21 September 1995

Andrew Consortium
School of Computer Science
Carnegie Mellon

Extensible Widgets for Andrew

Fred Hansen
Director, Andrew Consortium

Abstract: No existing Andrew widget set is routinely used for creating new applications, perhaps because of limited flexibility and far fewer setable attributes than from sets like Motif. This paper describes an approach to extensible widgets. The goals are to standardize modification and addition to the attributes of a given widget and to make it possible to build new widgets by starting from an existing one as a base.

Reviewing for this paper, I found that every single Andrew View newsletter has either discussed the need, or described plans, for a new widget set.
(WidgetMentionsInNewsletter)
The time has come.

Recently we have begun implementing a widget driver, as described in this paper. The design is based on experience gained in thinking about widgets for the course I taught this Spring, as described in my article in the most recent Andrew View.
(Andrew View, Summer 1995)

The design remains one where widgets are primarily described in widget resource files. The executable code can be written in either C++ or an extension language. The image is a figure, as possibly modified dynamically through tools provided to the code.

Source code naming

We have decided to give classes capitalized names, but store them in files with all lower-case names. (ATK.C has been slightly modified to implement this.)

Resources give Attribute values
Attribute values stored in Slots

Resources are represented in files having the usual X11 format for resource files, though there is not yet a dependence on X11. (We will introduce that dependence if we introduce inheritance of attributes by using the X resource data base mechanism. We will try to encapsulate this to aid porting to other systems.

When read in to a widget object, attribute values reside in "slots" associated with the widget. Slots can be declared in the object so they can be accessed with normal C++ syntax or methods, but they can also be added to existing objects at resource-read time, or during execution. In the latter case access is via a table lookup. (C++ does not provide run-time name binding.)

Each slot is an object in a class derived from ASlot, a base class which defines most of the slot access machinery. ASlot derived classes correspond to types, of which those currently defined are ASlotInt, ASlotReal, ASlotFunction, ASlotPtr, ASlotString, ASlotATK. Coming soon are ASlotDataObj and ASlotFigure.

A typical .H file will declare an attribute thus:

something.H

```
class something : AWidget {
      . . .
      ASlotInt height;
      . . .
      // if 'height' is private,
      //   here are methods to export it
      inline int GetHeight() {
            return height;
      }
      inline void SetHeight(int v) {
            height = v;
      }
}
```

The .C file in initializing an object instance must inform the system of the name of each slot. (We are considering instead a table based approach.)

something.C

```
something::something() {
      . . .
```

```
        Baptize(height, "height");
        . . .
}
. . .
something::Read (...) {
        . . .
        // get widget class name from stream
        FetchInitialResources(widgetclassname);
        . . .
        // may later find a value for 'height'
        //    in the data stream itself
}
```

Assignments to and accesses from height can be made as shown in the inline methods of something.C. (If height is public it can be accessed directly by name or modified with an assignment. If private, the GetHeight and SetHeight methods are used.)

If s is of type class something *, a string attribute, say flameColor can be added to it as a slot which can be given values. Access to the flameColor slot is more arcane since there is no C++ support:

other.C:

```
{
        something *s;
        slotname(flameColor);  //macro makes
                    //  flameColor into a slotname
        . . .
        s->Set(flameColor, "bright red");
              // Set adds the slot if absent
        . . .
        char *fcol = s->Get(flameColor);
}
```

An ASlot value is just five words:

the value (two words to allow for *double*)
the name of the slot (for access from extension languages)
pointer to a function to be called to set the value (usually NULL)
flags

The ASlot class itself defines quite a bit of the machinery on slots, but subclasses must define two sorts of operators, one for assignment to the slot and the other for converting the stored value to an assignable C++ value. For instance, consider ASlotString which stores a char * value. ASlotString defines these value operators:

```
operator const char ()
        // returns first character
operator const char *()
        // returns ptr to the value
ASlotString &operator=(const char *s)
        // copies s and stores a pointer to the copy
```

```
ASlot &operator=(const ASlot *s)
      // checks that s can convert to char *
      //    and then copies that value
ASlotString &operator=(const char s)
      // makes the value a pointer to
      //    a one character string
```

Subclasses also define ReadValue and WriteValue methods which translate between the stored value and a string representation. These are used when saving a widget to a datastream.

AWidget dataobject

The AWidget dataobject is fundamentally an ordinary dataobject, but provides for storing arrays of ASlots and for reading attribute values from resource files. The Baptize, Set, and Get methods support access by an extension language and the addition of new slots to existing objects. The FetchInitialResources method fetches resources from the resource files for whatever resourceclassname has been given to the AWidget. The FetchResources method reads resources from a particular file.

AWidgetView

AWidgetView is the widget driver for most simple widget classes. It performs all the actions of a usual view, but uses predefined slots in AWidget to supply parameters for its operations. For instance, the slots desiredWidth and desiredHeight are supplied as responses by the AWidgetView::DesiredSize method.

Image display is done by two function slots and a slot holding a figure. The AWidgetView::FullUpdate method first executes the function in the fullUpdate slot and then fetches the screenImage slot, which holds a figure. It draws that figure in its assigned screen space. Later when the AWidgetView::Update method is called, it calls the function in the update slot and then again fetches the screenImage figure and displays it. If the screenImage pointer has not changed, AWidgetValue does not redraw the entire figure, but instead calls the figureview's Update method.

User input is primarily via mouse operations. To support this, six function slots are defined: mouse{Left,Right}{Down,Move,Up}. These are called when the given mouse operation occurs.

Summary

With the AWidget and AWidgetView drivers, it should be quite straightforward to devlop new widget classes, including those needed for HTML. As experience accrues, the driver classes can be made to provide ever greater support for widgets, making their creation progressively less painful.