Andrew Consortium
School of Computer Science
Carnegie Mellon

The Andrew View

Volume 2, Number 3
December, 1993

This month finds the Andrew Consortium staff relocated to temporary quarters in "Building D" while our former offices are being renovated. We are finishing up the C++ conversion, porting the C version to new platforms, working on printing, and thinking hard about how to do widgets. All this despite a bit of travel.

IP Address Changes

As part of our move, our machines were allocated new IP addresses. In particular:

emsworth.andrew.cmu.edu is now 128.2.203.197
atk.itc.cmu.edu is now 128.2.203.218

Please use these addresses if the machines are not accessible by name from your environment.

First individual release delivered

The annual meeting considered and accepted the proposal to allow individuals to purchase copies of the most up-to-date AUIS sources. We are proud to announce that the first such has been delivered.

My own View

Wilfred J. Hansen, Director

I reported in the last issue that I was about to present a paper at the Visual Languages Symposium in Bergen. Entitled "Andrew as a Multiparadigm Environment for Visual Languages" the paper showed how Andrew can serve as a basis for implementing multiple languages in a cooperating environment. Drop me a note if you'd like a copy of this paper.

The Visual Languages Symposium as a whole was very interesting. As usual there were both visual languages and language visualization. The most interesting papers reported actual experiments. R. K. Panday and M. Burnett showed that users found it easier to write matrix operations in their visual language than in Pascal or APL. A. Apte and T. Kimura showed that drawing graphics took half as long with a pen as with a mouse. The results in another pen vs. mouse experiment reported by V. Citrin were less clear, but the paper reported interesting observations about the number of erasures and the tendency to get into complex and time consuming copy schemes in MacDraw.

After my return from Bergen, I was invited to a workshop on visual languages at Scottsdale, Arizona, under the aegis of the Jet Propulsion Laboratory. I presented my visual languages talk there and then extended the trip to present the talk at Stanford University, Lawrence Livermore Laboratories, and Silicon Graphics. At the latter, I learned more about the state of Fresco from Mark Linton.

Fresco is the C++ successor to Interviews, one of Linton's earlier projects, and is being defined and developed in conjunction with an industry-wide group. Fresco will be a toolkit offering a collection of widgets in an environment that has some of the properties of AUIS and some new features. The coordinate space is screen independent, and there is support for color and multi-threading. Some of the basic elements of Fresco are defined and implemented, but most of the superstructure is still undefined.

At SIGCHI (April 24-28, 1994, Boston, Massachusetts) I will present a tutorial entitled "Introduction to User Interface Systems for HCI Developers and Researchers". This will discuss the Human-Computer Interface challenges and opportunities of user interface systems like AUIS, Fresco, OLE from MicroSoft, and OpenDoc originally from Apple and now from the Component Integration Laboratory (see article below). These systems offer the challenge of defining user interface schemes that extend across a wide spread of applications; they offer the opportunity to conveniently capture user interactions in order to study user behavior.

Annual Meeting Priorities for the year ahead

Discussion at the annual meeting, 24 June, 1993, was lively and established valuable priorities for the year ahead. The first priority was recognized as "Robustness": ease of use, no crashes, no installation problems. The last C release will have some of these properties, and we are working to build on them in the C++ release.

Platforms to be supported with C++ release: RS/6000, HP/PA, PMAX Ultrix, Solaris 2, Alpha, Telmat, Pmax/Mach, Sun4/Mach, SGI, Linux. We would like to consider Win32 and Windows/NT, but resources may not permit. Dynamic loading is considered below.

Widgets / Application Builder: Regularize the widget suite by cleaning up the API and the user interfaces. Improve consistency and documentation. The meeting decided that we should plan for an application builder, but that it is not a high priority. See the discussion of widgets below.

Printing. This is the year to bite the troff bullet and spit it out. The printing methods, especially text, will directly produce PostScript; every inset will have a printing method. See the discussion below.

Mail. No consensus on a mail strategy emerged. We have the MAC and PC interfaces used by CMU, but do not have manpower to exploit them. Moreover, CMU is converting to an IMAP based mail server system, which will require, at least, extensive revision of the existing messages package.

Note, however, that between the 1992 and 1993 meetings we made the commitment to convert to C++. This is requiring most of 1993, so it is likely that the projects described above will not all be complete until the end of 1994.

Release plans

Release 6.1 is available and has been picked up by most members, though under the name 6.0.1. This release supports these platforms

IBM RS/6000 AIX 3.1 and 3.2
DEC PMAX UL4 and Mach
Sun OS 4c, 411, and Mach
HP UX for levels 80 and 90 on 300, 700, and 800

We are in the process of completing ports to Sun Solaris, Linux, and Telmat. Solaris is completed except for a bug in VUI; however, the messages package fails on AFS due to an AFS bug that is corrected in the next AFS release. Members may get the Solaris or Linux version by asking for it. The Telmat version will be completed later this year. When these ports are completed, the software will be version 6.2. If there are no problems, this version will go on the X V11R6 tape; however, if corrections are needed, the X version will be 6.3.

(The sources have some support for SGI, SCO Unix, and the Dec Alpha; however, we have not tested these nor debugged them completely. The IBM RT is no longer supported.)

The first C++ version will be version 7.1 and will be released in January. (Previously I said it would be version 8, but this no longer seems appropriate.) Version 7.0 has been delivered already to one member and could be released to others. The C++ version currently has been tested on IBM RS/6000 AIX 3.1 and 3.2, HP UX level 80 and 90, and DEC PMAX Ultrix 4. Version 7.1 will support the Sun and will likely support the other platforms supported by version 6.2.

Version 7.1 will not have support for dynamic loading, but it will be added in the next version. See below for a discussion of the design.

New and Revised Features in Version 6.1

Release 6.1 is a beta version of that which will appear on the X11 tape Release 6, at which time it will be available for users everywhere. The list below of changes from version 5.2 is derived from the code-change log entries. A more complete, and less polished, listing of those entries is available on request.

User visible changes

Added **CheckOwnerHome** preference description. This value determines whether a lookup is done in order to find the home directory of the owner of a file being viewed by EZ. The file-owner's home directory is used to *fold* the full pathname down to something shorter, like ˜foo/bar for displaying in the title-bar of the EZ window. Defaults to: yes.

Fixed and expanded the **XStyleSelection** support. Added the preference CopyOnSelect. When yes, selecting text will automatically put it in the cutbuffer. (When/if any other views support the concept of "selection" in the X sense it will happen for them too.)

Get rid of annoying dialog box when you run with **ispell** 3; Added code to check to make sure ispell allows you to accept words and add them to the private dictionary (in -a mode, which is what spell.c uses.) If it does not, print an error message when the 'a' or 'i' key is hit.

Rewrote help files to encourage use of **lookz** over troff format notes.

Added documentation of the **tmacgrofffile** preference.

Figure has been revised to include printing in landscape mode. Added FigureMatteColor preferences (default white) which overrides the BackgroundColor preference for figure insets.

Added a preference, **ReadOnlyTitle**, which lets you set the "(readonly)" marker in a window's title bar to some other string.

Implemented import of **ppm** images into the image inset.

Renamed the program "prefs" to "**prefed**" so that help on the name will get the right help file. Made the default file the preferences file currently in use. Fixed so that the global.prf file will be read if it exists, and it's values will update the default values.

Tab stops are now measured from the beginning of each line, instead of the left margin. Justification/discussion available upon request.

In path.c, we added a quoting mechanism so that environment variable expansion can be suppressed. A $$ sequence in a **path** is converted to a single $. The argument to visit-file is canonicalized twice at present, so the file name '$HOME' has to be referenced as '$$$$HOME'.

Changed "**Append to [Raw] File**" options in messages to use datacat.

A few improvements have been made to **zip**: De-coupled show-grid and constrain-to-grid so you can see the grid independently of being locked onto it. Or you could constrain your drawing to the grid without waiting for it to redisplay all the time. Changed the algorithm for redrawing the grid to display the grid points correctly more often. Added setting of a print size which was carefully chosen not to break reading by older versions of zip. Added commands to specify the print size in inches and points. Changed zip to come up in absolute mode by default so that what you see is much closer to what you get. Set the initial print size, by a heuristic: PrintSize overrides ViewWidth/ViewHeight overrides ObjectWidth/ObjectHeight.

System Maintainer visible changes

The code has been revised throughout to increase gcc and ANSI compatibility.

Ports have been incorporated for **DEC Alpha, Sun Solaris, Linux, HP/UX 9.01, and System V.** Most of these still have some problems because we are unable to test them fully.

Added compile and installation of **genstatl** and associated files. These provide for building AUIS binaries that do not utilize dynamic loading.

LINKINSTALL_ENV is no longer the default. Its use is being discouraged.

Programmer visible changes

Had to add **super_LinkTree** as first line of LinkTree in several cases-- if you don't have this, an inherited colormap will be set improperly.

Added **SetPatternOrigin**() method to atk/basics/common/graphic; this allows programs to align their grey-stipples however they want. Added empty, override-able version of graphic__ReadImage. Added empty stubs for SetBGColorCell and SetFGColorCell.

Added **ResizeWindow and MoveWindow** methods to im class.

Added **IsObserver** method to the observable class.

New documentation for the **traced** class (traced.doc).

Added documentation explaining how to write **preference descriptions** for the preferences editor (writing.prs).

Added a programmer's interface to **datacat**.

Added **AlwaysWrapViewChar** to wrap a viewref environment with a particular type of view and a particular dataobject, around an already existing viewref char.

Introduced two new methods for text: {Always,}**CopyWithAllStyles**. This is Fred's version of CopyText which copies all the styles enclosing the region being copied.

Fixed **Ness** to use CopyWithAllStyles. Now a copy of text will look exactly like the original.

Modified the class preprocessor to allow machine independent trampoline code (written in C).

Renamed **regexp** functions to reg_*.

Dynamic Loading and C++

One of the most useful features of AUIS in C code--at least for program development--has been dynamic loading. We are implementing dynamic loading for the C++ version, but there will be some restrictions. Here are some of our considerations.

Desired features

0. No compiler modifications.
1. Automatically load class code on demand, and not before.
2. Quick compile-link cycle for development.
3. The ability to develop objects which can be used with a runapp which has no knowledge of them.
4. Minimal run-time overhead for calls to dynamically loaded code, and for calls from dynamically loaded code to statically loaded code. (After the first time through.)
4. Support for global object constructors/destructors.
5. Retainment of the "debug-ability" of the code. (As long as the debugger supports dynamic loading.)
6. Minimal dependence on the symbol name mangling technique.

7. Retainment of the C++ style of function calls. (this->Blah())
8. Portability

Deficiencies of available options

The AUIS C code dynamic loader relies on a preprocessor for the header files to define a macro for each method and static member function. For the latter, these macros initially call a piece of assembly language trampoline code which saves the register state, loads and registers the class, replaces the function pointer with a pointer to the real function, and finally calls the actual function. For C code, this approach makes function calls look normal, but the C++ syntax is different and cannot be emulated with macros.

One possible approach with C++ is to generate a stub function for each function to be dynamically loaded. This stub would load the function, if needed, and then call it. Several problems arise in contemplating this approach, beginning with the difficulty of parsing C++ header files to generate the appropriate stubs. The fact that function names are mangled in some implementations is another complication. In any case, there would be an extra overhead for the call through the stub function.

Design chosen: Inheritance

To avoid the many problems, we have chosen a design that relies on inheritance. For each class C which is to be dynamically loaded, there must be a base class, CBase, which declares all the methods of C as virtual methods. CBase must have a static member function for creating objects of class C; this latter function will call on the system mechanism that dynamically loads the class implementation. In practice, ATK's view and dataojbect classes serve as CBase for all views and data objects, so there is no additional coding effort to implement new, dynamically-loadable objects of these classes.

Comments and questions on this approach are welcome.

Printing direct to PostScript

The new printing architecture will print directly through PostScript, instead of via troff. One major effort will be implementation of line- and page-break algorithms, but considerable effort must also go to general organization.

A print will now be initiated through the print::GeneratePS function,

```
static boolean print::GeneratePS(FILE *outfile, view *viewtoprint) ,
```

which sends a complete PostScript document to *outfile*. It initializes the printing process, and then calls *viewtoprint*->PrintPSDoc(), which causes *viewtoprint* to print as a top-level document.

The print class maintains a set of variables which affect how the document will print: landscape / portrait mode, scaling, printing multiple copies, printing a range of pages, output to file or printer, printer name, and view-dependent options such as *print-footnotes-as-endnotes and print-table-of-contents*. With the exception of printer name, each view has its own copy of these variables.

The PrintPSDoc method for a view is entitled to consider the entire page as its own and to generate PostScript for as many pages as it likes. When a parent view wishes to print an inset view, it first calls the inset's GetPrintInterface method to determine how it wants to be handled. At the moment, three printtypes are defined:

> "**generic**": The inset is given a rectangle, and draws itself in that rectangle.
>
> "**styledgeneric**": Like "generic", but the parent provides information about the current font and style.
>
> "**text**": The inset adapts itself within the parent text. It may provide some particular text data which the parent textview will stitch into the text stream it is laying out (textref, link, and timestamp do this, for example.) It may call back some special methods of textview which change how the textview is laid out (pagebreak, header, footnote). It may do stranger things.

The low-level text printing mechanism must be given a "layout plan"--a list of how wide the page (or column) will be at each point, and where logical breaks occur. (Logical breaks are the points which kept-together segments must not cross. They also mark where footnotes are to be inserted.) The mechanism then generates a data structure which represents the formatted text. This low-level structure contains everything one needs to print the lines of text: words, word positions, inset positions, fonts, underline positions.

The textview::PrintPSDoc method uses this low-level structure to lay out the text on pages. Its layout plan is a constant width (the page-width) with logical breaks at regular intervals (the page-height). If the textview is being printed as a "generic" inset, the low-level structure is drawn into the printing rectangle. The layout plan is the same constant width, but there are no logical breaks. If the textview is being printed as an inset with the "text" printtype, the low-level structure is not used at all; instead, the text object is laid out in the parent's data structure, as if the child view had been cut and pasted into the parent. (I think this capacity is important, but I don't know of any current application for it. Things like annotations usually want to be printed in rectangles, the way they appear on the screen.)

The low-level structure encodes all the formatting information implied by current AUIS styles, including:

> multiple fonts, font styles, font sizes

insets that produce rectangular images on the page
insets that produce text which is inserted in the text stream being printed
indentation, line spacing, paragraph spacing
baseline adjustment (super/subscripts)
justification, margins, tabs
color
foreign characters (ISO 8859)

The following features are implemented by generating appropriate entries in the low-level data stream:

footnotes
index, table of contents information
page numbering, headers/footers
page breaks, setting the page number at a page break
blocks of text which should always be kept together

Several features are not yet planned for implementation:

multiple-pages- per-physical-page
multiple columns
flowing text around insets
texttag, textref (for cross-references by page number)
leader dots
embedded user-written PostScript code
print-time computations
local spacing control
overlaying characters
hyphenation
blocks of text that must not begin near the bottom of the page

It may be that other organizations of the print machinery could make it more expressive. If you have a suggestion, please let us know.

Widgets

In the most general terms, a widget is an image component available for use as an element of the image for an application. Typical widgets are buttons and sliders, but text and drawings can serve as well. A widget presents an on-screen image and interacts in some way with the user. Currently the most well known and widely used toolkits are based on Xt so generally the term "widget" is used in relation to an Xt widget. However, the Xt concept of a widget is quite limited relative to Interviews or AUIS in that Xt provides no generalized protocols for maintaining widgets in relation to a set of data (ala the NotifyObservers and view/data separation in ATK.) Accordingly, AUIS widgets will have considerably more functionality than a typical Xt widget.

ATK currently has at least 3 1/2 widget sets: ADEW, Apt, Prefed, and various assorted insets. All have deficiencies. Most offer few options to the C programmer and almost none to the extension language author. The prefed objects are parochial to the preferences editor. Apt provides the best interfaces, but lacks user customizability and is limited to character strings for the most part. ADEW is probably the most complete of the widget sets, but ADEW's interface is restricted to a long value, a string, and an index into an array of strings. All of the available insets lack the degree of customization which has been customary in Xt based widget sets. (The closest to Xt style is "sbutton", which offers controls for the shadow coloring and spacing, but in Xt it would also provide for setting the button label...)

Requirements - User apparent

- Cut/paste/selections
- Search-able
- Persistent storage
- Mail-ability
- Printing
- Internationalization
- Attractive and intuitive default look and feel
- Optional Motif look and feel
- Keyboard control of all functions (in addition to mouse control)
- Continuous and clear feedback on application state

Requirements - Author (Constructing applications using widgets)

- Customization of look and feel
- Full access and control via a meta-x type interface and an extension language

Requirements - Developer (Extending widgets or building new ones)

- Extend NotifyObservers and WantUpdate to deal with partial modifications
- Multiple views on the same data
- Clean, consistent and easy-to-use programmatic interfaces
- Ease of writing new insets which support the standard protocols
- Generally applicable insets for most common application needs

Requirements - Optional

- Inter-operation with other X applications
- Drag and drop interfaces

Widgets planned

High level widgets and substrates (existing and little changed):

- eq - equation inset
- fad - rudimentary animation editor

figure - drawing editor
layout - inset for arbitrary layout of enclosed insets
lset - display two adjacent insets
ness - extension and string processing language
org - display and edit hierarchies
page - allow flipping between pages
raster - editor for monochrome bit-mapped images
table - table / spreadsheet inset
text - text, document, and program editor (the heart of ez and other applications)
image - display images

Other existing widgets that will remain unchanged. These are seldom used as applications.

clock - analog clock
header - specify document headers and footers
link - hyper-text-like link
month - display a one month calendar
note - annotation
timeoday - digital clock
writestamp - time file was written

The following lists of widgets are adapted from the list of widgets available in Motif.

Input widgets

ArrowButton - arrow w/ drop shadow
ColorChoice - select a color
DateChoice - select a date and time
DrawingArea - null widget
DrawnButton - button with only an outline; application provides image
FileSelectionBox - select a file
FontList - list of fonts
Integer- input an integer via scale and/or text entry
Label - simple text label button
List - choose from list of strings
OptionMenu - menu which pops up from a button
PathList - display a list of paths; allow editing
PushButton - pushbutton
Scale - select a value from a range. slider, text, thumb-wheel, ...
ScrollBar - scrollbar
SelectionBox - select from a list or type in text
Separator - empty, but draws lines
TextResponse - a small text entry area

ToggleButton - on/off

Output widgets

BarGraph -- display a value as a bar
Dial -- display a value as hands on a dial
EKGGraph -- scaled line graph
Gauge -- dashboard gauge (half a dial)
Indicator -- display text or icon

special-purpose, defined-format dialog boxes:

ErrorDialog - box noting an error
InformationDialog - box for displaying information
QuestionDialog - box that asks a question
WarningDialog - box for warning
WorkingDialog - box explaining that action is in progress

compound widgets

BulletinBoard - simple widget container. base widget for most dialog widgets (layout)
Command - essentially a typescript for connection to an application (typescript)
DialogShell - contains a widget and displays it as a dialog box ()
Form - tiled layout widget container (figure, augmented)
Frame - decorations around single child (matte, sort of)
MainWindow - standard main window:
menubar, command window, work area, scrollbars (frame)
PanedWindow - vertical tiled format (constrained, smart mono-dimensional lset)
Pager - display multiple widgets in same space
RadioBox - multiple buttons, only one of which may be pressed
RowColumn - tiling for menus (lset)
ScrolledList - List with Scrollbar
ScrolledText - Text with Scrollbar
ScrolledWindow - scrollbar(s) with work area (scrollbar)

In addition, it would be valuable to convert the existing pop-up and pull-down menus to widgets so they would behave like other insets.

Before going on to an architecture for widgets, a short digression on resources is appropriate. Resources and resource management were adapted into X from the Macintosh software. They aid in nationalization of software and in otherwise generally making it easier to adapt to local and personal needs and taste. Thus one resource for a widget might be the contents of a message and another the color of the background. The resource architecture provides for both general and specific targeting of a resource specification; the color may be set for all instances of some widget class or only for instances within a certain application or subtree. For example the text might be set for the widget instance named "showResults" in the "Text" widget in the "Radiologist" application.

Widgets can be tailored in so many ways by resource files that one is forced to ask whether a widget can be specified *completely* in a resource file. I believe the answer is yes. What is needed is to add to the resource language the means to specify the image and the functions to be performed in response to user actions on that image. The nub of the argument is that

> images are specified with a language that is
> interpreted as images by the figure inset
> functions are specified in Ness, C++, or some other language

Note that this results in machine independent dynamic loading: the widget description is interpreted from an ASCII file.

Image Specification

This proposal hypothesizes that images can be described to the figure inset in four ways: drawing, C++ code, Ness code, or a display list notation. For example, consider a dialog box containing a text and two buttons. This would be drawn by selecting the text tool and placing text in the image; and then selecting the button (or object) tool and inserting the buttons at the desired location. If desired to tie sizes to the box size, the constraint tool would be used. In C++ code, the image is created by calling a series of functions which insert objects and set their size and position. The Ness code would be similar, although possibly simpler. Ideally, it could be as simple as:

```
errorMessage: text(20, 20, 150, 30, "Fatal error")
abortButton: button(30, 80, "Abort")
okayButton: button(60, 80, "Okay")
```

The integers in these expressions are coordinates and sizes of the image. Since the code is Ness, any arbitrary expression can be used to set the coordinates. The name before the colon is a name by which the object can be later modified; for instance, the message can be changed with

```
setAttribute(errorMessage, msgfield, "Non-fatal error")
```

The display list notation would be the same or similar, but would not appear in the context of a Ness script. Possibly the expressions allowed in attributes would be restricted, or possibly Ness could be used to evaluate them.

Event Handlers

The event handler attributes of a view can be expressed in C++ code or in Ness, or possibly in some other language. For instance, in Ness the handler resource for left mouse up might be

```
on event "left mouse up"
        ...
end event
```

The notion is that this text would appear as the value of an attribute in a resource file. To match resource file conventions (unfortunately) each line would end with a backslash.

Work Scope

Revisions are needed in many areas to realize this widget approach:

system - arrange to get insets from resource files in addition to code files
attribute lists - develop as an adjunct to views and data objects
figure - modify to interpret display lists and to utilize resource manager
ness - add notation for expressing images simply

Once the basic changes are in place, the collection of widgets must still be implemented in terms of the new tools. It would be faster to simply implement the widget set in C++ code, but then the results would be far less flexible. Over the next two months, we will be evaluating whether the additional flexibility justifies the effort. Comments are welcome.

CIL and OpenDoc

A new group called the Component Integration Laboratories (CIL) has been formed by Apple, IBM, Novell, Oracle, Taligent, WordPerfect, Xerox, and others to "Speed Development of ‘Plug and Play’ Software Interoperability Across Multiple Computer Platforms." The formation announcement goes on to say

> The purpose of CIL is to establish, promote, and certify specifications and reference implementations of key technologies that integrate information and media from many applications within a networked environment. It is intended to develop certification processes for software products that conform to CIL specifications and documentation, as well as training for developers who want to use CIL technologies. In addition, CIL will serve as a distribution center for contributed software, design discussions and technology definition and evolution.

CIL offers four components at the present time:

> o The OpenDoc software architecture for Windows, OS/2, Macintosh and UNIX to enable embedding of features from different applications into a single working document.

o Bento for the storage and interchange of multi-media information.

o The Open Scripting Architecture (OSA) for the coexistence of multiple scripting systems.

o IBM’s System Object Model (SOM) architecture, a highly efficient platform and language-independent runtime mechanism for dynamic object linking.

More information is available via anonymous ftp in directory pub on host cil.org or by calling the offices at (415) 750-8352.

The OpenDoc proposal comes close to offering what AUIS has had for the past five years. Among the files at cil.org is a description of Open Doc from which the following description is adapted.

- - -

The OpenDoc architecture is designed to enable the construction of compound, collaborative, and customizable documents, which are interoperable across platforms and with other compound document architectures such as Microsoft’s OLE 2.0. It will be an open architecture, with source code available to vendors who want to implement the architecture in their products. The OpenDoc architecture is also flexible, providing replaceable facilities so platform vendors can implement their desired feature set.

Major concepts of the architecture include the following:

Documents - OpenDoc fundamentally changes the meaning of the term document. In today’s computing environment, a document has a type, which is tied to the application that will let the user view, edit, and print its content. With OpenDoc, a document is no longer a single block of content bound to a single application, but is instead composed of smaller blocks of content, or parts.

Parts - Parts are the fundamental building blocks in OpenDoc, replacing today’s monolithic applications with smaller units of content dynamically bound with related functionality. OpenDoc parts may be viewed in a number of ways:

= Content containers - These correspond rough to ATK data objects.

= Part editors -   These correspond to ATK views. OpenDoc parts will allow developers to create new applications in a manner similar to that of constructing a document template in today’s world.

= Frames - Frames within OpenDoc are areas of the display that represents a part. Frames provide a handle onto parts, allowing them to be manipulated as a whole, as well as allowing the user to see and edit a part’s contents. ... [A] frame can often show only a portion of the entire content of a part. Opening a large part into a window allows its the entire part to be viewed and edited.

Part handlers - When a part is being displayed or edited, a part handler is invoked to perform those tasks. A part handler is responsible for the following things:
=Displaying the part both on the screen and for printing purposes.
=Editing the part. The part handler gives the user access to edit and script the part.
=Storage management (both persistent and runtime) for the part. The part handler must be able to read the part from persistent storage into main memory, manage the runtime storage associated with the part, and write the part back out to persistent storage.

Part handlers are dynamically linked into the runtime world of the document, based on the part types that appear in the document. Because any sort of part might appear in any document at any time, the part handlers must be dynamically linked to provide a smooth user experience.

- - -

The Andrew Consortium staff will continue to evaluate OpenDoc to determine what interfaces and opportunities it offers which may be of value to Andrew users and Consortium members.

The Andrew User Interface System

The Andrew User Interface System (AUIS) is a portable user-interface environment and toolkit that runs under X11. It provides a dynamically-loadable object-oriented environment wherein objects can be embedded in one-another. Thus, one could use our 'generic-object' editor (ez) to edit text that, in addition to containing multiple fonts, contains embedded raster images, spreadsheets, drawings, equations, simple animations, etc. These embedded objects may themselves contain other objects, including text. The release includes many objects, including those mentioned above, along with a help system, a system monitoring tool (console), an editor based shell interface (typescript).

The underlying Andrew Toolkit architecture (ATK) supports not only screen display, but also file storage, cut/paste across windows, an application construction environment, an extension language, and printing. There is full support for programmers to create new objects and new applications.

The Andrew Message System(AMS) is a component of AUIS and provides a multi-media interface to mail and bulletin-boards. AMS contains many advanced features including authentication, return receipts, automatic sorting of mail, vote collection and tabulation, enclosures, audit trails of related messages, and subscription management. It also provides a variety of interfaces that support character-based terminals and low-function personal computers in addition to high-function workstations.

The following are components are available in version 5.2 and later versions. Some respond to frequently requested applications in X: word processor: ez, drawing editor: figure, mail and news reader: messages, font editor: bdffont, documentation presentation: help, directory browser: bush.

AUIS editors that are both applications and insets

eq - equation inset

fad - animation editor
figure - drawing editor
layout - inset for arbitrary layout of enclosed insets
lset - display two adjacent insets
ness - extension and string processing language
org - display and edit hierarchies
page - allow flipping between pages
raster - editor for monochrome bit-mapped images
table - table / spreadsheet inset
text - text, document, and program editor (the heart of ez and other applications)

AUIS applications

bush - directory browser
chump - schedule maintainer
console - shell interface / terminal replacement
ez - word processor and program editor
bdffont - font editor (for fonts in bdf format)
help - documentation browser (includes all AUIS help files)
launch - provides a menu of AUIS applications
messages - mail and news reader, manager, and composer
pipescript - viewer useful as stdout
prefed - preferences editor
sendmessage - application for sending a message or news posting
typescript - shell interface (terminal substitute)

Commonly used non-interactive AUIS applications

datacat - concatenate ATK files
ezprint - print an ATK document
nessrun - runs a Ness script (the extension and string processing language)
preview - preview an ATK document on screen

File format converters

from ATK to: RTF, ASCII, PostScript, troff
to ATK from: ppm, Scribe, RTF, troff, X window dump
convertraster - various raster formats

Embeddable insets not usually used as applications

clock - analog clock
eq - equations
fad - rudimentary animations
header - specify document headers and footers
image - display images
link - hypertext-like link
month - display one month calendar
note - annotation
text - the text object itself
timeoday - digital clock

writestamp - time file was written

Editing tools

compchar - type characters for European languages
compile - support compilation and error review
complete - filename completion for typescript
dired - directory browser
dsearch - dynamic search
ezdiff - compare two ASCII files
filter - process a region of a document through a shell command
isearch - incremental search
lookz - style table editor
spell - spelling correction (uses ispell)

Source text editing tools for

assembler, C++, C, Lisp, man pages, Modula, Modula-3, Pascal

Tools for constructing insets and applications

arb - ADEW application builder interface
bison - GNU parser generator (modified)
class - preprocessor for ATK object system header files
createcon - ADEW tool to generate C code for interface
createinset - create a prototypical inset under a new name
doindex - classC postprocessor
genmake - Makefile generator for classC
gentlex - lexical analyzer generator
makedo - classC post processor to create objects
runadew - ADEW application builder
whichdo - find an inset on the CLASSPATH

Additional insets available for building applications

button, display text string, entry of labeled text string, four position switch, multiple buttons, multiple sliders, multiple string entry, on/off switch, slider, slider controlling array of strings, text list, thumb knob

Consortium Services and Offerings

For information about services and offerings of the Andrew Toolkit Consortium please contact us at:

Information Requests
ATK Consortium
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
USA
phone: +1-412-268-6710
info-andrew-request@andrew.cmu.edu

We offer:

Memberships at three levels: Full, Contributing, and Associate
Source tape
CDrom with both source and executables
Bibliography
Copies of papers
Videotapes
Conference proceedings

Sources and binaries are available online via anonymous ftp from

emsworth.andrew.cmu.edu (128.2.203.197)

where the full CDrom release is in the ./cdrom directory. See the README there. Another ftp site is

export.lcs.mit.edu

in directory ./contrib/andrew; this is cloned on many servers, world-wide. On the nationwide AFS file system, AUIS is available in

/afs/cs.cmu.edu/project/atk-sm-X.V11R5/cdrom/

Remote Andrew Demo service:

finger help@atk.itc.cmu.edu (128.2.203.218)

News groups:

info-andrew+@andrew.cmu.edu

This is a distribution list offering the news with full AUIS formatting; send subscription requests to info-andrew-request@andrew.cmu.edu. For newsgroup in plain ASCII, see

comp.soft-sys.andrew

Andrew has been successfully used on (at least) these platforms:

IBM: RT AOS 3.4, RT AIX 2.2.1, RS/6000 AIX3.1, PS/2 AIX1.2
SUN: Sun3 3.5, Sun3 4.0, Sun4 4.0, Sun3 4.1, Sun4 4.1, Sun4 Mach
DEC: Vax Ultrix 3.1, Vax Ultrix 4.2, Vax BSD, DEC MIPS, Pmax Mach
other: HP, SCO I386, SGI IRIX 4.0, Apollo, Macintosh II MacMach, 486 Mach

Send requests to

info-andrew-requests@andrew.cmu.edu

Send bug reports to

info-andrew-bugs@andrew.cmu.edu