Amiga User Interface Directions

by Martin Taillefer

The Amiga's user interface is the most palpable aspect of the system. Users view the Amiga through Intuition. It is therefore critical that Intuition, and its ancillary components, continue to grow and become more functional and easier to operate.

This document outlines the major enhancements planned to the user interface. We wish to show the general direction in which we are heading. There are no guarantees that any of the material presented below is actually going to appear in the operating system.

Goals

There are two primary goals to current user interface developments:

❑ **Increased knowledge reuse**
A key motivating factor behind the original design of graphical user-interfaces was to increase the consistency and integration of personal computing environments. The refinement of the UI must engender more consistent applications, with a higher level of integration.

❑ **Shorter development cycles**
Simplifying and abstracting the API can substantially reduce application development time and development costs. This in turn promotes more and higher quality applications to be created.

The focus of UI development is on high user-benefit items. What can we do in the operating system that directly improves the usefulness and attractiveness of the system to current and new users? What can we do to make the system look like the right answer to the user's problems? This is why much attention is devoted to small details that have often been overlooked in the OS developments.

For example, putting a stronger emphasis on the graphical nature of the operating system. Although this has little direct impact on the usability of the software, it substantially improves its apparent quality and polish, which is an important consideration in the marketplace.

Low-Level User Interface Components

Windows

Windows are the heart of the Amiga's user interface. Their appearance, responsiveness, behavior, and manageability, determine in large part how the system looks and feels to a user.

Appearance

Amiga windows are fairly elegant, although there are a few details that could improve the overall look of things:

❑ **Variable Thickness Borders**

Except for the top border, window borders are fixed in thickness. Depending on the aspect ratio of the current screen, this might be adequate, or might make the window look bad. The current thin borders are a problem in high resolution modes where they become hard to see. The border dimensions of windows should adapt to the aspect ratio of the current screen, and should vary in width based on resolution. This also includes making system gadgets wider or taller based on the resolution.

Another issue with window borders is the presence of the very wide and often useless border needed when a sizing gadget is in place in a window. This is a big waste of screen real-estate on low-resolution displays. It is also a factor of confusion as there is no visual distinction between the drag bar and the blank useless borders. The blank borders can be removed by using an alternate sizing method (see the next section).

❑ **Better Looking Titles**

Along with variably-sized borders should come the repositioning of the window title text. The text currently touches the top border of the window, which looks quite bad and has often been reported as a bug in the system.

❑ **Consistent 3D Effect**

The current rendering for window borders does not create a consistent 3D effect. There are some rendering bugs that defeat the 3D effect.

❑ **Better Looking Window Movement**

The look of the user interface would greatly improved if windows were to follow the mouse pointer as they are being moved, instead of simply an outline of the windows. Whether this is actually implemented depends a lot on the performance attainable.

❑ **Better Looking System Requesters**
A mechanism similar to some PD utilities (ARQ) can be added to the system, further enhancing the graphical appearance of the UI. DOS requesters can automatically start using standard glyphs in them to indicate ''disk full'', or ''please insert disk'', and the EasyRequest() function can be extended to accept glyphs.

Behavior And Manageability

Windows are central to the operation of the system. Their generality and functionality should be maximized in order to remove as many hurdles as possible from the user attempting to navigate through the system. The windowing system should be a transparent environment that doesn't call attention to itself. It should do its job, not impose limits to annoy the user, and generally stay out of the way to allow real work to be done.

A few critical improvements can be made to our windowing system to substantially increase the usability of the Amiga as a whole.

First and foremost is the restructuring of the depth arrangement strategy. The current scheme in place since Release 2 has serious flaws. The main problem came with trying to shoehorn window zooming into old applications. This required the replacement of the dual depth arrangement gadgets by a single multi-function depth gadget. It is now very hard to depth arrange windows in a predictable way. It often takes multiple clicks and a lot of trial and error in order to get windows in the order you want them to be. Basically, the windowing system gets in the way.

Most competing systems do not have explicit user control over window depth arrangement. The active window is always brought to the front. We believe that user control over depth arrangement is preferable, and required for compatibility.

Certain things can be done to help in depth arrangement:

❑ **Revert To 1.x Depth Gadgets**
The original dual depth gadget scheme should be restored. It provided for consistent and determinate behavior and was easily understood.

❑ **Frontdrop Windows**
Certain utilities, or components within larger applications, require that specific windows always remain in front of other windows. For example, a clock program might want to hover in front of all other windows. Or a tool palette inside of a multi-window paint program would want to stay in front

of all active canvases. Frontdrop windows would be another stratum of windows, along the same lines as current backdrop and regular windows. This stratum would be on top of regular windows. Although general prioritized depth families could solve this problem in a general manner, such a scheme is likely overkill.

An important feature is the ability to resize a window from any of its corners or sides. Resizing a window currently involves locating and revealing the sizing gadget. Most window borders are not used for anything except as delimiters. Overloading them with the ability to resize the window provides a very nice increase in flexibility.

Another feature is the support of windows extending beyond the limits of their parent screen. Given the fact many Amigas are used on low-res displays, it is quite important to allow windows to be dragged off the sides of the display. It is much more important to do this on a system with a small number of pixels than it is on a system with a megapixel display.

Off-screen windows are also central to supporting the new concept of window iconification. The Zoom gadget introduced in Release 2 is a user interface failure. It has unpredictable behavior. Beyond that, what it does is just not that useful. The ability to iconify a window however is much more desirable from a user-standpoint, and can be substituted for the current Zoom gadget's functionality.

Clicking the Zoom gadget of a window will cause the system to move that window to a totally off-screen position. An icon representing that application will be displayed on the same screen the window was on. When the user double-clicks on the icon, the system removes the icon and brings back the window to its original position.

New applications can ask to be notified of iconification events so they can free resources. For example, a paint program can free pens that it had allocated, or a serial program can free the serial port for use by other software.

Screens will also be iconifiable. Basically, every window on the screen will get iconified, and then the screen is closed and turned into an icon on the Workbench screen. There are compatibility issues such as applications caching screen pointers. If Intuition cannot be sure that the screen owner and all visitors are aware of this new feature, the screen will be delinked though the screen memory would not be freed.

Mouse Pointer

There are three main issues involving the mouse pointer.

The most important issue concerns the tendency for the mouse pointer to freeze during lengthy Intuition operations such as layer handling, or when Intuition is blocked waiting for a semaphore to unlock. This property gives a very bad feel to the system. Anytime a window is opened, moved, sized, or closed, the whole system apparently freezes for a few seconds. The mouse stops moving and just sits there. On displays with many bitplanes, many windows, and no fast ram, the wait can be well over one second, sometimes close to five or six seconds (an A1200 running an 8 bitplane DblNTSC Workbench shows this). With all that said, preliminary investigations show that correcting this problem can best be described as extremely difficult, if not impossible.

The second pointer issue involves the inclusion of standard mouse pointer imagery for common tasks. For example:

- ❑ **Crosshair**
- ❑ **I-beam**
- ❑ **Object Selection Mode**
- ❑ **+ symbol for range demarcation**

The third issue involves pointer positioning from the keyboard. The current scheme is very difficult to use. A simpler and more functional approach is to use the left Amiga key qualifier and the numeric keypad keys for directional control of the pointer. Amiga-0 would be the equivalent of a left button click, and Amiga-Enter would be the equivalent of the right mouse button. Consideration in this scheme must also be given to adequately handle "sticky keys" (described later in the Preferences section). Specifically, it must be possible to extend the stickiness of the left Amiga key to cover multiple mouse moves in one press.

Gadgets

This section presents a series of new gadget classes aimed at improving the functionality of our low-level system controls. These classes are often reimplementations of existing code, but with an eye towards making them more extensible, flexible, enjoyable to use, and more enjoyable to look at.

GadTools Objects

Although GadTools provided the framework needed to support our UI look, its many inherent limitations often become problems that application writers have to work around:

❑ **No Relativity**
GadTools objects don't support any form of positioning relativity, making them difficult to use in sizable windows.

❑ **Not Extensible**
GadTools objects do not support inheritance, making them impossible to extend from within an application. You get what you get, nothing more.

❑ **Not Changeable**
Once a GadTools object is created, it is impossible to change many of its static attributes such as its label or font. The only way of accomplishing these tasks is to remove all gadgets from the window and recreate them from scratch.

❑ **Not Connectable**
GadTools objects do not support any form of object interconnections. This prevents the creation of ''self-driven'' user interfaces, where the application merely sets things up, and lets the UI objects do all the work by themselves.

BOOPSI objects have none of the problems denoted above. They are as easy to use and access as GadTools objects are, and quite a bit more flexible.

With this in mind, the obvious course to follow is to create a series of BOOPSI classes that parallel in functionality the existing GadTools objects. The new objects would not have the old problems and would add missing functionality along the way. Once the new classes are in place, GadTools can become one of their clients. The main reason behind this move would be to reduce the amount of redundant code in ROM. It also guarantees that the new classes provide all of GadTools' functionality, in a compatible and consistent manner.

The following sections go through the various new or modified BOOPSI classes, explaining their basic functionality. Many of these classes are direct supersets of GadTools gadget kinds, with extensions to address the needs of a more graphical and font-sensitive user interface.

One of the key enhancements that affects almost all gadget classes is the use of the label.image class for the rendering of labels and other components of various gadgets. The labelling class enables every gadget type to support multi-line labels with embedded graphics. This makes the system much easier to localize and make font-sensitive.

Another enhancement to all classes is support for font-sensitive layout, where the gadgets can be queried for various types of information about themselves, such as their preferred size, or minimum size.

Finally, system classes that are intended for subclassing, such as gadgetclass, will be rounded out to be more flexible and powerful. We intend on providing better documentation and clearer guidelines on subclassing system classes to create your own classes, while retaining future compatibility with super- class enhancements.

buttongclass

This class already exists and should be extended with:

- ❑ **New Look Borders**
  The class needs to be able to automatically render a border around a gadget's hit box. To give a bit more finesse to the look of these relatively boring gadgets, rounded corners could replace the current square corners.
- ❑ **Variable Repeat Rate**
  The class needs to adapt its repeat rate based on a user preference setting.

checkbox.gadget

Checkbox gadgets are fairly intuitive and easy to understand. A few issues need to be addressed:

- ❑ **Larger Clickable Region**
  Checkboxes present fairly small targets to mousers. This becomes a severe problem on large displays with small rendering. It is also a problem for handicapped users. Release 3 provides scalable checkmarks which helps a lot. The obvious way to further extend the clickable region is to listen for clicks on the checkbox's label and treat them as clicks on the checkbox itself.

There are a few open issues with checkbox gadgets:

- ❑ How would multi-line text labels look next to checkboxes? Would they also require a box around them to demark them clearly?
- ❑ Since it is likely that glyphs and multi-line labels will require some form of boxing, then would it be wise to always put a box around checkbox labels? This could also be used to increase the clickable region of this gadget type.
- ❑ The current rendering for the unselected state of a checkbox is simply an empty box. Is this an adequate cue to the user?

radiobutton.gadget

Radio buttons provide an attractive alternative to cycle gadgets and listviews. They tend to consume more screen real estate and more keyboard shortcuts than cycle gadgets and listviews. Nevertheless, radio buttons should be used when possible because they are the most intuitive type of gadget for the job: they clearly indicate all possible choices, and their rendering increases the graphical content of a display. A few issues should be addressed to make this type of gadget more palatable:

- ❑ **Strumming**
  The user should be able to strum the mouse across the items of a radio button group. This would increase consistency with other gadget types.

- ❑ **Larger Clickable Region**
  Radio buttons exhibit the same problem as checkboxes, they also present fairly small targets to mousers. As with checkboxes, the real solution is to listen for clicks on the button's label and treat them as clicks on the button itself.

- ❑ **Delimitation**
  Also like checkboxes, radio buttons are currently fairly unbounded. This becomes a problem when multiple columns of radio buttons are created. Different grouping schemes can be used to help make things clearer.

- ❑ **Disabling Individual Buttons**
  Individual buttons of a radio button group can be disabled and enabled independently.

popup.gadget

Cycle gadgets offer a nice compact alternative to radio buttons. However, the fact that not all available choices are visible at once is nonintuitive. The solution to this problem is the replacement of cycle gadgets with popup gadgets.

Clicking and holding down a popup gadget brings up a popup menu. You can then move the mouse within the menu. Releasing the left mouse button while on top of an item makes that item the current value of the popup gadget. Clicking the right mouse button while the menu is up cancels the gadget operation and closes the menu.

A point of contention with popup gadgets is whether they should retain the cycling nature of their ancestor the cycle gadget. A possibility is to keep the cycling behavior if the user clicks in a specific area of the gadget and bring up a pop up menu if the user clicks elsewhere in the gadget. Experiments will be conducted to find out whether this behavior is useful, tolerable, and desirable.

listview.gadget

Listviews offer a flexible concept for presenting variable lists of items. GadTools listviews however, have always been limited. This restricts their use, and prevents the easy realization of many viable user interfaces. Some features can greatly enhance their usefulness:

- ❑ **Multi-Selection**
  Multi-selection is the biggest feature missing in current listviews. This ability is needed, for example, in the ASL file requester. ASL must currently implement its own version of listviews because of this.

- ❑ **Drag Selection**
  Along with multi-selection comes drag selection. A group of items can be selected by dragging the mouse over these items while holding down the left mouse button.

- ❑ **Automatic Double-Click Detection**
  Many listviews benefit from directly supporting double-click selection of items. Programmers forget this useful feature. Providing direct support for the detection of double-clicks would encourage use of this. Double-click support is useful as a shortcut for experienced users, making the system more ''upwardly mobile''.

- ❑ **Horizontal Lists**
  It should be possible to create listviews that scroll data horizontally. This helps in the creation of font-sensitive layouts.

- ❑ **Multi-Column Display**
  Many lists contain multiple fields of information for each item in the list. Consider for example a file requester displaying a file's name, size, and creation date. Support for multi-column displays means that programmers suddenly have the convenience to create listviews with multiple columns of information, without concern for proportional fonts or inter-column clipping. It also means that the user can optionally be given control over the size and even existence of any of the columns of information. By dragging a separator bar, the user can easily allocate more space for filenames, and less for file dates. This is also a major issue in the ASL file requester.

- ❑ **Secondary Scroll Bars**
  In better support of large fonts, it is necessary to support scrollers to let the user move from left to right in a vertical scrolling list, or up and down in a horizontal one. This lets a user push some columns of information outside the visible area of a list, but still able to access them via the scroller.

scroller.gadget

Scroller gadgets are seldom used by themselves, and are mostly used as components in listviews. A few improvements help make them more useful:

- ❑ **Border Support**
  Scroller gadgets work in window borders and are able to be embedded in them. This is useful in many applications. It standardizes issues such as repeating speed of the scroll arrows.

- ❑ **Better Looking Arrows**
  The arrows currently in use in scroller gadgets are not very attractive and stand out in a world of snazzy looking 3D gadgetry. Arrow imagery should be taken from sysiclass. sysiclass will need to acquire freely scalable arrows instead of the currently limited arrows.

- ❑ **Right Mouse Button Cancellation**
  Clicking the right mouse button while one of these gadgets is activated terminates the activation and restores the scroller to its original value.

slider.gadget

The main improvements for slider gadgets will be:

- ❑ **Enhanced Imagery**
  The imagery should be more evocative than a simple black box. This would improve the look of the UI. It would also serve to distinguish sliders from scrollers. These have a very different purpose in life, this fact is not clear to the user since both gadget types currently look so much alike. Options of the new imagery include tick marks, and track filling to the left of (or below) the knob.

- ❑ **Repeating Container**
  Clicking and holding the mouse in the container area of a slider gadget should cause the slider to enter repeat mode and gradually move towards the mouse.

textentry.gadget

Text gadget s are used by most applications and are the most limited type of system gadget. Many enhancements are needed:

- ❑ **Multi_line Support**
  The biggest limitation of current text gadgets is their inability to handle multi-line data.

This new class will support multi-line editing and will include the use of optional scrollers to enter large amounts of data. It will also offer automatic word-wrapping, which is an important feature to make multi-line editing easier.

❑ **Text Editor Mode**

The multi-line support enables the simplification of a common UI composite object. The listview style in Workbench's Information window is a fairly complex gadget. It has a listview, a text gadget underneath it, and an associated "New" and "Del" pair of buttons. Operation of this composite object is fairly complex and unnatural. To edit a string, you must click on it within the listview area, but type the text into the text gadget way at the bottom of the listview. Deleting items requires selecting an item, which puts it into the text gadget, and then clicking the Del gadget.

Instead of the above composite listview object, a multi-line text gadget in non word-wrap mode can be used. To edit a line, you click on it, and start typing right on the place where you clicked. Text is not word wrapped and causes the gadget to scroll towards the left as you add more text.

❑ **Cut And Paste**

Clipboard cut and paste will be done in the expected manner.

❑ **Variable Cursor Styles**

The cursor style will be under preference control. Vertical bar or block cursors, blinking or not.

❑ **Standard Editing Rules**

Text entry gadgets will offer a complete set of editing abilities, with clearly defined behavior. Applications such as word processors are encouraged to adopt the same style. This includes standard keyboard controls, standard word selection algorithm, scrolling behavior, etc.

❑ **Templates**

This feature will allow a template to be provided to the text gadget, which describes the format of the data the user is allowed to enter. This is the main use programmers have for the current text gadget edit hook, provided with a much simpler interface. A template is composed of commands and literal text. The commands represent fields where data can be entered, while the string literal are displayed as is by the text gadget. Templates would allow easy creation of phone number gadgets, or ZIP code gadgets, or floating-point gadgets.

- ❑ **Deselection Detection**
  A bug that occurs in many applications is when the contents of a text gadget is only inspected whenever an IDCMP_GADGETUP event is received. Since it is possible for a user to modify the contents of a gadget without generating such a message, it is possible to have an application miss changes made to the text. IDCMP_GADGETOFF, discussed below, is the answer to this problem.
- ❑ **Optional History**
  Text entry gadgets will be able to optionally remember their previous contents thus providing automatic history.

filereq.gadget

Over the years, there have been many requests for the ability to add an application's own custom gadgets to the standard file requester. This would not be a very flexible approach, limiting the possible growth of the file requester. A much better approach is to create a BOOPSI class which encapsulates the logic of the file requester.

The file requester class will allow the display of any given named directory. The class implements the file requester's scrolling list (or possibly two lists, one for directories and one for files), and implements the file requester's three text gadgets. A client of the class, such as ASL, would simply embed the object within its window and send it messages asking it to display various directories, show the parent directory, show the volume list, etc. The file requester object would communicate selections made by the user back to the client. The client will be responsible for the control buttons at the bottom of the current ASL file requester (OK/Parent/Volumes/Cancel).

fontreq.gadget

This gadget type will exist for the same reasons the file requester class will. The class will offer the font and size lists, and optional style, pens, and rendering mode controls. The ASL requester's sample section, as well as the control buttons, fall within the client's domain, and not in the class itself.

screenmodereq.gadget

Follows the same concept as filereq.gadget and fontreq.gadget. The screen mode-related controls are in the class, and control gadgets remain in the client. Due to the vastly increased number of modes available under RTG, a different approach will be used to allow mode selection. The current flat list of modes becomes quite difficult to use when the system can display hundreds of modes. The new approach will be criteria-based, where the user can ask for a display that is 60Hz, 640x480 in 16 colors, and the system will figure out what is the best mode to display that.

colorselector.gadget

This class will provide the functionality to implement a standard color requester. It will be built-up from other object types including the colorwheel and the gradient slider.

printreq.gadget

This class will provide the functionality to implement a standard print requester within an application. It will allow per-printer options, and finally offer a consistent printer interface to the user. By providing printer-specific options, a printer driver could be written that controls a FAX. The printer-specific options would then allow control of items such as the destination phone number, header page, etc.

dragger.gadget

This class will allow the creation of draggable gadgets (icons). Draggable icons provide a very clear and elegant solution to many UI problems. They are unfortunately quite difficult to implement using the current system software, so almost no applications make use of them.

The dragger class will allow you to define an object on screen that the user can move around. Various features will allow control of the object behavior. For example, object motion can be restricted to within the current window, within windows of the same group, or allowed to go to windows of other applications.

calendar.gadget

This class will be in support of system tools such as Time prefs or Agenda. The class will provide a standard mechanism for the selection and display of dates. Features of this class will include:

- ❑ Renders a single page of a calendar
- ❑ User can click to select one or multiple days in the month
- ❑ Calendar pages can be made read-only
- ❑ Individual days can be ghosted or rendered in a different color
- ❑ Localization issues are handled transparently

pager.gadget

Some standard support needs to exist to create gadgets allowing the user to flip between various option pages from within a single window. ''Option pages'' means a display like PrinterPS which lets the user alternate between different sections of the available settings.

The technique currently in use in PrinterPS and Palette prefs involves a cycle gadget. This has the unfortunate effect that users think the current value of the cycle gadget is in fact an attribute maintained by the prefs editor, instead of a means to access additional settings. For example, the cycle gadget in Palette prefs contains two states: ''4 Color Settings'' and ''Multicolor Settings''. Users are under the impression that the cycle gadget determines whether the system is in ''4 color mode'' or ''multicolor mode''. What the gadget really means is that the 4 color settings or the multicolor settings are currently being displayed by the program. The setting of the gadget is not a preferences item, merely a control of the prefs editor.

To avoid the confusion, a different gadget style will be created to support this type of concept. The gadget will operate in a manner similar to tabs in a book. By clicking on the proper tab, you bring that page of the program on screen.

Font-Sensitive Layout

In order to completely support font-sensitive gadget layouts, the various system classes must be modified to understand a few new features. The main feature is one that asks each gadget what is its minimum size. This functionality is required in order to do true font-sensitive layout. Since only the gadgets themselves can know this information, they need to supply it to a client trying to use them in a UI.

Another benefit of this technique is that gadgets can easily get bigger when new functionality is needed. A gadget can grow and simply report its minimum size as larger. Using this method, gadgets can adapt to different screen resolutions and render differently (specifically, the thickness of the gadget borders can be variable to suit different aspect ratios)

Keyboard Control

Support for keyboard control of gadgets is currently very limited. Many things can be done by applications to fake it, some things are difficult, and some are impossible. Keyboard control is an important feature for power-users. Once again, this is a case of upward mobility of the system software. Keyboard control makes the system much faster to operate for experienced users, while not getting in the way of novice users.

We have already adopted the standard notation of an underlined letter to indicate the keyboard shortcut for a gadget. We are lacking adequate support to implement the standard's functionality correctly. Intuition doesn't have a very fine keyboard input focus. Our current interface has two possible targets to keyboard events: the active text gadget or the active window. This loose input focus can make certain things harder to do. For example, if there are two listviews in a window, which listview gets scrolled when the cursor keys are hit?

The UI style guide offers some guidance in the area of keyboard control, but there is a problem. A substantial amount of work is required on the part of application writers to implement proper keyboard control. And certain things are even impossible to implement legally (highlighting a button gadget programmatically). Current system support is limited to underlining a character within a gadget's label. That's not enough. We need to:

- ❑ Enable classes to inspect their labels to determine what is their keyboard shortcut. If no label is given, an explicit tag could be passed by the application giving the key to watch for.

- ❑ Enable classes to look at keyboard input as it is generated, and act in consequence. The key sequence would be swallowed and a message sent to the application in a manner similar to a direct gadget click.

With the above functionality in place, it becomes easy to set up keyboard shortcuts for an application's gadgets. Localization of these is also easy, as only the label of the gadget needs to change to have the keyboard shortcuts change with it. Input handling for the shortcuts is automatically done by the gadget classes, in the manner that best suits them. For example, a scroller would use its key shortcut to increase its value, while the same shortcut with the SHIFT key down would decrease it. The colorwheel class could use multiple qualifier to control the direction of its knob. Etc.

The style guide currently recommends using regular keystrokes as shortcuts for gadgets. This works best for windows that have no (or few) text entry areas. The left Amiga key can provide a stateless method of activating gadgets.

Images

The creation of sophisticated graphical and font-sensitive UI displays has always been a difficult task. Creating glyphs that are compact, quickly rendered, scalable, and correctly color mapped, is such a hassle that application developers generally don't bother.

Increasing the graphical appearance of the system can be made somewhat easier with the addition of a few basic BOOPSI image classes. The classes attempt to simplify the creation of font-sensitive displays containing graphics, as well as improve consistency of the UI.

label.image

A generic labelling class which can be used on its own, and is used by all gadgets classes for the rendering of their labels. The features of the labelling class will be:

- ❑ **Stand-Alone Or Linked Use**
  Label images will be able to be used on their own in much the same way TEXT_KIND gadgets are today. They can also be used as labels to gadget types.
- ❑ **Multi-Line Labels**
  Supports the creation of multi-line labels by embedding line-feeds within the label. Localization often generates much longer strings than the original English ones, and the ability to transparently split the text onto multiple lines greatly improves the appearance of both the original and localized version of an application.
- ❑ **Automatic Word-Wrapping**
  Also in support of localization, automatic word-wrapping helps UI layout tremendously. The application provides a box and some text, and the class ensures that everything fits within it.
- ❑ **Glyph Layout**
  It is important that any text label be able to incorporate glyphs. The mini layout engine needed to do the word-wrapping also handles embedded images automatically.
- ❑ **Keyboard Shortcut Indicator**
  The class supports underlining a single character within a label, to denote a keyboard shortcut.

drawlist.image

This class allows the creation of simple graphics in a resolution independent manner. The client defines the imagery in terms of commands in an abstract coordinate system, and the class arranges to scale everything to the requested pixel size upon rendering.

glyph.image

This class provides a series of predefined system glyphs rendered using the drawlist class. These glyphs would be used throughout the system and would become quickly familiar to users. This further increases consistency of the UI. Standard glyphs could include:

- ❑ Warning symbol
- ❑ Fatal Error
- ❑ VCR control symbols (Play, Stop, Pause, etc.)
- ❑ Help symbol
- ❑ "For your information" symbol
- ❑ Key top symbol (Function key, Help key, Alt, Ctrl, etc.)

fuelgauge.image

This class will be used by any applications needing to show the progress of a task. The class will offer standard rendering for a fuel gauge, with the following features:

- ❑ Horizontal or vertical orientation
- ❑ Optional tick marks below or to the right the gauge
- ❑ Optional milestone indicators (0, 50, 100%) rendered below or to the right of the gauge.
- ❑ Optional current percentage done indicator rendered to the left or right of the gauge.

frameiclass

This class needs to learn about two new features:

❑ **Titled Frames**
The frames created by this class should be able to have a title embedded in the top line of the frame. This would support the rendering of radio button borders, and is generally useful to separate a window into gadget groups.

❑ **Rounded Corners**
Frames with rounded corners are needed to support the new look for button borders.

Menus

The Amiga benefits from a flexible menuing system which has always had the ability to remain hidden and out of the way of the user. This is of great value on low-resolution displays. However, menus do require some work to better support higher resolution displays and add general functionality.

One of the points to watch for in a user interface is the minimization of mouse travel to accomplish common tasks. The current menuing system can lead to very large amounts of mouse travel on large displays, since the menu strip always appears at the top of the display. In addition, the horizontal organization of the menu strip requires a full horizontal sweep of the display to access all menus.

Both problems can be solved by making menu headers appear in a stack under the mouse pointer. This avoids the need to move the mouse to the top of the display to view the menus, and significantly reduces required mouse travel to scan through all the menus.

Additional functionality becomes possible with this new menu organization:

- ❑ **Moving Menus**
  While the user is holding down the mouse button over a menu page, if he moves the mouse over the ''move menu'' section of the menu, and presses the left mouse button, the menu starts to follow the mouse. When the left mouse button is released, the menu is dropped in place and menuing operations resume as normal.

- ❑ **Nailing Menus**
  By moving the mouse over the ''nail'' icon in a menu and clicking the left mouse button, a menu can be turned into a window. This window is managed totally by the system. Menu items are converted into appropriate gadget types and selections made in this window are converted to corresponding menu selections and sent to the application. The ability to nail a menu to the screen enables the user to easily display often used commands on the screen where they become instantly available with a single mouse click.

- ❑ **Amiga Menu**
  The main menu page for an application contains an ''Amiga'' menu, which provides a handy place for the system to add standard functionality to existing applications. Moving the mouse to the checkmark glyph reveals the Amiga menu. It contains standard commands including a list of common tools the user might want to start (Calculator, Agenda, etc.). This is similar to the current Tools menu in Workbench, but is much more general. The Amiga menu enables the easy addition of new items managed by the system. For example, a selection to bring up a list of running applications, or a list of available public screens, etc.

- ❑ **Keyboard Navigation**
  Keyboard navigation of the menus is another feature which becomes easier with the new menu organization. A standard keyboard sequence is entered

which causes the main menu page to be displayed. The menu can be cancelled by pressing Esc. The selected item is moved around by using the cursor keys. Pressing RETURN either brings up a submenu, or selects an item.

❑ **Visually Distinctive Mutually Exclusive Selections**
There needs to be special rendering to identify mutually exclusive menu items. They currently share the imagery of checkable items, which is misleading to the user.

Another important feature to improve menus involves submenu delays. Intuition would interpret high-speed mouse travel as meaning ''don't change the state of which menu panels are up and which aren't''. This would allow the user to ''cut the corner'' when heading towards a submenu, without fear of accidentally triggering some other submenu along the way.

Preferences

The ability to customize the work environment is one of the big features of modern user interfaces. It gives users an important sense of being in control. It allows them to become more productive as the computer can automatically adapt to their tastes, and not the opposite.

The Amiga has always had a rich variety of controllable attributes. In some respects, it has too many. One of the purposes of this section is to review the different preferences choices that we currently have, and see what additional functionality is needed. Another purpose is to find ways to increase the graphical contents of the preferences editors. Careful consideration is given to avoid overwhelming the user with too many options.

Fonts Prefs

Beyond an interface update, Font prefs needs to support the selection of more font types. The selection is currently very limited and doesn't allow adequate selections to cater to high resolution displays in a pleasing manner. The new font selections include:

❑ Window Title Font

❑ Screen Title Font

❑ Menu Font

❑ UI Font

❑ Shell Font

Keyboard Prefs

This is a new program which provides half of the functionality previously contained in the Input prefs editor.

❑ **Integrated KeyShow**
This shows the layout of the different keymaps as they are being selected. It eliminates KeyShow as a stand-alone utility.

Mouse Prefs

This is a new program which provides the other half of the functionality previously contained in the Input prefs editor.

❑ **Swap Mouse Buttons**
Will let the user swap the functionality of the mouse buttons. This makes the system feel more natural to new left handed users. The selection button can then always be under the user's index finger, where it belong.

❑ **Sticky Keys**
Sticky keys is a feature which makes it possible for users with hand coordination problems, or users with one or no functioning hand, to use a keyboard. Sticky keys causes qualifier keys such as SHIFT or ALT to be typed in separately from regular keys. A user can then generate an "A" by pressing the SHIFT key, release it, and press the A key.

❑ **Cursor Selection**
This will allow a specific cursor style to be chosen for use in text gadgets, consoles. and applications. Choices include block and bar cursors, blinking or static, and the blink rate.

❑ **Mouse Blanking**
The MouseBlanker commodity program will be replaced by a checkbox option in Mouse prefs. When turned on, the mouse pointer is blanker whenever a key is entered. This will eliminate a Workbench program and will mainstream this useful functionality.

Palette Prefs

To complete the implementation of the V39 palette preferences scheme, more pens need to be added to the system:

❑ Text Gadget Colors
❑ Screen Title Bar Color
❑ Cursor Color

Locale Prefs

The individual parameters currently controlled by the country selection in Locale prefs will become individually controlled. This will let users pick exactly which date or number format to use, instead of forcing per-country selections. The current country selections will become presets that automatically adjust all the editable fields to match the exact specifications for each country.

WBPattern Prefs

WBPattern should support centering and tiling of its backdrop pictures. This would improve the appearance and usefulness of using smaller pictures. It would also look better when switching the screen mode of the Workbench screen, without changing the picture.

Sound Prefs

Multiple types of beeps should be supported, each indicating something different:

- ❑ Warning
- ❑ Fatal Error
- ❑ Attention
- ❑ Task Completion

The different sounds give important feedback to the user when something happens.

Window Prefs

A new preferences program offering the following options:

❑ **AutoPoint Integration**
AutoPoint is a feature many users like. The current implementation is less than ideal as it is external to the windowing system. It also consumes a fair amount of memory and CPU time. Integrating autopoint functionality directly in Intuition eliminates memory requirements, and reduces additional CPU time consumed to almost nothing. Putting the option in the prefs editor also eliminates an obscure program from the system disks.

❑ **ClickToFront Integration**
For much the same reasons as integrating AutoPoint in Intuition, it is wise to also integrate ClickToFront.

❑ **System Requester Positioning**

Many users want to have system requesters appear in locations other than in the top left of the display. The user could choose to have the requesters positioned in any of the screen corners, have them centered, or have them appear under the mouse.

❑ **Border Control**

Control can be provided to alter the size of window borders.

ScreenMode Prefs

The current method for specifying screen modes is not intuitive. The variety of modes is overwhelming, and the current organization doesn't help the user understand the selection he is about to make. The future holds many more modes in support of AAA graphics, so a new approach must be devised to simplify the selection of modes names. The technique used in the ScreenMode preferences can also be directly applied to the ASL ScreenMode requester which will benefit applications.

Monitor Prefs

Monitor prefs will be a replacement for Overscan prefs. It will be functionally the same, with a few changes.

The major change will be the ability to control the aspect ratio for the various scan rates. This could be done by adding an extra gadget to the main window of the form ''Edit Aspect Ratio'', which brings up a screen where the user could graphically view and edit the aspect ratio of the scan rate.

Printer, PrinterGfx, and PrinterPS Prefs

Printer and PrinterGfx would greatly benefit from a face lift to give them the same basic interface as PrinterPS has. The ability to graphically edit the print attributes makes print control substantially easier. The current method used is very terse, complicated, and almost impossible to fully understand without having a manual in arm's reach.

These three program will also be merged into a single entity, which will use the pager.gadget to flip between the different option pages.

Pointer Prefs

Pointer prefs will be extended to support editing of the various pointer types added to the system's pointer class.

Default Preferences

The default set of preferences shipped with the system will be examined and updated if appropriate. Of specific interest are things that make the system look better in its default configuration. Possibilities include:

- ❑ Use of a proportional font as default
- ❑ Use of a higher resolution mode as default
- ❑ Use of more colors by default
- ❑ Use of a default Workbench backdrop pattern
- ❑ Use of centered system requesters

Programmability

There are a large number of subtleties in the Amiga's API. A few things can be done to substantially reduce programmer exposure to these subtleties. Following is a brief summary of the miscellaneous enhancements planned that will simplify the lives of programmers.

Windows

❑ **RastPort Clipping**

Two new features of future versions of graphics.library are RastPort-based clipping and RastPort-specific rendering origins. Once combined, these two features offer a lot to the Intuition programmer. Functionality similar to GimmeZeroZero windows can be provided using these features, without suffering all the performance problems of traditional GZZ windows. It also becomes possible to create separate panes within a window, each pane providing its own clipping, and its own rendering origin.

❑ **Smarter Window Refreshing**

Simple refresh layers could be made smarter. They would work mostly like current smart refresh windows, but would have the ability to dispose of allocated off-screen buffers when a memory panic occurs in the system. The reason for this window type is to enhance the performance of the windowing system. When possible, the very fast smart refresh algorithm is used, but if a memory failure occurs, the window behaves like if it was old style simple refresh.

❑ **Window Limit Specifications**
The ability to specify that the minimum and maximum dimensions of a window are to be interpreted as meaning "dimensions of the inner window region" would be of great help. These are currently very difficult values to set up correctly.

❑ **Window Fonts**
The ability to specify fonts to use in the window's title bar, and in the window's interior would add useful functionality and avoid much programmer work.

❑ **Window Ports**
Large applications frequently take advantage of the ability to share a single UserPort among several windows, but this adds some arcane complexity and ugliness to the code. A WA_UserPort tag would clean this up nicely, as well as mark that window in need of CloseWindowSafely() style processing inside Intuition.

❑ **Window Menus**
The ability to specify a menu strip to use in a window when the window is opened and the ability to close a window without having to call ClearMenuStrip() are two simple enhancements to make life simpler.

❑ **Window Event Buffering**
There is a need to buffer events that occur in a window prior to the window being completely ready to accept input. For example, when users bring up the file requester via keyboard shortcut, they start typing in their filename immediately after hitting the shortcut key. The problem is that the file requester is not ready to accept input yet. The result being the first few characters entered by the user for the filename are lost.

It is also fairly tricky to activate a string gadget in a window that is opening. Although this is better in V39, problems still exist. The application has to wait to receive an "IDCMP_ACTIVEWINDOW" event before it should try to activate the gadget. This is too much voodoo.

A way to solve both problems is to introduce a new OpenWindow() tag that means "this is the gadget that should be activated". Intuition would then buffer up any input events that occur while the window is in process of opening, and would send them all in one big gulp to the window when it is ready to receive them. It would also auto-activate the appropriate gadget, to make sure any keyboard sequences that are intended for it actually reach it.

❑ **HideWindow()/ExposeWindow() Functions**

These functions would move the given window totally off-screen, and would bring it back to its original location. The ExposeWindow() function can be used in concert with the ability to open a window in the hidden state. This lets an application completely render its window imagery, and once done rendered, blast it onto the screen in one blit by using ExposeWindow(). Not only will rendering in the window go faster because of the reduced number of clipping rectangles, it will also look a lot nicer to the user.

❑ **ToolBox Windows**

ToolBox windows provide services to other windows. For example, a strip of tools shared across many document windows in an application. Intuition can help coordinate the toolbox window with respect to its parent.

❑ **Unified Windows and Requesters**

This concept means having a window with the basic desirable properties currently only available through a requester: locking of parent window (either as a new property of some windows or through a LockWindow() call), and some kind of parent/child arrangement for depth/size.

❑ **Better support for autoscroll screens**

We can think of a few ways to make the use of autoscroll screens more convenient, including autoscrolling before the mouse has hit the very edge of the display, and automatic scrolling of the screen to bring newly opened windows into view.

❑ **Screen locking**

This is basically an Intuition-friendly LockLayers() function. The advantage over LockLayers() is that it would be safe to continue to transact with Intuition without fear of deadlocking. This function would be used by programs which need to do trans-window rendering such as custom pop-up menus or draggable icons.

Gadgets and Images

❑ **IDCMP_GADGETOFF**

A partner of IDCMP_GADGETUP. It tells you when a gadget's activation state is terminated, but no IDCMP_GADGETUP was sent.

❑ **DrawImageStateFrame()**

New function that sends the IM_DRAWFRAME method to the image object, instead of IM_DRAW.

❑ **IDCMP_MOUSEMOVE**
IDCMP_MOUSEMOVE events generated because of a gadget should have the gadget pointer in the IAddress of the IntuiMessage, if the window requests it.

❑ **BOOPSI**
Autoknob Support Allow the combination of the sizing properties of an autoknob with the custom image capabilities of a BOOPSI image. Currently, you can have either one but not both.

❑ **GMORE_FULLYRENDER**
Gadget flag which says "I fully render in my hitbox" or "I fully render in my bounding box." We could skip erasing GREL gadgets in the area that intersects their new position (or the new position of any other gadget with this property).

❑ **SGH_INITIAL**
String edit hooks should receive a SGH_INITIAL message to validate contents on startup, and possibly a SGH_FINAL on exit.

Tool Port

A new unifying concept of IPC is being developed which will greatly simplify the job of writing applications on the Amiga.

A tool port provides a central communication point for an application. All IPC directed to this application are funneled through the port. The definition of the port interface allows unlimited growth and expansion. Routers can easily be added by the application to direct different message types to different locations. All system communications, such as IDCMP or ARexx messages, will come into the tool port in a standard format.

The tool port will finally gives the system something it always lacked which is a unique handle by which an application formally identifies itself, and makes itself available to the outside world. The tool port will provide a set of pre-defined events, such as "shutdown", "show yourself", "open a file", etc., enabling the creation of system control tools.

The tool port will also simplify the programming model. All input coming into an application will come from a unique source. This serialization of messages guarantees correct processing sequence across all applications.

General Changes

There are a few general improvements that will affect most tools that come with the system software, and will benefit end-users and application writers.

Font-Sensitivity

All GUI-based system tools will become font-sensitive. This will let them adapt to any font the user chooses for his system. The layout task will be off-loaded to a new library called layout.library. This library will offer a generic rectangle layout engine, enabling effective font-sensitive layout.

layout.library will operate on a hierarchical organization of rectangles. Each rectangle describes its place within its parent rectangle. A rectangle can define its size in relation to its siblings, relative to the parent size, with a fixed number of pixels, etc. The layout procedure involves an iterative negotiation between the library and the rectangles, that causes the size and position of the rectangle to constantly adjust, until the requirements of all the rectangles are met.

On-Line Help

V39 added the necessary hooks to implement full context-sensitive help in applications. We plan on making all system tools provide help by making use of these mechanism. Some minimal additional support will likely be added to the new BOOPSI classes to make them easier to deal with. For example, each class can provide help about itself. In addition, some minor extensions to AmigaGuide are likely to happen, in order to make on-line help more usable. For example, support for simple help windows, where there are no prop gadgets, no system gadgets, and no AmigaGuide gadgets, could be handy to create light-weight help boxes.

Workbench

We are planning a complete rewrite and substantial redesign of the Workbench interface for the future. We are expecting a increase in performance, reduction in quirkiness, and substantial increase in functionality. However, plans are currently not detailed enough to be discussed.

As part of the Workbench rewrite, a tie-in with the ASL file requester will likely occur. The file requester will become a client of Workbench and will use its code to display file information. This will offer the user a consistent visual interface to the file system.
◆