CMU-ITC-88-093

User Interface Guidelines

for the Andrew System

Version 2.0
(Draft)

10/12/88

Authors:

Maria G. Wadlow
Christina Haas
Dan Boyarski
Paul G. Crumley

Preface

We want principles,
not only developed, - the work of the closet, -
but applied; which is the work of life.

Horace Mann
Thoughts, 1867

For the past several years, exciting work in computer system development has been in progress at Carnegie Mellon University. The Information Technology Center, a joint endeavor between Carnegie Mellon University and IBM Corporation, has been developing a computer system for use on campuses and in businesses around the country. This system, named Andrew in honor of Andrew Carnegie and Andrew Mellon, is comprised of three primary pieces: the Andrew File System, which provides remote access to files across the country; the Andrew Toolkit, which provides tools for building multimedia applications within the system; and the Andrew Message System, which enables mutimedia documents to be transferred via electronic mail.

The Andrew system is being developed to support a myriad of users. Professors and students, secretaries and campus administrators, system developers and documentors all make use of the system on a daily basis. For this reason, the user interface for the Andrew system is of tremendous importance to its acceptance and success.

The Andrew system is a relatively young system, still being developed and refined. In addition to ongoing development which is taking place at Carnegie Mellon, the Andrew system was designed to encourage modification and customization by its growing user community. Both of these development efforts are crucial to the success of the Andrew system.

This document consists of four major sections. Each section of the document is addressed to a specific audience with specific needs. The sections can be read and used separately, although they have been conceived as a whole.

Part 1: *Requirements for Andrew Inset Development*, presents specific requirements for inset behavior in the Andrew system and will be useful for developers who wish to understand inset behavior in the Andrew environment.

Part 2: *Related Issues in Andrew Inset Development*, represents the current thinking on complex interface issues in the Andrew system and outlines the considerations which should guide resolution of these issues.

Part 3: *Future Questions In Andrew Inset Development*, outlines some of the crucial interface issues which remain to be thoroughly analyzed and understood. Readers who are interested in complex, state-of-the-art decisions about interface design will find Parts 2 and 3 valuable.

Part 4: *The Importance of the User Interface: Annotations on Requirements for Andrew Inset Development*, presents some of the philosophy and guiding concepts behind the design of user interface guidelines, as well as specific discussion of the design decisions set forth in Part 1 of the document. This part would be particularly interesting for human factors researchers and those interested in the complex decision-making which goes into the design of interface requirements.

A glossary of selected terms and a short bibliography are also included.

This document presumes some amount of knowledge about the Andrew system. For the interested reader, more information about Andrew can be obtained from the following sources:

Borenstein, N., Everhart, C., Rosenberg, J., Stoller, A., A Multi-media Message System for Andrew, *Proceedings of the USENIX Technical Conference*, Winter 1988.

Howard, J.H., An Overview of the Andrew File System, *Proceedings of the USENIX Technical Conference*, Winter 1988.

Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S.H., and Smith F.D., Andrew: A Distributed Personal Computing Environment, *Communications of the ACM*, March 1986, Vol. 29, No. 3.

Palay, A.J., Hansen, W.J., Kazar, M.L.,Sherman, M., Wadlow, M.G., Neuendorffer, T.P., Stern, Z., Bader, M., Peters, T., The Andrew Toolkit: An Overview, *Proceedings of the USENIX Technical Conference*, Winter 1988.

Subasic, K., Robertson, J., Langston, D., and Grantham, D., *A Guide to Andrew*, Information Technology Center, 1988.

Information Technology Center
Carnegie Mellon University
Pittsburgh, PA 15213

Table of Contents

1 Requirements for Andrew Inset Development

This portion of the document (Part 1 of the larger document, *User Interface Guidelines for the Andrew System*) presents a set of requirements for inset behavior for the Andrew system. These requirements are based on specific design decisions and are the result of both collaboration by a number of interface experts and everyday experience by users of the system. These requirements deal only with those issues for which there is sufficient information to justify firm specifications for interface behavior. Issues which are still being investigated are discussed in Parts 2 and 3 of the larger document.

1.1 Objectives

Part 1 of this document is designed to furnish developers with access to information and answers to design questions pertaining to insets. It has been kept brief to improve the usability and accessibility of the information it presents. The requirements presented here deal primarily with default behavior. An annotation of the requirements, which presents a rationale for the design decisions, can be found in Part 4 of the larger document.

The goal of the requirements presented in Part 1 is two-fold: application of these requirements will lead to a user interface which is both consistent across insets and, easier to learn and use. In addition, these requirements will aid inset developers by providing a consistent model which can simplify the user interface design component of the inset development process.

1.2 What Are Insets and How Are They Used?

In many systems, users are able to create textual, graphical, raster or other types of files and documents. Relatively few systems enable the user to combine several types of objects within a single document. However, through the use of insets, the Andrew environment can support multimedia documents, documents which are not limited to one type of object.

Insets are software packages which have been designed for the display and modification of certain types of information. More specifically, an inset is the relationship between two types of objects: the data object, which describes *what* is to be displayed, and the view, which describes *how* it is to be displayed and manipulated. For instance, a table of numbers might be a data object that is displayed using a pie chart view. The distinction between the data object and the view will become more important as various properties of insets are discussed in the following sections. Currently available insets include text, tables, drawings, equations, animations and rasters.

Insets can be divided into two distinct categories, based on certain properties of the inset: *sequential insets* and *spatial insets*. This distinction applies to currently available insets but may hold for future insets as well. Sequential insets are those in which the *ordering of data* is important; for example, when reading, people process text in a primarily linear or sequential manner. Spatial insets, on the other hand, rely on the *relative position of the data*. The position of objects relative to other objects within a picture or drawing is crucial for comprehension. Of the currently available insets, text and eq would be classified as sequential insets, and zip, animation and raster would be classified as spatial objects. Table has both spatial and sequential properties: the table as a whole is spatial but the individual cells are sequential.

1.3 Requirements

1.3.1 Giving an Inset the Input Focus

In order to do any work with an inset beyond displaying it, that inset must have the input focus. Just as the user must be reminded which is the current window, an inset must give some visual indication that it is currently accepting input.

A single left click within an inset's boundaries gives that inset the input focus. Any visual indication of the current location of the input focus is removed from the inset which formerly contained the input focus and is now displayed in the current inset.

The inset which has the input focus is distinguished by: a border around the perimeter of the inset, menus appropriate to that inset and, in some cases, a cursor change.

1.3.2 Creating an Inset

A new inset can be inserted by choosing the Add Inset menu item from the menu stack (or the equivalent function from an exposed "palette" of tools) and selecting the inset type from a dialog box. The default inset type should be text.

Upon creation of a new inset, that new inset must have the input focus; that is, it must be the current focus of attention for any mouse or keyboard input. Any visual indication of the current location of the input focus is now displayed in the newly created inset. The default cursor for the new inset must be used and the inset's menus must be made available.

Upon creation, an inset must be edittable and any tools necessary for editing the inset must be displayed and available for use.

When an inset is created within a document, it must have the default shape (rectangular) and the default size. An inset's default desired size, the size that the inset would display itself given sufficient space, and initial position are determined by their own and their parent's spatial/sequential properties. Refer to the table in Figure 1.1 for details regarding default desired size and initial position.

		Parent: Spatial	Parent: Sequential
Child	Spatial	W=1/3 parent's shown width L=1/3 parent's shown length P=centered in shown parent	W=parent's shown width L=1/3 parent's shown length P=at caret
	Sequential	W=1/3 parent's shown width L=1 line P=centered in shown parent	W=parent's shown width L=1 character P=at caret

W - inset width

L - inset length

P - inset position

Figure 1.1

An inset's desired size is the size at which it was created. If the parent allocates more space than requested, an inset should use the desired size by default. If the parent allocates less space than requested, then the inset must resize and manipulate its data to fit within the given size. (See *Resizing an Inset.*)

1.3.3 Selecting an Inset

In order to perform any operations on an inset as a whole, the inset's parent must have the input focus and the inset must be selected, just as one would select any other object to manipulate it.

A double left mouse click in an inset (two left mouse clicks at the same screen position and within a prescribed timeframe) selects that inset for whole-inset operations such as cutting the inset out of the document. For some insets, a child inset may also be selected by positioning the cursor in the parent at some position before the inset and dragging to some position after the inset. For sequential insets, this may result in some portion of the parent inset being selected in addition to the inset.

A selected inset should be highlighted using the same mechanism to distinguish selected regions that is used by that inset's parent (usually outlining or the addition of selection "handles" on the borders), and the parent's menus should be displayed. Selecting an inset requires that any previous selection become unselected. That is, before the new selection can be highlighted, the highlighting of any former selections must be removed.

1.3.4 Resizing an Inset

An inset's desired size is used to allocate space for its display in the parent. Even if resized, insets should retain some information about their desired size, that is, the size at which they were originally drawn.

Selecting an inset and repositioning its boundaries by dragging them with the mouse is probably the easiest method of resizing an inset relative to the window in which it is being displayed.

Adding to the contents of an inset may also cause it to resize. In insets such as text or table, the user may add another line of text or a new column to the table which would cause the inset's desired size to increase automatically.

If, for some reason, the parent does not allocate sufficient space, then the inset must manipulate its representation of the data to fit within the space that has been allocated. All insets must be able to redisplay their data in the event that they are given insufficient space in which to display their contents.

Insets can be grouped into three categories, depending upon their default method of redisplaying given insufficient space: *cropping insets*, *wrapping insets* and *scaling insets*. At the lowest level, insets which are unable to display their contents within the given space may display only a portion of the full contents, cropping any data which doesn't fit within the boundaries. All insets must be able to crop. At the next level, insets may wrap their contents upon reaching a window boundary. Typically, only those insets which are sequential in nature (e.g., text) will wrap their contents to fit within the allocated space. Insets which wrap may crop their contents as well. At the highest level, insets may scale their contents to fit within the given boundaries. Insets which scale may also crop their contents and may be able to wrap them as well (although not necessarily simultaneously).

Figure 1.2 classifies the six basic insets in the Andrew system according to their redisplay capabilities.

	Table	Text	Eq	Raster	Zip	Fad
Scale		(opt.)	(opt.)	Default	Default	Default
Wrap		Default	Default			
Crop	Default	(opt.)	(opt.)	(opt.)	(opt.)	(opt.)

Default - default redisplay method

(opt.) - optional redisplay method

Figure 1.2

Using the table in Figure 1.2, a decision tree can be built which determines how insets redisplay when given insufficient space. (See Figure 1.3.)

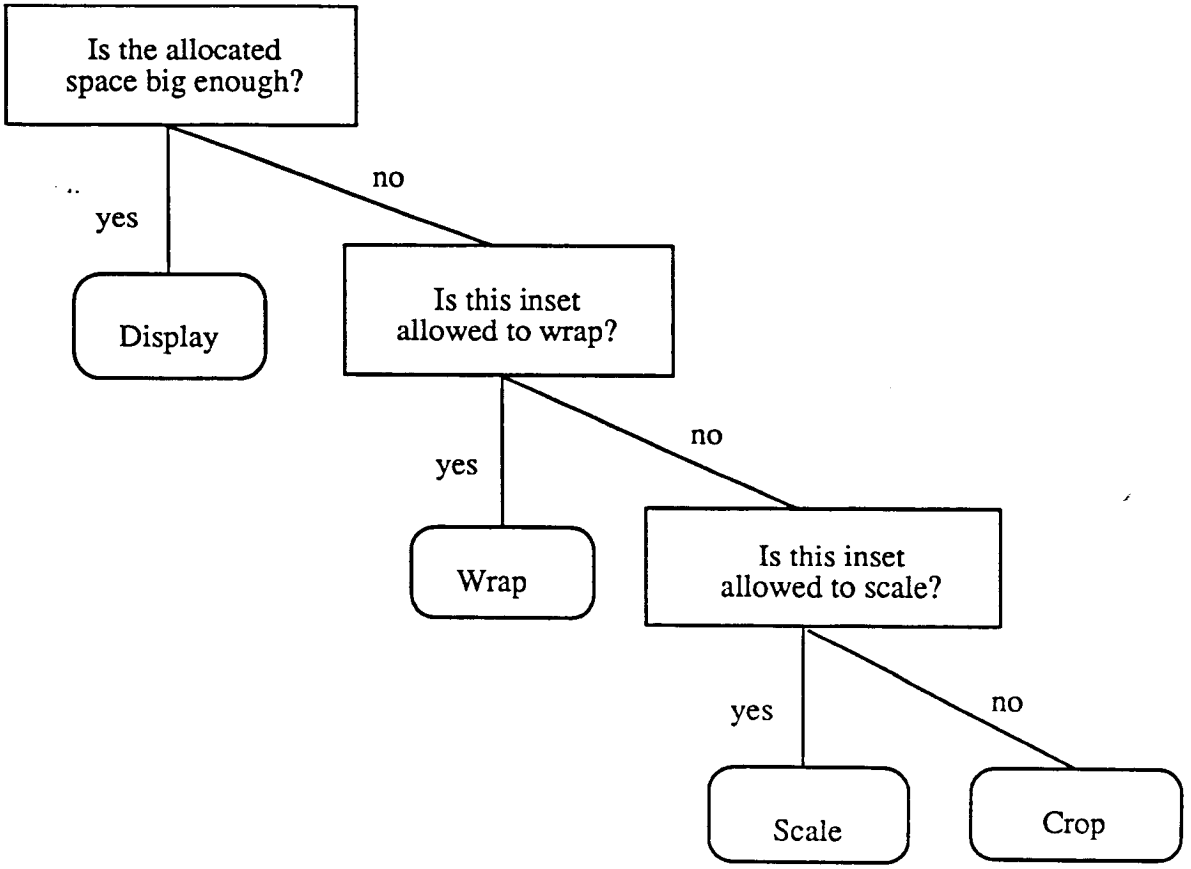


Figure 1.3

1.3.5 Scrolling an Inset

When an inset has been cropped, two issues arise: first, the user much be made aware that there is more of the inset to view, and second, the user must have a way to access the cropped part of the inset. The first of these issues is addressed by the varying kinds of inset borders. Border-related issues are discussed at greater length in Part 2 of this document. Scrollbars also signal that more information is available, and, more importantly, may provide access to cropped information.

Scrollbars are rectangular regions placed along the perimeter of an object. Scrollbars are used to indicate which portion of the data is currently visible, where the caret is located and the existence of a selected region. Scrollbars are also used to reposition the data that is displayed. The scrollbar should provide varying rates of movement -- from fine movement (e.g., a single line at a

time) to large scale movement (e.g., jump to top/bottom of the file). Figure 1.4 shows an Andrew scrollbar.

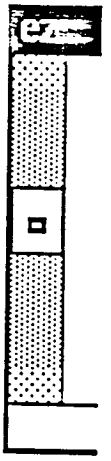


Figure 1.4

When used, horizontal scrollbars must appear across the bottom of the workspace, vertical scrollbars should appear to the left of the workspace. Scrollbars are most useful for sequential insets. Spatial insets may require some type of panning operation to allow simultaneous horizontal and vertical movement. Panning is described in more detail in Part 2 of this document.

1.3.6 Moving an Inset

Insets may be repositioned using any mechanisms available within the parent; for example, cutting and pasting or selecting the inset and dragging it with the mouse. Resizing or other mechanisms available through the parent inset (e.g., centered, right flush, etc.) may be used to fine tune an inset's position.

2 Related Issues in Andrew Inset Development

This section of the document (Part 2 of the larger document, *User Interface Guidelines for the Andrew System*) outlines current thinking about some of the complex issues of inset design and behavior. While considerable thinking and debate has gone into this section of the document, issues discussed here are still open to discussion and are not yet "requirements" for Andrew inset development. It is intended that the issues discussed here will be resolved and tested and eventually become part of the Requirements for Andrew Inset Development.

2.1 Inset Behavior

2.1.1 View Manipulation

2.1.1.1 Desired Size

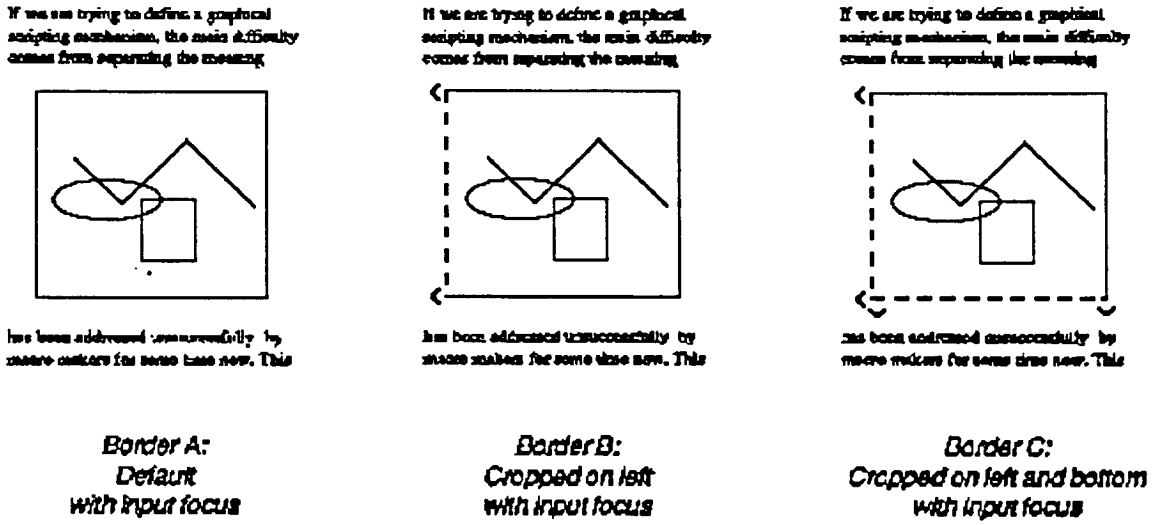
An inset's initial desired size should generally be small enough that the inset can print without being cropped. The inset's initial desired size should also be large enough that any tools or palettes associated with the inset have room to be adequately displayed.

When pasting an inset or inserting the contents of an existing document into the current document, insets should be inserted at their desired size.

2.1.1.2 Inset Borders

A more detailed description of the type of border which outlines the inset which has the input focus is currently being designed. This border should be sufficiently different from the one pixel wide lines which are commonly used in screen layout. This border should also give the user some visual indication that the inset contains more information than what is displayed within the border and some cue that scrolling the inset in a particular direction is possible. This border should be significantly distinct from the borders which surround dialog boxes, selected insets and insets which have the input focus and whose entire contents have been selected.

Figure 2.1 shows different borders for insets which have the input focus. Border A shows the default borders. Border B shows that the inset has been cropped in one direction (to the left). Border C shows that the inset has been cropped in two directions (to the left and to the bottom).



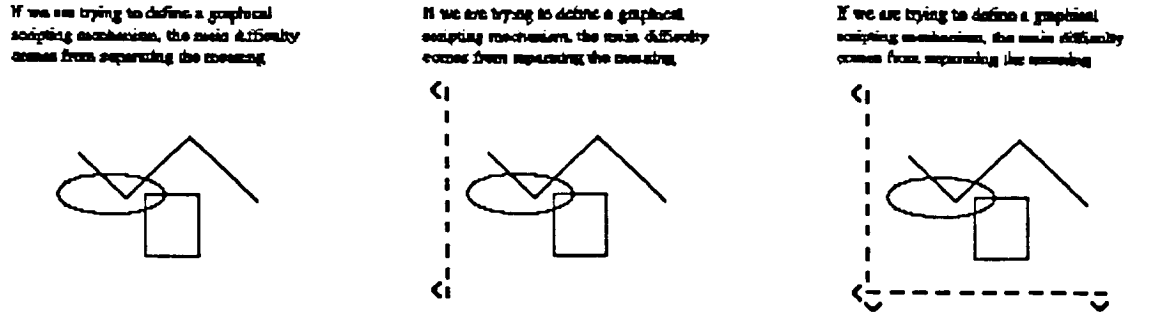
Border A: Default with input focus

Border B: Cropped on left with input focus

Border C: Cropped on left and bottom with input focus

Figure 2.1

Figure 2.2 shows different borders for insets which do not have the input focus. Border D shows the default borders (none). Border E shows that the inset has been cropped in one direction (to the left). Border F shows that the inset has been cropped in two directions (to the left and to the bottom).



Border D: Default without input focus

Border E: Cropped on left without input focus

Border F: Cropped on left and bottom without input focus

Figure 2.2

2.1.1.3 Panning

Panning is a method of moving through data without using scrollbars. To pan an inset, the user positions the cursor within the inset, presses a mouse button, and uses the mouse to drag the data to the desired position. When the mouse button is released, the data is updated to appear in its new position. Some insets may require the user to place them in a special mode in order to pan. In addition, panning may be used in conjunction with scrollbars, to allow simultaneous horizontal and vertical movement as well as providing a constant indication of the position of the data.

If panning is implemented as a special mode of the inset, some visual indication should be present to alert the user that panning is available.

2.1.1.4 Iconification

Under some circumstances, users may wish to iconify some or all of the insets in a document. Iconification might be used in previewing documents to obtain a sense of the structure of the document; for example, as in using an outline editor. Typically, iconified insets would appear in their expanded form when printed, although the user may wish to override this behavior.

2.1.2 Data Manipulation

2.1.2.1 Cutting/Copying the Contents of an Inset

When an inset has the input focus and the entire contents of the inset have been selected, choosing Cut or Copy from the inset's menu stack will cut or copy the contents of the inset but leave the inset itself intact. For example, after having cut all of the text out of a text inset, the inset will be empty, but will still exist for further manipulation. In order to remove the inset, it must be selected from its parent and cut using the parent menus (See *Cutting/Copying an Inset*).

2.1.2.2 Pasting/Replacing the Contents of an Inset

If the contents of an inset have been cut or copied, choosing Paste will cause a new inset to be created and the contents of the cutbuffer will be pasted into it. If the parent is of the same type as the inset whose contents were cut or copied, the contents of the cutbuffer will be pasted into it directly (i.e. no additional inset will be created). For example, if the contents of a text inset are cut from a document and then pasted into a text document, the result will be a simple text document and not a text inset within a text document. If the inset types differ, a dialog box will be presented to the user indicating that the data can not be inserted directly into the parent. The default response will be to create a new inset and embed that inset in the parent.

The Replace operation may be made available by an inset to allow all or some portion of the contents to be replaced. The user may replace the contents of an inset with the contents of another inset of the same type, if that inset supports a Replace operation.

2.1.2.3 Inserting a File

The user should have the capability of inserting data from another file into the current document. Information from another file can be inserted in full into the current document, in which case a copy of the data is inserted. This operation is equivalent to editing the file to be inserted, copying

the contents of the file, and then pasting the copied data into the current document.

2.1.2.4 Inserting a File Reference

The user should also have the option of inserting a reference to a file into a document. To the casual viewer of the document, there would be no visual difference between data which has been inserted into a document and data which has been referenced. However some means of distinguishing between the two should exist. In addition, the user should have some method of globally resolving all references to external files to allow the document to be transferred to another system. The user should be warned before transfer that references to external files exist within the document to be transferred. The user should then be given the opportunity to resolve those references.

2.1.2.5 Reading/Writing an Inset

The user should be able to replace the contents of the current inset with data from another file. This operation is similar to the Replace operation, except that the new data is imported from a file rather than from the cutbuffer.

The user should have the capability of writing the contents of an inset to another file. For example, when editing a table inset within a text document, the user should have the capability of saving the table inset into a separate file. This new file would contain a table document, and the saved table would be the top-level inset in the document.

2.1.2.6 Inheritance

The type of state information which would be of interest to insets varies widely from one type of inset to the next. To account for this disparity, upon creation of a child inset, that child inset should request state information from the parent inset regarding font family, size, face, etc. The parent inset should be responsible for keeping track of which children have requested this inheritance information and provide those children with updates as that information changes. See Part 3 for more discussion regarding inheritance.

Browse mode, where the viewer can look at a document but not modify it, should be a viewer-selectable option which affects the document as a whole.

Readonly mode, where the author chooses to disallow modification of the document, should be an author-selectable option which may vary on an inset-by-inset basis, but which is inherited by child insets.

2.1.3 Meta-Inset Operations

Meta-inset operations are those manipulations which are performed on the inset as a whole, such as creating, selecting or cutting an inset. Meta-inset operations are not dependent upon the particular type of inset being manipulated. Meta-inset operations are invoked from the parent, via mouse clicks or keystrokes, or through the parent's menus.

2.1.3.1 Cutting/Copying an Inset

The Cut operation is used to remove an object or objects from the current document. Upon cutting or copying, the object is placed in the cutbuffer and is available for pasting within the current or any other document. The Copy operation is used to copy an object or objects to the cutbuffer without removing them from the current document. When an inset has been selected, choosing Cut or Copy from the parent's menu stack will cut or copy the entire inset.

When the inset has been selected from the parent as described above, the entire inset, and any children insets, will be cut or copied. For example, if a table inset has been selected from within a text document and copied, when that inset is pasted into a zip document, the end result will be a table inset within a zip document. The fact that the table was originally included in a text document will not be evident unless some portion of the text had been copied as well.

2.1.3.2 Pasting/Replacing an Inset

The Paste operation is the logical inverse of the Cut or Copy operation. Care should be taken to ensure that if a user chooses Cut from the menu and then immediately chooses Paste, the resulting state should be identical to the state before Cut was chosen, although the cutbuffer contents may differ. The Replace operation allows the user to simultaneously cut any object or objects out of a document and paste another object into the document in the same position.

When any object or group of objects has been pasted or replaced in a document, those objects should become selected upon pasting and should remain selected until the user takes some action which removes that selection. This behavior should be the same regardless of whether the objects are strings of text, rectangles or insets. In addition, any previous selections should now become unselected. In sequential objects, such as text, the pasted objects should be inserted after the previous selection.

The Replace operation should be made available as an expert option for insets.

2.1.4 Cursors

The shape of the cursor should indicate mode changes. The standard cursor may vary from inset to inset, however they should be made as consistent as possible. In addition, the same wait cursor should be used throughout the system and its threshold should be consistent from inset to inset. Mode changes, for example when the user clicks on the rectangle icon in zip and enters "rectangle-drawing mode", should be reflected by a change in the cursor shape. See Figure 2.3 for examples of various cursors used in the Andrew system.

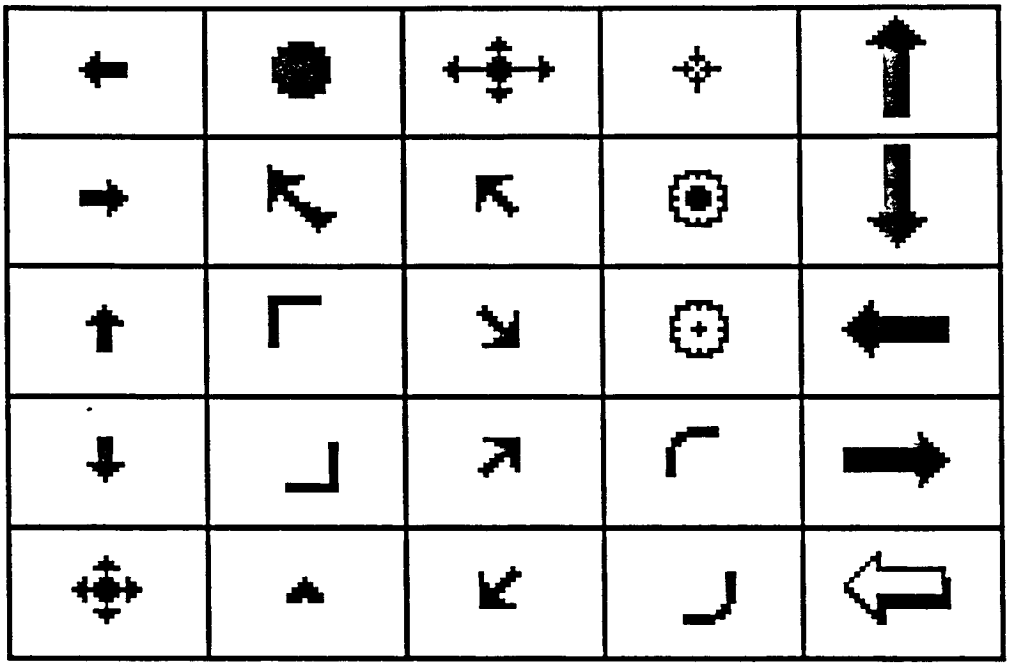


Figure 2.3

2.2 Author/User Specifications and Customizing

When the user has changed the scale of the inset, scrolled the inset or adjusted the level of detail (zoomed), the user should see some visual indication that the inset has been modified and should have the ability to easily normalize the inset.

The author of a document must have an easy way to ensure that future viewers of the document see it exactly the way the author intended.

It must be possible for the author of a document to "lock down" or configure the document in such a way that all of the author's preference, template, default value, etc. settings are contained in the document. This allows the author to be confident that a viewer's setting of these attributes will not alter the appearance of the document. This capability would not be the default behavior, but it must be an option. Figure 2.4 is an example of this capability for font selection. Figure 2.4 shows the decision path for determining which font should be displayed. Any number of other decisions might follow a similar path.

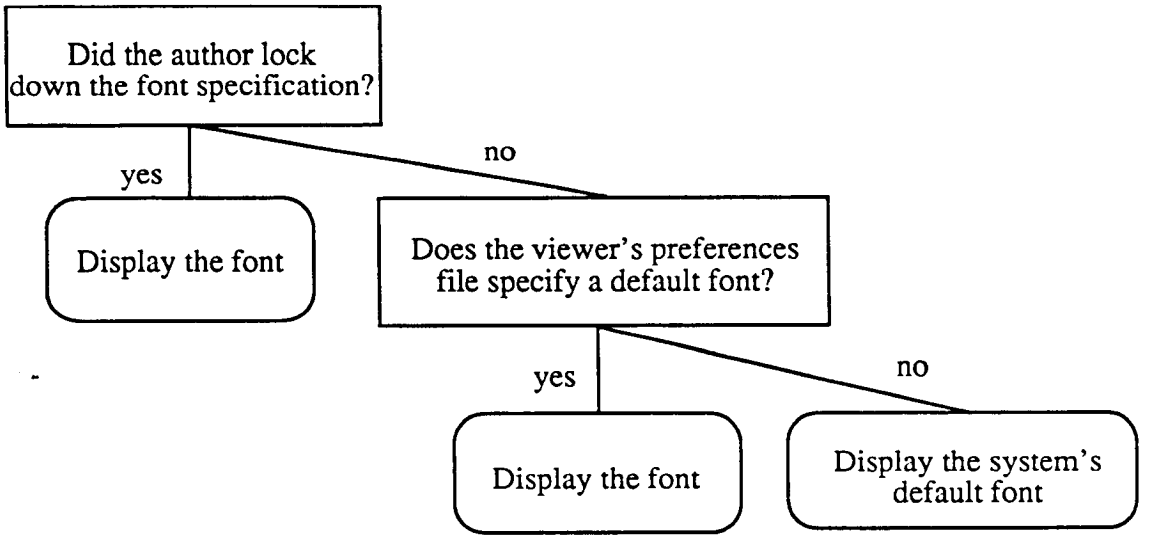

Did the author lock down the font specification?
yes
no
Display the font
Does the viewer's preferences file specify a default font?
yes
no
Display the font
Display the system's default font

Figure 2.4

Figure 2.4

The user must be able to interactively set attributes of the object which are normally set in preferences or templates. This allows the author to easily see how a document will appear when viewed by a user with different preference settings. The overridden values should be written to the file in place of the normal settings. The user should be able to revert to using the default values or preferences. There should be some distinction between the specification of a parameter which happens to be the default value and the lack of a specification for a parameter (which would result in the use of the default value).

The user should be able to set preferences for an object by executing a startup script of commands. This capability greatly enhances the ability of an object to do useful work for a user by allowing scripts as well as other types of templates to be used.

2.3 Dialog and Response Devices

Dialog and response devices provide methods for active communication between user and computer. Dialog and response devices are of four types: 1) *action devices*, which immediately initiate action and which typically do not require further information; 2) *choice devices*, which present the user with information about the currently available and selected options and allow the user to change the selected option; 3) *input devices*, which allow the user to enter names or numbers from the keyboard; and 4) *feedback devices*, which give the user information and may or may not require subsequent user action.

These different types of dialog and response devices provide a range of communication methods. Different dialog and response devices are more suitable to different scenarios within the system, depending upon the particular situation, the user's expertise and any preference options the user may have invoked. For example, keybindings (discussed below) may be useful for experienced users while menus of differing complexity may be suitable for users of differing levels of expertise.

2.3.1 Action Devices

2.3.1.1 Button

Definition: A small, visually distinct, area of the screen which responds to mouse clicks.

Advantages
- Buttons have a stable position; they don't move around the screen.
- Buttons are easy to learn/use.
- Buttons provide fast access to frequently used operations.

Disadvantages
- Buttons are limited in their functionality.

Example
- Messages "Punt" button.

2.3.1.2 Menu

Definition: A simulated three-dimensional arrangement of user selectable operations.

Advantages

Menus are of the pop-up variety, so they don't take up any screen space when not in use.
Pop-up menus are readily available from anywhere in the window.
Menus can contain many options.
Menus take advantage of muscle memory by positioning options in the same places in different menus.
Menus are easy to learn/use.
Using menus can reduce keystrokes.

Disadvantages

Menus are not displayed continuously and must be called up.
Menu items may be hard to find initially.
There is a danger of having too many menus items or menu cards.
Menus may be slower for frequent users.
Pop-up menus can be painful in slow window managers (menus require rapid redisplay rate).

Example

WM menus.

2.3.1.3 Keybindings

Definition: A method by which users may bypass menu selections through the use of keyboard operations.

Advantages

Keybindings are very fast.

Disadvantages

Keybindings may be difficult to remember, particularly for new users.

Example

Pressing ctl-g to abort an operation in the editor.

2.3.1.4 Mouse Operations

Definition: The building blocks upon which most dialog devices rely to receive user's requests.

Advantages

Mouse operations are intuitive.
Using the mouse is very efficient.

Disadvantages

The user's hands must be removed from the keyboard in order to make use of the mouse.

Example

Clicking the left mouse button to select an object on the screen.

2.3.2 Choice Devices

2.3.2.1 Palette, Selector, Switch

Definition: A two-dimensional arrangement of user options.

Advantages

Palettes provide fast access to options.

Palettes serve as a constant reminder of the current possibilities since all of the choices are visible.

Palettes are visual representations of things that are hard to describe verbally, such as colors and fill patterns.

Disadvantages

Palettes can require significant amounts of screen space.

Examples

Draw/Paint palettes.

Mathematical symbols for equation editor.

2.3.2.2 Slider

Definition: A representation of a range of values.

Advantages

Sliders provide fast access to options.

There is no need to list all of options for a slider. Sliders are especially useful for numeric ranges where all of the options would get tedious, (e.g., range from 1 to 1000).

Disadvantages

Sliders require more screen space than a simple input box.

Making the unavailable options grey is not possible with a slider.

Example

Volume slider.

2.3.2.3 List

Definition: A collection of possible values.

Advantages

Lists allow an operation to be performed on multiple objects at once.

Lists permit changeable values to be displayed as options.

Lists permit all options to be displayed simultaneously.

Lists serve as a reminder of the options when there are many choices (e.g. filenames).

Scrollable lists allow a large number of varied choices to be displayed in a fixed amount of screen space.

Disadvantages

Lists require more screen space than a simple input box.

Lists can be slow.

Example
Subscription list in messages.

2.3.2.4 Toggle Item

Definition: An item which toggles between two different (opposite) values.

Advantages
Toggle items on menus take advantage of muscle memory by locating options in the same places on the menu card.
Toggle items conserve space.

Disadvantages
Toggle items on menus are inconsistent with menu model presented above (violates consistent location and greying unavailable option rules).

Examples
Expand Menu/Shrink Menu toggle menu item in Console.
Checkboxes to turn attributes on or off.

2.3.3 Input Devices

2.3.3.1 Data Entry Field

Definition: An area of the screen that is used to accept strings of data.

Advantages
Data entry fields simplify input of data.
Data entry fields require very little training.

Disadvantages
Data entry fields require some amount of screen space.

Example
The area provided for inputting a file name after having selected "Save as..." in ez.

2.3.4 Feedback Devices

2.3.4.1 Dialog Box

Definition: A method for querying the user about some aspect of the system or informing the user of some important change in the system's operation. Dialog boxes may incorporate other types of dialog and response devices.

Advantage
Dialog boxes command attention from the user.
Dialog boxes allow more efficient display of information.
Dialog boxes are self-explanatory.

Disadvantages

Dialog boxes may be slow.
If overused, dialog boxes may lose primacy.

Example
"Purge deleted messages?" dialog box in messages program.

2.3.4.2 Message Line

Definition: An area of the screen in which feedback messages may be provided for the user.

Advantages
Message lines don't interfere with normal user work.

Disadvantages
Message lines don't command the user's attention.

Example
Checkpointing messages.

3 Future Questions in Andrew Inset Development

This portion of the document (Part 3 of the larger document, *User Interface Guidelines for the Andrew System*) outlines questions which will be important to discuss and resolve as the Andrew system evolves. Computer technology progresses and expands at an astonishing rate. In many ways, a document which describes the guidelines for a user interface is out-of-date almost as soon as it is written. Consequently, many interesting issues vital to the design of the interface have not been addressed in earlier sections of this document. This section (Part 3) outlines some of these issues.

3.1 Inset Manipulation

3.1.1 The 'Show Insets' Option

Some mechanism should exist to indicate to the user the presence of insets in a document and their types. This information should be continuously available and should be consistent between inset types.

A function similar in purpose to the Macintosh's "About" menu item would be a way to provide some information about the inset. However, some types of information about the inset should be available continuously, such as the fact that it is an inset, or that some action may be performed on the inset (e.g., Animate in a fad inset).

In addition, it would be helpful to have some way of displaying the structure of a document to the user, noting the top-level inset and the hierarchy of children insets.

3.1.2 Redirecting an Inset to a Different Window

Some mechanism should exist for displaying an inset in its own window to make scrolling and editting easier.

A menu option may be provided to allow an inset to be redirected to a subordinate window. Insets which are redirected in this manner have the same behavior as insets which have not be redirected, but are less constrained by screen space limitations. Palettes may be redirected to subordinate windows so that they can be repositioned by the user.

Some method of indicating the relationship between the subsidiary window and its controlling window should exist.

3.1.3 Default Size

An interesting problem can be derived from these using the default sizes for new insets as discussed in Part 2. If a sequential inset, embedded within a sequential parent, is to be only a single character wide as the default, what happens when the child inset is enlarged and must wrap? For example, an in-line equation which extends beyond the right border of the window would result in an inset which is not rectangular. Other scenarios can be imagined which would have the same result. Possible solutions to this problem are being considered, but work on this problem is still in progress.

3.1.4 Inheritance

Some method should be available to allow information to be passed from parent to child (for example, that the document is readonly or that the surrounding text is in a sans-serif font). It should be possible for the user to identify all of the variables which affect a selected item (including modes, preferences, templates, etc.).

If a document is displayed with an unexpected font, the user should have some easy way of determining which preference or template is responsible for it so that, if necessary, it can be changed.

3.1.5 Shared Data

There should be some method of informing the user that an inset's data is being shared with other insets.

There should be some method for displaying the structure of a document to show the boundaries and nesting of the insets within the document. This ability will be needed to represent non-sequential documents (i. e. hyper-documents) in order to construct, browse and manipulate these more complex document structures. When this capability is provided, it will be feasible to indicate to the user that multiple views are sharing common data.

3.2 Color

The use of color can highlight, emphasize structure and organization, create emotional responses and discriminate subtle distinctions. These capabilities can be very powerful when used to their best advantage. However there are dangers. Saturated colors can be tiring to the eyes, too many colors can be confusing and the use of color in the wrong place can be misleading.

The Andrew system uses color in the following ways:

- to enhance the dialog between the system and the user.
- to represent data in an effective manner.
- in an image, e.g. a color picture.

Each of these uses of color has benefits and problems compared with the use of two-level, or black and white, images. The following sections will address some general requirements for using color as well as specific requirements for the different uses of color.

3.2.1 Common Areas of Concern

Applications should be designed for monochrome first. All objects must be able to do something with a simple, two-level imaging system. This does not imply that the object must display its information in a manner that is simply a black and white version of the representation that would be used on a color screen. As an example, a pseudo-color image may refuse to display itself on a screen that does not have color capability. In this situation, the object should resort to the display of a short message describing what would have been shown ("A weather map", for example) and how to see the image ("To see this weather map, view this document on a color system."). This example is an extreme, but legitimate, example of how an object may handle the lack of color in the display system. In practice, all objects should be able to intelligently handle two-level imaging system since all objects should be able to print.

Color coding should be added only after displays have already been designed as effectively as possible in monochrome. Color coding should not be used to compensate for poor design.

The use of color should be considered carefully, as many systems do not yet support color screens. Even on color systems there are few standards that define how to display a particular color (chrominance, saturation, and intensity) on a given screen. This can lead to a number of colors being mapped to the same color or inaccurate color rendition.

Color should be used as a redundant indicator, in conjunction with symbology or some other display feature. Color should not be used as the sole code. Color blindness of potential users should be considered in the choice of color/intensity combinations for encoding critical information (It is estimated that approximately one male in four have some degree of color blindness.).

Objects must understand the concept of transparent vs. opaque. This is important when views of objects are stacked on top of each other. As an example, a graphics editor must understand that it's background color could be set to be transparent rather than opaque. This would allow the graphic object to be placed over a raster object, providing arrows or labels on the image.

3.2.2 Using Color to Enhance Communication

Brighter, more saturated colors should be used to draw attention to critical areas of the display.

Conventional color assignments should be used when choosing a color code (e.g. red = danger, yellow = caution, green = normal, white or grey = neutral, etc).

3.2.3 Using Color to Represent Data

Each color should be used as a unique indicator, each color should represent only one category of the data.

Tonal variations (different shades of the same color) should be used rather than spectral variations (different colors) to represent relative differences in data and order the color assignments so that the lightest and darkest shades correspond to the extreme values of the data.

When distinguishing between several discrete categories of data, distinct colors, colors which are easily discriminable, should be used to code the different categories.

Color coding should be used very conservatively, using few colors and only to designate critical categories of the displayed data.

Brightness may be used as a two-valued code to represent binary information, however brightness should not be used to distinguish more than four different values, particularly on screens with adjustable brightness.

3.2.4 Using Color in an Image

The use of color in an image is certainly the most intuitive of the methods listed. Still, there are a number of potential problems that can arise in the use of color in a system. It is important to allow the users to exercise their creative capacities as much as possible. At the same time, authors must be made aware of the situation where their work will not be displayed on a high-quality, color imaging system.

Users should be allowed to select colors from a palette rather than choosing colors by name, number or numeric parameters such as RGB or CIS. The use of sliders to create colors using RGB or CIS is a reasonable way to create new colors.

Authors should be provided with the ability to view how their work will be presented on imaging systems that have fewer capabilities in resolution, color and quality so they can "tune" the images to do well on lesser systems.

3.3 Printing

The user should be allowed to specify the parts of a document to be printed, it should not be required that the entire document be printed.

There should be defined methods for resolving the differences between printed and displayed versions of a document.

Insets should scale to fit within page boundaries. Insets which are unable to scale, such as the table inset, should use cropping or wrapping to enable their contents to fit on a page.

The user may wish to indicate where an inset may be broken across multiple pages.

4 The Importance of the User Interface:

Annotations on
Requirements for Andrew Inset Development

This portion of the document (Part 4 of the larger document, *User Interface Guidelines for the Andrew System*) presents a discussion of user interface philosophy and outlines some of the principles of user interface design which have been important in both the development of Andrew insets and the accompanying requirements for inset behavior. The section begins with a discussion of the importance of the user interface and presents some of the guiding philosophies behind the larger document. In addition, this section describes the effects that these guiding principles should have on system design. The section concludes with specific discussion of some of the design decisions presented in Part 1, *Requirements for Andrew Inset Development.*

4.1 The Importance of the User Interface

New computer technologies afford opportunities for users to become more productive and creative. However, users can take advantage of these opportunities only if the technology is presented in a way that is easy for users to understand, learn, and use. Therefore, the design of the user interface for computer system is crucial, for to the user the interface *is* the system. The user interface, ideally, should be simple and appealing, encouraging use and allowing the user to transfer experiential knowledge between applications.

The ability of the user to adopt, with support from the software itself, a mental model of system behavior is criticial to the success of the user interface. Carefully fostered mental models can enhance the user's perception and understanding of the system and how it works. Typically, applications are developed with some type of model in mind. The developer usually has some rather well-developed notion of how the application works which is consulted to determine whether additional features will be consistent with the rest of the system.

Users of the system also develop a mental model of the system. The user's mental model is modified and refined as the user becomes more familiar with the system. In fact, a well-developed mental model may eventually allow users to predict system behavior and continue to explore and learn about the system on their own.

Unfortunately, application interfaces do not always successfully convey the developer's model to users of the application. A user whose mental model of the system is inconsistent with that of the developer may find the application confusing and difficult to use. The user's expectations, which were based on his or her mental model of the system, may vary in important ways from the actual functionality of the application. Further, the developer's conceptual model may be too complex and detailed, lacking the appeal necessary to be of assistance to users of the application. In most instances there is no need for users to understand the application to the level of detail and complexity that the developer must. Ideally, the user's mental model should be structured in such as way that it will help users "bootstrap" themselves to greater understanding of the system.

It should be obvious that some knowledge of the user community is necessary to foster a coherent

and workable mental model. Developer's knowledge of the user community can be used to foster a mental model which aids the user in both initial learning and later independent use of the system. Knowledge of users can be acquired in a number of ways: research on the Andrew user community included a usage survey, conducted by Sandra Bond and on-going interviews and formal studies with students on the Carnegie Mellon campus, conducted by Christina Haas. In addition, since many departments on campus used Andrew for teaching or research, the university community provided feedback, both formal and informal, through bug-reports, bulletin boards, electronic mail, and other means.

4.2 Basic Philosophies

In developing these guidelines, the authors were guided by several user interface design principles:

Consistency: Consistency has been a primary goal: consistency in the kinds of objects users see, consistency in the kinds of operations that are possible within applications and in how those operations are accomplished, as well as consistency in the methods used to pass information from user to machine and back. This consistency is crucial in creating a system which appears to its users as a consolidated, working whole and which enables users to transfer skills and concepts between various insets and applications.

Predictability: Closely related to consistency is predictability: the ability of the user to correctly anticipate responses from the system. System predictability is vital in establishing the sense of trust needed for users to feel comfortable using the system. Surprising behavior might be considered entertaining in a game but will be considered frustrating and annoying in a tool.

User Control: In making choices about interface options, priority has been given to features which will put the locus of control on the user, rather than on the computer. The Andrew interface should be a tool which users employ to reach their own goals, rather than a system which appears to have goals of its own.

Appeal: The layout of the interface should be inviting, drawing the user in and creating a comfortable working environment. Visual appeal is vital to good communication. Complicated, awkward or cluttered interfaces are low on appeal and inhibit productivity. Well-articulated, straightforward, clean designs are appealing because they convey information clearly. Appealing systems encourage users to explore the system, creating an atmosphere which is conducive to learning.

Direct Manipulation: The users should have the sense that they are dealing directly with "real" objects, objects which, though they are only two-dimensional on a computer screen, behave as real world objects in the users' experience might. For instance, moving an object on a screen by adjusting numbers of an axis is not direct; moving that object via selection and dragging with the mouse is more direct.

Multiple Perspectives: This document has been compiled by a unique group of people. The backgrounds of the authors include formal training in computer science, engineering, psychology, graphic design and rhetoric. The authors also have a wide range of experiences with the Andrew system, including design, development, evaluation, implementation, analysis and testing new users, and using Andrew on a daily basis. Such a variety of individuals, with differing viewpoints, complementary training and diverse experience, provide the opportunity for a more complete and thorough treatment of the issues involved in interface design.

4.3 The Impact of Guidelines on System Development

The goal in the development of the document, *User Interface Guidelines for the Andrew System* has been to focus attention on critical design issues and establish specific design requirements. Furthermore, it is hoped that these guidelines will encourage the further discussion of complex user interface decisions and policies.

Consideration of such guidelines in the development process will help to assure consistency across applications. When using guidelines to design an application, the designer should keep in mind that, for a given task, using a computer should always be easier than not using a computer. In addition, the use of guidelines must be supported by a thorough understanding of the user population for which the application is being designed. Designers should keep in mind that accurate samples of the user population rarely consist of application designers. For maximum effectiveness, consideration of guidelines must take place early in the design process and prototype testing should be done to ensure good design.

One of the most important and difficult aspects of designing the Andrew User Interface has been to determine reasonable default behaviors for a user community that varies greatly in needs, experience, and knowledge. Reasonable defaults are crucial: it is default behavior which users first encounter in a system, defaults shape how users come to understand and to use a system, and default behavior comprises the mental model which is so important for successful interaction with a system. Simple and predictable default behaviors, then, should allow users to do what they need to do to get their work done but not overwhelm them with complexity or unnecessary information. In many ways, designing a system *is* designing its default behavior.

4.4 Annotations to Requirements Presented in Part 1

4.4.1 Creating an Inset

Note that the syntax involved in creating an inset is to specify the creation of the inset, then to specify the type of inset to be created: Operator + Object. If this syntax applies consistently across applications it can become quite automatic for users. The syntax holds whether the user is creating an inset via a menu item, through a palette, or by other means.

If the user does not specify an inset type, text should be the default value of a newly created inset. Presumably this will be the most widely used inset type.

Typically, users will create insets in order to do something with them. Consequently, when the inset is created, the appropriate editing and manipulation tools should be available without further user action.

To assure consistent initial behavior and appearance, default shape, size and placement of insets are necessary. The default should be rectangular. For a spatial object imbedded in a spatial object (for example, a raster imbedded in a line drawing) the default size should be approximately one-third the parent's size and placement should be centered. A spatial object imbedded within a sequential object (for example, a drawing inside text) should be the parent's shown width, one third the parent's shown length, and imbedded at the typing caret. Sequential insets increase in size linearly as users create them; e.g., a text inset "grows" as the user types. Therefore the default size for sequential insets doesn't follow the "1/3 rule." Rather, sequential insets in sequential parents are intially one character in size and located at the typing cursor; when imbedded in a spatial object they are one line long, centered.

4.4.2 Resizing Insets

Users can resize insets by dragging boundaries (spatial insets) or by adding to the contents of the inset (sequential insets).

An inset's desired size is the size at which it was created and saved. The inset should retain some information about this desired size and should display at the desired size whenever possible. Of course, often there will be more, or less, space than necessary for display. Insets should not "grow" to fill desired space, but should display at the desired size.

However, when there is insufficient space insets must have a means of adapting their contents in a way that will still convey useful information to the user. The method used to adapt the display is dependent upon the kind of data which is being viewed.

The distinction between *cropping insets*, *wrapping insets* and *scaling insets* is useful here. The simplest way in which insets modify their contents is cropping. Cropping simply cuts off parts of the inset so that it will fit into available space. All insets must be able to crop. With most types of insets, however, cropping is less than ideal, for the user may have trouble understanding or making sense of the cropped object.

At the next highest level, objects which are sequential may wrap their contents: this typically happens when a text document is longer than the window in which it is displayed. Wrapping and cropping tend to work best with sequential objects, those types of objects (like text) which are perceived and processed in a primarily linear way. With both cropping and wrapping, users must have some signal that more information is available, but not visible, as well as a means of accessing that hidden information.

The third method of redisplay, given insufficient space, does not have this restriction. Scaling, while it may diminish the level of detail or even readability at some level, shows the entire object but in a smaller scale. For spatial objects, scaling is usually the most sensible means of redisplay since spatial objects are typically perceived as a whole.

4.4.3 Scrolling an Inset

With both cropping and wrapping, users must have some signal that more information is available but not visible and a means to access that available but hidden information. Often, the incomplete nature of wrapped (sequential) information will be a signal that more data is available, but hidden. With cropped objects, the portion of the inset that is displayed may appear to be complete when it is not. Scrollbars can signal that more information is available. However, they take up considerable space and it may be inefficient to have them visible at all times. When visible, scrollbars can aid the user in moving through the document.

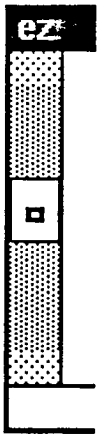


Figure 4.1

The "end zone" regions of the scrollbar allow users (via a mouse click) to move quickly to the beginning or end of a document. To move through the document incrementally, users click with the left button next to a line they wish to move to the top of the screeen. Clicking with right button undoes this action; i.e., moves the line at the top of the screen back down to the cursor.

Typically, scrollbars are positioned to the left and/or across the bottom of the workspace. Clicking in the grey area of a scrollbar causes the view within the work area to shift, the clicked position is relocated to the extreme top or left of the workspace (e.g. "Move this (clicked) line to the top." or "Move this (clicked) column to the left."). Positioning the scrollbars to the left and/or below the workspace facilitates this repositioning.

Scrollbars also furnish users with information about the size of the document relative to the portion currently being viewed, about the position of the cursor, and about the size and position of the selection (if any).

4.4.4 Moving an Inset

Typically, users will use mechanisms that they are familiar with--ie, those available in the parent--to move and position an inset.

Glossary

"When I use a word," Humpty Dumpty said, in rather a scornful tone, "it means just what I choose it to mean - neither more nor less."

Through the Looking Glass
Lewis Carroll

appeal: *n.*, the quality of inviting and encouraging use.

Andrew: *n.*, an environment where a variety of applications and insets can be used as a family rather than as individual objects. This is accomplished by providing a rich set of primitives in the forms of objects and services that programmers can use to create new applications and insets. If the guidelines presented in this document are followed in the process of assembling these primitives, the user of the Andrew system will benefit because experience gained with one application or inset is transferrable to other applications and insets. This work is being done by the Information Technology Center at Carnegie Mellon University with the financial support of the IBM corporation.

application: *n.*, a tool, or a family of tools, which allows a user to complete some task. An application may be comprised of several objects such as rasters or zip drawings. The term application is relative. For instance, complications exist with this term in the case of insets which can be used as both embedded objects and as applications. Table is such an inset: it can be embedded in a document and it can also be used as an application to perform spreadsheet operations. See also *inset*.

author: *n.*, the creator of a document or object. This distinction between the author and viewer is used to indicate the types of attributes each of these users can specify for a document or inset. See also *viewer*.

browse mode: *n.*, a mode of operation for an application where all the insets in that application are made read-only. This mode is used for viewing documents and provides two advantages over viewing a document in a the the same mode in which a document is created. First, this mode allows the user to view a document without worrying about inadvertently modifying that document. Second, for many insets, a mode that does not allow the modification of the data will be more efficient for both the system and the user as the editing operations are not needed.

button: *n.*, a small, visually distinct, area of the screen which responds to mouse clicks by initiating an action or series of actions. See also *mouse click*.

CRT: *n.*, See *cathode ray tube*.

caret: *n.*, a marker on the screen which indicates the current position in a collection of data. The shape of the caret can change from inset to inset or within an inset to indicate different modes. See also *mode* and *cursor*.

cathode ray tube: *n.*, a device that produces images by propelling a focused beam of electrons against a phosphor coated, glass tube. Many televisions use cathode ray tubes.

child: *n.*, a second order inset, an inset which is embedded within another inset. See also *parent*.

click: *v.*, the act of performing a mouse click. See also *mouse click*.

clip: *v.*, the act of discarding or altering extreme values in a set of data because the original data cannot be represented, usually due to space constraints. See also *crop*.

common inset operations: *n.*, operations which are handled by each inset individually, but which are commonly done across insets, such as scrolling or scaling; may be dependent of inset type.

communication: *n.*, the movement of information. In the Andrew system users communicate with the system using dialog devices. See also *dialog device*.

conceptual model: *n.*, the developer's understanding of the system and how it operates. See also *mental model*.

consistency: *n.*, the level of conformity among the various components of the system.

copy: 1., *n.*, a replica of some object or piece of data.; 2., *v.*, the act of placing a replication of some selected portion of data in the cutbuffer. See also *cut*, *cutbuffer*, *paste*, *replace* and *select*.

crop: *v.*, one of the actions available to an inset when too little space is provided to display the entire contents of that inset. Cropping is done by simply discarding or clipping the information that would not fit in the allocated area. See also *scale* and *wrap*.

current inset: *n.*, the inset which contains the input focus.

current position: *n.*, the position of interest in a collection of data. This is often the place where new information will be added or altered by the user.

current selection: See *selected region*.

cursor: 1., *n.*, a marker on the screen which tracks the movement of the mouse. The shape of the cursor may change to indicate the mode of operation.; 2. *n.*, in some applications the term cursor may be used when the term caret is meant. See also *mouse*, *mode* and *caret*.

cut: *v.*, the act of removing a selected portion of data from its place in an inset and placing that data in the cutbuffer. See also *copy*, *cutbuffer*, *paste*, *replace* and *select*.

cutbuffer: *n.*, the area of the Andrew system where data that has been cut or copied is stored until it is pasted or replaced. See also *copy*, *cut*, *paste* and *replace*.

data: *n.*, plural of datum.

data object: *n.*, the portion of an inset which describes what is to be displayed in that inset. See also *view*.

datum: *n.*, a piece of information.

default value: *n.*, a predetermined, frequently used, value for a data entry, intended to reduce required user entry actions.

dialog box: *n.*, a dialog device which may either come up in response to a user action (like choosing a menu item) or may be system-initiated and which presents the user with a choice of options to choose by clicking the mouse; typically, no other action may be taken by the user until the dialog box has been "answered." Example: a dialog box which asks users to answer "You have unsaved changes. Do you want to save them?" in response to a command to quit.

dialog device: *n.*, a feature for giving commands or in other ways communicating with the system; typically, dialog devices are visual and employ the mouse (menus, palettes, dialog boxes, buttons) although keystrokes are also dialog devices.

discrete selection: *n.*, a property which permits only one of a range of values to be operative at any one time.

display: 1., *n.*, an image presented to the user. This image is often the primary method of output from an application or inset.; 2., *v.*, the act of presenting an image to a user. See also *cathode ray tube*, *liquid crystal display*, *plasma display* and *output*.

document: *n.*, a collection of data. In the Andrew environment, a document contains both the structural information about a document and the raw data and insets.

double click: 1., *n.*, the operation of quickly pressing and releasing (clicking) a mouse button twice with a small delay between each click. This is differentiated by the Andrew system from clicking a mouse button, waiting some period of time, then clicking the mouse button again. Double clicks are used to select extended regions of data in many applications.; 2., *v.*, the action of performing a double click. See also *mouse click* and *mouse*.

double mouse click: See *double mouse click*.

embed: embedded 1., *adj.*, a property of a particular relationship between two object. This type of relationship is created when one object is completely contained in the other object. This type of relationship occurs often in the Andrew environment when insets are placed in documents.; 2. *v.*, the act of placing one object inside another. See also *child, parent and inset*.

end-zone: *n.*, an area at the end of a scrollbar which is used to quickly move to the beginning or end of an inset's data. See also *scrollbar and mouse*.

eq: *n.*, an inset for displaying and manipulating equations in the Andrew system.

event: *n.*, an occurance of an operation. The use of dialog devices causes events to occur. For example, the act of cutting some data is an event.

export: *v.*, the act of moving information out of an object to another object.

export menu: *n.*, a class of menus or menu items by which a user manipulates the communication of an inset or application with the non-Andrew world. Example: a menu item which allows users to import a statistics program from another system to analyze results.

fad: *n.*, an inset for displaying and manipulating animations in the Andrew system.

feedback: *n.*, system messages for the user which indicate problems or status of a task. Feedback should not be confused with output.

file: 1., *n.*, a collection of information. In computers, files are usually represented as a sequence of bytes or characters. The documents that users create, browse, or modify are files.; 2., *v.*, the act of storing information or a collection of information in such an item.

guidelines: *n.*, this document.

hierarchy: *n.*, a method of organizing a collection of items such as a set of parent-child relationships. A family tree is an example of a hierachy. See also *child*, *parent* and *sibling*.

hysteresis: *n.*, the particular property of path dependence in the relationship between a stimulus and response. This property is that of an object to maintain its current state and require a relatively large force to move to a new state. An example of an object with hysteresis is a light switch. The light switch tends to remain in the on or off position even if small forces are applied. In order to move to the other state one must press hard enough to make the switch "click" into the other state where it will remain.

image: *n.*, a picture or graphical representation of information. See also *display*.

imbed: alternate spelling of embed. See *embed*.

import: *v.*, the act of moving information into an object from another object.

information: *n.*, data which has meaning. This term in oftern used in the field of communication theory where it carries ver;y precise meaning. In the area of user interface design, the term 'information' is often interchangably with the term data. One subtle difference is that information usually implies that the data has meaning or value for the user while any collection of facts or figures devoid of perceived meaning can be data. See also *data*.

inherit: *v.*, to receive relevant information or data from a parent or other application in a hierarchy.

input: 1., *n.*, any information that the user directs at the system to create, manipulate or modify objects in the system.; 2., *v.*, the process by which information is conveyed to the system.

input focus: *n.*, the inset that is currently accepting input; indicated to the user by some visual means.

inset: *n.*, a software package which allows the display and possible modification of a certain type of information within an area (usually rectangular) on the screen. Currently available insets include text, table, zip, eq, fad and raster.

LCD: *n.*, See *liquid crystal display*.

liquid crystal display: n., a device that produces images by selectively reflecting light in different regions of the device. Many digital watches that produce black on grey images use liquid crystal displays.

manipulation: *n.*, general actions such as creating, moving, saving insets. See also *meta-inset operations.*

mental model: *n.*, the user's perception of the system and how it functions. See also *conceptual model.*

menu: *n.*, a dialog device, usually popped up via the mouse, which consists of cards arranged systematically on which are menu items which, when chosen with the mouse, cause a command to be given to the system.

menu card: *n.*, a list of similar commands appearing as one plane in a menu stack; typically, menu cards reflect functionally related sets of menu items.

menu card title: *n.*, the name of a menu card appearing at the top of a menu card; titles should reflect groupings within cards.

menu item: *n.*, specific commands within menu cards; typcially menu items are verbs and connote action; they do not generally convey system information.

menu item name: *n.*, the name given to specific menu items; the name should be brief and descriptive.

Menu Repeat Button: *n.*, a small "bull's-eye" button which appears on the first menu card after an initial menu selection. Pointing at the Menu Repeat Button with the cursor is a quick method of moving to the last selected menu item. The Menu Repeat Button has also been called a "mousehole" or "wormhole".

menu stack: *n.*, a collection of menu cards.

message-line: *n.*, many applications set aside a single line for text at the bottom of their window. This line is used to present non-critical messages to the user. For examples of the use of the message-line see the Chapter 3 on Dialog and Response.

meta-inset operations: *n.*, manipulations which are performed on the inset as a whole, such as creating, selecting or cutting an inset; usually independent of inset type.

mode: *n.*, in some applications and insets there may be different ways to display and/or modify information. These different ways of doing things are called modes.

mouse: 1., *n.*, a device used to input information to a computer. A mouse is capable of transmitting information about changes in its position and the state of buttons located on the mouse, to the computer system to which the mouse is connected. A mouse is useful for selecting an option from a menu of choices and for drawing.; 2., *v.*, slang, the act of using a mouse.

mouse-ahead: 1., n., mouse movements and button clicks that are made by the user while the system is busy doing other work. These mouse operations are saved by the system and used when the system requests mouse data.; 2., v., the act of storing up such data.

mouse click: 1., *n.*, the operation of pressing and releasing a mouse button. The cursor is placed over the desired area on the display and either the left, right, middle or some combination of these buttons is pressed and released. Clicks are used to select items from palettes, press buttons and perform similar operations in many applications and insets.; 2., v., the action of performing this operation. See also *click and mouse.*

multimedia document: *n.*, a document which can contain more than one type of object, such as text, graphics, images, voice, etc. In the Andrew system, insets are the basic objects contained in documents and multimedia documents are ones which can contain multiple types of insets.

muscle memory: *n.*, automaticity; the ability to make a motor movement without conscious attention.

natural menus: *n.*, the default set of menus which, will be used by the many Andrew users when they have acquired a basic familiarity with the Andrew system..

novice menus: *n.*, a special set of simplified menus created for new users; typically, after some experience users will employ natural menus.

object: 1., *n.*, in the Andrew system, applications, insets, primitives and dialog devices are objects which have well-defined properties and thus can be used to build other objects. Some of these properties include the ability to be created and destroyed, to be stored or retrieved from a file, and to support the manipulation of the contents of the object..; 2., *n.*, the target of some operation.

operation: *v.*, an action that is performed on an object. This action often modifies the object or the way the object is presented.

option: *n.*, one of several choices.

output: *n.*, any information that the system provides regarding the state of objects or the progress of operations on those objects.; *v.*, the process by which information is received from the system.

palette: *n.*, a dialog device which presents a range of choices for users to select; typically, a palette stays on the screen for some amount of time; a palette may contain information for users about their previous choices or the default choices. Example: a set of tools in a drawing editor.

pan: *v.*, the act of moving an area of an object's graphical representation so that various portions of that area may be viewed in a limited amount of space.

parent: *n.*, upon insertion of one inset within another, a parent-child relationship is established, where the outer inset is considered the parent and the inner, or second order, inset is referred to as the child. See also *child*, *sibling* and *hierarchy*.

paste: *v.*, the act of inserting the contents of the cutbuffer at the position of the caret. See also *caret*, *copy*, *cut*, *cutbuffer* and *replace*.

plasma display: *n.*, a device that produces images by selectively ionizing gas in regions of a thin glass chamber.

predictability: *n.*, the ability to correctly anticipate behavior.

preference: *n.*, a particular, user-selected, value of a parameter that alters the way an inset or application interacts with the user. This is one way a user can tailor the system in order to use the system more effectively.

preferences: *n.*, a file in which a user places preferences. In the Andrew environment this file is called "preferences" and is placed in a user's home directory. See also *preference*.

primacy: *n.*, the state of having importance, commanding attention or taking precedence.

program menus: *n.*, the set of menus which appear when the editor is in its normal state.

raster: *n.*, an inset for displaying and manipulating rasters in the Andrew system.

read-only: *adj.*, the property of an object which prevents that object from being altered in any way.

read-only mode: *n.*, a mode of operation for an application or inset where there is no way to alter the data of the inset(s).

recommendation: *n.*, a suggested method of handling problems which are subjective in nature, are matters of individual preference or for which further study may be necessary.

region-sensitive menus: *n.*,

replace: *v.*, the act of discarding some portion of selected data immediately followed by inserting the contents of the cutbuffer in the position from which the discarded data was removed. These two operations are carried out as as a single, uninterruptable action. See also *copy*, *cut*, *cutbuffer*, *paste* and *select*.

requirement: *n.*, a specification which must be met in order to meet the basic and minimal user interface standards developed for the Andrew system.

scale: 1., *n.*, an indication of the size of an object or the data an object represents.; 2. *n.*, a device used to provide such an indication.; 3., *v.*, transforming the feature of an object in a well defined manner resulting in a new representation of that object. One of the actions available to an inset when too little space is provided to display the entire contents of that inset at the desired size is to scale the visual representation to a smaller size. This scaling is usually performed by using a linear transformation. See also *crop* and *wrap*.

screen: *n.*, a device used to convey information to the user by the drawing of images. Usually a screen is a common cathode ray tube (CRT) such as a television screen. See also *cathode ray tube* and *display*.

scrollbar: *n.*, a device for traversing a document or inset either vertically or horizontally. Scrollbars provide information about the size of the current view relative to the whole document or inset as well as identifiying the location and size of the caret.

scroll: *v.*, to move to a new view of an object via the scrollbar.

select: *v.*, the act of identifying the portion of the data at which future operations or manipulations will be directed. The method of selecting this data varies between inset types.

selected inset: *n.*, an inset which has been selected for meta-inset operations.

selected region: *n.*, a portion of data that is to be used as the object of an operation. These regions are often specified by defining a starting and ending point of the region though it is

possible that regions could contain discrete portions of data.

selected-region menus: *n.*, the menus which appear when an object or group of objects has been selected.

selection: See *selected region.*

sequential insets: *n.*, insets where the ordering of data is crucial for understanding, for example the text inset. Sequential insets tend to be processed in a primarily linear fashion. See also *spatial insets.*

sibling: *n.*, child insets of the same parent.

single click: See *mouse click.*

single mouse click: See *mouse click.*

spatial insets: *n.*, insets where the relative positioning of data is crucial, for example a drawing inset. Spatial insets tend to be perceived and processed holistically. See also *sequential insets.*

subordinate window: *n.*, a window whose existence is tied to a process in another window.

table: *n.*, an inset for displaying and manipulating tables in the Andrew system.

tailored menus: *n.*, a set of menus, usually customized by the user and typically used by very experienced users.

text: *n.*, an inset for displaying and manipulating text in the Andrew system.

titlebar: *n.*, in the Andrew system's original window manager, wm, all windows had a bar across the top of the window. This bar is used to indicate the application in window as well as other information about the contents of the window. In this area, a special set of titlebar menus are available. See also *titlebar menus, window* and *window manager.*

titlebar menus: *n.*, when the mouse is placed in a window's titlebar a special set of menus is available that allows the user to perform operations on the window rather than the contents of the window. See also *window* and *titlebar.*

type-ahead: 1., *n.*, keystrokes that are typed by the user while the system is busy doing other work. These keystrokes are saved by the system and used when the system requests keyed data.; 2., v., the act of storing up such data.

user: *n.*, any person who uses an information system in performing their job.

user control: *n.*, when ultimate control of the system and its behavior lies with the user.

user interface: *n.*, all aspects of information system design which affect a user's participation in data transactions. The user interface can be critical in task success and user satisfaction.

view: *n.*, the portion of an inset which describes how that inset's data is to be displayed.; *v.*, to perceive and process data. See also *data object.*

viewer: *n.*, the group or individual that is reading or viewing a document. See also *author*.

window: *n.*, the rectangular area of a display allocated to an application. It is possible for one application to control many windows.

window manager: *n.*, a special application that allows a user to manipulate the various window on a display. The window manager can only perform operations on the whole window, not on the contents of the window. Examples of operations that a window manager can perform include moving and resizing windows, hiding windows, and controlling the characteristics of the input and output devices, such as keyboards and mice.

window-manager menus: *n.*, menus which are invoked by clicking the mouse in a window's titlebar or in the grey space around windows; window manager menus contain menu items for issuing commands to the window manager.

wrap: *v.*, one of the actions available to an inset when too little space is provided to display the entire contents of that inset. Wrapping is done by flowing the contents of the object from one line to the next and cropping any information that that continues past the space provided. Wrapping is usually only useful for objects that are sequential in nature such as text or equations. See also *crop* and *scale*.

zip: *n.*, an inset for displaying and manipulating drawings in the Andrew system.

zoom: *v.*, the act of displaying more or fewer details of an object. The act of zooming ofter requires more or less cropping of the visual representation to occur.

Bibliography

Hansen, Wilfred J. & Haas, Christina (1988). Reading and writing with computers: A framework for explaining differences in performance. *Communications of the ACM, 31* (9).

Mayhew, Deborah J. (1988). *Basic Principles and Guidelines in User Interface Design*, SIGCHI Tutorial Notes. ACM Special Interest Group on Computer Human Interaction Annual Meeting: May, Washington, D.C.

Microsoft Windows Software Development Kit: Application Style Guide. (1986). Version 1.03.

OPEN LOOK Graphical User Interface Functional Specification. (1988). Sun Microsystems, Inc. (Prerelease Version).

Ramsey, Rudy. (1979). *Human Factors in Computer Systems: A Review of the Literature*, Science Applications Incorporated.

Rose, Caroline. (1985). *Inside Macintosh.* Addison-Wesley Publishing.

Rubenstein, Richard & Hersh, Harry. (1984). *The Human Factor.* Digital Equipment Corporation.

Schneiderman, Ben. (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Reading, Massachusetts: Addison-Wesley.

Smith, Sidney L. & Mosier, Jane N. (1986). *Guidelines for Designing User Interface Software*, Bedford, Massachusetts: Mitre Corporation (ESD-TR-86-278).

Thomas, Frank & Johnston, Ollie. (1984). *Disney Animation: The Illusion of Life.* Abbeville Press.