no RDA_Buffer to ReadArgs(), it now properly clears out RDA_Buffer on FreeArgs(), so you can reuse it safely (without having to clear it out yourself).

o StrToLong() was returning the number of white-space characters if no digits were found. It now properly returns -1.

o Fixed GVF_DONT_NULL_TERM for global variables.

o The initial console window on bootup now opens the size of the Workbench DClip.

o Fixes the CliInitNewCLI() open of S:Shell-Startup when no FROM file is specified.

o Fixed "Copy CONSOLE: foo" by making GetDeviceProc() know about CONSOLE:.

o Localized "Software Failure".

filesystem

o Added support for DOS\4 and DOS\5 file systems which offer directory caching. DOS\4 and DOS\5 are orders of magnitude faster at directories than other file systems, since they keep a cached copy of all the ExNext()/ExAll() data appended to the directory. This does require a few extra block accesses on create and delete, and also after modifying the file (in Close()). Create speed dropped about 30%. However, directory speed is 7-20 times faster.

For floppies, this usually means that dir or list take about 3/4-1 second to start, and then you get most or all of the directory instantly, or within 1/2 second or so (it may take 1 or 2 seconds for really big directories).

o Fixed a bug with delete for non-DOS\5 partitions.

o Fixed a random memory trash in the filesystem in a race condition. When two renames hit just the right timing, one has to wait on the other, and the wait routine used the wrong (garbage) register to get the head of the list. This trashed 1 longword of semi-random memory, and then hung the rename forever.

gadtools.library

o LayoutMenus() and LayoutMenuItems() recognize some new tags to support NewLook menus:

```
GTMN_NewLookMenus (BOOL): requests NewLook menu treatment.
GTMN_Checkmark (struct Image *): checkmark you'll use in menus
GTMN_AmigaKey (struct Image *): Amiga-key image you'll use in menus
```

Basically, if you open your window with WA_NewLookMenus, also lay out your menus with GTMN_NewLookMenus. If the menu-item font will be the screen's font, that's all you need to do. If the menu-item font is something else, you must create a checkmark and an Amiga-key image, and pass each one to both Intuition (WA_Checkmark and WA_AmigaKey) and to GadTools (GTMN_Checkmark and GTMN_AmigaKey).

GTMN_FrontPen is now recognized. If GTMN_NewLookMenus is specified, this attribute defaults to the screen's BARDETAILPEN, else it defaults to "do nothing", which allows the GTMN_FrontPen tag that may have been passed to CreateMenus() to still hold.

o STRING_KIND, INTEGER_KIND, and BUTTON_KIND gadgets now support the GA_Immediate tag.

o You can now put an arbitrary command string in the right-hand side of a menu, where the Amiga-key equivalent goes. To do this, point the NewMenu nm_CommKey field at the string (eg. "Shift-Alt-F1), and set the new NM_COMMANDSTRING flag in nm_Flags.

o If a window has multiple checkboxes or radio buttons, a separate image is no longer allocated for each one.

o The bevel box of the slider and listview now refresh with the gadget, instead of with GT_RefreshWindow().

o Scrollers with GA_RelVerify set weren't sending IDCMP_GADGETUP messages when the arrow buttons were released.

o GadTools now uses SetABPenDrMd() when advantageous.

o New GTMX_Scaled and GTCB_Scaled tags instruct GadTools to scale the mx button and checkmark respectively to the dimensions supplied in the NewGadget ng_Width and ng_Height fields. Under V37, or in the absence of these tags, the dimensions are fixed. Added #defines for those dimensions.

o GadTools now has a GT_GetGadgetAttrsA() function (and a GT_GetGadgetAttrs() varargs version). This function retrieves attributes of the specified gadget, according to the attributes chosen in the tag list. For each entry in the tag list, ti_Tag identifies the attribute, and ti_Data is a pointer to the long variable where you wish the result to be stored.

o Checkboxes now return their "selected" state in the IntuiMessage->Code field.

o Many new tags for CreateGadgetA() were added:

GTTX_FrontPen and GTTX_BackPen to let the color of TEXT_KIND gadgets be controlled.

GTNM_FrontPen and GTNM_BackPen to let the color of NUMBER_KIND gadgets be controlled.

GTNM_Format to specify the formatting string to use with NUMBER_KIND gadgets. This is so a localized number format using "%lD" instead of "%ld" can be used.

GTNM_MaxFormatLen to specify the maximum length of the string that can be generated by GTNM_Format.

GTTX_Justification and GTNM_Justification to allow for right and center justification on TEXT_KIND and NUMBER_KIND gadgets.

GTSL_MaxPixelLen lets you specify the maximum pixel length the level display of a SLIDER_KIND gadget will occupy. This allows proportional fonts to be used with sliders.

GTSL_Justification specifies how the level display of a SLIDER_KIND gadget is to be justified within the width allocated by GTSL_MaxPixelLen.

GTLV_MakeVisible for listviews. You pass it an item number and it makes sure it is visible within the listview display.

o Many new tags for GT_SetGadgetAttrsA():

GTTX_FrontPen, GTTX_BackPen, GTNM_FrontPen, GTNM_BackPen, GTNM_Format GTTX_Justification, GTNM_Justification, GTSL_Justification, and GTLV_MakeVisible all have the same purpose as described for CreateGadgetA() above.

MX_KIND gadgets now support GA_Disabled.

GTSL_DispFunc and GTSL_LevelFormat are now changeable via GT_SetGadgetAttrs() instead of being create-time only attributes.

o Added the GTBB_FrameType tag which gives access to the new frame types available in Intuition. You pass the tag to DrawBevelBox() and can specify BBFT_BUTTON, BBFT_RIDGE or BBFT_ICONDROPBOX.

o GT_SetGadgetAttrs() can now safely be called when the gadget being affected is not attached to a window, by passing a NULL window parameter

o Specifying GTTX_CopyText and not GTTX_Text now works for TEXT_KIND gadgets.

o TEXT_KIND or NUMBER_KIND gadgets that have the GTTX_BackPen or GTNM_BackPen tags specified look visually cleaner when changing the gadget text using GTTX_Text or GTNM_Number than those without. Listviews take advantage of this when applicable.

o The value of GTSL_Level is now bounds checked at CreateGadget() time in addition of at GT_SetGadgetAttrs() time.

o Fixed bug where doing GT_SetGadgetAttrs() on a MX_KIND gadget and not passing the GTMX_Active tag would reset the active selection to #0 instead of leaving it alone.

o The NewGadget.ng_TextAttr field can now be NULL whenever a gadget is created. In such a case, the screen's TextAttr is used (screen's TextAttr is determined from the VisualInfo in the NewGadget structure).

o The level display of SLIDER_KIND gadgets is now rendered with background set to BACKGROUNDPEN instead of 0, which is more "correct".

o Now uses SetWriteMask() instead of SetWrMsk().

o Added support for gadget help in all gadget types.

o GadTools now handles the new ExtIntuiMessage generated by Intuition.

o Many enhancements to PALETTE_KIND gadgets:

Palette gadgets no longer display a box filled with the selected color. The selected color is instead denoted by a box drawn around the color square in the main palette area.

Palette gadgets now allow strumming, and right mouse button cancellation.

GTPA_ColorTable is a new tag to support sparce color tables in gadtools. The tag can be passed at create/set/get time.

GTPA_NumColors is a new tag to specify the total number of colors to display. This allows amounts of colors that are not powers of 2. This tag is also good at create/get time.

GTPA_ColorOffset is now supported at get/set time.

Renders itself much faster, this makes a big difference on 256 color screens.

Now does quite smart layout of the color squares. An attempt is made to keep them as square as possible, based on the aspect ratio information obtained from the gfx database. As many colors as possible are put on the screen, until things get too small in which case the upper colors are thrown away.

o MX_KIND gadgets now support ng_GadgetText and will display the label in relation to the group of mx gadgets.

o Added GTMX_TitlePlace tag. This determines where the title of a MX_KIND gadget is displayed. If this tag is not provided, the title is not displayed. This is required for compatibility.

o Fixed size calculation errors in listview present since V37. This may cause certain listviews to change in size from their V37 size.

- o Revamp of listviews:

    ListView lines can no longer end up complemented in certain tricky situations involving detaching lists.

    ListViews correctly track the selected line when you click in them.

    ListViews were clipping the text four pixels early on the right.

    Added GTLV_CallBack. This tag allows a callback hook to be provided to gadtools for listview handling. Currently, the hook will only be called to render an individual item. This adds the very useful ability to define a callback hook which is used to scroll complex items such as graphics, etc.

    Listviews now allow strumming. That is, holding down the left mouse button and moving the mouse up or down causes the active selection to track the mouse. Moving the mouse off the top or bottom of the listview causes the list to scroll.

    Listviews that used to have a display or string gadget underneath them now have a highlight bar to indicate the selected item. This is in anticipation of listview multi-selection. If the listview had a display gadget, it no longer does as the highlight bar is used. If a listview had a string gadget, it retains it.

    Listviews highlighting is done using the pen-spec method instead of the 1.3 complementing method.

    Listviews are much faster at rendering and scrolling, which makes a noticeable difference in 8 bit planes

    Added GTLV_MaxPen tag to specify the maximum pen number used by a custom rendering callback hook. This enables more optimal scrolling and screen updates.

    {GTLV_Selected, ~0} is now supported at both create and set times.

    {GTLV_Labels, ~0} is now supported at create time.

    Listviews now support GA_Disabled. This causes the list area to be ghosted, but the scroller and arrows remain unghosted.

- o When cloning a rastport for internal use, no longer copies the TmpRas field of the original rastport, which should eliminate some potential bugs.

- o Changed the definition of TEXTIDCMP and NUMBERIDCMP in gadtools.h to be (0) instead of (NULL), to keep the Manx compiler happy.

- o It is now safe to call GT_GetGadgetAttrs() and GT_SetGadgetAttrs() with NULL gadget pointers.

- o When GT_SetGadgetAttrs() is called on an active STRING_KIND or NUMBER_KIND gadget, the gadget is automatically reactivated after its string is changed. Although this reactivation flickers, the functionality is quite useful.

- o Now copies a complete TTextAttr structure when needed to fix potential problems with WeightTAMatch(). This is only done to create underlines under gadget labels.

- o Fixed example in CreateGadget() autodoc. Only had a single argument in the call to GT_RefreshWindow()

- o BOOPSI images are now allowed in gadtools menus.

- o Added the GTTX_Clipped tag for TEXT_KIND gadgets.

ramdrive.device

- o Uses AllocMem(xxxx,MEMF_REVERSE|MEMF_KICK|MEMF_NO_EXPUNGE) instead of private AllocHigh() code.

- o Uses CopyMem() instead of an unrolled loop. CopyMem()'s MOVEM's are faster than the unrolled loop used previously.

- o Protects KickTag/KickMem list with FORBID/PERMIT inside of KillRad() vector used by REMRAD. Fixes possible crash if some other task is fiddling with these lists at the same time.

ram-handler

- o Fixes the long standing bug where if the file you were examining with ExNext() is deleted, RAM: goes off into never-never land (and your system follows). If the file is deleted, it will restart at the beginning of the directory.

- o Disabled softlinks in RAM: to save ROM space.

shell

- o NewShell/NewCLI now open full with (like Shell from WB).

- o NewShell/NewCLI now handle FROM fields up to 255 long (up from 127) and errors out if FROM or WINDOW are too long.

- o Resident module handling now properly Forbid()s around seg_UC++/--.

- o Prompt now handles %%.

- o Removed two harmless enforcer hits at boottime.

- o If a "command 'command...' ..." fails, it no longer inserts the error message and continues (FailAt is used to determine failure).

- o Added evil kludge to solve the problem of 1.3 SetClock crashing on 68040s.

- o Fixes the write to rom on <> redirection.

- o NewShell/NewCLI no longer print error messages if no S:Shell-Startup is present.

timer.device

- o Timer keys off new GfxBase flag for determining EClock frequency since PAL/NTSC is now software selectable.

trackdisk.device

- o Post-write delay has been moved from 2ms (spec is 1.2ms) to 3ms, and side settle delay from 1ms (spec is 0.1ms) to 1.5ms. This should fix most A1010's out there. In addition, both of those values have been made part of the public unit structure like settle delay and step delay, so people with REALLY bad A1010's can back it off as far as they need to (or setpatch can).

- o Fixes a nasty oversight in the HD floppy handling. After switching from a HD floppy to LD floppy and back to HD floppy, you could never safely write to an HD floppy unless you formatted an HD floppy first.

    What happened was that the extra slop area at the front of the write wasn't getting set to $aaaaaaaa, it was being left with garbage from the last LD read (since LD uses less slop, it's start-read spot is earlier). Format re-inits the entire buffer, as does the first switch to an HD floppy (only the first, since it switches to a larger buffer then).

- o Fixed a bug where if a track was totally unreadable it returned the number of retries as the IO_ERROR instead of TDERR_NoSecHdr (this was causing the "Error 11" stuff when you popped a disk while copying from it).

```
utility.library

  o Downcoded all tag calls from C to assembly which yields substantially
    faster performance.

  o Removed 68020-specific versions of the date conversion routines.

  o Fixed bug in MapTags() where the "includeMiss" parameter didn't work.

  o Cleaned up and expanded autodocs.

  o Cleaned up public include files.

  o Added comments in the autodoc entries for the 4 32-bit math routines,
    to the effect that they preserve address registers, and that A6
    does NOT have to be loaded in order to call the routines.  This is an
    exception to the standard rule, but can avoid register shuffling which
    is important in low-level math routines.

  o Made the math routines several cycles faster on 68000 machines.

  o Added SMult64() and UMult64() which do 32x32=64 bit integer math.

  o Added ApplyTagChanges().

  o Added two new library calls that are mainly here to help intuition get
    smaller.  These are PackStructureTags() and UnpackStructureTags().

  o Added the NameSpace code.

  o Added GetUniqueID().
```

Appendix D: Release 3.01 ROM Changes

Following is a description of most of the important changes made to the ROM-based system software between Release 3 and Release 3.01. This doesn't cover changes made to modules not discussed in this talk. It also doesn't cover any changes after December 1992. Since 3.01 is still under development as of this writing, more changes are likely to be made to the software prior to release.

BootMenu

- o Fixed bug where the chip type mutual exclude gadget was being displayed even on pre-ECS machines.

exec.library

- o Added the full support for the Zorro-III quick interrupts. The new LVO (in an old slot) ObtainQuickVector() is used to allocate the vector. There is no deallocation since this is basically a configuration issue and not a dynamic thing.
- o On machines with PCMCIA cards, EXEC now makes sure the interface is turned on at boot time and then will turn it off before configuration. This should let a full 8-meg of RAM be added in the Zorro-II space. This change requires an update to the credit card resource/device such that it will correctly turn on the interface if needed.
- o The Quick Interrupt vectors that have not yet been added used to be -1. Now they point at an Alert that is the new Unexpected Quick Interrupt.

gadtools.library

- o Fixed bugs in clipping code in TEXT_KIND and NUMBER_KIND gadgets. The clipping didn't work correctly on right and center justified text, and was under-evaluating the number of pixels available for the text in a gadget that didn't have borders.
- o Fixed bug in the calculation of the default value for the GTSL_MaxPixelLen tag. This caused odd clipping of the number display for sliders whenever the title of the gadget wasn't on the same side of the slider as the display of its current value.
- o Fixed GTJ_CENTER option for the various GTXX_Justification tags. The way centering was done could cause certain characters to get lost.

carddisk.device

- o Now flushes cache during data writes in anticipation of 040 copyback cache on A1200 (no hardware support for PCMCIA memory space data cache control provided, so the data cache is still potentially a problem when programming flash rom; means turning off the data cache globally for 030/040 A1200's to support 6-10us write/verify timing).

card.resource

- o Now leaves PCMCIA slot disabled if any RAM is configured at $600000; this allows use of >4Megs of 24bit RAM on the A1200 at the expense of being unable to use the PCMCIA slot.
- o Partial work around for a hardware bug in our PCMCIA implementation which presents 2Meg+ addresses everytime we access ATTRIBUTE memory. This causes a problem when a >2Meg card which ignores REG is used (and a potential problem with any card which tries to decode the entire address when REG is set). The former problem is kludged around by trying to sniff out mirroring of 4 bytes at $A00000 and $800000 but not mirrored at $600000.

  - Tested with:
    - Fujitsu 512K SRAM (ignores REG)
    - Fujitsu 128K SRAM (decodes entire address)
    - Panasonic 512K SRAM (returns $FF for attribute memory)
    - HP 128K SRAM (has 16 bytes of attribute memory)
    - NewMedia 2M PSRAM (ignores REG)
    - NewMedia 4M PSRAM (ignores REG - this is the card which demonstrates the problem)

- o Considerably faster memory sizing for SRAM/DRAM cards (does test of every 256 words/long-words) - tested with:

  - Fujitsu 512K SRAM (ignores REG)
  - Fujitsu 128K SRAM (decodes entire address)
  - Panasonic 512K SRAM (returns $FF for attribute memory)
  - HP 128K SRAM (has 16 bytes of attribute memory)
  - NewMedia 2M PSRAM (ignores REG)
  - NewMedia 4M PSRAM (ignores REG - this is the card which demonstrates the problem)

- o CardMemoryMap structure extended for V39 card.resource. Now has COMMON/ATTR/IO Memory Zone size for lookup via structure. Will be used to provide splitting of memory zones in the future if needed. No change for existing software.
- o BVD1/SC, BVD2/DA, and BSY/IRQ status change interrupts can now be individually enabled/disabled. WR (Write-Protect) status change interrupts are always enabled (rare), and there is no change in the defaults. This is intended for future use if needed (e.g., Flash-ROM which expects software to poll SC during programming; better performance can be obtained if interrupts are not generated). If needed on the A600, this can be implemented as documented work around, or SeFunction() of CardMiscControl(). No expected change for existing software; defaults are the same as they use to be in V37 card.resource. Spurious interrupts (change true, but interrupt disabled) are cleared by the resource software, and hidden from the status change callback hook.
- o Secondary callback option for status change interrupts; allows high-performance hardware to be serviced via interrupts only (instead of signalling a task).
- o Flush Cache when ReleaseCard() is called. A flush before full release ensures that no more writes will occur once the caller returns from ReleaseCard(). This is to support the 040 copyback cache when/if an 040 becomes available for the A1200. Would prefer control over the data cache for PCMCIA space independent of the first 4MEG of 24bit Fast RAM, but we don't have this feature. Lack of Data Cache control for PCMCIA space is still potentially problematic for use of FlashROM programming which requires disabling the DATA cache for 030/040 equipped A1200's so that fast (6-10us) write/read operations can be performed during programming. Disabling the DATA cache during FlashROM writes means disabling globally.

dos.library

- o Fixed bug in RemAssignList(): it wouldn't remove the first lock in the assign.
- o AttemptLockDosList() was returning NULL or 1 for failure instead of NULL.
- o Made RunCommand() free any memory added to the tc_MemEntryList by the command being run. tc_MemEntry is now saved and emptied before calling the command, and restored after any added memory is freed.
- o Fixed ExAll() emulation to not lose 1 file each time the list is broken up into multiple ExAll() calls.
- o Removed broken attempted fix for rda_Buffer. Autodocs now reflect that you must restore rda_Buffer before each call to ReadArgs() if you pass in an RDArgs structure. Now always clears rda_Buffer in FreeArgs().
- o SetVBuf() enabled.

o Changed some prototypes to avoid c++ reserved word "template". Changed VPrintf()/VFPrintf() prototypes to VOID * from LONG * to reduce useless compiler warnings/casts.

o GetDeviceProc() now returns errors better (especially ERROR_NO_MORE_ENTRIES). It used to lose error codes by calling UnLock().

o SetVBuf() now updates the filehandle so it won't overwrite the buffer with a smaller one if SetVBuf() is called before doing buffered IO. Also it doesn't allocate anything if the new size is the same as the old.

o SetVar() now creates subdirectories as needed (including multiple ones) if they do not exist already (in ENV: and in ENVARC: if GVF_SAVE_VAR is set). Also, it now preserves any IoErr() and won't try to save to ENVARC if there is an error saving to ENV:.

o Modified to fix an edge condition which existed when making the mod to SetVBuf().

expansion.library

o New A1200-specific build that can detect CPU Slot RAM ($08000000) if you have a 32-bit addressing CPU installed. The CPU slot area is 128meg in size (just like the A3000) but has the addition of a wrap check at each 1meg of space in the CPU address space to make low-cost RAM expansion possible without jumpers. (It is now possible to get 128Meg SIMMs so a single SIMM on a CPU card could make a 128Meg of FAST RAM system)

The reason that this has to be A1200 specific (at least for now) is that the behavior of the existing A500/A2000 CPU cards with respect to 32-bit addresses is very undefined. They act very strangely and differently making it very difficult to safely figure out if these cards are operating correctly or not.

filesystem

o Fixed deletion of the destination of a hardlink - this was badly broken in all versions of the FS, DCFS just made it easier to hit. This was causing spurious "Checksum Error on Block 0" errors (and potentially others), especially when UUCP was using a DCFS partition.

o Fixed a return code which would make softlinks not work if a softlink to a directory is in the middle of a path.

o Fixed the buffer overrun on ExAll() with ED_COMMENT if the first character was >$80 (and lost the first character of comments).

o Fixes updating the date of a directory that changes in the parent of that directory's dircache.

o There were old offsetting bugs in the ExAll() filename/comment copying code. When I fixed the code not to copy too many bytes, the clear was being done to the wrong byte.

filesysres.resource

o Now matches the FS version change.

workbench.library

o Adjusted the sizes of the OK/CANCEL and SAVE/CANCEL gadgets in the Workbench requesters to match the rest of the system.

o Fixed a long standing bug that was just found: The system would crash (sometimes) or cause Enforcer hits if files were deleted within a drawer that was also selected for deletion. This one has a fundamental flaw in Workbench which had to be patched with some rather tricky organization of tests...

IFF

by Carolyn Scheppner

ILBM Files: V39 and AA Support Issues

For compatibility with and support of enhanced Amiga graphics capabilities, both the short-term and long-term possibilities, you must modify your software to remove any built-in limitations that will prevent you from growing *with* the Amiga.

Get Rid of Hardcoded Limits, Write Software that Adapts

Many of the Amiga graphics software packages currently on the market are hardcoded like the old "DF0: DH0:" file requesters.

Such hardcoded graphics application software limits include:

- ❑ offering a fixed set of display modes or sizes
- ❑ offering a fixed range of depths or sizes for certain display modes
- ❑ loading or handling a maximum of 32 colors
- ❑ dealing with color guns as 4-bit values

The first thing you need to think about when upgrading your application for V39, is *not* to upgrade it for V39. You must upgrade your software so that it can adapt to arbitrary display modes, depths, and sizes.

If you offer different display modes, do not arbitrarily restrict the modes that you offer. If 8-bitplane hires, or hires HAM, are supported by a new chip set, and your software restricts a user to 5-bitplane hires and lores HAM, then your software will be obsolete.

Rewrite your software to use features such as the 2.1 display mode requester. Make sure that your software can adapt to larger palettes. Handle R G and B values internally as at least 8-bits, not 4.

Proper IFF ILBM Support

When loading, saving, and processing ILBM files, care must be taken to properly support the current and future graphics capabilities of the Amiga. Hardcoded limits and assumptions often exist in ILBM handling code. The NewIFF39 code modules attempt to properly implement all of the following concepts in a backward (and hopefully forward) compatible manner.

Proper ILBM.CAMG Chunk

Saving

If running under V36 or higher, save a 32-bit mode id in the CAMG ULONG. Some of these modes have all zeros in the upper word and are classic Amiga viewmodes. Others are more complex but generally provide a good bit pattern in the low word to allow reasonable results if displayed with an old viewer. You may wish to let your user decide if a different mode id should be saved. For example, a user may prefer to work in DBLNTSC but need to save his images as plain Hires LACE for genlocking. See V39 graphics/modeid.h for current modes. Use the 2.1 ASL screen mode requester or the display database if you wish to offer only all modes available on the user's system. Use ULONG modeid = GetVPModeID(viewport) if you are saving the mode id of a display.

Loading

Support reading and using full 32-bit mode ids from CAMG, with some screening for bad IDs, and fallback code if ModeNotAvailable(). Screening is required because there are some CAMGs out there with garbage in the upper word. See sample "getcamg" code at end. Under V39 or higher, the new BestModeIDA() graphics function can be used to query for a suitable and available replacement mode when an ILBM CAMG mode is not available. Stop looking at the bits of graphics mode ids. See the NewIFF39 code modules for example usage of BestModeIDA().

Proper ILBM.BMHD X and Y aspect

Saving

Use the display database in a simple manner to get the correct x and y aspect values for the ILBM.BMHD. See the "getaspect" code below. This code gets the correct aspect ratio for any viewport modeid from the display database. If running under < V36, it falls back to updated 2.0-compatible aspect values for old modes.

Loading

Perhaps you can start to expect reasonable information in the ILBM.BMHD x and y aspect fields.

Detect and use all 8 bits of CMAP color, if provided, when running under V39 or higher machine. If you see the new BMHDF_CMAPOK flag set in BMHD.Flags (described above), then you can be sure that the CMAP already contains 8 significant bits per gun of color. If you are running under V39 or higher, scale each 8-bit gun value to 32-bits (by duplicating it in all 4 bytes of a ULONG), and load the colors using one of the V39 32-bit color loading functions (LoadRGB32(), SetRGB32(), SetRGB32CM()).

If the BMHDF_CMAPOK bit is not set in the BMHD, then to display the CMAP colors correctly on an AA system, you need to determine if the stored CMAP color gun values are shifted 4-bit or full 8-bit values. If you do not examine the CMAP and instead just assume that it has 8 significant bits, you will display the colors of many ILBMs incorrectly dim, while older LoadRGB4() loaders will actually display the correct colors (because the V39 4-bit color loading functions will scale the passed 4-bit values properly to 32-bits).

You can try to determine if the CMAP contains all 4-bit left-justified RGB values by examining the low nibble of every CMAP entry you intend to use (do not examine additional registers or garbage which may be stored in the CMAP). If every examined low nibble is 0, then you would probably be correct to assume that the CMAP contains 4-bit left-justified values. In this case, you can correctly scale these values to 32-bits by first scaling to 8 bits (i.e., duplicate the upper nibble of each gun into its low nibble), then scaling to 32-bits (as described above). If any of the examined CMAP nibbles is non-zero, then the 8-bit CMAP values are directly scaled to 32-bits. Then load the colors using one of the V39 32-bit color loading functions.

When loading on a pre-V39 machine, just use the upper (most significant) nibble of each 8-bit CMAP gun value when calling the old 4-bit-per-gun pre-V39 graphics functions (LoadRGB4, SetRGB4, SetRGB4CM). Be very careful to AND out the low nibble of each gun before shifting and OR'ing R, G, and B guns to create an xRGB word for pre-V39 functions.

Stop Limiting Color Register Load Counts to 32

Older IFF code, and even the early V37 NewIFF code would read any number of colors from an ILBM CMAP, but would only set a maximum 32 colors in the display. Instead, the maximum number of colors set in the display should be limited by the display Viewport's

important. A new advisory BMHD flag, BMHDF_CMAPOK, has been defined to indicate that an ILBM contains an 8 bit significant CMAP, not an old 4-bit left-justified CMAP. The advisory flag is the high bit (1 << 7) of the BMHD.Flags byte (formerly named BMHD.Pad1 or BMHD.Reserved1). Whenever you save an 8-bit-significant CMAP (either 4 bits scaled to 8 bits or true 8 bits), we suggest that you set this flag bit in your BMHD.Flags to advise aware loaders that your CMAP values are definitely not 4-bit shifted values and do not need scaling to 8 bits.

```
#define BMHDB_CMAPOK     7
#define BMHDF_CMAPOK     (1 << BMHDB_CMAPOK)
```

[redacted]

Stop Limiting Depth to 5/6

Older IFF code had fixed limits for the maximum allowable depth for displays and ILBMs. Remove your limits. Display as much as the system can handle. Don't reject depths and depth/mode combinations arbitrarily. Also, you may want to stop assuming that a 6-plane ILBM with no CAMG is HAM or HALFBRITE (although that might still be a good assumption since only a pretty lame program would write a HAM or HALFBRITE ILBM with no CAMG chunk).

Watch out for BitMap->BytesPerRow

We have seen a number of products which have problems with BitMap->BytesPerRow rounding changes under V39 with AA.

BitMap->BytesPerRow is a modulo - it is the number that must be added to the address of a bitmap byte to get to the same byte one scan line down. Prior to V39, the value of BitMap->BytesPerRow always happened to be the width of a bitmap scan line rounded up to the nearest multiple of 16, then divided by 8. And all BitMaps, whether allocated via AllocRaster, or AllocMem using the RASSIZE macro, or via OpenScreen, had this same BytesPerRow rounding. This rounding aligned every scanline on a word boundary to allow fetching by the word-oriented custom chips.

In addition, the IFF ILBM spec states that the scan lines of an ILBM BODY must be saved as width rounded up to the nearest multiple of 16 pixels (i.e., as an even number of bytes width). So it is not surprising that assumptions about BitMap->ByesPerRow crept into ILBM handling code.

ILBM BODY scan lines must still be stored as pixel width rounded up to a multiple of 16. But under V39 and AA, the higher bandwidth required to support new graphics modes requires that the scan lines of a displayable BitMap be aligned on larger boundaries. Therefore under V39 and AA, the BytesPerRow of a BitMap must not be confused with the correct storage width of an ILBM.

In addition, graphic applications must be very careful not to assume that all BitMaps of the same width will have the same BytesPerRow. >From V39 onwards, BitMaps allocated by different methods (e.g., OpenScreen(), AllocRaster(), AllocMem(), AllocBitMap()), or allocated for display by different chipsets (ECS, AA, etc.) or for different display modes or a promoted display mode, may have different scanline alignment restrictions and therefore a different BytesPerRow. Do not assume that bitplanes allocated by different methods can be swapped, or processor copied across scanline boundaries.

To test for BitMap->BytesPerRow problems, you generally must test on a AA machine with either mode promotion or explicit use of higher bandwidth (greater alignment restriction) modes.

Common symptoms of BitMap->BytesPerRow problems are

1. a skewing of the display, where the pixels or each successive scan line all appear shifted to either the left or right, creating a diagonal effect, or
2. excess blank space at the right edge of an ILBM or a display.

Watch out for interleaved bitmaps

If your application supports capturing any screen, you must watch out for the new interleaved bitmaps. An interleaved BitMap's BytesPerRow field is still the modulo for getting from any one pixel to the pixel directly below it, *but* it is no longer even related to the rounded up width of the screen or viewport. Instead, it is a *much* larger value which is actually the rounded up BitMap scan line width *times* the depth. Do not assume that BytesPerRow is related to the width of the display.

> *Note*: The 2.0 Native Developer Update release of the NewIFF code had 2 major bugs. The screen.c module had a 1.3 incompatibility, and the ilbmr.c module could not properly save an interleaved bitmap (such as the V39 Workbench screen). See the newer version 37.10 of the NewIFF code. This has been placed in our listings area on BIX, and sent to ADSP and Fred Fish. For full AA color support, see the NewIFF39 code (available to developers via BIX, ADSP, CIX and DevCon disks, and planned to be provided on the 3.0 Native Developer Update and given to Fred Fish).

Under V39, an interleaved bitmap can be detected by:

```
if(GetBitMapAttr(bitmap_ptr,BMA_FLAGS) & BMF_INTERLEAVED)
    printf("is interleaved");
```

Proper Printing of New Display Modes

When dumping a RastPort to printer under V36 and higher, the following IORequest field must contain a 32-bit modeid such as that returned by GetVPModeID(viewport). You may want to allow the user the ability to print a display with a different modeid than it is being displayed in. Passing the full modeid allows the printer.device to properly control the aspect of the output, as long as the mode is available.

```
ULONG   io_Modes;              /* graphics viewport modes */

-------------------------- getcamg -------------------------------
From: /* ilbmr.c --- ILBM loading routines for use with iffparse */

/*
 * Returns CAMG or computed mode for storage in ilbm->camg
 *
 * ilbm->Bmhd structure must be initialized prior to this call.
 */

ULONG getcamg(struct ILBMInfo *ilbm)
{
struct IFFHandle *iff;
struct StoredProperty *sp;
UWORD  wide,high,deep;
ULONG modeid = 0L;

if(!(iff=ilbm->ParseInfo.iff))return(0L);

wide = ilbm->Bmhd.pageWidth;
high = ilbm->Bmhd.pageHeight;
deep = ilbm->Bmhd.nPlanes;

D(bug("Getting CAMG for w=%ld h=%ld d=%ld ILBM",wide,high,deep));

/* Grab CAMG's idea of the viewmodes. */
if(sp = FindProp (iff, ID_ILBM, ID_CAMG))
  {
  modeid = (* (ULONG *) sp->sp_Data);

  /* knock bad bits out of old-style 16-bit viewmode CAMGs */
  if((!(modeid & MONITOR_ID_MASK)) || ((modeid & EXTENDED_MODE)&&(!(modeid & 0xFFFF0000))))

  modeid &= (~(EXTENDED_MODE|SPRITES|GENLOCK_AUDIO|GENLOCK_VIDEO|VP_HIDE));

  /* check for bogus CAMG like DPaintII brushes with junk in upper word and extended bit
   * not set in lower word.
   */
  if((modeid & 0xFFFF0000)&&(!(modeid & 0x00001000))) sp=NULL;
  }

if(!sp)
  {
  /* No CAMG (or bad CAMG) present; use computed modes. */
 modeid = 0L;
  if (wide >= 640)        modeid = HIRES;
  if (high >= 400)        modeid |= LACE;

  /* This 6 planes == HAM or HALFBRITE is not necessarily true anymore, but hopefully
   * all NEW programs are writing a proper CAMG chunk!!
   */

   if(deep == 6)
     {
     modeid |= ilbm->EHB ? EXTRA_HALFBRITE : HAM;
     }

   D(bug("No CAMG found - using mode $%08lx",modeid));
   }

D(bug("getcamg: modeid = $%08lx",modeid));
return(modeid);
}
-------------------------- getaspect -------------------------------

bmhd->xAspect = 0;  /* So we can tell when we've got it */
if(GfxBase->lib_Version >=36)
  {
  if(GetDisplayInfoData(NULL, (UBYTE *)&DI,sizeof(struct DisplayInfo), DTAG_DISP, modeid))
    {
    bmhd->xAspect =  DI.Resolution.x;
    bmhd->yAspect =  DI.Resolution.y;
    }
  }

  /* If running under 1.3 or GetDisplayInfoData failed, use old method of guessing aspect ratio
   */
 if(! bmhd->xAspect)
   {
   bmhd->xAspect =  44;
   bmhd->yAspect =  ((struct GfxBase *)GfxBase)->DisplayFlags & PAL ? 44 : 52;
   if(modeid & HIRES)      bmhd->xAspect = bmhd->xAspect >> 1;
   if(modeid & LACE)       bmhd->yAspect = bmhd->yAspect >> 1;
   }
```

NewIFF39: IFF Modules with AA Support

The NewIFF39 code modules and examples based on iffparse.library are designed as replacements for the original EA IFF code. The object code modules (with source provided) contain many high-level support functions for reading and writing IFF files and clipboard data, and for loading, saving, and displaying ILBM files in a 1.3 through 3.0/AA compatible manner. In some modules, it has been possible to retain much of the original EA IFF code. However, most structures and most higher level function interfaces have changed.

On the plus side, these new modules contain many easy-to-use functions for querying, loading, displaying, and saving ILBMs. Modules similar to these have been used in-house at Commodore during the development of the Display program and several other ILBM applications. The screen.c module provides powerful display-opening functions which are 1.3-compatible yet provide a host of new options under 2.0 and 3.0 such as centered overscan screens, full-video display clips, border transparency control, and autoscroll. These 39.x releases of the NewIFF code also support V39 and the AA chipset for full 8-bit-per-gun color and AA modes. Modules are provided for printing (screendump) and for preserving or adding chunks (copychunks). And the 8SVX example now actually plays samples and instruments. In addition, clipboard support is automatic for all applications that use the IFFP modules because parse.c's openifile() interprets the filename -c[n] (i.e., "-c", "-c1", "-c2", etc.) as clipboard unit n.

All of the applications with the NewIFF code require iffparse.library which has been part of the OS since Workbench 2.0. Please note that the 2.04 Workbench version of iffparse.library is a 1.3-compatible library, and that all of the modules and examples (with the exception of Save8) have been designed to take advantage of 2.0/3.0, but also work under 1.3. Developers who wish to distribute iffparse.library on their commercial products may execute a 2.0 Workbench license, or may get an amendment to their 1.3 Workbench license to allow distribution of iffparse.library.

The NewIFF39 modules contain enhanced code in many areas to take advantage of 2.0 and 3.0 features while remaining functional under 1.3 (with 2.04 iffparse.library). In addition, great effort has been put into identifying and replacing code sections which were inflexible or not upwards compatible. The code implements 32-bit CAMG, display database aspect ratios, 8-bit CMAP from 4, 8, or 32-bit color data, mode fallback when an ILBM's display mode is not available, overscan centering (except under 1.3) based on the user's closest Preference setting with display clips properly constrained to maximum limits, setting of as many colors as the destination can handle, default detection and adjustment of old 4-bit left-shifted CMAP values, and built-in support for up to 24 planes of data plus a Mask plane. Note that currently, the 24 planes plus 1 mask plane limit is hardcoded, but we expect to make this more flexible in a future release of the code, perhaps also adding alpha channel support.

Through the use of tag-like arrays of desired chunk IDs, applications can control which chunks are gathered by the parsing module. It is also possible to clone chunks for rewriting and for applications to add chunks to written FORMs. Applications may also pass additional screen tags to the display-opening module (screen.c) and may control much of this module's default behavior through flags (see iffp/ilbmapp.h of NewIFF39).

Most of the high-level function pairs provided in these modules have been designed to provide safe cleanup for themselves. For example, a loadbrush() that succeeds or fails at any point can be cleaned up via unloadbrush. The cleanup routines null out the appropriate pointers so that allocations will not be freed twice.

All applications use the parse.c module. The basic steps for using the parse.c module are:

- ❑ Define tag-like arrays of your desired chunks (readers only)
- ❑ Allocate one or more [form]Info structures as defined in iffp/[form]app.h (for example an ILBMInfo defined in iffp/ilbmapp.h).
- ❑ Initialize the ParseInfo part of these structures to the desired chunk arrays, and to an IFFHandle allocated via iffparse AllocIFF().
- ❑ Use the provided high level load/save functions, or use the lower level parse.c openifile(), reader-only parseifile()/ getcontext()/nextcontext(), and closeifile(). The filename -c[n] may be used to read/write clipboard unit n.
- ❑ Clean up, FreeIFF(), and deallocate [form]Infos.

V39/AA Support

For NewIFF39, the ILBMInfo was extended to support a 32-bit-per-gun representation of the CMAP for use with the V39 color setting functions.

```
/* --- New --- */
WORD  *colorrecord;       /* Passed to LoadRGB32 (ncolors,firstreg,table) */
Color32 *colortable32;    /* 32-bit-per-gun representation of colors      */
ULONG crecsize;           /* Bytes allocated including colorrecord WORDs  */
```

For compatibility, the old style WORD array colortable of xRGB 4-bit values is still created for all loaded ILBMs. But the NewIFF39 code will also automatically allocate and create the 32-bit color table under V39 or higher, unless the calling application asks it not to by setting IFFPF_NOCOLOR32 in ILBMInfo->IFFPFlags:

```
/* Don't allocate or use a 32-bit-per-gun Color Table under V39 or above */
#define IFFPB_NOCOLOR32 0
#define IFFPF_NOCOLOR32 (1L << IFFPB_NOCOLOR32)
```

By default, the NewIFF39 code will examine the low nibbles of an ILBM.CMAP to determine if it is an old-style left-shifted 4-bit-per-gun CMAP, and if all of the low nibbles of all usable registers are zero, the code will assume a 4-bit CMAP and scale the 4-bit values properly to 32 bits. This CMAP examination is disabled if either the ILBM.BMHD->Flags contains the flags BMHDF_CMAPOK (1L << 7), or if the calling application passes the IFFPF_CMAPOK flag in ILBMInfo->IFFPFlags. In either of these cases, the 8-bit CMAP values will be accepted as-is, and scaled to 32 bits.

Since V37, the NewIFF code has supported saving of 8-bit-significant CMAP data, and the function interface will accept 4-bit, 8-bit, or 32-bit per gun color data.

Currently, the NewIFF code does not make use of V39 Datatypes, and therefore does not have the capability to load non-IFF file formats. You may wish to link with the NewIFF modules but also add conditional application code to make use of Datatypes when datatypes.library is present. This would allow your program to be backwards compatible while taking even greater advantage of V39.

Locale Support

The NewIFF39 code does not use locale.library as-is. However, the code is prepared for localization. A catalog file of all module strings is provided (iffp.cd), and the module include file iffpstrings.h is generated from this iffp.cd file using CatComp 39.x. All string handling for the modules has been centralized in modules/iffpstrings.c which includes comments regarding locale support. See the Locale.Readme in the NewIFF39 archive for additional tips on writing localized IFF applications.

Important Notes

❑ Clipboard and File Handles
Most of the higher-level load functions keep the IFFHandle (file or clipboard) open. While the handle is open, you may use parse.c functions (such as findpropdata) *or* direct iffparse functions (FindCollection(), FindProp(),) for accessing the gathered chunks. However, it is not a good idea to keep a filehandle *or* the clipboard open. While a clipboard unit is open, no other applications can clip to the unit. And while a file is open, you can't write the file back out. So, instead of keeping the file or unit open, you can use copychunks (in copychunks.c) to create a copy of your gathered chunks, and do an early closeifile() (parse.c). Then access and later write back out (if you wish) and deallocate your copied chunks via the routines in the copychunks module (findchunk, writechunklist, freechunklist).

❑ Complex Forms
The parse.c module will enter complex formats such as CATs, LISTs, and nested FORMs to find the FORM that you are interested in. This is great. However, if you are a read-modify-write program, you should warn your user when this occurs unless *you* are capable of recreating the complex format. Otherwise, your user may unknowingly destroy his complex file by writing over it with your program. Example - a paint program could read an ILBM out of a complex LIST containing pictures and music, and then save it back out as a simple ILBM, causing the user to lose his music and other pictures. To determine if a complex form was entered after a load, check the (form)Info.ParseInfo.hunt field. If TRUE (non-zero), then your file was found inside a complex format.

LIST OF IFFP MODULES AND APPLICATIONS

Note - Some useful functions are listed with each module See module source code for docs on each function. See application examples for usage.

Applications (these require linkage with modules - see Makefiles)	
ILBMDemo	Displays an ILBM, loads a brush, saves an ILBM, opt. print
ILBMLoad	Queries an ILBM and loads it into an existing screen
ILBMtoC	Outputs an ILBM as C source code
ILBMtoRaw	Converts an ILBM to raw plane/color file
RawtoILBM	Converts raw plane/color file (from ILBMtoRaw) to an ILBM
24bitDemo	Saves a simple 24-bit ILBM and then shows it 4 planes at a time (if given filename, just does the show part)
Play8SVX	Reads and plays an 8SVX sound effect or instrument - LoadSample, UnloadSample, PlaySample, OpenAudio, CloseAudio, and body load/unpack functions
PlaySMUS	Just a skeleton for a SMUS player - it loads the SMUS and samples.
ScreenSave	Save the front screen or viewport as an ILBM, with an icon
Save8	Create and save an 8-plane 256 color screen (V39/AA)

Other Examples (use iffparse.library directly and require no modules)	
Sift	Checks and prints outline of any IFF file (uses RAWSTEP)
ILBMScan	Prints out useful info about any ILBM
ClipFTXT	Demonstrates simply clipping of FTXT to/from clipboard apack.asm Dr. Gerald Hull's assembler replacement for packer.c

Also Provided	
Display	1.3 through 3.0 compatible ILBM display and slideshow program - supports AA.
Bumprev	updated version of bumprev revision file creation utility

General IFFParse Support Module

parse.c	File/clipboard I/O and general parsing - openifile, closeifile, parseifile, getcontext, nextcontext, contextis, currentchunkis, PutCk chunk writing function, and IFFerr text error routine
iffpstrings.c	Centralized string and message handling for modules.

ILBM Read Modules

loadilbm.c	High level ILBM load routines which are passed filenames (calls getbitmap) - loadbrush/unloadbrush, loadilbm/unloadilbm, and queryilbm
getbitmap.c	brush/bitmap loading (non-display, calls ilbmr.c) - createbrush/deletebrush, getbitmap/freebitmap
getdisplay.c	bitmap load/display (calls screen.c, ilbmr.c) - showilbm/unshowilbm, createdisplay/deletedisplay
screen.c	1.3/2.0/3.0 AA/ECS/non-ECS compatible screen/window module - opendisplay, openidscreen, modefallback, clipit
ilbmr.c	Lower level ILBM body/color load routines (calls unpacker.c) - loadbody, loadcmap, getcolors/freecolors, alloccolortable, getcamg (gets or creates modeid)
unpacker.c	BODY unpacker

ILBM Write Modules

saveilbm.c	High level ILBM saving routines which are passed filenames (calls ilbmw.c) - screensave and saveilbm
ilbmw.c	Lower level ILBM body/color save routines (calls packer.c) - InitBMHD, PutCMAP, PutBODY
packer.c	BODY packer

Extra Modules

copychunks.c	Chunk cloning and chunk list writing routines - copychunks, findchunk, writechunklist, freechunklist
screendump.c	Screen printing module (iffparse not required)
bmprintc.c	Module to output ILBM as C code

Include Files

iffp/#?.h	This subdirectory may be kept in your current directory or in your main include directory.

Thanks to Steve Walton for his code changes for Manx/SAS compatibility, and to Bill Barton and John Bittner for their comments and suggestions.

IFF FORM AND CHUNK REGISTRY

The following is an alphabetical list of registered FORMs, generic chunks (shown as (any).chunkname), and registered new chunks for existing FORMs (shown as formname.chunkname). The center column describes where additional information on the FORM or chunk may be found. Items marked "EA_IFF" are described in the main chapters of the EA IFF specs. Those marked "IFF_TP" are described in the third-party specications section. Items marked "propos" are proposals which have been submitted to CATS, some of which are private. And items marked with "------" are private or yet unreleased specifications. New chunks and FORMS should be registered with CATS US, IFF Registry, 1200 Wilson Drive, West Chester, PA. 19380. Please make all submissions on Amiga diskette and include your address, phone, fax.

(any).ANNO	EA_IFF	EA IFF 85 Generic Annotation chunk
(any).AUTH	EA_IFF	EA IFF 85 Generic Author chunk
(any).CHRS	EA_IFF	EA IFF 85 Generic character string chunk
(any).CSET.doc	IFF_TP	chunk for specifying character set
(any).FRED	—	Private ASDG global chunk.
(any).FVER.doc	IFF_TP	chunk for 2.0 VERSION string of an IFF file
(any).HLID.doc	IFF_TP	HotLink IDentification (Soft-Logik)
(any).INFO.proposa	propos	This chunk contains data usually found in a file's .info file.
(any).JUNK.doc	IFF_TP	Always ignore this chunk.
(any).NAME	EA_IFF	EA IFF 85 Generic Name of art, music, etc. chunk
(any).TEXT	EA_IFF	EA IFF 85 Generic unformatted ASCII text chunk
(any).(c)	EA_IFF	EA IFF 85 Generic Copyright text chunk
8SVX	EA_IFF	EA IFF 85 8-bit sound sample form
8SVX.CHAN.PAN.doc	IFF_TP	Stereo chunks for 8SVX form
8SVX.SEQN.FADE.doc	IFF_TP	Looping chunks for 8SVX form
ACBM.doc	IFF_TP	Amiga Contiguous Bitmap form
AHAM	—	unregistered (???)
AIFF.doc	IFF_TP	Audio 1-32 bit samples (Mac,AppleII,Synthia Pro)
ANBM.doc	IFF_TP	Animated bitmap form (Framer, Deluxe Video)
ANIM.brush.doc	IFF_TP	ANIM brush format
ANIM.doc	IFF_TP	Cel animation form
ANIM.op6.doc	IFF_TP	Stereo (3D) Animations
ANIM.op7	—	unregistered (???)
ARC.proposal	propos	archive format proposal (old)
ARES	—	unregistered (???)
ATXT	—	temporariliy reserved
AVCF.doc	IFF_TP	AmigaVision Course File
AVCO.doc	IFF_TP	AmigaVision COmmand (Nested in AVCF)
AVEV.doc	IFF_TP	AmigaVision EVent (Nested in AVCF)
BANK	—	Soundquest Editor/Librarian MIDI Sysex dump
BBSD	—	BBS Database, F.Patnaude,Jr., Phalanx Software
C100	—	Cloanto Italia private format
CAT	EA_IFF	EA IFF 85 group identifier
CHBM	—	Chunky bitmap (name reserved by Eric Lavitsky)
CLIP	—	CAT CLIP to hold various formats in clipboard
CMUS.proposal	propos	Common MUsical Score
CPFM	—	Cloanto Personal FontMaker (doc in their manual)

DCCL	—	DCCL - DCTV paint clip
DCPA	—	DCPA - DCTV paint palette
DCTV	—	DCTV - DCTV raw picture file
DECK	—	private format for Inovatronics CanDo
DEEP.doc	IFF_TP	Chunky pixel image files (Used in TV Paint)
DR2D.doc	IFF_TP	2-D Object standard format
DRAW	—	reserved by Jim Bayless, 12/90
DTYP.doc	IFF_TP	DataTypes Identification
EXEC.proposal	propos	Proposed FORM for executable (loadseg-able) code.
FANT.doc	IFF_TP	Fantavision movie format
FAX3	—	private GPSoftware FAX format, no longer used.
FAXX.GPHD.doc	IFF_TP	Additional header info for FAXX FORMs
FAXX.doc	IFF_TP	FAXX (Facsimile image FORM)
FIGR	—	Deluxe Video - reserved
FILM	—	LIST FILM - For storing ILBMs with interleaved 8SVX audio
FNTR	EA_IFF	EA IFF 85 reserved for raster font
FNTV	EA_IFF	EA IFF 85 reserved for vector font
FORM	EA_IFF	EA IFF 85 group identifier
FTXT	EA_IFF	EA IFF 85 formatted text form
GRYP.proposal	propos	byteplane storage proposal (copyrighted)
GSCR	EA_IFF	EA IFF 85 reserved gen. music score
GUI.proposal	propos	user interface storage proposal (private)
HEAD.doc	IFF_TP	Flow - New Horizons Software
ILBM	EA_IFF	EA IFF 85 raster bitmap form
ILBM.3DCM	—	reserved by Haitex
ILBM.3DPA	—	reserved by Haitex
ILBM.ASDG	—	private ASDG application chunk
ILBM.BHBA	—	private Photon Paint chunk (brushes)
ILBM.BHCP	—	private Photon Paint chunk (screens)
ILBM.BHSM	—	private Photon Paint chunk
ILBM.CLUT.doc	IFF_TP	Color Lookup Table chunk
ILBM.CMYK.doc	IFF_TP	Cyan, Magenta, Yellow, & Black color map (Soft-Logik)
ILBM.CNAM.doc	IFF_TP	Color naming chunk (Soft-Logik)
ILBM.CTBL.DYCP.doc	IFF_TP	Newtek Dynamic Ham color chunks
ILBM.DCTV	—	reserved
ILBM.DGVW	—	private Newtek DigiView chunk
ILBM.DPI.doc	IFF_TP	Dots per inch chunk
ILBM.DPPV.doc	IFF_TP	DPaint perspective chunk (EA)
ILBM.DRNG.doc	IFF_TP	DPaint IV enhanced color cycle chunk (EA)
ILBM.EPSF.doc	IFF_TP	Encapsulated Postscript chunk
ILBM.PCHG.doc	IFF_TP	Line by line palette control information (Sebastiano Vigna)
ILBM.PRVW.proposal	propos	A mini duplicate ILBM used for preview (Gary Bonham)
ILBM.TMAP	—	Transparency map (temporarily reserved)
ILBM.VTAG.proposal	propos	Viewmode tags chunk suggestion
ILBM.XBMI.doc	IFF_TP	eXtended BitMap Information (Soft-Logik)
ILBM.XSSL.doc	IFF_TP	Identifier chunk for 3d X-Specs image. (Haitex)
IOBJ	—	reserved by Seven Seas Software
IODK	—	reserved for Jean-Marc Porchet at Merging Technologies
ITRF	—	reserved
JMOV	—	Reserved for Merging Technologies
LIST	EA_IFF	EA IFF 85 group identifier
MFAX	—	Reserved for TKR GmbH & Co.
MIDI	—	Circum Design
MOVI	—	LIST MOVI - private format

MSCX	---	private Music-X format
MSMP	---	temporarily reserved
MTRX.doc	IFF_TP	Numerical data storage (MathVision - Seven Seas)
NSEQ	---	Numerical sequence (Stockhausen GmbH)
OB3D.proposal	propos	Proposal for a stadard 3D object format
OCMP	EA_IFF	EA IFF 85 reserved computer prop
OCPU	EA_IFF	EA IFF 85 reserved processor prop
OPGM	EA_IFF	EA IFF 85 reserved program prop
OSN	EA_IFF	EA IFF 85 reserved serial num prop
PGTB.doc	IFF_TP	Program traceback (SAS Institute)
PICS	EA_IFF	EA IFF 85 reserved Macintosh picture
PLBM	EA_IFF	EA IFF 85 reserved obsolete name
PMBC.proposal	propos	Reserved for Black Belt Systems 91.12.01
PREF	---	Reserved by Commodore for user preferences data, currently private.
PROP	EA_IFF	EA IFF 85 group identifier
PRSP.doc	IFF_TP	DPaint IV perspective move form (EA)
PTCH	---	Patch file format (SAS Institute)
PTXT	---	temporarily reserved
RGB4	---	4-bit RGB (format not available)
RGBN-RGB8.doc	IFF_TP	RGB image forms, Turbo Silver (Impulse)
RGBX	---	temporarily reserved
ROXN	---	private animation form
SAMP.doc	IFF_TP	Sampled sound format
SC3D	---	private scene format (Sculpt-3D)
SHAK	---	private Shakespeare format
SHO1	---	Reserved by Gary Bonham (private)
SHOW	---	Reserved by Gary Bonham (private)
SMUS	EA_IFF	EA IFF 85 simple music score form
SPLT.doc	IFF_TP	ASDG's File SPLiTting system
SSRE	---	Reserved for Merging Technologies 92.05.04
SWRT	---	unregistered (???)
SYTH	---	SoundQuest Master Librarian MIDI System driver
TCDE	---	reserved by Merging Technologies
TDDD.doc	IFF_TP	3-D rendering data, Turbo Silver (Impulse)
TERM	---	unregistered (???)
TREE.doc	IFF_TP	Storage of arbitrary data structures as trees (or nested lists).
TRKR.proposal	propos	TRacKeR style music module format proposal
UNAM	EA_IFF	EA IFF 85 reserved user name prop
USCR	EA_IFF	EA IFF 85 reserved Uhuru score
UVOX	EA_IFF	EA IFF 85 reserved Uhuru Mac voice
VDEO	---	private Deluxe Video format
WORD.doc	IFF_TP	ProWrite document format (New Horizons)
YUVN.doc	IFF_TP	For storage of Y:U:V image data (MacroSystem)

◆

3.0 AmigaGuide™

by David N. Junod

Introduction

The standard Amiga keyboard sports a HELP key, yet there has been no system provided support for this key. Now, there is AmigaGuide, which provides a standard method of displaying help and other on-line documentation to the user.

Sections marked with ** indicate that the option is only available in pre-3.0 versions of AmigaGuide.

Capabilities

AmigaGuide uses an Intuition window that contains a scroll bar, buttons and pull-down menus, to display plain ASCII text files or AmigaGuide databases.

An AmigaGuide database is a set of related documents contained in one file. Each document may contain references to other documents, using what is called a link. A document may contain any number of links, pointing to any number of other documents. When the user selects a link, the document that the link points to will be displayed. The user may then use the links to read through the database, following whatever path he may choose. The technical term for AmigaGuide's abilities is hypertext.

The user may at any time print a document or a portion of the document. **He may also send portions of a document to the clipboard, for use in other applications.

Using ARexx, the user may write scripts, or an application could provide scripts, to control AmigaGuide.

**Cross-reference tables can be loaded that specify where a keyword, or phrase is defined. The user can then use AmigaGuide's Find Document facility to quickly display a document based on keyword, without having to know the name of the database that it is located in.

AmigaGuide provides a unique feature to hypertext systems, called Dynamic Nodes. A Dynamic Node is a hypertext or plain text document that is generated in real-time as opposed to coming from a static file. An application that generates Dynamic Nodes is called a Dynamic Node Host.

Interfacing

AmigaGuide databases are accessed in three different ways:

- ❑ Databases can be browsed directly from the Workbench or Shell using the MultiView utility.
- ❑ AmigaGuide support can be added to an existing application that supports ARexx by using AmigaGuide's ARexx function host capabilities.
- ❑ Applications can use the functions of AmigaGuide to provide help on gadgets, menus and windows. For example, the user could position the pointer over any gadget or menu item, press help, and the appropriate document would be displayed in the AmigaGuide window. The application could also have AmigaGuide display a pertinent portion of the current project.

Other Uses

In addition to help or on-line documentation, AmigaGuide has other possible uses.

Tutorials

An application that has an ARexx port and supports AmigaGuide could set up a help system that not only provides help, but also gives examples. The user could read about a feature and click on the EXAMPLE button, which would run an ARexx script that would give an example of use. For instance, to show Pattern Fill, the script could draw a circle, select a pattern, and then fill the circle.

Computer Aided Instruction

The student could read about different topics, following links. A multiple choice quiz could be set up at the end where the questions and answers run ARexx scripts to accumulate the score.

Program by Query

Many programmers develop using a Cut & Paste technique. They clip modules from various applications or utilities they have written and paste them together to build new applications. A database of these different code fragments could be set up (such as loading and saving ILBMs, playing sounds, etc.) and you could step through, answering questions, while the sections you need are being appended to a new source file.

USER PREFERENCES

AmigaGuide allows a number of items to be tailored to the user's preference. These preference items are stored in environment variables. The AmigaDOS command SetEnv can be used to set any of these variables.

In order to set any of the following environment variables, an ENV:AmigaGuide directory must be made.

```
makedir ENV:AmigaGuide
```

A Preferences Editor that sets AmigaGuide preferences would write in the ENV:AmigaGuide directory when "Use" is selected, and write in the ENVARC:AmigaGuide directory when "Save" is selected.

Following is a list of the variable names, and what they control.

Path

This variable contains the list of directory names that AmigaGuide will search through when it attempts to open a database. The directory names are separated by a space.

```
SetEnv AmigaGuide/Path "Workbench:Autodocs Workbench:Includes"
```

**Pens

This variable provides the user with the ability to specify the colors to use for the various renderings that AmigaGuide performs.

```
SetEnv AmigaGuide/Pens <abcdefgh>
```

AmigaGuide Pens

a = Background pen	b = Button text pen
c = Button background pen	d = Highlighted button text pen
e = Highlighted button background pen	f = Outline pen
g = Highlight outline pen	h = Text on background pen

```
SetEnv AmigaGuide/Pens 21213001
```

Internally, AmigaGuide subtracts "0" from the pen number, so values can range from 0 to 207.

**Text
Used to specify the graphical style that the links are presented in. The possible styles are:

BUTTON	Draw a raised border around the text (default).
UNDERLINE	Underline the text.
BOLD	Bold the text.
ITALIC	Italicize the text.

```
SetEnv AmigaGuide/Text BUTTON
```

Authoring AmigaGuide Documents

Authoring an AmigaGuide database, or any hypertext database for that matter, is a difficult task. It takes a lot of insight into the subject matter and how the pieces relate to each other. A database must consist of documents that are related. Documents must be broken into manageable chunks, and links carefully thought out. A document should consist of information dealing with one topic and should contain links to other related documents.

An AmigaGuide database is ASCII text with embedded commands that tell AmigaGuide how to interpret the database. A database should consist of a main table of contents and a number of related documents.

Label Commands

These are commands that can be used within a database. Commands must start in the first column of a line. If a line begins with an @ sign, then it is interpreted as a command.

@DATABASE <name>
Must be the very first line of an AmigaGuide document.

@MASTER <path>
Complete path of the source document used to define this AmigaGuide database.

@AUTHOR <name>
The author of the database.

@(C) <copyright>
The copyright notice for the database.

@$VER: <AmigaDOS version string>
Specify the version of the database. This command must always be uppercase.

@FONT <name> <size>
The font to use for the database.
@INDEX <name/node>
The name of the index node, which will be accessed by the "Index" button. Can be a node in an external database.
@HELP <name/node>
The name of the help node, which will be accessed by the "Help" button. Can be a node in an external database.
@NODE <name> <title>
Indicate the start of a node (page/article/section). The first node, or main node, must be named MAIN. MAIN must be the master table of contents for the database.
@DNODE <name>
Indicates the start of a dynamic node. The AmigaGuide system uses the callback hooks to obtain the document from a document provider.
@WIDTH <chars>
How wide, in characters, the largest document is.
@HEIGHT <chars>
How high, in characters, the largest document is.
## <remark> @REMARK <remark>
Remark (not displayed to the user).

Node Label Commands

These are commands that can be used within an @NODE.

@ENDNODE <name>
Indicate the end of a node.
@TITLE <title>
Title to display in the title bar of the window during the display of this node.
@TOC <node name>
Name of the node that contains the table of contents for this node. Defaults to MAIN. This is the node that is displayed when the user presses the "Contents" button.
@PREV <node name>
Node to display when the user selects "< Browse"
@NEXT <node name>
Node to display when the user selects "Browse >"

@FONT <name> <size>

Specify the font to use for the node.

@{<label> <command>}

Indicate a textual link point. Can be anywhere in a line. Starting with 3.0, AmigaGuide can can link to graphics, sounds, animations and other DataTypes.

\@

A backslash in front of the @ sign is used to escape it.

Attributes

The following list of attributes can be applied to the text within a node. This feature is only supported with the 3.0 version of AmigaGuide.

@{B}

Turn bold on.

@{UB}

Turn bold off.

@{I}

Turn italic on.

@{UI}

Turn italic off.

@{U}

Turn underline on.

@{UU}

Turn underline off.

@{FG <color>}

Change the foreground text color. Color can be:

Text	Shine
Shadow	Fill
FillText	Background
Highlight	

@{bg <color>}

Change the background color. The same colors can be used as in the FG command.

Action Commands

These are commands that can be assigned to a link point.

****ALINK <name> <line>**
Load the named node into a new window, with <line> at the top of the display.

BEEP
Cause a display beep in the screen that the AmigaGuide window resides in.

CLOSE
Close the window (should only be used on windows that where started with alink).

LINK <name> <line>
Load the named node, with <line> at the top of the display.

RX <command>
Execute an ARexx macro.

RXS <command>
Execute an ARexx string file. To display a picture, use 'ADDRESS COMMAND DISPLAY <picture name>', to display a text file 'ADDRESS COMMAND MORE <doc>'. Note that with 3.0 it is possible to display text, graphics or sounds within the AmigaGuide window.

SYSTEM <command>
Execute an AmigaDOS command.

QUIT
Shutdown the current database.

Example AmigaGuide Database

The following is an example of an AmigaGuide database. It doesn't contain any 'useful' information, but it does show the usage of some of the commands.

```
@database "example.guide"
@master "example.doc"

@node Main "Example AmigaGuide database"

Table of Contents
@{"ARexx" link ARexx}
@{"Shell" link Shell}
@{"Workbench" link Workbench}
@endnode

@node ARexx
Put something here about  @{b}ARexx@{ub}.
```

```
@endnode

@node Shell
Put something here about the @{b}Shell@{ub}.
@endnode

@node Workbench
Put something here about @{b}Workbench@{ub}. Say that it has @{"icons" link icon}.
@endnode

@icon "Workbench Icons"
Those little pictures that you can drag around.
@endnode
```

AREXX SCRIPTS

It is possible to control AmigaGuide using ARexx. Each occurrence of AmigaGuide has an ARexx port. The AmigaGuide shared system library is also an ARexx function host.

Port Naming

The default port name is AMIGAGUIDE.# where # is the occurrence. With the **AmigaGuide utility, a port name can be specified as a command line argument. An application with an AmigaGuide interface can also provide the port name.

ARexx Commands

Any of the following action commands are also ARexx commands. All commands are not case-sensitive.

****ALINK <name> <line>**

Load the named node into a new window, with <line> at the top of the display.

BEEP

Cause a display beep in the screen that the AmigaGuide window resides in.

CLOSE

Close the window (should only be used on windows that were started with alink).

LINK <name> <line>

Load the named node, with <line> at the top of the display.

SYSTEM <command>

Execute an AmigaDOS command.

QUIT

Shutdown the current database.

ARexx Functions

The amigaguide.library is an ARexx function library. The library can be added as a function host with the following lines:

```
/* Load the AmigaGuide library as a function host */
IF ~SHOW('L','amigaguide.library') THEN
   CALL ADDLIB('amigaguide.library',0,-30)
```

It supports the following functions (function names are not case-sensitive).

ShowNode PUBSCREEN/K,DATABASE/K,NODE/K,LINE/N
  Display a node on the named screen. Defaults to the Main node on the Workbench screen. If DATABASE isn't specified, then will search through the cross-reference list to get the database name.

LoadXRef NAME/K
  Load a cross-reference file into memory.

GetXRef NODE/K
  Return information on NODE. Format of the text string returned is "NODE" "DATABASE" TYPE LINE.

ExpungeXRef
  Flush the cross-reference list from memory.

Adding an AmigaGuide Interface to Your Application

Applications can add AmigaGuide support using the functions within the amigaguide.library.

The AGHelp example on disk illustrates how to add simple context sensitive help to an application. It uses the new 3.0 Intuition gadget help routines to determine which gadget the pointer is over, and uses AmigaGuide to display the help text.

The AdvAGHelp example on disk shows a more complicated context sensitive help system. Like AGHelp, it uses the new 3.0 Intuition functions, but also shows:

Continuous Help
  Display help information as the user moves the mouse around over the objects of the user interface.

Project Information
  Using current project information or user interface state information (such as a gadget or menu being disabled) in your help documents.

Cross Reference Files

AmigaGuide allows cross-reference tables to be loaded that specify what document a keyword is defined in. **This cross-reference table is used by the "Find Document" requester to locate a node. It is also used by the AD2AG utility to construct hypertext versions of the system Autodoc files.

A cross-reference file follows a layout similar to the devs:mountlist format. The table itself starts with a line that consists of the keyword XREF: and ends with a line that contains a # as the only uncommented character. Comments can be included in C-style format, beginning with "/*" and ending with "*/".

```
/* This is a comment */
XREF:
   ...    "Gadget"  "intuition/intuition.h"        215 3
   ...
#
```

A cross-reference entry consists of four words:

Keyword
  The keyword that is being defined.
File
  The ASCII file or database that the keyword is defined in.
Line
  The line within the node that the keyword is defined on.
Type
  This field indicates the type of keyword. Possible values are.

0	Generic AmigaGuide link.	1	Describes a function.
2	Describes a command.	3	Points to an include file.
4	Describes a macro.	5	Describes a structure.
6	Describes a structure field.	7	Describes a type definition.
8	Describes a define.		

Loading a Cross Reference List

A global cross-reference list can be loaded from disk using the LoadXRef() function. The format is.

```
LONG success;
BPTR lock;
STRPTR name;

success = LoadXRef(lock, name);
```

The arguments are

lock
Lock on the directory where the file is located. May be NULL.
name
Name of the cross-reference file to load. LoadXRef will search the user preference path.

It returns

-1
Indicates that the load was aborted by a Ctrl-C.
0
Unable to load the file.
1
Successfully loaded the file.
2
No changes have been made since the last time that this file was loaded.

Access to the Cross Reference List

An application can use the GetAmigaGuideAttr() function to obtain a pointer to the cross-reference list. The application then may search through the list, or even save the list to disk. Note that access to this list is read-only, and must be enclosed between a call to LockAmigaGuideBase() and UnlockAmigaGuideBase().

```
struct List *list;
LONG key;

/* Lock the AmigaGuideBase for exclusive access */
key = LockAmigaGuideBase(NULL);

/* Get a pointer to the cross-reference list */
if (GetAmigaGuideAttr(AGA_XRefList, NULL, &list))
    {
    /* Do something with the list */
    }

/* Unlock AmigaGuideBase */
UnlockAmigaGuideBase(key);
```

A cross-reference list consists of nodes of struct XRef, defined in <libraries/amigaguide.h>.

```
/* Cross-reference node */
struct XRef     {
   struct Node xr_Node;     /* Embedded node */
   UWORD xr_Pad;            /* Padding */
   struct DocFile *xr_DF;   /* (Private) Document defined in */
   STRPTR xr_File;          /* Name of document file */
   STRPTR xr_Name;          /* Name of item */
   LONG xr_Line;            /* Line defined at */
   };

#define   XRSIZE (sizeof (struct XRef))
```

Following are the field definitions.

xr_Node
  Embedded node structure. xr_Node.ln_Name points to xr_Name.
  xr_Node.ln_Type contains the type of the keyword.

xr_Pad
  Used to align the remaining fields.

xr_DF
  Private pointer.

xr_File
  Pointer to the name of the file that xr_Name is defined in.

xr_Name
  Pointer to the keyword.

xr_Line
  The line, within xr_File, that xr_Name is defined on.

Dynamic Node Host

AmigaGuide provides a unique feature to hypertext systems called Dynamic Nodes. A Dynamic Node is a hypertext or plain text document that is generated in real-time as opposed to coming from a static file. An application that generates Dynamic Nodes is called a Dynamic Node Host.

If a link point within a document isn't resolved, it will query a list of Dynamic Node Hosts to see if any one of these external applications can resolve the node. This feature allows for dynamic interaction with constantly changing data. It is useful for AmigaGuide authoring tools, interactive development environments and extremely context sensitive help systems, to name a few.

Dynamic Nodes has been implemented using an Object Oriented Programming paradigm. When a link point hasn't been resolved an HM_FINDNODE message is sent to each Dynamic Node Host on the list. Once the node has been found, an HM_OPENNODE is sent to the Dynamic Node Host that the node belongs to. HM_CLOSENODE is sent to the host once the node is exited.

Initializing a Dynamic Node Host

In order for an application to register itself as a Dynamic Node Host, it must initialize a hook and add the hook to the AmigaGuide Dynamic Node list, using the AddAmigaGuideHost()

The hook structure as defined in <utility/hooks.h>.

```
/* Standard hook structure */
struct Hook     {
   struct MinNode h_MinNode;
   ULONG (*h_Entry)();     /* assembler entry point */
   ULONG (*h_SubEntry)(); /* often HLL entry point */
   VOID *h_Data;           /* owner specific */
   };
```

The AddAmigaGuideHost() function returns a pointer to an AmigaGuideHost structure. This structure, defined in <libraries/amigaguide.h>, is as follows.

```
/* Callback handle */
struct AmigaGuideHost {
   struct Hook agh_Dispatcher;          /* Dispatcher */
   ULONG agh_Reserved;                  /* Must be 0 */
   ULONG agh_Flags;
   ULONG agh_UseCnt;                    /* Number of open nodes */
   APTR agh_SystemData;                 /* Reserved for system use */
   APTR agh_UserData;                   /* Anything you want... */
   };
```

Following are the field definitions for the AmigaGuideHost structure.

agh_Dispatcher
   This is a copy of the Hook that was passed to AddAmigaGuideHost().

agh_UserData
   Can be manipulated by the Dynamic Node Host any way it sees fit.

The other fields are not to be manipulated in any way.

Removing a Dynamic Node Host

A Dynamic Node Host is removed using the RemoveAmigaGuideHost() library function. The application must successfully remove the hook before exiting, otherwise AmigaGuide would end up calling the hook function, that has been unloaded from the system, causing a system crash.

The following code fragment illustrates how to initialize and remove a Dynamic Node Host.

```
#include <exec/types.h>
#include <libraries/amigaguide.h>
#include <clib/exec_protos.h>
#include <clib/amigaguide_protos.h>
#include <pragmas/exec_pragmas.h>
#include <pragmas/amigaguide_pragmas.h>
#include <stdio.h>

extern struct Library *SysBase, *DOSBase;
struct Library *AmigaGuideBase;

#define   ASM __asm
#define   REG(x) register __ ## x

ULONG __saveds dispatchDNH(struct Hook *, STRPTR, Msg);
ULONG ASM hookEntry(REG(a0) struct Hook *,REG(a2) VOID *,REG(a1) VOID *);

/* Callback hook dispatcher */
ULONG __asm hookEntry (
        REG(a0) struct Hook *h,
        REG(a2) VOID *obj,
        REG(a1) VOID *msg)
{
/* Pass the parameters on the stack */
    return ((h->h_SubEntry)(h, obj, msg));
}

main (int argc, char **argv)
{
struct Hook hook;
AMIGAGUIDEHOST hh;

/* amigaguide.library works with 1.3 and newer versions of the OS */
if (AmigaGuideBase = OpenLibrary ("amigaguide.library", 33))
    {       /* Initialize the hook */
    hook.h_Entry = hookEntry;
    hook.h_SubEntry = dispatchDNH;

    /* Add the AmigaGuideHost to the system */
    if (hh = AddAmigaGuideHost (&hook, "ExampleHost", NULL))
        {
        printf("Added AmigaGuideHost 0x%lx", hh);
        /* Wait until we're told to quit */
        Wait (SIGBREAKF_CTRL_C);

        printf ("Remove AmigaGuideHost 0x%lx", hh);

        /* Try removing the host */
        while (RemoveAmigaGuideHost (hh, NULL) > 0)
            {          /* Wait a while */
            printf (".");
            Delay (250);
            }
        printf ("");
        }
        else
        {
        printf ("Couldn't add AmigaGuideHost");
        }

    /* close the library */
    CloseLibrary (AmigaGuideBase);
    }
}
```

Handling Dynamic Node Host Messages

Once the Dynamic Node Host has been added to AmigaGuide, it can start receiving messages for different requests.

Currently, AmigaGuide supports the following methods, or message types, for a Dynamic Node Host.

HM_FINDNODE
: When AmigaGuide can't resolve a link, then it sends an HM_FINDNODE message to all Dynamic Node Hosts to see which host defines the node.

HM_OPENNODE
: Once AmigaGuide locates the host that defines a node, using the HM_FINDNODE message, then the HM_OPENNODE message is sent to that host to ask it to open the node.

HM_CLOSENODE
: Once the user has closed all occurrences of a Dynamic Node,then AmigaGuide sends the HM_CLOSENODE message to the host that opened the node.

HM_EXPUNGE
: AmigaGuide sends this message to all Dynamic Node Hosts when the Expunge vector of amigaguide.library is invoked, or the ExpungeDataBases() function is called.

Several of the methods receive a TagItem array as an argument. Currently the following tags are supported. The tag values are defined in <libraries/amigaguide.h>.

HTNA_Screen
A pointer to the screen on which source AmigaGuide window resides.

HTNA_Pens
The pen array associated with the screen.

HTNA_Rectangle
A Rectangle structure (defined in <graphics/gfx.h>) containing the dimensions of the window.

HTNA_HelpGroup
A unique numeric identifier associated with an AmigaGuide client. This tag is new for 3.0.

Each method requires one or more parameters. The MethodID is the only common parameter for each method.

HM_FINDNODE

Used to locate the Dynamic Node Host that a node is defined by. When a Dynamic Node Host receives a HM_FINDNODE message for a node that it owns, it should reply with TRUE, otherwise it must respond with FALSE.

The HM_FINDNODE method receives the following arguments:

```
/* HM_FINDNODE */
struct opFindHost      {
    ULONG MethodID;
    struct TagItem *ofh_Attrs; /*  R: Additional attributes */
    STRPTR ofh_Node;           /*  R: Name of node */
    STRPTR ofh_TOC;            /*  W: Table of Contents */
    STRPTR ofh_Title;          /*  W: Title to give to the node */
    STRPTR ofh_Next;           /*  W: Next node to browse to */
    STRPTR ofh_Prev;           /*  W: Previous node to browse to */
};
```

The field definitions are as follows

ofh_Attrs
This field contains a pointer to a TagItem array of attributes for the message. This field is read-only.

ofh_Node

The name of the node to open. This field is read-only. It is possible for this name to contain parameters that need to be parsed. For example, the command that triggered the link could have been:

```
Link "snd/beep 320"
```

In which case, the ofh_Node field would contain:

```
beep 320
```

ofh_TOC

The Table of Contents to assign to this node. This is the name of the node to link to, if the ''Contents'' button is pressed. This field can be written to (not implemented).

ofh_Title

The title to assigned to this node. This field can be written to.

ofh_Next

The name of the logical next node. This is the name of the node to link to if the ''Browse >'' button is pressed. This field can be written to.

ofh_Prev

The name of the logical previous node. This is the name of the node to link to if the ''< Browse'' button is pressed. This field can be written to.

HM_OPENNODE

Once AmigaGuide locates the host that defines a node, using the HM_FINDNODE message, then the HM_OPENNODE message is sent to that host to ask it to open the node. If the Dynamic Node Host is able to open the node, then it should respond with TRUE, otherwise respond with FALSE.

The HM_OPENNODE method receives the following arguments:

```
/* HM_OPENNODE, HM_CLOSENODE */
struct opNodeIO    {
    ULONG MethodID;
    struct TagItem *onm_Attrs; /*  R: Additional attributes */
    STRPTR onm_Node;     /*  R: Node name and arguments */
    STRPTR onm_FileName;    /*  W: File name buffer */
    STRPTR onm_DocBuffer;   /*  W: Node buffer */
    ULONG onm_BuffLen;      /*  W: Size of buffer */
    ULONG onm_Flags;        /* RW: Control flags */
);
```

The field definitions are as follows

onm_Attrs

This field contains a pointer to a TagItem array of attributes for the message. This field is read-only.

onm_Node

The name of the node to open. This field is read-only. It is possible for this name to contain parameters that need to be parsed. For example, the command that triggered the link could have been:

```
        Link "snd/beep 320"
```

In which case, the onm_Node field would contain:

```
        beep 320
```

onm_FileName

If you want AmigaGuide to read a particular node from disk, then supply the file name here. The file can either be a straight ASCII file or an AmigaGuide document (not a database). The application can write to this field.

onm_DocBuffer

If you are dynamically creating a node in memory, then use this field to point to the buffer. If this field is used, then the onm_BuffLen field must be filled in also. The application is in charge of freeing onm_DocBuffer when it is done (indicated by a HM_CLOSENODE message). The application can write to this field.

onm_BuffLen

The length of the buffer that onm_DocBuffer points to. The application can write to this field.

onm_Flags

These are control flags that the Dynamic Node Host can set.

HTNF_KEEP

Don't flush this node from memory until the database is closed. This will delay the HM_CLOSENODE message until the database is closed.

HTNF_ASCII

The node is straight ASCII, doesn't contain any AmigaGuide keywords.

HTNF_CLEAN

Remove the node from the database as soon as it is closed.

HTNF_DONE

This flag is used to indicate to AmigaGuide that the Dynamic Node Host already took care of presenting the node, and that there is no need for AmigaGuide to present it. This is useful for playing sounds, animations, or even debugging information.

HM_CLOSENODE

Once the user has closed all occurrences of a Dynamic Node, then AmigaGuide sends the HM_CLOSENODE message to the host that opened the node. If the Dynamic Node Host is able to close the node, then respond with TRUE, otherwise respond with FALSE.

The HM_CLOSENODE message uses the same message structure as HM_OPENNODE.

HM_EXPUNGE

AmigaGuide sends this message to all Dynamic Node Hosts when the Expunge vector of amigaguide.library is invoked, or the ExpungeDataBases() function is called. The Dynamic Node Host should free as much memory as it possibly can.

The HM_EXPUNGE method receives the following arguments:

```
/* HM_EXPUNGE */
struct opExpungeNode {
   ULONG MethodID;
   struct TagItem *oen_Attrs; /*  R: Additional attributes */
};
```

The field definitions are as follows

oen_Attrs
   Currently, no attributes passed.

Message Dispatcher

The following is an example of Dynamic Node Host message dispatcher. Since this is executed with a callback hook, it is being run on the calling process' task, not the Dynamic Node Host process. Because of that, it is necessary to load the global data segment using geta4() or __saveds.

```
ULONG __saveds
dispatchAmigaGuideHost (struct Hook *h, STRPTR db, Msg msg)
{
struct opNodeIO *onm = (struct opNodeIO *) msg;
ULONG retval = 0;

switch (msg->MethodID)
   {      /* Does this node belong to you? */
   case HM_FINDNODE:
    {
    struct opFindHost *ofh = (struct opFindHost *) msg;

    DB (kprintf ("Find [%s] in %s", ofh->ofh_Node, db));

    /* See if they want to find our table of contents */
    if ((stricmp (ofh->ofh_Node, "main")) == 0)
        {
        /* Return TRUE to indicate that it's your node, otherwise return FALSE. */
        retval = TRUE;
        }
    else
        {
        Display (onm); /* Display the name of the node */

        /* Return TRUE to indicate that it's your node, otherwise return FALSE. */
        retval = FALSE;
        }
    }
    break;

/* Open a node. */
case HM_OPENNODE:
    DB (kprintf ("Open [%s] in %s", ofh->onm_Node, db));
/* See if they want to display our table of contents */
if ((stricmp (onm->onm_Node, "main")) == 0)
    {
    /* Provide the contents of the node */
    onm->onm_DocBuffer = TEMP_NODE;
    onm->onm_BuffLen = strlen (TEMP_NODE);
    }
else
    {
    Display(onm);     /* Display the name of the node */

/* Indicate that we want the node removed from our database, and that we handled
 * the display of the node
 */
onm->onm_Flags |= (HTNF_CLEAN | HTNF_DONE);
    }

/* Indicate that we were able to open the node */
retval = TRUE;
break;

/* Close a node that has no users. */
case HM_CLOSENODE:
    DB (kprintf("Close[%s] in %s", onm->onm_Node, db));

/* Indicate that we were able to close the node */
retval = TRUE;
break;

/* Free any extra memory */
```

```
    case HM_EXPUNGE:
    DB (kprintf ("Expunge [%s]", db));
    break;

    default:
    DB (kprintf ("Unknown method %ld",msg->MethodID));
    break;
    }
return (retval);
}
```

Developer Specific Utilities

On the DevCon disks are a number of utilities for AmigaGuide that would be of special interest to the developer.

AD2AG

Scans Autodocs for function and command names, scans the INCLUDE: directory for .h include files, and scans the include files for structure definitions and typedefs. Also parses Autodoc files and constructs a corresponding AmigaGuide database. It resolves links to functions, commands, include files, structures and typedefs.

Glossary

Autodoc
Documentation extracted from source code.

Browse
Navigate sequentially through a series of documents, instead of via links.

Cross-reference table
A table that consists of the following information:

Keyword	A word or phrase
Database	Name of the database that the keyword is defined in.
Line	The line that the keyword is defined on within the database. This only applies if the database is a straight text file (such as an include file).
Type	What type the keyword is, such as a ''normal'', function, command, include file, or structure.

Database
A file that consists of multiple documents.

Document
A block of text, constrained to one subject. Also called a node.

Dynamic Node
A Dynamic Node is a hypertext, or plain text, document that is generated in real-time, or from live data, as opposed to coming from a static file.

Dynamic Node Host
An application that generates Dynamic Nodes.

Link
A word, or phrase, within a document that is linked to another document.

Node
A block of text, constrained to one subject. Also called a document.

Retrace
To follow, in a reverse direction, the path taken through a series of documents.

Table of Contents
A list of documents, categorized by type.

Recommended Reading

Ben Shneiderman & Greg Kearsley, *Hypertext Hands-On!*, Addison-Wesley Publishing Company, ISBN 0-201-13546-9

Philip Seyer, *Understanding Hypertext Concepts and Applications*, Windcrest Books, ISBN 0-8306-9108-1 (hardcover) 0-8306-3308-1 (paperback)

◆