Volume 2. Interface Guide

Contents

1. Introduction to Screen Management

Getting Started

A+ provides a straightforward, easy to understand scheme for visually representing variables in a variety of ways, including graphs. Compound representations can be formed for collections of variables and used as screen interfaces for applications. The connection between the values of A+ variables and their visual representations is maintained by A+ so that the two are always in agreement: a change to the value of a variable will be immediately visible on the screen, and a change to the screen will be immediately reflected in the value of the appropriate variable. (If, however, a mapped file is displayed in one process and a change to that file is made in another process, the change will not be reflected automatically in the display, since the displaying process is not notified of changes made to the file in other processes.) As a simple example, consider:

```
     $load s     ⍝ All the screen management functions are in s;
     b←⍳2 3      ⍝ the principal six are loaded into the root context also.
     `b is `array
     show `b
```

Run this example and you will see a display of the array `b` on the screen. Change a number on the screen and then examine the value of `b` in the A+ session; it has changed accordingly. Change an element of `b` and note the corresponding change to the screen display. And the same consistency between the view of `b` on the screen and value of `b` in the A+ session is maintained for any visual representation of `b`, even a graph. Not only are changes to values reflected in changes to graphs, but when points on a graph are dragged to new positions, values change accordingly.

(At various points throughout this chapter examples of visual displays will be discussed, and it is assumed that the reader is actually running these examples and has the displays available for reference. There are tutorial scripts available under the "Tutorials" item on the "On-Line Documents" section of the home page. They are ASCII text files; use Emacs to access them and the **F2** key to execute the lines of code in them.)

The phrase `` `b is `array `` in the above example specifies the way in which the variable `b` will be visually represented, in the display format called *array*. Array is said to be the name of a *class* of A+ objects. It is also said that:

- the phrase `` `b is `array `` *binds* the variable `b` to the array class;
- `b` is *bound to the class array* when the phrase `` `b is `array `` is executed.

The term *object* b refers to a variable b which has been bound to a class. The terms *visual class* and *display class* will be used for *class* when the emphasis is useful.

Not all variables can be bound to the array class, and usually there is more than one class to which a variable can be bound (but it can be bound to only one class at a time). Each class has it own visual appearance. For example, a character vector can be in the array class:

```
     c←"Visual Displays"
     `c is `array
     show `c
```

or in the label class:

```
     d←"Visual Displays"
     `d is `label
     show `d
```

A numeric matrix, however, which can be bound to the array class, cannot be bound to the label class.

```
     m←⍳2 3
     `m is `label

⍝      !!  .m: variable cannot be bound to label
```

There are several display classes of compound, or *container*, objects, by which complex displays can be created by arranging collections of objects of simpler classes. The layout display class is one of these.

```
     w←()         ⍝ Initialize a layout.
     `w is `layout
 `.w
     x←"Array Format"
     `x is `label
 `.x
     w←w,`x       ⍝ Add label to layout; context assumed same as container's.
     y←⍳2 4
     `y is `array
 `.y
     w←w,`y        ⍝ Add an array to the layout.
     show `w
```

The display of `w` should suggest the potential power of the layout class, but it also reveals some characteristics of displays that one might like to modify. For instance, in this example the values of `x` and `y` may convey all the information needed, and so it is preferable to remove the variable names `y` and `w` from the display. The characteristics that can be modified are called the *attributes* of the display class. Each class has its own set of attributes; some are shared with other classes, while others are specific to that class.

Continuing with the above example, the variable names, `y` and `w`, are the default values of the attribute called *title*, which is an attribute of both the array and layout classes. A new title can be specified by modifying this attribute:

```
     `w has (`title;"New Title")
```

Or the title can be removed by specifying it to be empty:

```
     `w has (`title;"")
```

Similarly, the title can be removed from the object `y`:

```
     `y has (`title;"")
```

Just as attributes can be altered, their current values can be determined. For instance, to find out what background color is used for the label in the above example:

```
     `bg of `x
```

```
<   `grey
```

And to change it:

```
      `x has (`bg;`yellow)
```

Another way to alter the behavior of the screen management system is to redefine various global variables in the s context. These variables are discussed at the appropriate places throughout the chapters that follow; in particular, see "The s Context".

There are now several displays on the screen, and in an application one might want to control potential screen clutter by specifying what is seen and unseen. A+ provides two functions for doing that. It is possible to hide one or more of the displays, perhaps replacing them with icons:

```
      hide `w
```

The hide function is an inverse of *show*. There is also an inverse to the class-binding function *is*, called *free*, that releases an object from its specified class and has the side effect of removing its display from the screen:

```
      free `d
```

Note that in the case of *hide `w*, the variable *w* is still bound to the display class layout; it is simply not visible. Showing *w* again and getting back the original display verifies this:

```
      show `w
```

However, in the case of *free `d* the variable *d* is no longer in the label display class; if the variable *d* is shown again it will be bound to the array class by default, and its display will look different from before:

```
      show `d
⍝     .d: variable bound to `array
```

All the basic primitive functions for visual representation, namely *is*, *free*, *show*, *hide*, *of*, and *has*, have now been shown. (These functions are put in both the root and the s contexts by *$load s*.) Examples have been shown of several display classes, namely array, label, and layout, and two attributes, title and bg (background color). Complete definitions are given in "Display Classes", and in the chapters that follow.

Controlling Which Screen the Windows Are Displayed On

A+ does not (yet, at least) provide a means to let a single process write to several X displays. You can, however, choose which display your s windows appear on. If, for example, your machine, "mymachine", say, has two displays and you want to show your s windows on the second of them, you can execute

```
      sys.setenv{"DISPLAY=mymachine:0.1"}
      $load s
```

Note that you must set DISPLAY before you load s.

Default Behavior

The primitives for visual display often report information in the A+ session when executed, such as binding class assumptions. This information has been mostly left out of the simulated terminal sessions shown here. Whether or not messages appear is controlled by the s context variable `s.QUIET`. The default value is 0, which causes all messages to be printed. The value 1 causes only error messages to be printed, suppressing any warning or informational messages, and 2 only messages for severe errors. No messages are printed if the value is -1. In addition, the behavior of s context functions when errors occur is controlled by the s context variable `s.ABORT`, whose default value is 1, meaning that error messages will be displayed (if `s.QUIET` allows) and function execution will simply end when error conditions arise. The value 0 means that errors will be signalled and execution suspended (see "Signal").

Example 1. Manipulating Layouts

If you are executing the above layout example, at this point you should have on the screen a label object `x` and an array object `y` inside a layout object `w`. Now append a view object to `w`. (A view is simply a character matrix.)

```
    s←⊤`name `address `phone
    `s is `view
    w←w,`s
    show `s
```

It is worthwhile to experiment with respecifying `w` as `` `s `y``, then as `` `y `s `x``, and so on, watching the changes on the screen. In particular, you will note that as objects are removed from `w` they remain on the screen. This behavior, which is the same for tables, graphs, and windows, is controlled by two s context variables, `s.AUTOREPARENT` and `s.AUTOSHOW`, both with default value 1. For example, if

```
    w←1↓w
```

the object that was named in `w[0]`, say `x`, is redisplayed outside the layout. However, if `s.AUTORESHOW` had been 0 then `x` would not have been redisplayed. It would, however, have remained bound to the label class. If `s.AUTOREPARENT` had been 0, then x would have been freed from the label class (and consequently not displayed).

As you proceed with the experiment you may notice that the relative positions of `x`, `y`, and `s` on the screen are not necessarily related to the arrangement of these names in the variable `w`. The geometry of the layout can be controlled by specifying `w` as a matrix or a nested array. For example, to put `x` on the left, `y` in the middle, and `s` on the right, set

```
    w←1 3ρ`x `y `s
```

Or, to put `` `y`` above `` `x`` and `` `x`` above `` `s``, set

```
    w←3 1ρ`y `x `s
```

Or, to put `y` to the left of `s` and `x` above them both, set

```
    w←`x,1 2ρ`y `s
```

or

```
    w←(`x;`y `s)
```

A layout can also be composed on the screen. Set `w` to Null, so that all the objects that were contained in it will be shown on the screen as independent displays:

```
    w←()
```

Reset `x`, `y`, and `s` to their original display classes if necessary. Move the displays around so you can see them all, and then move them into an agreeable arrangement. For example, make `s` larger and `x` smaller, put `x` next to `y`, and `s` on top of them both. Now set

```
      `w has (`build;1)      ⍝ Or s.AUTOBUILD←1
      w←`x `y `s
```

The independent objects have now been captured in a layout reflecting their independent arrangement on the screen, and their displays have been replaced with the layout display.

Once a display has been arranged to your liking, it can be saved and redisplayed later using the s context functions `s.save` and `s.load`. For example,

```
      s.save 'testfile.+'
```

will save all objects created in this session, including the layout `w`, in the file `testfile`. Then if the arrays `x`, `y`, and `s` are recreated in some future session, the layout can be recreated and redisplayed as follows:

```
      s.load 'testfile'
```

More details can be found in "Layouts, Geometry, Constraints".

Example 2. A Text-Graph Dependency

As a second example, consider devising a display with a text input area and a graph area, so that when an expression for a scalar function is entered in the text area, its graph appears. Set

```
      xc←0.1×⍳1000
```

Functional expressions can be captured on the screen by a scalar object. For example:

```
      fn←20⍴' '
      show `fn is `scalar
```

The value area of a scalar can be edited. Move the mouse onto the blank area to the right of the text `fn`, enter `1○xc` (i.e., sine of `xc`), and press the **Enter** key. Now examine `fn` and you will see that this expression has been captured.

A *trace* for a graph of this expression is a two-column matrix, with the domain to the left of the range. To make the trace a dependency on `xc` and `fn` (or rather the expression in it):

```
      tr:xc(,@0)⍎fn
      ⍴tr
 1000 2
```

Now put the graph on the screen:

```
      g←`tr
      show `g is `graph
```

Quite possibly the graph display has covered the scalar display. Move it so that you can see both. Now enter another expression in the slot. The graph will change automatically and immediately because it displays the value of the dependency `tr` on the screen. Finally, these two independent displays can be put in a layout:

```
l←(`fn;`g)
show `l is `layout
```

Default Class Binding

If a global variable or dependency is not bound to a display class and its name is used as the argument to `show` or as the left argument to the screen management primitive function `has` when the right argument contains a nonpersistent attribute (see "Persistent Attributes (key = p)"), it will be bound to its *default display class*[1].

The default display class of an object is determined according to the following rules, which are applied in order:

1. `ι0` is bound to class array and the Null to class scalar;
2. if the variable is of type box, attempt to bind it to class radio, and if that fails try class check, and so on through the list of slot, layout, array, scalar (any object can be bound to class scalar);
3. if it is of type function, bind it to class button;
4. if it is a matrix, attempt to bind it, in order, to class layout, array, scalar;
5. if it is a vector, attempt to bind it, in order, to class table, graph, layout, array, scalar;
6. bind it to class scalar.

Children (see next section) of a table object are always bound to the tableField class and children of a graph are always bound to the graphTrace class.

Display Classes

The descriptions in the following chapters assume default values for the attributes of the display classes. In particular, whenever a display class has a title area, the name of the variable bound to the class is displayed there. Settings other than defaults are controlled by the attributes of the various classes, which are described in the several chapters on attributes as well as in the chapters on the various classes. The classes can be grouped in seven categories:

- simple classes for displaying individual global variables: array, hgauge, label, matrix, page, scalar, vgauge, and view;
- classes for entry: command, hscale, password, text, and vscale;
- the class for functions: button;
- classes for slotfillers: action, check, choice, hmenu, radio, slot, tree, and vmenu;
- general container classes, which consist of objects made up of other objects: hgrid, hpane, layout, notebook, vgrid, vpane, and window;
- special container classes: table and graph, and their special contained classes, tableField and graphTrace;
- the class for printing reports: report, which cannot actually be displayed.

The objects that can be bound to container classes are variables whose values consist of collections of other objects. For example, in this chapter the value of the layout object *w* ended up as `` `x `y``, where *x* is a label object and *y* is an array object. An object bound to a container class is said to be a *parent* of the objects named in its value, and these objects are said to be *children* of the parent. The children of a child are also children of its parent. In A+, no object can be a child (or parent) of itself.

When a child is bound to a container class, its context need not be given if it is the same as the parent's, because the same context is assumed. Different contexts are accommodated by qualifying the child's name.

Entering and Editing Information on the Screen

A+ provides a very simple scheme for entering and editing on the screen. Not all display classes are designed for entering and editing, but those that are all use this same scheme. The way the area to be edited is identified does vary, however, and so this point is discussed first. (A more detailed account can be found in "User Interactions with Displays".)

Traversal: Activating the Area for Input or Editing

The screen is composed of a series of windows, and to enable keyboard entry the one that holds the area for input or editing must be selected. The usual way to do this is simply to move the mouse pointer onto that window. The appearance of the window will change to indicate that it now has *keyboard focus*, such as by a change in the background of the title area at the very top of the window.

Assuming that this is a window created by the screen management system, it must have been the result of executing `show `a` for some object `` `a``. Depending upon the display class of `` `a``, the editing state will be different:

- If `` `a`` is bound to the `` `scalar`` display class, a highlight band will be displayed around it and you can begin entry or editing.
- If `` `a`` is bound to the `` `slot`` display class, it will look like a collection of scalars, and one of the scalar lookalikes will have a highlight band around. If you want to work in some other area in the slot, use the **Tab** key or **arrow** keys to move to it.
- If `` `a`` is bound to the `` `array``, `` `table``, or `` `matrix`` display class, a highlight band will appear around the object, not around an individual row or cell. Press the left mouse button when the mouse pointer is in one its rows or cells, and that row or cell will then appear raised. It is called the *selected* row or cell. If you want to enter or edit some other row or cell, use the **Tab** key or **arrow** keys to move to it.
- If `` `a`` is bound to the `` `layout`` display class, one of the objects in it will have keyboard focus, as indicated in the previous descriptions. Use **Meta-Tab** on Sun keyboards and **Alt-Tab** on IBM keyboards to move around the layout until the object you want has keyboard focus. You can also move the pointer onto the area you want and press the left mouse button to get keyboard focus on that area.

Input and Editing

Once the row or cell you want has been selected, you can simply start typing to enter input; input mode begins automatically on the first key stroke. Edit mode is ended for that row or cell and the input is sent to the appropriate A+ variable when the **Enter** key is pressed, or the **Tab** key, **Meta-Tab** on a Sun keyboard or **Alt-Tab** on an IBM keyboard, or an **arrow** key is used to move to another row or cell.

There are actually three input modes: complete replacement, overwriting, and insertion. When input mode is entered by starting to type, anything previously displayed in the input area is lost; it is completely replaced. If the mouse pointer is on a character in the area to be edited and the middle or right mouse button is pressed, the area takes on the appearance of edit mode and the keyboard focus is on that character. If it is the middle button that is pressed, anything entered overwrites the existing text, starting at that character. If it is the right button that is pressed, anything entered is inserted just to the left of that character.

Windows

A+ windows are tree structures of bound variables. A bound variable is often referred to as an *object*. For example, in the window consisting of a layout `m` containing a scalar `s`, a table `t` of fields `a`, `b`, and `c`, and a button `f`:

```
      a←b←c←⍳10
      t←`a`b`c
      s←101
      foo{s;c;v}:⍞'pressed'
      f←<{foo}
      `t is `table
      `s is `scalar
      `f is `button
      m←(`s;`f;`t)
      `m is `layout
      show `m
```

there are seven objects:

```
        m            `layout
            s            `scalar
            f            `button
            t            `table
                a            `tableField
                b            `tableField
                c            `tableField

      `descendents of `m
<   `.t `.s `.f `.a `.b `.c
```

Toplevels and Popups

An A+ process will usually contain several windows. The windows in a process also form a tree. For example, continue the example above by creating three more arrays:

```
      v1←v2←v3←⍳10
      `v1 `v2 `v3 is¨ `array
      show¨ `v1 `v2 `v3
```

The first thing to note is that the control borders of `m` and the `v`-arrays are different. `m` is a top-level window and the `v`-arrays are popup windows. The default shelltitle of `m` is the value of `_argv`, if it is nonnull, or "A+", and those of the `v`-arrays are "v1", "v2", and "v3".

There Is Normally at Most One Toplevel

An A+ process normally has at most one top-level window. By default, this property - top-level windowhood - is given to the first object bound. In the above example, it is `t`:

```
      `t is `table          ⍝ First object bound.
```

Subsequently bound objects which do not contain t are created as popups. Finally, when a container for t is bound:

```
      `m is `layout         ⍝ Layout containing t
```

the property of top-level windowhood is passed to `m`.

This can be demonstrated by creating a new layout `n` containing `m`.

```
      n←`m `v1
      `n is `layout
      show `n
```

Top-level windowhood is given to `n`, since `n` contains `m`.

Note that this property can be lost:

```
      n←()                      ⍝ m reparented to popup.
      free `n                   ⍝ Toplevel freed.
```

All windows are now popups.

The Workspace

When the A+ process has a top-level window `x`, that top-level window is called the *workspace* of that process. It usually has the shelltitle "A+".

The Workspace Variable `s.WS`

The variable `s.WS` is either `()` (there is no workspace for the process) or `` `cx `var``, where `cx.var` is the top-level window. `s.WS` can be assigned the symbol of any popup window in the process, and no matter what the form in the assigment, fully qualified name, context and unqualified name, or (for the root context) just unqualified name, the value of s.WS will be a two-element symbol vector. Continuing the above example, we can restore the property of top-level windowhood to `m`:

```
      s.WS←`m
      s.WS
 ` `m
```

The Screen Management Function `hide{}`

The function `hide{}` dismisses a popup window if its pin is in the 0 state. That is, `hide{}` will not dismiss a pinned window.

The `` `exit`` Callback

Quitting from a top-level window causes a quit event; dismissing a popup window causes a dismiss event: generically, an `` `exit`` event. The default `` `exit`` callback function frees a toplevel and hides an unpinned popup. This callback can be reset:

```
      `popup has (`exit;my.dismiss)
      `top has (`exit;my.quit)
```

Leaders and Followers

Construct a set of windows:

```
      a←b←c←d←e←⍳3
      show¨`a`b`c`d`e
```

The workspace is `a`:

```
      s.WS
 ` `a
```

`b`, `c`, `d`, and `e` are popup windows.

Now hide the workspace:

```
     hide `a
```

Notice that if they are unpinned the popups "follow" *a* into hiding.

Now, show the workspace:

```
     show `a
```

The popups follow *a* back into visibility.

Now pin one of the windows, say *b*, and hide the workspace again. This time, *b* remains visible.

By default, all popups follow the workspace:

```
     `followers of `a
< `.b `.c `.d `.e
     `leader of `b
< `.a
     `leader of `c
< `.a
```

Note that

```
     `x has `leader `x
```

is equivalent to

```
     s.WS←`x
```

I.e., only the workspace follows itself.

Also note that

```
     `x has (`leader;)
```

causes *x* to follow the workspace if there is one. Otherwise, it has the same effect as

```
     `x has `leader `x
```

Follower Trees

More complex hierarchies are possible:

```
     `c has `leader `b
     `followers of `a
< `.b `.d `.e
     `followers of `b
< `.c
```

A convenient form for describing the whole leader-follower tree of a process is a nested slotfiller:

```
     ↓k←0⊃    `followertree of `a
<  `.a
< < <  `.e `.d `.b
  < <
    <
    < <  `.c
    < <
     `.a `.b⊃k
<  `.c
```

The `iconic Attribute

Any popup window can be converted to a toplevel, and vice versa. For example:

```
    a←b←⍳10
    show `a is `array       ⍝ First object is toplevel.
    show `b is `array       ⍝ Subsequent objects are popups.
    `b has (`iconic;1)      ⍝ Convert to toplevel.
    `b has (`iconic;0)      ⍝ Convert back to popup.
```

s.WSNAME Contains the Workspace Shelltitle

By default, the shelltitle on the workspace is either `0⊃1↑_argv`, or `"A+"` if `_argv` is Null. The default workspace shelltitle can be set to an arbitrary string:

```
    s.WSNAME←"y Application"
```

s.SHELL Determines whether Windows are Top-level or Popup.

The first object bound is a toplevel, the workspace; subsequent objects are popup windows. This behavior can be altered at any point by changing the value of the global variable `s.SHELL`:

```
    s.SHELL
¯1                     ⍝ ¯1 means popup window.
    show ⍳10           ⍝ First object bound is a toplevel.
    show ⍳10           ⍝ Subsequent objects depend on s.SHELL
    s.SHELL←0          ⍝ 0 means top-level window.
    show ⍳10           ⍝ This is a top-level window.
```

The s.wstree{sym} Function

This function creates a nested slotfiller dependency which represents the window hierarchy:

```
    s.wstree{`my.var}
    show `my.var is `tree
```

The default assignment callback function on `my.var` shows and raises the selected window. (When a window is raised, wherever it overlaps another window it is shown and the other window is obscured.)

Layouts, Geometry, Constraints

Containers

A+ windows are trees of screen objects. An A+ screen object is an A+ variable bound to a display class. E.g.,

```
    a←⍳10                 ⍝ A variable
    `a is `array          ⍝ bound to array display class.
```

The A+ display classes are of two kinds: data and container. Typically, an A+ window will consist of a container with children, some of which may in turn be containers. All objects at the leaves are data objects, which are childless.

There are eight container classes in A+: table, graph, window, layout, hpane, vpane, hgrid, and vgrid. The following sections explain how to use layouts to create geometry - pictorial structure - in a window tree.

Layouts

A layout *m* is a structure of virtual rows and columns. The position of an object *p* in a layout can be specified by four numbers, called the `at` of *p* (in *m*):

virtual row
virtual column
number of virtual rows spanned
number of virtual columns spanned.

E.g.,

```
     `at of `p
< 1 2 1 4
```

indicates that *p* is at row 1, column 2, and spans one row and four columns.

Ultimately, all methods of positioning objects in layouts reduce to that of specifying the `at` of the object.

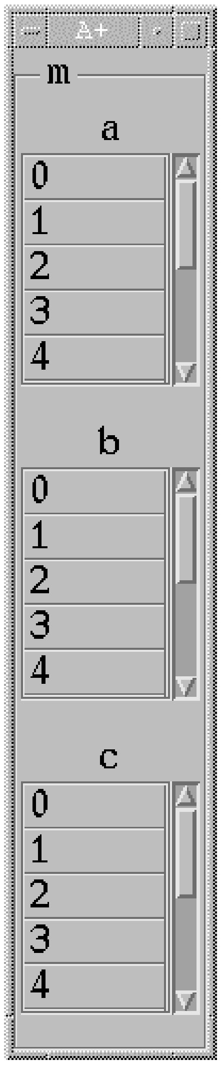
The Natural Size of Objects

Every A+ screen object has a natural size. The natural size of an object is a function of its data, font, and other attributes, which might be specific to the object. For example, the natural size of an array class object is a function of the shape of the data, the width of the formatted data, the font, and the settings of the rows and columns attributes for the object. The natural size of a container is a function of the natural sizes of its children and their arrangement within the container. Natural size can always be expressed in pixels (xs and ys).

The Structure of a Layout

The Simple Symbolic Vector Form

The structure of an A+ layout, as it is manifested on the screen, is a function of the array which is bound to that layout. That is, a layout *m* containing children *a*, *b*, and *c* is a variable *m* containing the symbols `` `a``, `` `b``, and `` `c``. In the simplest case, *m* is just:



```
     m←`a `b `c
```

which represents vertical stacking of *a*, *b*, and *c*, with *at* of:

```
     >0⊃¨`at of¨⊃m
 0  0  1  1
 1  0  1  1
 2  0  1  1
```

Simple symbolic vectors are ambiguous representations. Two attributes control the interpretation of a simple symbolic vector:

- build:
  if 1, approximate the current arrangement of *a*, *b*, and *c* on the screen;
  if 0, use the default arrangement for `` `a`b`c ``, taking the position attribute into account;
- position (effective when build is 0):
  if 1, place `` `a`b`c `` in a vertical column (equivalent to `` (`a;`b;`c) ``);
  if 0, position each of them at 0 0 for 1 1 (stacked one atop another).

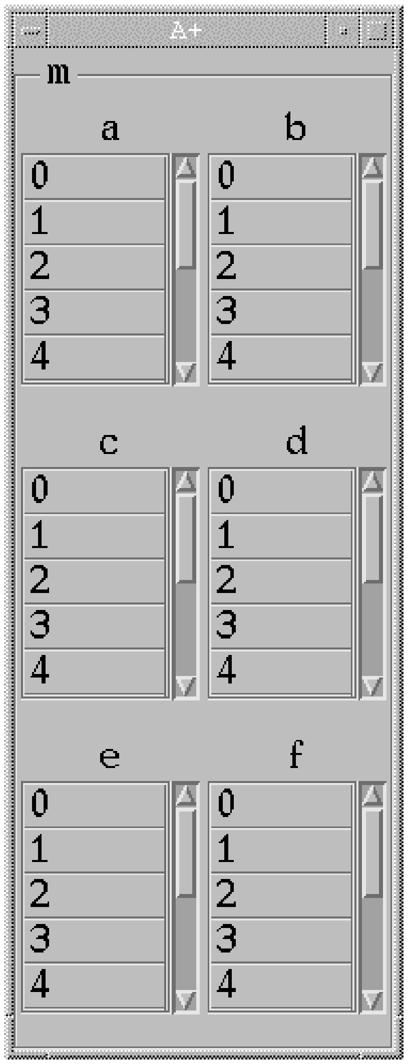
That is, if build is set to 0, then *m* (i.e., `` `a`b`c ``) is taken to represent the geometry of the layout, with the orientation depending on the position attribute. To be effective, position must be set before the layout is built, when *m* is `()`, say.

If build is set to 1, *a*, *b*, and *c* retain the same relative positions and sizes when they are orphaned ("zero parented").

```
      `m has (`build;0; `position;0) ⊣ m←()
      `m is `layout ⊣ m←`a`b`c
      >0⊃¨`at of¨⊃m
 0  0  1  1
 0  0  1  1
 0  0  1  1
```

The Nested Vector Form

The simplest method of representing geometry is confined to depth 1 vectors of symbolic vectors. Each symbolic vector in a variable of this form represents a new virtual row of the layout. E.g.,

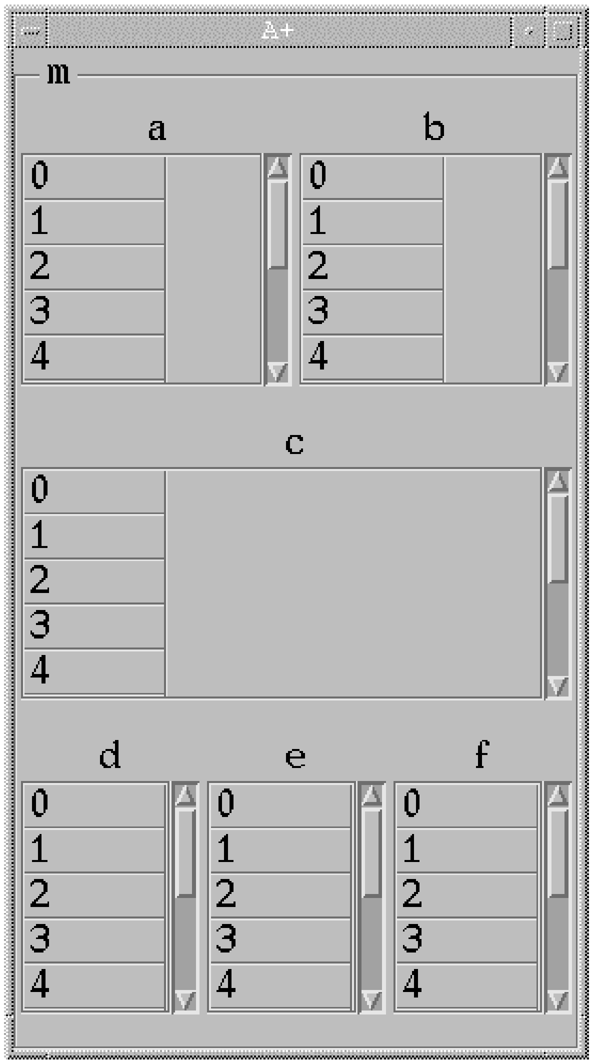


```
      m←(`a `b;`c `d;`e `f)
      `m is `layout
      >0⊃¨`at of¨⊃m
 0  0  1  1
 0  1  1  1
 1  0  1  1
 1  1  1  1
 2  0  1  1
 2  1  1  1
```

giving the representation shown. Each object spans exactly one row and column; each vector in m represents one virtual row of the layout. (Hence, the ";" in the expression above can be read as a sort of virtual "new line" character.)

The Span of Objects

This nested vector form works fine in the case where the same number of objects is found on each virtual row of the layout, but what about the case where each row contains a different number of objects? For example,



```
      m←(`a `b;`c;`d `e `f)
```

In this case, `c` is meant to span the entire width of the structure, `a` and `b` half each, and `d`, `e`, and `f` a third each, as shown, with `at` of:

```
      >0⊃¨`at of¨⊃m
 0  0  1  3
 0  3  1  3
 1  0  1  6
 2  0  1  2
 2  2  1  2
 2  4  1  2
```

The total number of virtual columns is the least common multiple of the number of objects in each row.

Note that symbols can appear exactly once in a form of this type, and that depth and rank must be less than or equal to 1, and that the simple symbolic vector `v` is treated as `⊢¨v`.

The Simple Matrix, or Canonical Form

Only certain geometries can be represented using vectors of vectors, namely, those in which row-span is one for all objects. Consider the case in which buttons `a` and `b` and table `t` have the following spatial arrangement:

```
          a   t
          b   t
```

`a` and `b` are located on successive rows, and `t` spans both rows. In order to describe this arrangement, we use the canonical A+ notation for layout geometry:

```
     m←2 2⍴`a `t `b `t
```

which precisely pictures the arrangement described above:

```
     !@¯1 ⊤m
 at
 bt
```

The rules of canonical representation are more intuitive than accurate formalization would suggest: where `u←unique{,m}` is the vector of unique symbols in `m`, there must be a rank 2 subarray `mi←m[j;k]` for each `u[i]` in which all occurrences of `u[i]` in `m` are found and no occurrences of any other element of *u*, and the shape of *mi* specifies the number of virtual rows and columns spanned by *u*[*i*]. Hence, neither of the following counts as a canonical representation:

```
2 2ρ`a`t`t`b
2 2ρ`a`a`a`b
```

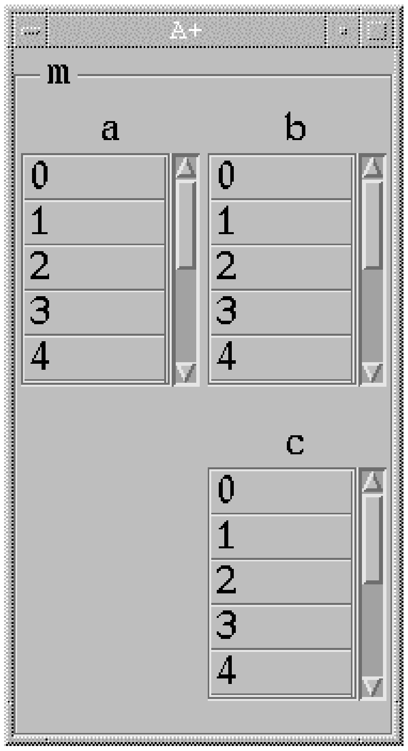
Note that any layout array in nested vector form can be translated into canonical form. E.g.,

```
m←(`a;`b `c;`d `e `f)
n←3 6ρ`a`a`a`a`a`a`b`b`b`c`c`c`d`d`e`e`f`f
```

represent the same geometry.

The Placeholder in Simple Matrix Layouts

Consider the geometry of three objects, *a*, *b*, and *c*, which are to be arranged in a rectangle whose lower left quadrant is intended to be "empty". That is:



```
      >0⊃¨`at of¨`a`b`c
 0 0 1 1
 0 1 1 1
 1 1 1 1
```

To obtain this geometry, use the "placeholder" symbol "`" in the layout variable:

```
      m←(`a `b;` `c)
```

allowing the *c* object to span a single column, as shown.

Note that ` is *not* the symbol of any object.

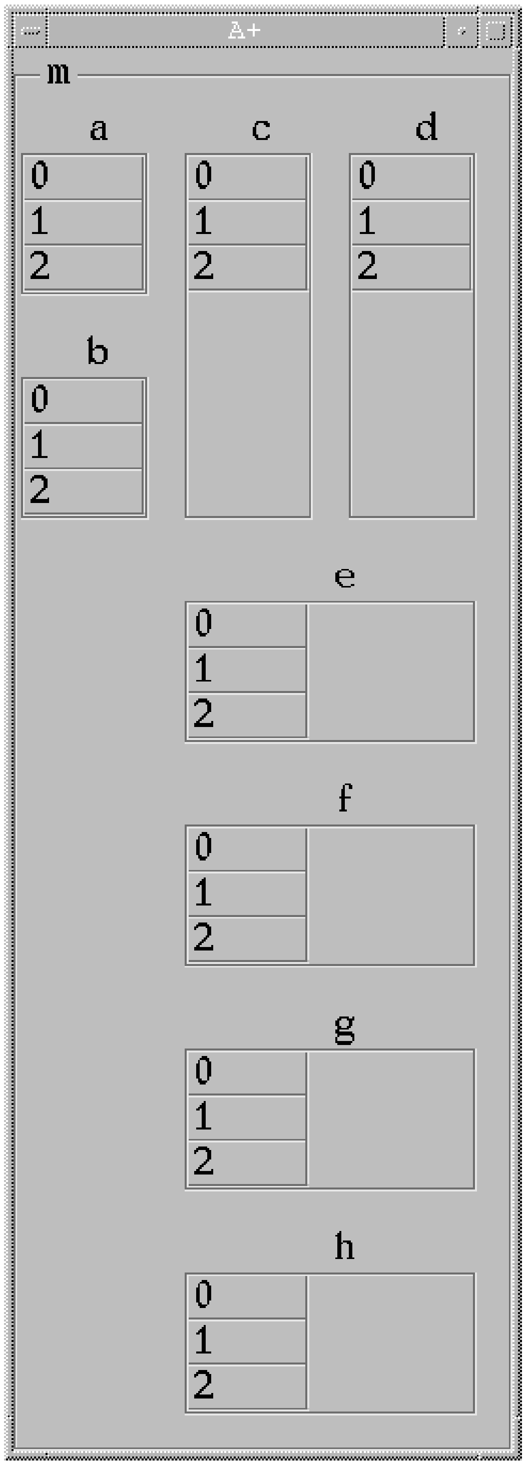
The Nested Matrix Form

Consider the array *m*:

```
  a←b←c←d←e←f←g←h←ι3
  m←2 2ρ<()
  m[0;0]←<2 1ρ`a`b
  m[0;1]←<1 2ρ`c`d
  m[1;1]←<4 1ρ`e`f`g`h
```

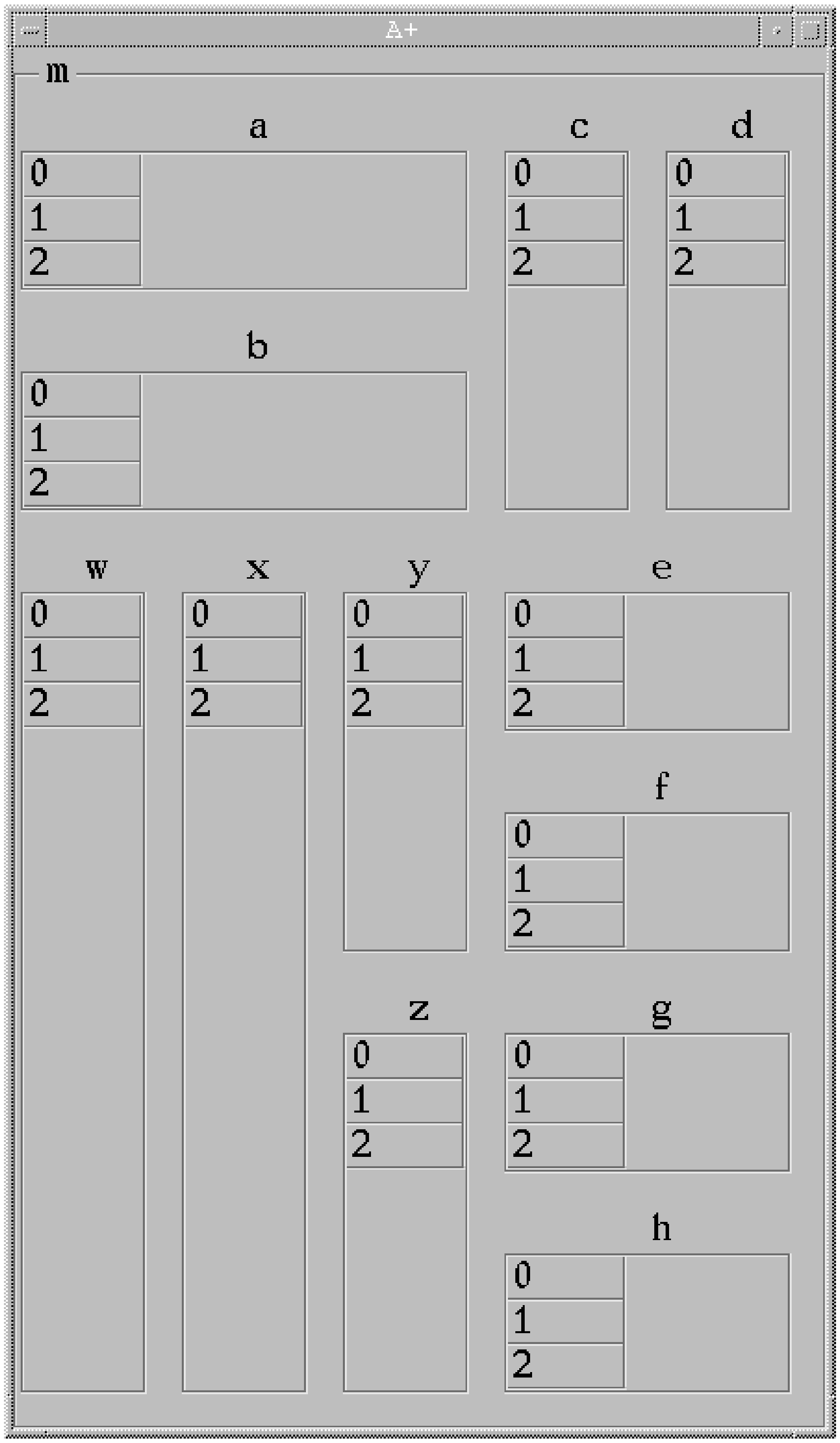
m is a 2x2 array of symbolic matrices, where the shapes of the submatrices implicitly specify the relative spans of their children, as shown.

```
  `m is `layout
   >0⊃¨`at of¨⊃⊃,,¨m
0  0  1  1
1  0  1  1
0  1  2  1
0  2  2  1
2  1  1  2
3  1  1  2
4  1  1  2
5  1  1  2
```



The ability to manipulate blocks of objects without the intervening step of assigning to a global variable can be extremely convenient in applications with dynamic structures. Thus:

```
    w←x←y←z←ι3
    r←1 2ρ(`w `x;2 1ρ`y `z)
    m[1;0]←<r
```

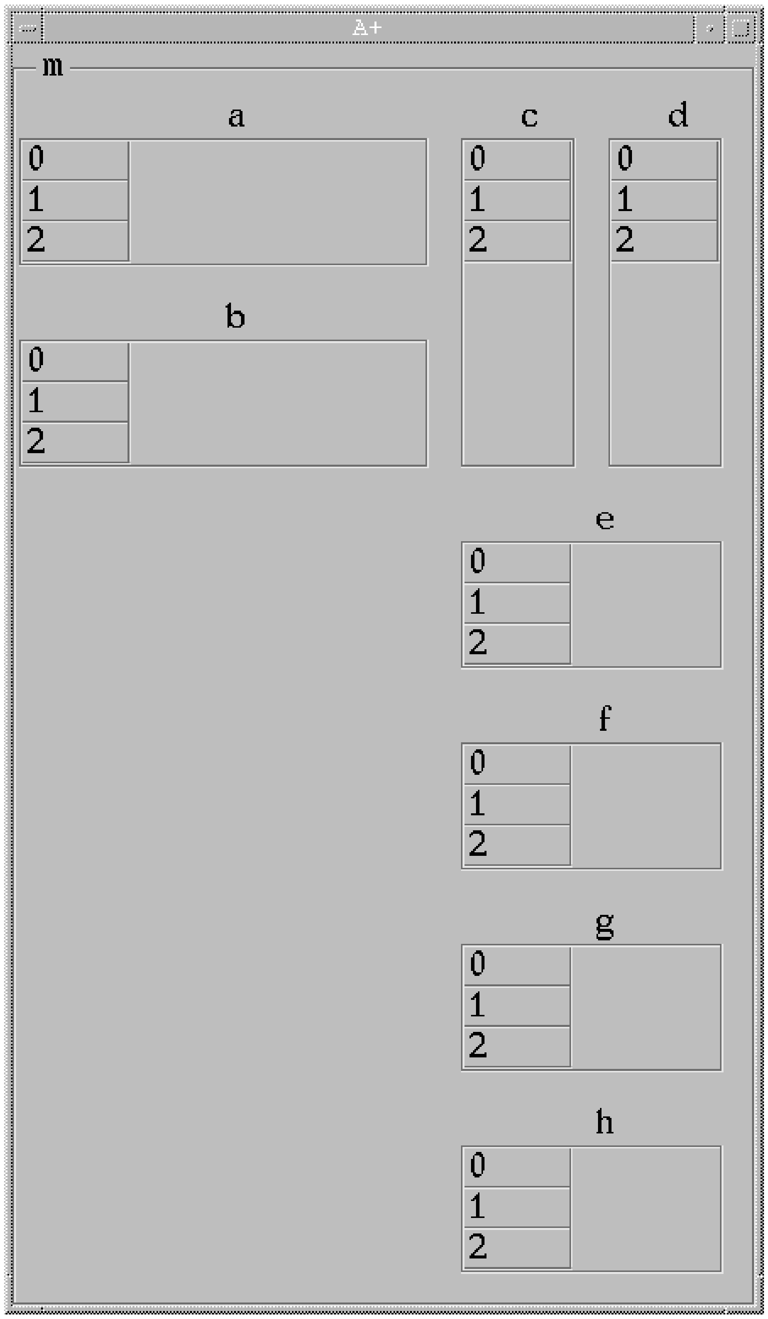


The display of *m* is modified as shown; note how `()` is replaced wholesale by the arbitrary structure containing *w*, *x*, *y*, and *z*.

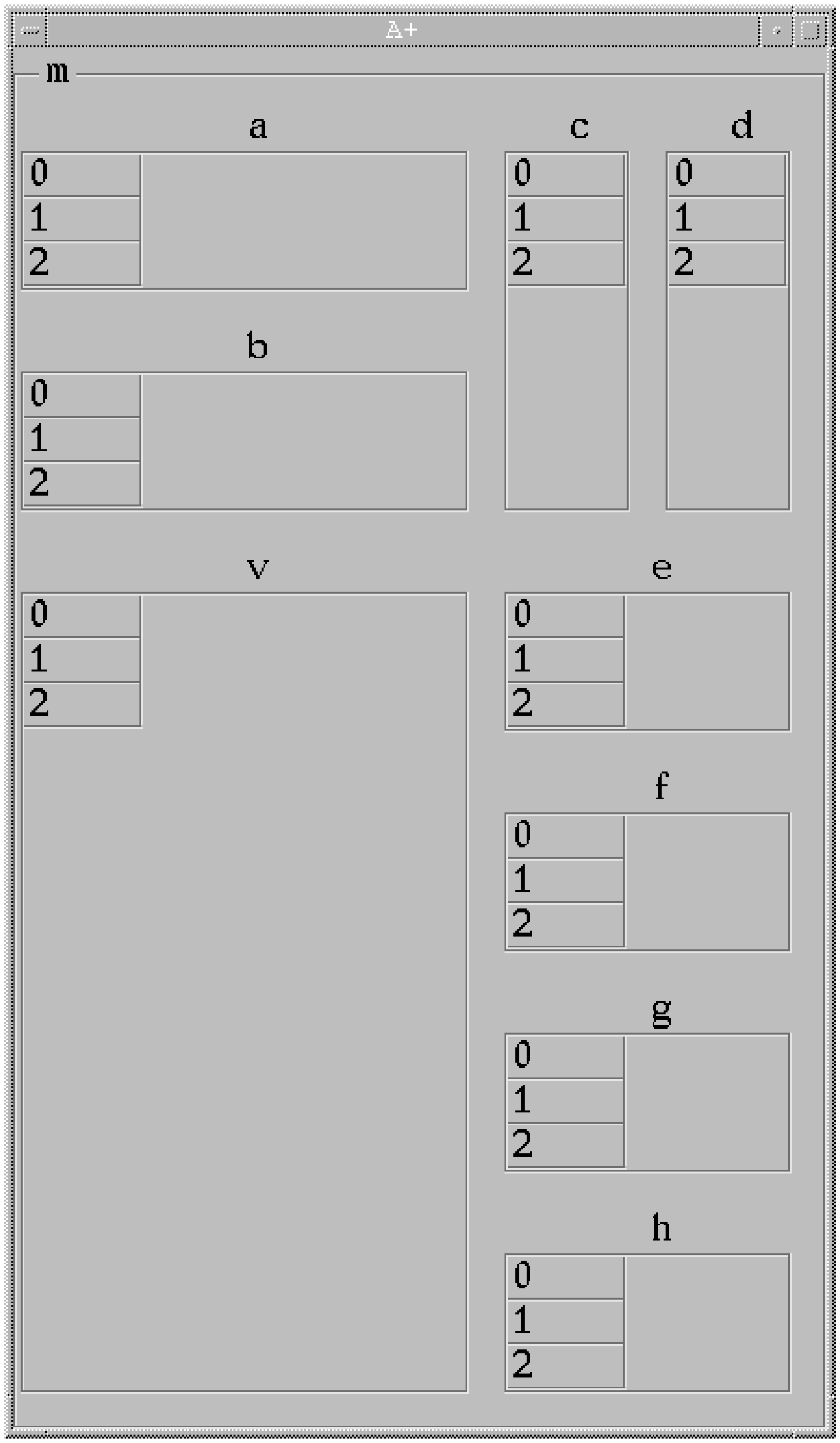
```
      >0⊃¨`at of⊃¨,¨⊃,,¨m
0  0  1  3
1  0  1  3
0  3  2  1
0  4  2  1
2  0  4  1
2  1  4  1
2  2  2  1
4  2  2  1
2  3  1  2
3  3  1  2
4  3  1  2
5  3  1  2
```

The placeholder can be restored or another variable inserted:

```
      m[1;0]←<()
```



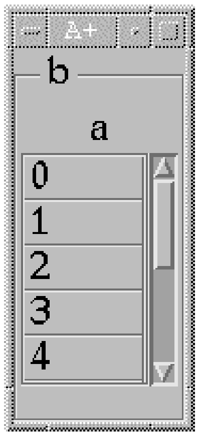
```
v←ι3
m[1;0]←<3 1ρ`v
```



with the results shown. The compactness shown in previous figures can be obtained by freeing *m* and then rebinding it after inserting the placeholder or variable.

Constraints on Resizing Objects

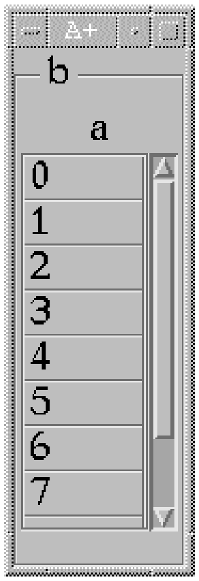
Objects can be resized in either of two ways: directly, by specifying values for certain size-dependent attributes (e.g., font, or rows or cols); or indirectly, by resizing the object's container. E.g.,



```
      a←⍳10
      b←`a
      show `b is `layout
```

The natural size of the array-class object *a* is determined by (among other things) the default number of rows displayed, which in this case is:

```
      `rows of `a
<   5
```



Now grab the frame of the window with the mouse and pull down until eight rows of *a* are exposed.

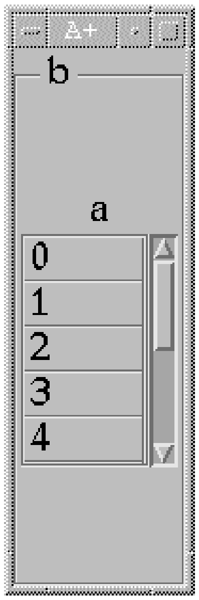
```
      `rows of `a
<   8
```

This should look pretty much the same as executing the code

```
      `a has (`rows;8)
```

Intuitively, however, they seem to be quite different. In the first case, the size of *a* is a function of the size of its parent *b*, while in the second case, the size of *b* is a function of the size of its child *a*.

Since the second case will only arise when the program explicitly sets an attribute, there is no real need to constrain or control that kind of resizing. But in the first case, you might want to keep *a* from being resized no matter what the user does to change the size of *b*. In this example, you will want to set the "don't grow in height" constraint on a:

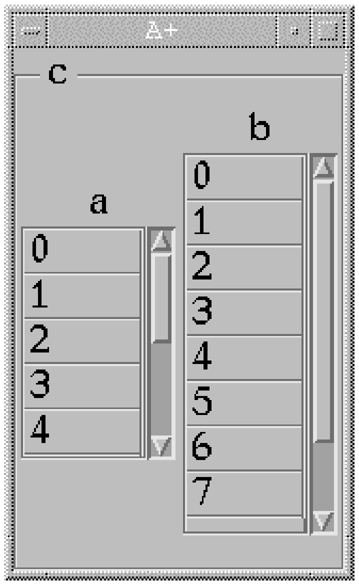


```
    `a has (`resize;'H')
```

As the window *b* is resized, *a* will hang on to its five-row display, centering itself in *b*-space.

What is actually going on, and what will help in understanding some of the examples to come, is this: suppose you stretch *b* to include 100 additional height pixels. The layout receives the message "distribute 100 additional pixels over your rows" in response to which it expands its only row by 100 pixels. The row of the layout then sends to each of its children - in this case, just *a* - "grow by 100 additional pixels" But *a* is H-constrained, and refuses to comply.

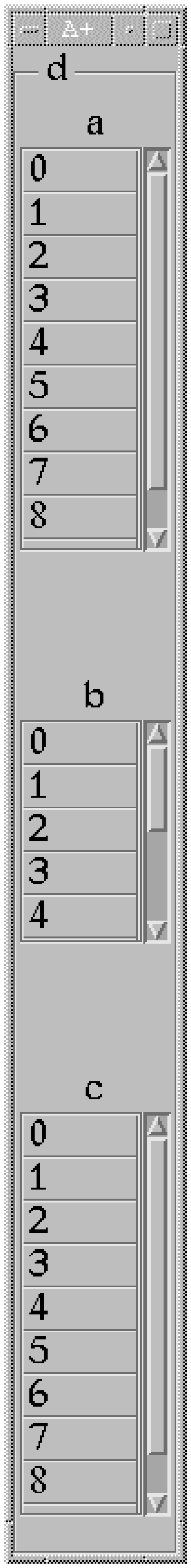
Consider the case:



```
    a←b←ι10
    c←<`a`b
    `a has (`resize;'H')
    show `c is `layout
```

In this example, only *a* is constrained. Resizing the layout *c* has the effect of giving extra space to the sole row of *c*, which in turn has the effect of giving extra space to each child in that row. Array *a* rejects the space while *b* accepts it.

In an example involving more than one row:



```
     a←b←c←⍳10
     d←(`a;`b;`c)
     show `d is `layout
```

Resizing *d* 300 pixels larger will cause *d* to send three messages, one to each of its rows: "grow by ⌊300÷3 additional pixels" which in turn causes each row to send to its child the message to grow by that amount. In this case, all three children will expand. But if we were to set the H-constraint on *b*:

```
     `b has (`resize;'H')
```

then we would see that *a* and *c* each take their share of the extra space, but *b* does not, as shown.

The "H" attribute is paired with one for constraining width, called, appropriately enough, "W".

Constraints on Resizing Layout Structures

Where only constraints on objects are set, resizing the layout causes each of its rows to accept a share of the extra space. You will have noticed that while this method keeps constrained objects from changing size, it does not keep the rows or columns containing these objects from being resized. In the example given above, where `b` is constrained by "H", what you might have expected was that `a` and `c` would absorb all the extra space, leaving `b` the same size, *and* leaving the amount of space taken up by the second virtual row of the layout unchanged. Instead, `b` appears to be floating in empty layout-space. How do you keep `b` "glued" to `a` above and `c` below?

Recall that a resize causes the layout to send messages of the form "grow/shrink by n pixels" to each of its rows (or columns), which in turn send messages of the form "grow/shrink by n pixels" to each of their children. "H" and "W", set on objects, allow these objects to reject such requests, and remain the same size. We want a similar ability for the virtual rows and columns of a layout, which would enable them to reject resize requests they receive from the layout:

```
a←b←c←⍳10
d←`a`b`c
`b has (`resize;'h')
show `d is `layout
```

Now suppose you expand `d` by 300 height-pixels. `d` receives the message: "grow in height by 300 pixels." It then asks each of its rows, "can you grow in height?" to which rows 0 and 2 reply "yes" and 1 replies "no". `d` then sends the following messages:

to row 0: "grow by `⌊300÷2` pixels"
to row 2: "grow by `⌊300÷2` pixels"

which in turn send resize messages to their children. But note that row 1, having replied "no" to the message asking whether it could be resized, receives no such message from the layout. Consequently, no "H" constraint need be placed on any object in row 1.

The "h" attribute is similarly paired with one for constraining column width, "w".

Constraining Layout Structure by Way of the Objects

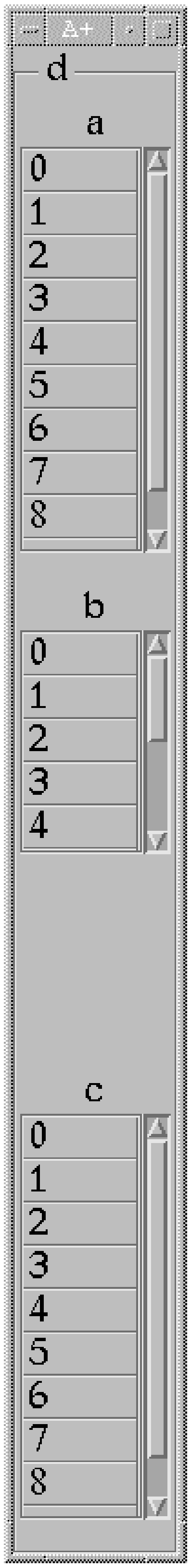
While syntactically both forms of resize constraint are placed on objects in a layout, semantically they are quite distinct. "H" and "W" constrain the resizing of objects; "h" and "w" constrain the resizing of the structure containing the objects on which these constraints are placed. What this means is that "h" or "w" placed on an object will indirectly constrain all other objects on the same row or column. E.g.,

```
a←b←c←⍳10
d←(`a;`b `c)
`b has (`resize;'h')
`d is `layout
```

"h", set on `b`, propagates upward to row 1 of the layout, and hence downward in its effects to `c`. All extra space accumulates in row 0 of the layout.

Repositioning Objects in a Layout

Consider once again the example:



```
a←b←c←ι10
d←(`a;`b;`c)
show `d is `layout
`b has (`resize;'H')
```

Resizing the layout by 300 pixels causes row 1 to grow, but *b* is constrained. The extra space is allocated: half above *b*, half below. In other words, *b* is vertically centered in the row. Suppose we wanted *b* to stick to the top of its row as it grows:

```
`b has (`resize;'t')
```

This causes the extra space given to row 1 to be allocated: all below *b*, as shown.

Other justification options are:

```
     `b has (`resize;'b')        ⍝ Stick to the bottom.
     `b has (`resize;'l')        ⍝ Stick to the left.
     `b has (`resize;'r')        ⍝ Stick to the right.
```

Justification options can be paired in obvious ways:

```
     `b has (`resize;'lt')       ⍝ Stick left and top.
```

but, of course, the following pair is inconsistent, and s will use the one it prefers, top, for:

```
     `b has (`resize;'tb')       ⍝ Stick top and bottom.
```

Default Constraints

A+ classes such as table and array which come with the concepts of rows and columns are instantiated with no resize constraints. It is assumed that most uses of array will allow for resizing. Other classes, such as label and button, are constrained in the height direction, and are instantiated with both "h" and "H" constraints. No width constraints are set by default on any classes.

For default constraint information, enter, e.g.:

```
     `resize                         s.defaultOf                        `button
<   hH
```

The resize Attribute

The resize attribute is cumulative in its settings unless a period is included somewhere in its value:

```
     `a is `button
     `resize of `a
<   hH                         ⍝ Default constraints.

     `a has (`resize;'W')      ⍝ Set 'W' cumulatively.
     `resize of `a

<   hHW
     `a has (`resize;'lt')     ⍝ Set 'l' and 't' cumulatively.
     `resize of `a

<   lthHW                      ⍝ Accumulated.
```

To set constraints noncumulatively:

```
     `a has (`resize;'lt.')  ⍝ Set 'l' and 't'
                             ⍝ noncumulatively, using a period.
     `resize of `a
<   lt                     ⍝ Not accumulated: just 'l' and 't'
```

To reset the default constraints:

```
     `a has (`resize;)     ⍝ Set to null.
     `resize of `a
<   hH                     ⍝ Default constraints.
```

To eliminate all constraints:

```
     `a has (`resize;'') ⍝ Set to ''. Could have used
                         ⍝ '.' with the same result.
     `resize of `a
<                        ⍝ No constraints.
```

The Geometry of Slotfiller Objects

A+ currently supports five slotfiller classes: slot, radio, choice, check, and action. Typically, the A+ variable of a slotfiller class has the form:

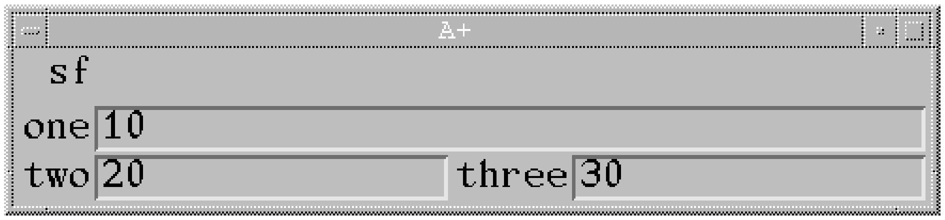
```
     (`sym ... `sym;(val;...;val))
```

Partition Vector Form

The method for controlling the geometry of slotfiller objects is strongly analogous to the one used to position objects in a layout. Conceptually, slotfillers possess row-column structure, within which components can be positioned and which they may span. For example, the slotfiller

```
     sf←(`one `two `three;(10;20;30))
```

can be laid out as



```
10
20
30
```

through the geometry setting:

```
     `sf has (`geometry;1 2)
```

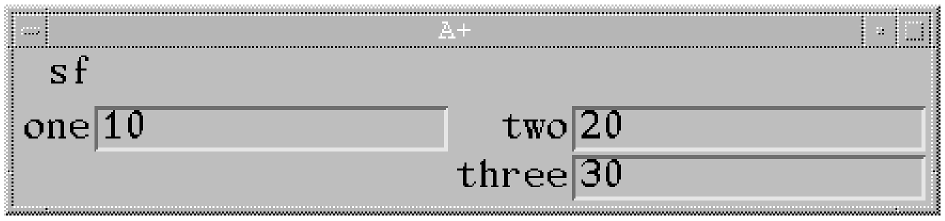
which specifies: one object on the first row, two objects on the second row. Implicitly, the first object spans two columns, each of the other two span one column.

Note that geometry settings of this form are partition vectors:

```
     1 2⊂ 0⊃sf
<   `one
<   `two `three
```

Index Matrix, or Canonical Form

A more general method for controlling geometry is, once again, strongly analogous to the method of canonical representation for layouts. Consider the geometry for `sf` shown as



```
10
20
30
```

where the object *one* spans two rows, and *two* and *three* each span a single row. This is obtained by means of:

```
     `sf has (`geometry;2 2ρ0 1 0 2)
```

Note that

```
     !@ ¯1 ⊤(2 2ρ0 1 0 2)#0⊃sf
 one   two
 one   three
```

Placeholder Indices in the Canonical Form

Consider the geometry:

```
one: ___10     two: ___20
               three: ___30
```

where the lower left quadrant is empty. This is obtained by means of:

```
     `sf has (`geometry;2 2ρ0 1 ¯1 2)
```

where -1 plays the same role in geometry arrays as ` does in layout variables.

Symbolic Geometry

The s functions will translate symbolic matrices into geometry matrices:

```
     `sf has (`geometry;2 2ρ `one `two ` `three)
```

is equivalent to

```
     `sf has (`geometry;2 2ρ0 1 ¯1 2)
```

for slotfillers containing the symbols `` `one``, `` `two``, and `` `three``.

Horizontal Geometry

The default geometry for a slot is 1; i.e., `(#0⊃sf)ρ1`. Horizontal geometry for any slot is `#0⊃sf`. For convenience, this can be specified simply as

```
     `sf has (`geometry;¯2)
```

Making Objects Equal Sizes

Consider

```
     a←(`one `thirteen `twelve `seven;(;;;))
     `a has (`geometry;¯2)

     show `a is `action
```

The buttons have different widths, and grow and shrink proportionally as the object is resized. Objects with geometry - layouts and slotfillers - can be constrained by the R and C attributes:

```
     `a has (`C;1)
```

causes all buttons in *a* to assume the size of the button with maximum width, namely, `thirteen`.

The R constraint has the same effect on rows:

```
     a←ι10
     b←20
     c←`a`b
     show `c is `layout
     `c has (`R;1)
```

now causes `b` to assume the height of `a`.

One consequence of making sizes equal is that virtual row or column ratios can be used to indicate the proportional sizing of objects. For example, compare

```
     a←ι3 4
     b←ι10
     c←`a`b
     show `c is `layout
```

with

```
     aa←ι3 4
     bb←ι10
     cc←1 4ρ`aa`aa`aa`bb
     `cc is `layout
     `cc has (`C;1)
     show `cc
```

Both `a` and `aa` are at `0 0 1 3` and `b` and `bb` are at `0 3 1 1`, but only in `cc` does virtual-columns span determine relative width.

Locking and Zooming

Suppose you have a layout of four graphs:

```
     aa←bb←cc←dd←ι10
     (a;b;c;d)←(`aa;`bb;`cc;`dd)
     `a`b`c`d is¨`graph
     m←`a`b`c`d
     `m is `layout
     m has¨((`at;0 0);(`at;0 1);(`at;1 0);(`at;1 1))
     show `m
```

You might like a mechanism whereby one of the graphs can be "zoomed" to fill the entire layout and later restored to its original cell. Note that `m` is not representational: the graphs are explicitly positioned using `` `at``.

```
     Z←()      ⍝ Symbol of the graph that has been zoomed.
```

Consider the function

```
zoom{x}:
      {
      if (x≡())
              {
              if (~Z≡())
                    {
                    i←mιZ;
                    s←2 2⊤i;
                    yxs←`ys `xs of 0#(m≠Z)/m;
                    hide Z;
                    Z has (`ys `xs `at;yxs,<s,1 1);
                    show Z;
                    }
              }
     else
              {
              if (~Z≡()) &{()};
              x has (`at `raise;(0 0 2 2;1));
              };
```

```
        .Z←x;
        }
```

If the argument to *zoom* is nonnull, then (1) if something is zoomed, "unzoom" it (see below), and then (2) zoom the object named. To zoom an object, reposition it at 0 0 with a span of 2 2 and raise it to the front.

If the argument to *zoom* is null and something is zoomed, "unzoom" it by translating its index in *m* to a position in the layout, get the pixel extent of some other (arbitrary) child of *m*, hide the object, reset its position and extent, and reshow the object.

These roundabout procedures are necessary because an attempt to reposition the object that was zoomed will cause the layout to expand to accommodate the extent of the formerly zoomed object.

The attribute lock on layouts (and their subclasses) can be turned on, after showing, to force the downsizing of an object that is being positioned so that the object will fit into the cell. In other words, the layout will not expand to fit the object, but will force the object to fit the layout. Now consider the *zoom* function, rewritten to operate on locked layouts;

```
`m has (`lock;1)

zoom{x}:
        {
        if (x≡())
           {
           if (~Z≡()) Z has (`at;(2 2⊤m⍳Z),1 1);
           }
        else
           {
           if (~Z≡()) &{()};
           x has (`at `raise;(0 0 2 2;1));
           };
        .Z←x;
        }
```

Natural Size Action

Consider

```
a←b←c←⍳10
d←(`a;`b `c)
show `d is `layout
```

Resize the layout *d*. Now execute

```
`d has `naturalsize
```

The layout sends the naturalsize message to each of its children. A primitive object that receives the naturalsize message recomputes its natural size (see "The Natural Size of Objects"). After each child recomputes its natural size and transmits it to the parent, the parent recomputes its own size.

2. Screen Management Functions

"Introduction to Screen Management" shows all the basic primitive functions for visual representation in action, and in fact includes most forms of their definitions. This chapter gives complete details. The command `$load s` establishes these functions in both the root and the s contexts.

In this chapter, an *object* is a symbol holding the name of a global variable or dependency. (It may also be a two-element vector of symbols naming context and variable separately, but it will be described in this chapter as a single symbol, for simplicity.) A *class* is a visual representation format, e.g., array or label. An *attribute* is a modifiable characteristic of a class such as title or bg. An object which is not bound to a display class is said to be *free*.

Definitions of the Screen Management Functions

free

Syntax

`free obj`

Argument and Result

The argument is a symbol scalar or one-element vector. The result is a one-element symbol vector or the Null.

Definition

If the object (i.e., global variable or dependency) named in the right argument `obj` is bound to a class, it is unbound. In this case the result is `1⍴obj`. Otherwise, the function has no effect and the result is Null. If the class to which the object is bound is a container (see the function `is`), then all children of the object are also freed. If the object is a child of another object, it is freed but its parent is not.

has

Syntax

`obj has rescs`

Arguments and Result

The left argument `obj` is a symbol scalar or one-element vector. The result is a symbol scalar.

Definition

This function sets the attributes named in the right argument for the object (i.e., global variable or dependency) named in the left argument. The attributes can be specified in one of two general ways:

- as a slotfiller array: `(`attribute1 `attribute2 ...; (value1;value2;...))`
- as an association list: `(`attribute1;value1; `attribute2;value2; ...).`

In addition, if all of the values to be set are symbols, then the following symbol array is also permitted:

```
`attribute1,value1,`attribute2,value2, ...
```

There is an additional variation. If a dot is placed before the attribute name, as in `` `.attribute1 ``, the symbol given for that attribute is not an attribute value itself, but a variable, function, or dependency whose value (at any time of use) will be used for that attribute value. A variable or function provides an "electrical connection", and a dependency does so if it is bound to the reference class (so that it is evaluated as necessary each time the interpreter traverses the mainloop); with such a connection, the display is changed to reflect the new attribute value whenever the variable or function is respecified or the dependency is invalidated. If a Null is given as a value for a dotted attribute, instead of a name, the previous attribute default variable (if any) is no longer connected to this attribute of this object; the value of the attribute does not change. (See "Attribute Default Variables".) See the examples.

The result is `''⍴obj`. The expression `` `obj has (...;) `` sets the specified attribute to its default value.

If the left argument holds a name that has no value and is not local to a function, then a free global variable with the value Null is assigned to that name - which might possibly cause a binding requested by the right argument, such as to radio, to fail.

If the left argument (after the action just described if necessary) names a free variable or dependency and the right argument does not specify a class binding, then the variable or dependency is bound to its default display class. (In fact, because the attributes are applied sequentially, if the right argument specifies a class binding but first specifies a nonpersistent attribute, the free variable or dependency is bound first to its default display class and then to the specified class.)

Examples

```
`header has (`title `bg;
("Visual Display"; `yellow))

⍝ Specified as a slotfiller.
⍝ (The responses, `.header and so on, are omitted here.)

`header has (`title;"Visual Display"; `bg;`yellow)

⍝ Given as an association list.

`a has `bg `red     ⍝ Specify actual value, a symbol so simple
form is okay.
`bg of `a
`red
var←`blue
`a has `.bg `var  ⍝ Specify a variable whose value

⍝   will be used, by using `.bg
`bg of `a         ⍝ Get the value of the attribute.
`blue
`.bg of `a        ⍝ Get the name of the variable
`var                  ⍝   holding the attribute's value.
```

hide

Syntax

`hide obj`

Argument and Result

The argument is a one-element symbol array. The result is Null.

Definition

Every object displayed on the screen appears in a window under the control of the window manager of the underlying system, and every window created through A+ is the result of applying the function `show` to some object; that object is called the *root object* of the window. Windows are either top-level windows or popups, a distinction that is discussed in "Windows" and "Quit, and Open and Close". If the object named in the argument `obj` is the root object of a top-level window, the effect of this function is to replace the window with an icon. If the object is the root object of a popup window, and the pin is out, the effect is to remove the window from the screen; there is no effect if the pin is in. If the object is a child of a layout, the effect is to remove the object from the screen (but not from the layout definition) and reconfigure the layout if necessary. Otherwise, the function has no effect.

is

Syntax

`obj is cls`

Arguments and Result

The left argument is an array and the right argument is a symbol scalar. The result is also a symbol scalar.

Definition

This function binds the object represented by `obj` to the display class named in `cls`. There are three cases for the left argument, depending on whether it is a global name with a value, just a name, or just a value.
Case 1. The left argument is a one-element symbol array naming a global variable, which may be a dependency. This is the ordinary case, and this global variable is the one that will be bound to a display class `cls`.

Case 2. The left argument is a one-element symbol array holding a name, but that name has no value. If the name is not local to a function, it is automatically given a prototypical value for an object bound to class `cls`, and the resulting global variable is bound to that class. (In this case, `obj has (`class;cls)` would assign the value Null to the object, no matter what the class.)

Case 3. The left argument is an array other than a one-element symbol array. A global variable is created whose value is this array, and this variable is bound to the class named in `cls`. See the example for `show` for the rules governing the creation of the variable name.

There are also three cases for the right argument; a container display class (graph, hpane, layout, table, vpane, window), simple classes (all the other display classes except graphTrace and tableField), and a default setting. The graphTrace and tableField classes are not in the domain of `is`.
Case 1. The right argument names a simple class. The object represented by the left argument is bound to that class and the name of the object is returned as the result.

Case 2. The right argument names a container class. The global variable named by the left argument must be an array of objects. The function behaves as in Case 1, and additionally binds to their default classes any free children of the object being bound. The geometry of the newly bound object is determined by the form of the object's value. "Example 1. Manipulating Layouts" shows how the value of the left argument can determine the geometry of the object.

Case 3. The right argument is the Null. The object named in `obj` (or created, for Case 3 of the left argument) is bound to its default class. See "Default Class Binding".

If the object to be bound is already bound to a class, it is first freed.

The result in the first two cases for the left argument is `''⍴obj`, and in the other case it is a symbol scalar holding the name of the created variable.

See "Assignment of Objects Bound to Display Classes" for a discussion of assigning values to objects that are bound to display classes and the refusals and error messages that can occur.

of

Syntax

```
rscs of obj
```

Arguments and Result

The right argument `obj` is a symbol scalar or one-element vector.

In the usual case, the left argument `rscs` is a symbol array, and the result is a nested vector of the same shape. The left argument can also be a slotfiller or association list suitable as a right argument for `has`, and the result is a nested vector of shape `⍴0⊃rscs` or `(⍴rscs)÷2`, respectively.

In the attributed data case, the left argument is either ``(`out;value)`` or ``(`in;string)``, where `value` is appropriate for an element of `obj` and `string` is appropriate for an entry in a cell of a display of `obj`, and the result is a nested scalar.

Definition

Consider the usual case first, and since only the attribute names in a slotfiller or association list are used (the values being ignored), assume that the left argument is a symbol array. For every `i` of `⍳#,rscs`, if `res` is the result, `i⊃res` is the value of the resource `i#,rscs` for the object (i.e., global variable or dependency) named in the right argument `obj`. Place a dot before the attribute name to indicate that you want the variable holding the value, not the value itself. (See "Attribute Default Variables"). If there is no such variable, a Null is returned. See the examples for *has*.

Attributed Data

For `` `out ``, the result is the enclosure of the formatted string, the string that would appear for an element of *obj* with value *value* in a display. For `` `in ``, the result is the enclosure of the converted value, the value that would be appropriately placed in *obj* as a result of the entry of *string* in a cell of a display of *obj*. It is not necessary for *obj* to be shown, although it must be bound.

Examples

```
`title `bg of `header ⍝ Continued from has example. Usual case.
< Visual Display
<  `yellow
(`in;'142.376') of `a ⍝ Attributed data, assuming appropriate a
<  142.376                ⍝ Converted value, a floating-point number.
(`out;142.376) of `a
< 142.376                 ⍝ A character string.
`a has (`out;'f12.3') ⍝ Now change output format.
`.a
(`out;142.376) of `a
<      142.376
```

show

Syntax

```
show x
```

Argument and Result

The argument is any array. The result is Null.

Definition

If *x* is a one-element symbol array representing a bound object, that object is displayed on the screen in the format to which it is bound. If *x* is a one-element symbol array representing a free object, that object is bound to its default class (see "Default Class Binding") and then displayed. Otherwise, an object is created whose value is that of the expression *x*, and the object is bound to its default class and displayed on the screen.

If the displayed object is a table or a graph and its children were already bound, either explicitly by the user or as a result of being displayed on the screen, they are re-bound as appropriate for *x*.

Example

```
show ι2 3
⍝      gn: variable generated
⍝      gn: S will bind variable to array
⍝      gn: variable bound to array
```

gn is a global variable name generated by *show*. The value of that variable is `ι2 3`. The name is chosen to be the first available name in the root context from the list of names:
`.a, .b, .c, ... , .y, .z, .a0, .a1, .a2, ... .`

Dependencies and Displayed Objects

Dependencies marked for evaluation will ordinarily be reevaluated only when explicitly referenced. If, however, a dependency is displayed on the screen, it will be automatically referenced upon being marked for evaluation (see the discussion following the next example). The effect is that its value is always current relative to the objects on which it depends, which is the most generally useful behavior for dependencies displayed on a screen. They are, after all, being referenced at unknown times by the user. For example, if a dependency represents a view of a database and the database changes, one would expect to see the view change accordingly. The same automatic referencing can be obtained without actual display on the screen by binding a dependency to the reference display class, as in `` `d .is `reference``.

Immediately referenced dependencies, however, require extra consideration similar to cyclic dependencies. Consider the following simple example:

```
  a←0 3 2 ¯1 5
  b←9 2 5 4 3
  c:a+b
  show `c is `array
  b←b,10
  +: length
*
```

This dependency set should be organized in a commit and cancel style. When `a`, `b`, and `c` are displayed and edited on the screen, the decision to commit or cancel can be put on the screen as well, in the form of a commit button and cancel button. There is then no need for a callback function to record update activity or for tests as to when commitments can be done.

There is a general rule for when dependencies that are displayed (or bound to the reference display class) and marked for evaluation will be automatically evaluated. The A+ process is always executing in an *event loop*. Within this loop it processes, one at a time, messages from Emacs or an XTerm (keyboard entry, or stdin in Unix terminology), from other processes through adap, and from the screen (e.g., key and button presses). Some of these messages are processed by evaluating A+ expressions or invoking callback functions, during which dependencies might be marked for evaluation. During each pass through the event loop, a series of messages is processed and then the screen is refreshed, i.e., all dependencies that are displayed (or bound to the reference display class) and marked for evaluation are evaluated. Since the programmer can't be sure how many messages are processed on each pass, the prudent thing is to assume that it is only one. The assumption, therefore, must be that the screen is refreshed after each message is processed.

For instance, in the above example the expression `b←b,10` was evaluated, which constitutes a message and caused the displayed dependency `c` to be marked for evaluation. When execution was complete, `c` was automatically evaluated. Since this happened before `a` was adjusted, which would have required a separate message, an error occurred. If, however, the expression `(a;b)←(a,5;b,10)` had been entered, then both `a` and `b` would have been updated in the evaluation of this single message, and the subsequent automatic evaluation of `c` would not have failed because of a length error. Note that had the expression contained a defined function, that function would have been executed to completion before `c` was automatically evaluated. The respecifications of `a` and `b` could have occurred anywhere within that function execution, even in different functions that it called, without a length error.

In sum, if, as in this example, a displayed dependency requires more than one thing to be done before it can be correctly evaluated, then either all those things must be done within a single message, or the evaluation must be deferred, say through a commit and cancel formulation.

3. Display Attributes

The colors, fonts, layout arrangements, and many other aspects of the appearance of a screen display are controlled by *display attributes*. "Screen Management Functions" tells how to use the screen management primitive functions `of` and `has` to set and reference values of display attributes.

Every display attribute is listed in "Display Attributes", together with a brief description and a set of keys. The meanings of the alphabetic keys are given in the next section.

- Keys with capital letters indicate the display classes for which an attribute is meaningful.
- Keys in lower case letters generally indicate characteristics of the attributes.
- Links to additional information about the attributes are indicated by "more".

For some display attributes, the information in "Display Attributes" is only a summary. For more information on:

- Functional attributes, see "Functional Attributes";
- Attributes with callbacks, i.e., event attributes, consult "Attributes with Callbacks";
- User interactions related to certain attributes, such as select, see "User Interactions with Displays";
- Variables and additional functions related to attributes, consult "The s Context", including the "Attribute Default Variables" section. "Null (0)" or "Null (1)" in the Default column indicates the attribute will mirror a default variable whose default value is 0 or 1, respectively.

Among the display attributes described here are those for specifying colors and fonts. Both colors and fonts are specified by symbols or character strings, as in `` `name`` or `'name'`. A list of all available color names can be found in "`/usr/X11R6/lib/X11/rgb.txt`".

For convenience, a list pruned to the unique colors appears in "Colors". A list of all available font names can be produced by executing `xlsfonts` in an XTerm session (or `$xlsfonts` from A+). A list of preferred fonts (those available on the screen and for printing) is in "Preferred Fonts".

Display Attribute Characteristics

Persistent Attributes (Key = p)

Display attributes are categorized as *persistent* and *nonpersistent*. Persistent attributes can be thought of as more directly connected to the concept of an array than nonpersistent ones. For example, the font in which the values of a variable will appear is a persistent attribute, while the thickness of the highlight border in its display is nonpersistent. The most distinguishing feature is that once a persistent attribute is set for a variable *x*, it will remain set, even as x is bound, freed, and bound again to another class; in effect, a persistent attribute is an attribute of A+ variables, not of particular display classes. Indeed, it can be set on an unbound variable. For this reason persistent attributes are also called *attributes of variables*. On the other hand, the setting of a nonpersistent attribute will not be maintained when a variable is freed, and some nonpersistent attributes cannot even be specified for variables bound to certain classes. Nonpersistent attributes are attributes of display classes, not of variables bound to those classes. Nonpersistent attributes are also called *attributes of classes*, or widget attributes.

Even though a persistent attribute can be given a value for an object of any class, that value is not necessarily meaningful. For example, the font attribute, which specifies the font in which values are type set, is not meaningful for line graphs. For each persistent attribute, "Display Attributes" shows "ALL" as the key for the set of display classes to which it applies and - unless it is meaningful for all classes - lists in parentheses the classes for which it is actually meaningful, i.e., for which it has a visual effect. Note that an object contained in a table or graph is bound to the class tableField or graphTrace, respectively, no matter to what class it may have previously belonged.

Applicable Classes; the ALL and CNT Keys

Keys with capital letters in the second column of "Display Attributes" indicate classes to which the attributes apply. The ALL key means all classes except report, which is fundamentally different from the others, since it cannot be shown, and in some cases, such as navigation, children of reports. Only Rp refers to the report class. CNT means the container classes hgrid, hpane, layout, notebook, vgrid, and vpane.

Attributes of Objects in Layouts (Key = CNFT)

Objects in layouts and panes have attributes for establishing their relative positions and behavior when the layouts are resized. See "Layouts, Geometry, Constraints", and also "The Layout Display Class".

Attributes for Top-level Objects (Key = TOP)

Top-level objects, in the broad sense[1], are objects that are not contained in other objects (and not reports). It is these objects that appear with certain window manager decorations, and there are attributes associated with this appearance and their interrelations among top-level objects. See "Windows" in "Introduction to Screen Management".

Graph Attributes for Axes (Keys = o, x, y)

Graphs have four axes, the x axis on the bottom of the plot area, the y axis to the left, an alternate x axis above, and an alternate y axis to the right. Some attributes come in sets of four, with one attribute for each axis. For example, the font for the labels can be set separately for each axis, as xlabelfont for the x axis, ylabelfont for the y axis, Xlabelfont for the alternate x axis, and Ylabelfont for the alternate y axis. Whenever there is such a foursome, only the attribute for the x axis is given, with the key o.

Similarly, for several attributes there are x-axis and alternate x-axis versions, but no y-axis or alternate y-axis versions. For these attributes, only the attribute for the x-axis is given, and with the key x. Analogously, for attributes with y-axis and alternate y-axis versions, but no x-axis or alternate x-axis versions, only the attribute for the y-axis is given, with the key y.

Action Attributes (Key = a)

Certain attributes express an action to be taken, rather than a quality. For example, naturalsize is the attribute which causes an object to be resized to a size appropriate for its current value and attribute settings. It is not necessary to specify a value for this attribute, but only that the resizing actions should take place, as in

```
`object has `naturalsize
```

Some action attributes also take values, but it is not necessary to specify a value to get the primary action to take place. For example, the show attribute can be given the value 1 to show an object or 0 to hide it, and
`` `object has (`show;1) `` is equivalent to `` `object has `show ``.

Attributes that Can Only Be Set (Key = s) or Referenced (Key = r)

Some attributes can only be set, and have no stored value; s will complain if asked for a value. Other attributes can only be referenced; s will complain about any attempts to set values for them.

Attributes with Callbacks (Key = cb)

These attributes are associated with user interactions such as key and button presses. See "Attributes with Callbacks". This key is used as a link to the corresponding table entry in that chapter. Some attributes with this key have a default value of 0 or 1 and can also be given the other value of 1 or 0; in these cases either `` `false `` or `` `off `` can be used in place of 0, and either `` `true `` or `` `on `` can be used in place of 1.

Functional Attributes (Key = fn)

These attributes provide efficient means for dynamically modifying their values based on changing circumstances in running applications. See "Functional Attributes". This key is used as a link to the corresponding table entry in that chapter.

Table of All Display Attributes

Keys for the table

The second column of this large table shows *keys* which explain the applicability of each attribute. To see a brief description of what a key represents, simply point to they key with your mouse... no clicking required.

Alternatively, assuming you are using the frames version of the online manual, when you are looking at an entry in the table and are not sure what a key stands for, you can click on "Keys" under "Table of Attributes" in the lefthand frame. After ascertaining the key's meaning, you can return to the table by pressing the righthand mouse button with the pointer in the main frame and selecting "Back" from the menu that pops up.

Attribute	Meaning
A	array
B	label
C	command
D	password
E	choice
G	graph

gT	graphTrace
hG	hgauge
hM	hmenu
hP	hpane
hR	hgrid
hS	hscale
I	radio
K	check
L	layout
M	matrix
N	action
O	notebook
P	page
Q	scalar
R	tree
Rp	report
S	slot
T	table
tF	tableField
U	button
V	view
vG	vgauge
vM	vmenu
vP	vpane
vR	vgrid
vS	vscale
W	window
X	text
ALL	all objects except reports (fundamentally different; can't be shown)
ALL (*list*)	ALL, but meaningful only for the classes in *list*
CNFT	NFT but only objects in containers
CNT	containers: layout, grids, panes, and notebook
GS	hgauge, vgauge, hscale, and vscale
NFT	not tableField or graphTrace
TOP	top-level and popup objects only
o	also for graph attributes prefixed X, y, Y, or

x	also for the attribute with prefix X
y	also for graph attribute prefixed Y
a	action
cb	attribute with callback
fn	functional
p	persistent
r	reference only (cannot be set)
s	set only

Attribute	**Keys** (POINT TO KEY FOR INFO)	**Description**	**Default**
acceptfocus	E, hM, I, K, N, U, V, vM	If 1, this object will accept input focus during traversal; if 0, it will not.	1
active	ALL, r	If 1, this object will be automatically updated when marked for reevaluation; if 0, it will not. The value 0 occurs when the show attribute is 0 for this object or one of its ancestors.	1
addtexttrace	G, □, ▭	The action taken on an add-a-text-trace event.	
addtrace	G, □, ▭	The action taken on an add-a-trace event.	
ancestors	ALL, r	Vector of objects containing this object.	
arrowbuttons	C, D, Q, S	If 1, incrementing and decrementing arrows (up- and down-pointing triangles) are shown, for each entry in the case of a slot; if 0, they are not shown; if (for a slot) a boolean vector of the same length as the slotfiller, they are shown for the entries corresponding to ones in arrowbuttons. For the action associated with these buttons, see the increment and decrement attributes.	Null (0) for slot, 0 for the others.
arrowdown	ALL, cb	Event callback function, called for a **down-arrow** keypress. If 1, default action; if 0, no callback. [Future use]	Move selection down 1.
arrowkeys	ALL, r	Slotfiller of **arrow** keys and callback functions. Cf. arrowdown, arrowleft, arrowright, arrowup. [Future use]	
arrowleft	ALL, cb	Event callback function, called for a **left-arrow** keypress. If 1, default action; if 0, no callback.	Move selection

		[Future use]	left 1.
arrowlist	NFT	Matrix with items "fr dir to". dir is one of `left `right `up `down and fr and to name objects. Connected to upto, downto, etc. [Future use]	
arrowright	ALL, cb	Event callback function, called for a **right-arrow** keypress. If 1, default action; if 0, no callback. [Future use]	Move selection right 1.
arrowup	ALL, cb	Event callback function, called for an **up-arrow** keypress. If 1, default action; if 0, no callback. [Future use]	Move selection up 1.
at	CNFT, □	Position of this object in a layout: (vrow, vcol, vrows, vcols).	
atsector	TOP	A two-element vector designating the (row, col) position of the virtual desktop sector where this object is displayed, or `here for the currently active sector (see s.desktop{}, s.beHere{}).	0, 0
axis	G, □	Specify which axes appear on a graph.	`std
b	CNFT	If 1, bottom justify this object in its layout cell.	0
backpagebg	O	Background color of tabs and page edges.	`deep skyblue4
backpagefg	O	Foreground color of tabs and page edges, but not the titles on the tabs, which get their color from fg.	`black
backpages	O	Number of apparent unshown pages, i.e., page edges.	3
backpage thickness	O	Thickness of each (unshown) page edge, in pixels.	4
banner	Rp	Text of banner to be printed diagonally in background on each page, e.g., "Draft", "Internal use only".	
barwidth	G, □	The maximum pixel width of bars in bar and stack graphs.	10
be	ALL (CNT), □, P	If (cuv) has (`be;(f;s)) and cuv is a container, then f{s;d;i;p;c;v} is called to bind the child d of cuv to a display class: g	

		may be a display class, a list of display classes, or whatever will help in this process.	
bg	ALL, p; □for A, M, tF, V	The background color of this object. For tables and arrays, does not change the scrollbar background color if that color is different from the bg color that is being changed. A tableField inherits its value from its table, and when the value for tableField is reset to Null it uses the current value for table.	grey
bggrayscale	tF, fn	Background shade value in reports, 1 - white to 0 - black. A scalar or a function (in the callback `i` is the row).	1
bindingwidth	O	The width (diameter) of the spiral binding, in pixels.	36
blank	ALL (A, M, T, tF), p	What is to be displayed for a NA, e.g., on insertion of a row. Cf. `na` and `s.AUTOBLANK`.	Null
blink	P, □, ▭	Boolean mask controlling whether or not a cell blinks, i.e., its foreground and background colors oscillate. (The blink timer runs only when something is actively blinking).	
blinkrate	P	The rate of blinking in milliseconds.	250
bold	P, □	Boolean mask indicating which characters are bold.	
borderheight	O	Border area above and below the notebook, in pixels.	7
borderwidth	O	Border area on each side of the notebook, in pixels.	7
bottom	G, ▭	Distance from the bottom of the graph window to the x-axis rule (visible or not), as a percentage of the window height.	0
bottom margin	Rp	Bottom margin of report, in inches.	1
bound	ALL, r, p	If 1, this object is bound; if 0, this object is free.	
box	P, ▭	An n by 4 matrix of boxes, where each row represents a box as first row, first col, number of rows, number of columns.	`0 4 ρ 0`
boxcolor	P	A symbolic vector of box colors. If boxcolor is specified with fewer colors than boxes, then the remaining boxes stay the color that they were before the specification.	

breakbggray scale	tF, □	Background shade for breaks in reports, 1 - white to 0 - black. A scalar or function.	1
breakcriteria func	tF, □	Determines whether a break occurs for this row (propagated to other columns). Must be functional and return 0 or 1.	If row (item) differs from previous row.
breakfggray scale	tF, □	Foreground shade for breaks in reports, 1 - white to 0 - black. A scalar or function.	0
breakfont	T, tF, □	Font to be used for printing breaks. Value for table is used as a default if no value set for a tableField.	
breakleading	tF, □	Distance in points between break and next row.	2
breakoffset	tF, □	Distance in points between break and previous row.	2
breakon	tF	Whether column can initiate breaks or not.	0
break processfunc	tF, □	Function to be called to process data at a break if breakprocesson is 1. If none, computation mode is used.	
break processon	tF	Whether to perform computations at breaks.	1
breakstyle	T, tF, □	Style for break: `` `left ``, `` `right ``, `` `top ``, `` `bottom ``, or a combination. Value for table used as default.	`` `left ``
breaktext	tF	Text to be inserted at every break.	
buffer	C, X, r	The character buffer of the command or text display class; a vector.	
build	hR, L, vR, □	If 1 and this object is a simple vector layout, use autobuilder; if 0, use a default arrangement; if Null, use the value of *s.AUTOBUILD*.	Null (0)
C	hP, I, K, L, N, S, vP	If 1, make all virtual columns of this object the same size.	
cancel	Rp	Set to 1 (in a callback) to cancel report generation.	0
children	CNT, G, T, W	A vector of all objects named in this variable.	
class	ALL, p	The display class of this object.	

clear	NFT, cb, ☐	Furnishes a callback function for a clear event. See "Accessing the Primary Selection Buffer".	
col	A, M, T, V	The column index of the selected cell. Setting this attribute causes the selected column to appear in the display. In addition, if the row attribute is set to a valid row index, the cell at position (row, col) is highlighted. Setting it to -1 signifies that no column is selected (so none is highlighted).	0
colindex	M	A vector of indices of the selected column labels when the selectcol attribute is 1, in the order of selection.	ι0
colindexbg	M	The background color of the indexed column labels.	medium aquamarine
collabelrows	M	The maximum number of rows in the column labels.	1
color	P, ☐	Row indices for colormap attribute, for foreground, background color pairs for cells. Functional result must be same shape as variable; if nonfunctional, it is reshaped to match.	Null (0)
colormap	P	k by 2 matrix of color symbols; each row is a foreground, background pair. Indexed into by the color attribute.	`1 2 ρ `black `grey`
colors	A, C, D, M, Q, S, T, tF, V, ☐	A vector of colors that will be cycled through when a cell is updated by indexed specification (which can be caused by editing the cell on screen); see the cycle attribute.	
cols	A, E, hM, M, T, V, vM, X	The number of visible columns.	Table: at most 5; text: 40.
colsep	A, M, T, V	If n, column separators appear every n columns.	1
colspace	M, ☐	An integer vector specifying column widths. The first element applies to the row-label column. If there is only one element, it applies to all columns. Otherwise, all elements except the first are used cyclically, as in Reshape.	Null
column	tF	Horizontal alignment of data in the tableField. One	`` `left ``

alignment		of `` `left ``, `` `center ``, `` `right ``.	
column control	T	Vector of columns per page.	
column pagespan	T	Number of pages that the table's columns are to span.	
column resize	T, ▭	Controls whether columns can be resized by dragging their right borders (column separators): 1 - yes, 0 - no.	0
column spacing	T	Space between columns in reports, in inches.	
computation mode	tF	Specify the default computation mode: `` `sum ``, `` `avg ``, `` `max ``, `` `min ``, `` `stddev ``, `` `variance ``. Used when breakprocesson is 1 and breakprocessfunc is not specified.	
compute pagebreakcb	Rp, cb	For a callback that occurs at the completion of the pagebreak computation.	
compute sizecb	Rp, cb	For a callback that occurs at the completion of the computation of the report size.	
constraints	hP, L, O, vP	A slotfiller containing the resize attribute of each child, indexed by child's name.	Defaults for the children
coordinate	G, ▭	The pointer's x, y coordinates when a refer event occurs.	
	gT, ▭	All the x axis, y axis coordinates that make up the trace when a refer event occurs.	
Coordinate	G, ▭	The pointer's X, Y coordinates when a refer event occurs.	
copy	ALL (A, M, T, tF), p	For an inserted line. If 1, copy the line that the new line is being inserted above or below. If 0, use na value. Cf. `s.AUTOCOPY`.	Null (0)
copy texttrace	G, □, ▭	The action taken on a copy-a-text-trace event.	

copytrace	G, □, ▭	The action taken on a copy-a-trace event.	
cornerindex	M	If 1, the corner label is "flagged"; if 0, it is not; see the selectcorner attribute.	0
corner indexbg	M	The background color of the corner label when it is flagged.	medium aquamarine
current breakcolumn	T, r	Column that is currently causing a break.	
current page	O	Name of the object that is being shown. Set it to show page.	
cursor	C, P, X	The position of the cursor (number of characters from the left and, for page, top).	page: ¯1 ¯1; others: 0.
cycle	A, C, D, M, Q, S, T, tF, V	The duration in milliseconds of the appearance of each color in the value of the colors attribute. If set on a table, it applies to all tableFields.	
cyclemode	A, M, T, V	If `fg, color cycling affects the foreground, if `bg, the background; if `reverse, color cycling is implemented by reverse video and the colors attribute is ignored.	`fg
decrement	C, D, Q, S, cb	An event callback function to be associated with the down arrow button that appears when arrowbuttons is 1. For a slot, there may be several down arrow buttons; the selected attribute gives the symbol for the entry whose button was pressed.	
deiconized	TOP(not popups), □	Function for event callback when the widget is deiconized, whether or not the window manager is CDE. See the caveats in the callback section. Cf. iconized.	
delete	A, E, G (▭), I, K, M, N, S, T, □	Controls the action when a user presses **Meta-Delete** or, on IBM, **Alt-Delete**. The default action is to delete the selected row. Set delete to 0 to turn off the **Delete** key, and 1 to get the default action. In the case of a graph, it applies only to selectable traces. See the selectable attribute.	0
delimiter	Rp	To be defined.	

descendents	CNT, G, T, W, r	A vector of all objects contained in this object.	
disclaimer	Rp	Governs disclaimer: `` `none``; `` `text`` only; text in `` `box``; text with `` `rule`` above and below; text below `` `toprule``; or text `` `appended`` to top-level window.	`` `none``
disclaimer bottom margin	Rp	The bottom margin of the disclaimer, in inches.	.2777777778
disclaimer file	Rp	The file containing the disclaimer text.	
disclaimer leftmargin	Rp	The left margin of the disclaimer, in inches.	.2777777778
disclaimer right margin	Rp	The right margin of the disclaimer, in inches.	.2777777778
disclaimer rule width	Rp	Width of any ruling for the disclaimer; see the disclaimer attribute.	
disclaimer topmargin	Rp	The top margin of the disclaimer, in inches.	.2777777778
disclaimer orientation	Rp	Orientation of the disclaimer: same as the object if `` `none``; else `` `portrait`` or `` `landscape`` regardless of the object.	`` `none``
distribution method	3	`` `linear`` or `` `table``, indicating distributionlevels evenly spaced separators or the separators in distributiontable.	`` `linear``

doc	ALL, p	An attribute that allows documentation, presumably text, to be attached to each object. Cf. *`s.AUTODOC`*.	`>,<`
			`s.AUTODOC`

done	ALL (A, M, P, Q, S, tF), □, p	Holds the name of a function to be called at the end of the screen entry and refresh cycle. (The cycle is: in, preset callback, assignment, set callback, out	

		done.)	
downto	NFT	Object to which **Shift-down-arrow** moves keyboard focus. Connected to arrowlist. [Future use]	Null
dragdrop	T, □	Controls whether columns can be moved by dragging: 1 - yes, 0 - no.	0
dynamic	NFT	Set to 1 for this object to be resized automatically when necessary due to a font or text change.	0
edit	A, C, D, hG, hS, M, Q, S, T, vG, vS, a	When edit is set to any nonzero value, if there is a selected cell and it is not protected, it is put in input (editing) mode. When edit is set to 0, if there is a cell in input mode, its contents are accepted and input mode ended, as if the viewer had pressed **Enter** while the pointer was in the object.	
editbegincb	A, C, D, hS, M, Q, S, T, vS, □	The callback function to be executed when a user begins editing the widget.	Null (none)
editbg	A, C, D, hG, hS, M, Q, S, T, vG, vS	The background color of the cell being edited.	black
editendcb	A, C, D, hS, M, Q, S, T, vS, □	The callback function to be executed when a user ends editing of the widget.	Null (none)
editfg	A, C, D, hG, hS, M, Q, S, T, vG, vS	The color of the text being entered while in edit mode.	grey
editspace	ALL (A, C, D, hS, M, Q, S, T, tF, vS), p	If 0, the space for editing is the same as the current value of the space attribute; if 1, the value of *s.EDITSPACE*; if Null, the value of *s.AUTOEDITSPACE*.	
eval	ALL, cb, p	Nested pair: variable callback function, static data. Function is called after the variable is evaluated (by %).	
evaluate	ALL, p	If 0, the variable is not to be evaluated in determining proper attributes for display; they will be set by the application programmer. Use with great caution.	Null (1)

execute	ALL (A, hS, M, Q, S, T, tF, vS), p	If 1, execute input expressions to obtain a value; if 0, don't. The table value is used for a tableField if the tableField's value is null.	Null (1)
exit	TOP, □, ▭	Controls the action taken when a user attempts to remove this object from the screen. The default iconification-or-removal behavior occurs for the default value.	1
extent	NFT	Location and size of an object that is shown: the vector y, x, ys, xs. For a child y and x are relative to its parent.	
extents	NFT (toplevel CNT), r	A slotfiller whose indices are the names of all non-container, non-tableField, non-graphTrace descendents and each of whose values would be the extent attribute for the corresponding descendent *except that* the locations are absolute (relative to the screen, not to parents).	Always a null slotfiller except for top-level CNT.
f1 through f12	ALL, □	Controls the action when the **F1 - F12** keys are pressed and this object has focus.	
familytree	CNT, G, T, W, r	A nested slotfiller of all objects contained in this object.	
fg	ALL except gT and W, □, p	The foreground color of this object, usually text or values. For tables, cells of any tableField for which fg is not explicitly set. For tableFields, cells. For page objects, the boxes on the page. For check and radio objects, a value area button when the corresponding value is 1. For check, choice, label, radio, tableField, this attribute can be a list of values, which will be used cyclically - in the manner of Reshape - to assign colors to the value area buttons or rows individually; note that for choice, the chosen item is displayed in the value area, whereas the items in the dropdown list are labels. (If this attribute is a list for a table, only the first value is used.) For graph, applies to everything except the title and traces.	

fggrayscale	tF, fn	Foreground shade value in reports, 1 - white to 0 - black. A scalar or a function (in the callback `i` is the row).	0
field	T	The symbol of the selected field, i.e. column. The field and col attributes always indicate the same column. Setting field to Null is the same as setting col to -1: no field is selected.	
fields	T	The number of visible fields, or columns.	
filename	Rp	The name of the output file.	report.ps
fill	D	The character displayed for every keyboard entry in the value area; `''` for no fill.	*
fillcolor	gT, □, ▭	The fill color of various graph styles.	
firstcol	A, M, T, V	The index of the first visible column.	0
firstfield	T	The symbol of the leftmost displayed field of this object.	
firstrow	A, M, T, V	The index of the first visible row of this object.	0
fixedfields	T	The number of fixed fields of this object, i.e., fields on the left that do not participate in horizontal scrolling. The screen-display analogue of fixedreportcolumns.	0
fixedreport columns	T	The number of leftmost table columns that are to appear on the left side of each page of a report for this table. The printed analogue of fixedfields.	0
fkeys	ALL, r	A slotfiller of function key, callback fn pairs for this object.	
focus	NFT	Set to any value to give this object keyboard focus.	
followers	TOP, ▭	A vector of symbols whose objects are removed from the screen when this object is iconified.	
followertree	TOP, ▭	The followers of this object, in a boxed array.	
font	ALL except gT, W, □, p	The font in which the value of this variable is set. For table, used for any tableFields for which font	kaplgallant

		is not explicitly set. For graph, applies only to characters used as symbols in scatter and line-scatter traces.	
foot	TOP	If 1, include a footnote area at the bottom of this object.	
footer	Rp	The footer text.	
footer offset	Rp	The space above the footer text, in inches.	2
footnote	G(▭)	The footnote text, a character matrix or vector of vectors. If empty or null, no footnote area is shown.	Null (none)
footnote font	G(▭)	The font in which the footnote is type set.	kaplgallant-19
footnote justify	G(▭)	Justify the footnote relative to the margins set with the bottom, top, left, and right attributes.	
format breakfunc	tF, fn	The variable callback function that formats the break text. The `d` parameter contains the unformatted text.	
framebg	O	The color of the frame background.	`deepskyblue4
frame linewidth	Rp, T	Page frame line width.	0 (thinnest)
frameoffset	Rp, T	Page frame offset; the spacing between frames, in inches.	2
framestyle	Rp, T	Page frame style: `` `center``, `` `left``, `` `right``, `` `top``, `` `bottom``, or a combination, or `` `none``.	`` `none`` (null)
frame thickness	O	The frame thickness in pixels.	2
freeze	NFT	If 1, setting attributes or respecifying variables will not alter the display of the object and its descendents; if 0, they will. Resetting to 0 will cause at least redrawing; for a layout, repositioning. **Warning:** if color cycling is on for an object, updates that occur while it is frozen will miss the first color, including updates followed	1

		immediately by unfreezing.	
fullscreen	TOP, a	Make the window exactly as large as the screen and put it at location 0, 0. Setting fullscreen to 0 has no effect.	0
geometry	ALL (I, K, N, S), □, p, ▭	The arrangement of slots in the display of a slotfiller object.	1
gradient	gT, □, ▭	If 1, cycle through [`s.FILLCOLORS`] for every data point in a scatter plot or bar graph. If a function, call it for each point.	0
grid	G, ▭	Specify the axes that control the grid lines, or none.	
gridfg	G, ▭	The color of the grid lines.	
gridstyle	G, ▭	The style of the grid lines (solid, dotted, etc.).	
gridwidth	G, ▭	The width in pixels of the grid lines.	0 (thinnest)
h	CNFT	If 1, don't change the height of this object's row on resize.	1 for hmenu
H	CNFT	If 1, do not change the height of this object on resize.	1 for hmenu
has	ALL, p	If 1, whenever an attribute value is changed, report the new setting in the A+ session log, as a set of comments.	0
head	TOP (popups only)	If 0, remove the window manager's header on this object (including, of course, the pin).	1
header	Rp	The header text.	
header offset	Rp	The space below the header text, in inches.	2
headingbg grayscale	tF	Background shade value in reports, 1 - white to 0 - black. A scalar or a function (in the callback `i` is the row).	1
headingfg grayscale	tF	Foreground shade value in reports, 1 - white to 0 - black. A scalar or a function (in the callback `i` is the row).	0
heading style	T, tF	Heading style in reports: `` `left``, `` `right``, `` `top``, `` `bottom``, or a combination, or `` `none``.	`` `none`` (null)

hide	NFT, a	Set to any value to hide this object. In a child of a layout, it must be set after the layout is shown, since show for a layout sets show for its children. See `` `show ``.	
hl	NFT	The highlight border color when the object has focus.	yellow
hlthickness	NFT	The width in pixels of the highlight border around the object.	2; 0 for menus.
horizontal space	R	The minimum distance between nodes in the same row of the widget, in pixels.	15
hscrollsize	A, M, T, V	The height in pixels of horizontal scrollbars on this object.	15
hscrollwith	A, M, T, V, ☐	A list of names, in symbol form, of objects whose horizontal scrolling and column selection is to be synchronized with this object's; see the setfirstcol and setcol attributes.	`` 0ρ` ``
icon	TOP	An n x m boolean matrix of the bitmap pattern for the icon of this object, where n and m are at most 64. Appears in reverse video.	
iconic	TOP	If 1, this object is a top-level window; if 0, a popup. This attribute has no effect on the object named in `s.WS` when `s.AUTOWS` is 1. `` s.WS←`a;`a has (`iconic;0) `` produces an error message.	
iconized	TOP, ☐	Function for event callback when the widget is iconized. Also see deiconized.	
icontitle	TOP	The title of the icon for this object.	
in	ALL (A, C, hS, M, Q, S, T, tF, vS), ☐, p	Used to parse values entered on the screen, which are always in character vector form. Values of this attribute are usually functions. However, if the value is a character matrix and the character vector read from the screen is a row of this matrix, the value created for the workspace is the row index.	`in x:x` for character workspace values; `in x:⍎x` otherwise.
inc	GS	The change in value when an **arrow** key is pressed: **left** or **up**, an increase; **right** or **down**, a decrease. Accepted for gauges, but meaningful only for scales.	1
increment	C, D, O, S	An event callback to be associated with the up	

	cb	arrow button that appears when arrowbuttons is 1. For a slot, there may be several up arrow buttons; the selected attribute gives the symbol for the entry whose button was pressed.	
incurrent workspace	TOP, cb	In a CDE window, an event callback that is called when the window receives presence in the current workspace: that is, when the presence of a window is changed to include the current workspace or when the current workspace is changed to a workspace which includes this window from a workspace which did not include it. Cf. outofcurrentworkspace. See the caveats for deiconized in the callback section. Ignored outside CDE.	
index	A, M, T, V	A vector of indices of the rows that have been chosen (when refer is nonzero) since the object was bound or index was reset; sorted (formerly, in the order of selection). If selectionmode is `single, index is ι0 and can't be set; cf. selected for the index of the selected row.	ι0
insertabove	A, M, T, □	Controls the action when a user presses **Shift-Meta-Insert** or, on IBM keyboards, **Shift-Alt-Insert**. The default action is to insert a new row above the selected one. Set insertabove to 0 to turn off this key combination, and 1 to get the default action.	0
insertbelow	A, M, T, □	Controls the action when a user presses **Meta-Insert** or, on IBM keyboards, **Alt-Insert**. The default action is to insert a new row below the selected one. Set insertbelow to 0 to turn off this key combination, and 1 to get the default action.	0
is	ALL, □, p	Controls the action when the object is bound to a display class.	
justify	B, N, U	Justify the text in the title area. Settings `left, `right, `top, `bottom, 'l', 'r', 't', and 'b' are cumulative, to give all nine possible positions, unless settings begin with a dot, and several symbols or several characters can be included in one setting. Set to Null to restore the default position, centered both ways. has returns the character vector form.	Null (centered)
key	C, P, □	Controls the action whenever a key is pressed.	
keysym	P, r, □	The value represents the latest key press.	

l	CNFT	If 1, left justify this object in its layout cell.	0
label	ALL (E, I, K, M, N, R, S), ⎕, p	The text in the label areas. For a display class for slotfillers, the value can be a single character vector or symbol that applies to every label, or a vector of character vectors or symbols that are used cyclically - in the manner of Reshape - to label each slot individually. A character matrix *m* can also be used, and it is treated as if it were `<@1 m`. The values do not replace the symbolic indices of the underlying slotfiller object. For matrix, a single value can given, or a vector that applies to all labels, or a nested vector of length three, whose items apply to corner, row and column labels, respectively, in the manner just described. Column labels can have more than one row (see collabelrows); these labels can be specified as nested vectors of character vectors, or character matrices.	For a class for slotfillers: the symbolic indices. For a matrix: `('';('0';'1';...);('A';'B';...)).`
labelfg	ALL (E, I, K, M, N, R, S, GS), ⎕, p	The color of the text in the label areas. For a display class for slotfillers, this attribute can be given a vector of values that apply cyclically to the label areas. For matrix, a single value can be given, or a nested vector of three items, each a scalar or vector, that apply to the corner label, the row labels, and the column labels, respectively. For (gauges and) scales, the foreground color for the scale label.	`` `black ``
labelfont	ALL (E, I, K, M, N, R, S, GS), ⎕, p	The font in which the text in the label areas is type set. This attribute applies in the same manner as labelfg (above). For (gauges and) scales, the font in which the scale label appears.	kaplgallant. If null, for matrix use font attribute.
labelinc	GS	The increment between labels on the scale. A value of 0 means that s is to select an appropriate increment.	0
labeljustify	GS	The justification of the scale label. One of `` `left ``, `` `right ``, `` `bottom ``, `` `top ``, `` `none ``.	h: `` `bottom ``; v: `` `left ``.
labelout	GS, fn	A format specification for the scale labels. If not functional, it is one of the forms shown in the next table.	`` `float ``

leader	TOP, □	The leader of this object for iconification (when the leader is iconified, this object is removed from the screen).	
leading	T	Default leading for reports: vertical spacing in pixels.	
left	G, □	The distance from the left side of the graph window to the rule of the y axis (visible or not), as a percentage of the window width.	0
left margin	Rp	The left margin of the report, in inches.	1
leftto	NFT	Object to which **Shift-left-arrow** moves keyboard focus. Connected to arrowlist. [Future use]	Null
legend	G, □	Position of the legend on the graph; e.g., `` `tl `` for top left. Includes settings for outside the graph area.	`` `tl ``
	gT, □, □	The text in the legend that identifies this trace.	Null (varname[:n])
legendbg	G(□)	Color of rectangle containing legend; follows bg if not set.	Null (grey)
legendfg	G(□)	The color of the text naming the traces in the legend.	Null (black)
legend font	G(□)	The font in which the legend text is type set.	lucidasans typewriter-10
legendhl thickness	G, □	The width in pixels of the legend highlight area.	1
legend shadow thickness	G(□)	The width in pixels of the shadow area around the legend area, if present.	1
legend style	G(□)	Display the legend vertically, horizontally, horizontally with the last trace values, or not at all, using `` `ver ``, `` `hor ``, `` `lastvalue ``, `` `none ``.	`` `ver ``
line	P	A k by 4 matrix of line coordinates. Each row represents a line with the format (r, c , nr, nc). Vertical lines should have nc equal 0 and horizontal lines should have nr equal 0. If both nr	

		were 0. A vertical line begins at the top center of the character whose row and column indices in the page variable are r and c, respectively, and extends downward to the bottom center of the character nr rows below. A horizontal line starts at the left center and extends to the right center of the character nc columns to the right. Line settings are not cumulative. Each setting of this attribute removes previous settings.	
linecolor	gT, R, □, ▭	The color of line graphs.	from s.LINE COLORS
linestyle	gT, □, ▭	The style of line graphs, e.g., solid or dashed.	from s.LINE STYLES
linewidth	gT, □, ▭	The width in pixels of line graphs.	0 (thinnest)
	P	The percent of the character width to be used to draw the lines specified by the line attribute.	10 (%).
literal	ALL, p	If 1, display character values in single quotes; if 0, don't.	0
lock position	hP, L, O, vP	If 1, do not reposition the children in this object when some other children are removed (e.g., freed) or unmapped.	
locksize	hP, L, O, vP	If 1, do not resize the layout when some children are removed (e.g., freed) or unmapped.	
lower	TOP, a, s	Specify this attribute to send this object to the back of the visible windows.	
majortick size	GS	The length in pixels of the major ticks on the scale.	10
mapped	ALL (NFT)	The value is 1 if this object is mapped (to the screen), and 0 if unmapped. There are two cases: 1. If this object is a top-level or popup window, its show attribute has value 1, and it is unmapped, then it is shown with blank contents. 2. If it is not a top-level or popup window, its show attribute has value 1, and it or one of	

		its ancestors is unmapped, then if it's shown, it is shown blank.	
margin	B, U	Pixel width around text.	
margin height	O	The top and bottom margins of an object in its page, in pixels.	4
margins	Rp, s	Use to set all four margins to the same value.	
margin width	O	The left and right margins of an object in its page, in pixels.	4
max	GS	The maximum graphically displayed value. It is possible for it to be less than the value label.	100
maxtitle	GS	The label that is shown for max. If null, there is no label and no space allowance for a label.	Null
maxtitlefg	GS	The color in which maxtitle appears.	black
maxtitle font	GS	The font in which maxtitle appears.	lucidasanstype writer-bold-12
maxtitle justify	GS	The justification for maxtitle. One of *`none* (no maxtitle), *`top*, *`bottom* (meaningful for hG, hS), *`left*, *`right* (meaningful for vG, vS), *`center*.	*`center*
min	GS	The minimum graphically displayed value. It is possible for it to be greater than the value label.	0
minortick count	GS	The number of minor ticks between major ticks on the scale.	1
minortick size	GS	The length in pixels of the minor ticks on the scale.	6
mintitle	GS	The same as maxtitle, but for min.	Null
mintitlefg	GS	The same as maxtitlefg, but for min.	black
mintitle font	GS	The same as maxtitlefont, but for min.	lucidasanstype writer-bold-12
mintitle justify	GS	The same as maxtitlefg, but for min.	*`center*

mnemonics	hM, vM	Keyboard accelerators for menus; a slotfiller of the same shape as the menu variable. The accelerator is the first letter of the symbolic index; the values are ignored. If several buttons at the same level are given the same accelerator, it is underlined in all of them but that key affects only the first of those buttons. Note that the default for accelerators when `s.MENUDEFAULTMNEMONIC` is 1 (menu variable initials) does not address this potential problem; where duplicate initials exist, mnemonics must be set specifically to provide a full set of accelerators.	`2ρ<nρ()`
mode	G, ▭	Set up interactive entry (`` `addtrace ``) or set up text entry at `` `coordinate `` (`` `addtexttrace ``), or terminate entry, like **Enter** or double click (`` `normal ``).	
movelimit	gT, ☐, ▭	Specify whether the x coordinate (`` `x ``) or y coordinate (`` `y ``) or neither (`` `none ``) remains fixed when the pointer is used to move a trace point.	`` `x ``
na	ALL (A, M, T, tF), p	The value used for NA in the object, e.g., when new rows or cells are inserted. Cf. `blank`, and `s.NA` and `t.NA`.	`(∨x)⊃s.NA` for variable *x*.
naturalsize	NFT, a, s	The number of visible rows and columns may change with changes in the size of an object or its column widths. Specify naturalsize to make the number of visible rows and columns the same as when it was first mapped - or, for a container, to do so for its contained objects. At present, at least, setting naturalsize also sets firstrow and firstcol to zero.	
newshow	CNT, G, T, W	For an existing container: if 1, a reparented child (see reparent) is always reshown and an unbound variable is shown; if 0, a reparented child is reshown only if mapped and an unbound variable is not shown; if Null, use the value of `s.AUTONEWSHOW`.	0
newspaper column	T	Number of columns in which the table is arranged on the report page (allows a narrow table to fill a page better).	1
nodebg	R	The background color of the nodes.	`` `violetred ``

nodefg	R	The foreground color of the nodes.	`` `yellow ``
notify	NFT, ☐	Furnishes a callback function for a notify event. See "Accessing the Primary Selection Buffer".	
orientation	O, R, Rp	Notebook (orientation on page) and tree: `` `horizontal `` or `` `vertical ``. Report (orientation of report on paper): `` `landscape `` or `` `portrait ``.	`` `horizontal `` or `` `landscape ``
out	ALL (A, C, M, Q, S, T, tF), ☐, p	Specifies the format in which data appears in a display. The value can be a valid left argument to ⍕ (Format) or *_fmt*, in which case the data are formatted according to the rules of those functions. In addition, the value can be one of those in the "out Attribute Format Samples" table. Also see the *_sfmt* function. If the object is a slotfiller, individual formats can be given, as in ``(`out;(fmt1;fmt2;...;fmtn))``. If the value of this attribute is a character matrix and the data to be displayed are valid row indices of the matrix, then each data element is replaced in the display by the text of the row it indexes.	*out x:{ 1↓⍕x}* for numeric; *out x:x* for character; *out x:{ s.box{x}}* otherwise. Sensitive to the blank attribute.
outofcurrent workspace	TOP, cb	In a CDE window, an event callback that is called when the window loses presence in the current workspace: that is, when the presence of a window is changed to exclude the current workspace or when the current workspace is changed to a workspace which does not include this window from a workspace which included it. Cf. incurrentworkspace. See the caveats for deiconized in the callback section. Ignored outside CDE.	
outputmode	Rp	`` `ps ``: PostScript; `` `eps ``: Encapsulated PostScript; `` `ppm ``: Portable Pixmap.	`` `ps ``
outputstyle	T	`` `columnmajor ``, `` `rowmajor `` - report page arrangement. `` `aligned ``, `` `noheadings `` - for ASCII tables.	`` `rowmajor ``
pagebreakcb	Rp, ☐	Function and optional static data for page breaks.	
page	tF	When set to 1, a page break is inserted after a	0

breakon		column break.	
page changecb	O, □	Callback when the current page is changed by a user.	Null (none)
pagecontrol	Rp	Vector listing pages to be printed.	Null (all)
pagecount	Rp, r	Number of pages printed so far.	
pagecount total	Rp, r	Total number of pages in the whole report.	
pageinc	GS	The amount of the increase in value when **PageUp** is pressed and decrease when **PageDown** is pressed. Accepted for gauges, but meaningful only for scales.	10
page numbering	Rp	Print page numbers (1) or not (0).	1
page numbers	Rp	Vector of numbers: i-th element is page number of i-th page.	Null (i for i)
pagenumber text	Rp	Vector or nested vector of text to be printed with page numbers.	
page selection	ALL (except gT, Rp, tF)	If 1, when the object is a child of a notebook, its page can be selected; if 0, its page cannot be selected and its pagetitle (on the tab) is lightened. If its page is already selected when its pageselection attribute is set to 0, the page remains displayed (but its pagetitle is lightened).	Null (1; *s.of* says 1 when in notebook and null when not).
pagesize	Rp	Page size: one of `legal (8.5x14"); `letter or `a (both 8.5x11"); `b (11x17"); `a4 (210x279mm); or `b5 (182x257mm). *May not work on non-HP printers.*	`letter
pagetitle	ALL (except gT, Rp, tF), but *only* when the child of a notebook	The title to be shown on the notebook tab for an object that is a child of a notebook.	Null (the title, which itself has the variable name as default).
parent	ALL (except	The object whose variable explicitly contains this object. Top level objects are their own parents	

	report),	Inclusion in a report is ignored - a report object cannot be a parent -, and the parent attribute is not defined for report objects.	
pieangle	gT	If pieprimaryslicealign is `` `none ``, pieangle gives the orientation of the primary slice of a pie chart in counterclockwise degrees from the three o'clock position.	0
pie aspectratio	gT	The extent to which a pie chart is tilted towards the viewer, ranging from 0.0 to 1.0. 1.0 is a top view, or flat pie.	0.6
pie depthfactor	gT	The maximum height of the pie in a pie chart, as a fraction of the standard depth factor. Cf. pieprofiles.	1.0
pie legendalign	gT	Where the legend for each slice of a pie chart is placed. Values are null, `` `none ``, `` `inside ``, `` `outside ``, `` `left ``, `` `right ``, `` `top ``, `` `center ``, and `` `bottom ``. The last five control placement with regard to values and percentages and will be ignored if not consistent with the values of pievaluealign and piepercentalign. Several values may be concatenated, as in `` `inside`top`left ``. The color is controlled by the trace's titlefg attribute.	`` `center ``
pie offsetmargin	G	The margin around pie charts, as a fraction of the available space. The value ranges between 0 (inclusive) and 1 (exclusive). 0 makes pie charts as large as possible in the graph. 0.99 makes them dots. Values close to 1 cause calculation overflows, which appear as A+ domain errors.	Approx 0.3
pieoffsets	gT	The extent to which each slice is separated (pulled out) from the center of a pie chart. Values between 0.0 and 1.0 (inclusive) are treated as fractions of the pie's radius. Values greater than 1.0 and equal to or less than 100.0 are treated as percentages of the pie's radius. The value or values given are used cyclically for the slices.	0
pie percentalign	gT	Where to place the display of the percentage that each slice is of a pie. Values are null, `` `none ``, `` `inside ``, `` `outside ``, `` `left ``, `` `right ``, `` `top ``, `` `center ``, and `` `bottom ``. The last five control placement with regard to values and legends and and will be ignored if not consistent with the values of pievaluealign and pielegendalign. Several values may be	Null

		concatenated, as in `` `inside`top`left``. The color is controlled by the trace's titlefg attribute.	
pie primaryslice	gT	The trace index of the slice to be considered the "primary" slice of a pie chart, the slice whose display is controlled by pieprimaryslicealign or pieangle. The value -1 means that the largest slice will be considered the primary slice. For a brief time, called primaryslice.	-1
pieprimary slice align	gT	Where to place the primary slice of a pie chart (by rotating the pie). The possible values are null, `` `none`` (see pieangle), `` `left``, `` `right``, `` `top``, `` `bottom``. For a brief time, called primaryslicealign.	Null (`` `top``)
pieprofiles	gT	The height of each slice of a pie, as compared to the maximum height of the pie. Values between 0.0 and 1.0 (inclusive) are treated as fractions of the pie's maximum height. Values greater than 1.0 and equal to or less than 100.0 are treated as percentages of the pie's maximum height. The value or values given are used cyclically for the slices.	1.0
pie valuealign	gT	Where to place the display of the value of each slice of a pie chart. Values are null, `` `none``, `` `inside``, `` `outside``, `` `left``, `` `right``, `` `top``, `` `center``, and `` `bottom``. The last five control placement with regard to percentages and legends and will be ignored if not consistent with the values of piepercentalign and pielegendalign. Several values may be concatenated, as in `` `inside`top`left``. The format of the value display is controlled by the out attribute, and the color is controlled by the trace's titlefg attribute.	Null
pin	TOP (popups only), ☐	If 1 and this object is a popup window, the window is pinned; if 0, it is unpinned.	0 if a popup
position	hR, L, vR, ☐	Applies to simple vector layouts unless `s.AUTOBUILD` is 1 and the objects in the layout already appear on the screen. If 1, the layout will be in the default vertical form; if 0, all its children have their at attribute equal to 0 0 1 1, so that they overlap; if Null, use the value of `s.AUTOPOSITION`	1

		Note: this attribute must be set before the layout, `lout`, say, is built: i.e., start with `lout←()`, set the attribute value, and then set `lout` to its correct value.	
preset	ALL, p	Specify a preset callback on this object.	
primary	NFT	Set this attribute in order to specify the primary selection buffer, and reference it to read that buffer. See "Accessing the Primary Selection Buffer".	
primaryslice	gT	Briefly called this, but now called pieprimaryslice.	
primaryslice align	gT	Briefly called this, but now called pieprimaryslicealign.	
print	NFT, a	Specify this attribute to "print" the object in a file. If running in batch mode, not in Emacs, allow a slight delay between mapping a file and printing it, to avoid a race condition: for example, if a legend is specified for a graph and the graph is printed *immediately*, the legend may not appear in the print file.	
printable	ALL, p	If 1, printable by the opt gadget. See /aplus/lib/s/ If Null, use *`s.AUTOPRINTABLE`*.	1
printbottom	NFT	The bottom page margin, in inches.	1
print defaultfont	NFT	The printer font used when the display font is unavailable.	courier
printdis bottom margin	NFT	Disclaimer offset from bottom edge of paper, whether in portrait or landscape orientation.	.27 in.
printdisfile	NFT	Path and file name of an ASCII text file. Spacing between words is honored and newlines are ignored.	none
printdisfont	NFT	The font in which to print the disclaimer.	times-roman-6
printdis left margin	NFT	Disclaimer offset from left edge of paper, whether in portrait or landscape orientation.	.27 in.
printdis	NFT	Controls the orientation of the disclaimer:	*`` `none ``*

orientation		same as object if `none or printdisstyle is `append; else `portrait or `landscape regardless of object orientation.	
printdis right margin	NFT	Disclaimer offset from right edge of paper, whether in portrait or landscape orientation.	.27 in.
printdis rulewidth	NFT	The width of the disclaimer rules. See printdisstyle, below.	0 (very thin)
printdis style	NFT	Governs printed disclaimer: `none; `text only; text in `box; text with `rule above and below; text below `toprule; or text `appended to top-level window (scrollbars cause offset in both directions by their width).	`none
printdistop margin	NFT	Minimum distance between disclaimer text and bottom of the object being printed. Maximum setting is one inch.	.27 in.
printfile	NFT	Name of the file to receive the printed object. Set printfile to null to restore the default. Use the pageview or ghostview command to show print.ps on a display screen.	print.ps
printleft	NFT	The left page margin, in inches.	1
printmode	NFT	Rendition of the object: `mono - black and white;  `color - color on a color printer, grayscale on others; `reverse - reverse black and white;  `colorfg - white background and true foreground colors (i.e., colors set with fg attributes) on a color printer, variable shading on a noncolor printer.	`mono
print orientation	NFT	`portrait or `landscape (rotated 90 degrees).	`landscape
printoutput mode	NFT	`ps for Postscript. `eps for encapsulated Postscript and `ppm for	`ps

		portable pixmap; in these modes, printing the disclaimer is disabled, and if printfile is of the form `fn.ps` the file that receives the printed object is `fn.eps` or `fn.ppm`, respectively. A ppm file can be converted to a gif file using the Unix command ppmtogif, and Mosaic and Netscape can display gif files directly, as image references in html files.	
printright	NFT	The right page margin, in inches.	1
printsize	NFT	Page size, one of: `` `legal`` (8.5x14"); `` `letter`` or `` `a`` (both 8.5x11"); `` `b`` (11x17"); `` `a4`` (210x279mm); or `` `b5`` (182x257mm). *May not work on non-HP printers.*	`` `letter``
printstyle	NFT	Placement on the paper. One or more of `` `left``, `` `right``, `` `top``, `` `bottom``, `` `center``, and `` `none``.	`` `center``
printtop	NFT	The top page margin, in inches.	1
protect	ALL (A, C, E, gT, I, K, M, N, Q, S, T, tF, U), □, P	If 1, the values in this object cannot be edited (in the case of a graph trace, a point can't be dragged to a new position); an attempt to do so will elicit a beep. For array, check, page, slot, and tableField objects, this attribute can be given a list of values, which is used cyclically - in the manner of the A+ primitive function Reshape - to protect each cell, or value area, individually. A table can be protected entirely or not at all; if it is protected, then so are all its fields, independent of their individual protect settings; see protected, immediately below. Note that the setting of this attribute has no effect on behavior controlled by the insertabove, insertbelow, select and delete attributes.	0
protected	ALL (A, C	If 1, protection, as defined by the protect attribute	

	, E, gT, K, M, N, Q, S, T, tF, U), p, r	is on for this object. This attribute is most useful for tableFields because the protect setting for the table overrides the settings on the individual tableFields.	
r	CNFT	If 1, right justify this object in its layout cell.	0
R	hP, I, K, L, N, O, S, vP	If 1, make all virtual rows of this object the same size.	
raise	TOP, a, s	Specify this attribute to bring this object to the front.	
rband	P, ☐, ☐, ☐	The action taken when a rubberband box is created. If rband is 0, none. If 1, the default. When a rubberband box is created, the rbandbox attribute is set no matter what the setting of rband, and even for boxes whose number of rows and columns is 0 0.	Draw outline and append new rbandbox setting to box, *unless* 0 0 size.
rbandbox	P, r, ☐	(row, col, #rows, #cols) of the most recent rubberband box.	0 0 0 0
realize	CNFT, a, s	Specify this attribute to completely prepare this object for showing. Then when this layout object is shown, it will appear all at once, and not in a staggered fashion.	
recursively	CNT, G, T, W, a, s	Propagates an attribute value to the object and all its descendents; the value of recursively is an attribute-value pair, as in `` `v has (`recursively; (`attr;val)) `` which is tantamount to `` (`v,⊃`descendents of `v) has¨ <(`attr;val) ``	
refer	A, C, D; G, gT(☐); M, Q, R, S, T (☐), V; ☐(G, gT), ☐ (the rest), ☐	"Double click" left mouse button in a cell. The default action is to highlight the referred row. See index. Set refer to 0 to turn off the left mouse button, and 1 to get the default behavior. Press the left mouse button on a referred row to remove the reference.	0
referpoint	gT, ☐, ☐	This action controls a referpoint event. Note, for coincident points: trace search order is the same as the drawing order.	
refresh	NFT, a, s	Refresh the visual appearance of this object to	

		reflect recent settings of display related attributes such as fg and out.	
reparent	CNT, G, T, W	If 1, a child removed from the container will become a top-level object; if 0, it is freed; if Null, use the value of *s.AUTOREPARENT*.	Null (1)
report	Rp, a	Print all of the object, not just what is shown on the screen, perhaps on several pages.	
reportfont	ALL (T, tF), fn, p	Font for report. If it is not set on tableField, the value for table is used.	
report headingfont	ALL (T, tF), fn	Font for column heading in report. If it is not set on tableField, the value for table is used.	
report totalfont	T	The font in which the totals (if shown) appear.	
reporttotal leading	T	The vertical space, in points, around totals, if shown.	2
report totalon	T	Whether totals are shown in reports (1) or not (0).	0
report totalstyle	T	One or more of `` `left `right `top `bottom``.	`` `left``
request	NFT, a, s	Specify this attribute to trigger a subsequent notify event. See "Accessing the Primary Selection Buffer".	
reshow	CNT, G, T, W	If 1, a reparented child (see reparent) is reshown if it was shown in its previous parent; if 0, it is not; if Null, use the value of *s.AUTORESHOW*.	Null (0)
resize	CNFT, ▭, ▭	Constrains resizing the object in a layout, and controls its position within its allotted layout area. Each setting (character string or symbol) is the total constraint if it contains a period; otherwise it is additional to previous settings. Set to Null to get back to the default setting, and to `''` or `'.'` to eliminate all constraints. Setting h, H, w, or W to 1 (but not to 0) sets resize; setting resize sets these four attributes.	button, label, scalar, check, radio, and slot: `'hH'` (or `` `hH``); all others: `''`.
resizeable	TOP	If set to 1, this object can be resized interactively by a viewer; if 0, it cannot. This attribute must be	

		set before this object is mapped.	
respace	ALL (A, C, M, Q, S, T, tF), p	If respace is 1, the space attribute value is increased if it is less than the length of the result of the *default* out function; if 0, it is not; if Null, use the value of *`s.AUTORESPACE`*.	Null (0)
right	G, ▭	Distance from the right side of the graph window to the Y-axis rule (visible or not), as a percentage of window width.	0
right margin	Rp	The right margin of the report, in inches.	1
rightto	NFT	Object to which **Shift-right-arrow** moves keyboard focus. Connected to arrowlist. [Future use]	Null
row	A, M, T, V	The row index of the selected cell. The value can be -1 to indicate that no row or cell is selected. Setting this value causes the selected row to appear in the display, and, if the col attribute is set to a valid column index, the selected cell at position (row, col) to be highlighted.	-1
rowbg	A, M, T, V	The background color of the selected row. The selected cell is always in the selected row; the value of the selectbg attribute is used in place of the rowbg attribute for this cell.	lightsteelblue3
rowcontrol	T	Vector of table rows to be shown on page.	
rowindex	M	A vector of indices of the selected row labels when the selectrow attribute is 1, in the order of selection.	`ι0`
row indexbg	M	The background color of the indexed row labels.	medium aquamarine
row pagespan	T	Number of pages that the table rows are to span.	0

rows	A, M, T, V, X	The number of visible rows in the display of this object.	at most 5;
			for text, 5.

rowsep	A, M, T, V	If n, row separators appear every n rows. 0: no separators.	1
rule	G, ▭	Specify which axis rules appear.	`` `std ``

rulewidth	G, ▭	The width of the axes rules and tick marks, in pixels.	0
save	X, □	If 1, the value of the underlying character vector is updated to match the text that appears on the screen (or that can be made to appear with the scrollbars) whenever the user presses the **Control-s** key combination. If save is 0, pressing **Control-s** has no effect.	1
scalefont	Rp	Whether or not to scale the table font when row or column constraints exceed the page size. If the font is not scaled, the exceeded constraints are ignored.	1
scalefontcb	Rp, □	Function and optional static data for variable-type callback when fonts are scaled.	
script	ALL (NFT), p	If 1, the s definition of this object will appear in the result of `s.script{}`; if 0, it will not appear; if Null, use the value of *`s.AUTOSCRIPT`*.	Null (1)
scrollbg	A, M, T, V, W	If one element, the background color of any scrollbars on this object; if two, the vertical, horizontal scrollbar colors. Also see bg.	grey, grey
scrollsize	A, M, T, V, W	The height in pixels of any scrollbars on this object.	15
select	A, M, R, T, V, □, ▭	Function and static data for an event callback triggered by a left mouse button click on a value area, movement from one field to another by an **arrow** key, or a deletion that deletes the last row (changing the row attribute).	Select the cell holding the pointer.
selectable	gT, □, ▭	If 1, refer events are active for this line or text trace; if 0, they are not. For selectable to be nonzero, the trace's trace set must have exactly two columns (i.e., just one trace in the set, with x coordinates explicitly given).	1 if the trace is user created; 0 otherwise.
selectbg	A, M, T, V	The background color of a selected cell.	grey
selectcol	M, □, ▭	This action controls a select column event.	0

select corner	M, □, ▭	This action controls a select corner event.	0
select distance	G, ▭	The maximum distance in pixels between the pointer and a trace point in order that the point be selected for moving.	10
selected	A, G, gT (▭), I, K, M, N, R, S, T, V	Row and column indices or symbolic index of selected cell or trace point; for a graph, the selected trace set object. Also for tree.	
selected field	T	The value is a pair consisting of the row index and the tableField name (as a symbol) of the selected cell. A row index of -1 means none selected.	`(-1;`*`x`*`)`, *`x`* being the object in the leftmost column.
selected nodebg	R	The background color of the currently selected node. Changes when nodefg is changed if the two are alike.	`` ` ``*`nodefg`*
selected nodefg	R	The foreground color of the currently selected node. Changes when nodebg is changed if the two are alike.	`` ` ``*`nodebg`*
selected pagebg	O	The background color of the currently selected page.	`` ` ``*`grey`*
selected pagefg	O	The foreground color of the currently selected page.	`` ` ``*`black`*
select field	T, □, ▭	Mouse button event on the title area of a field. The default action is to select the field whose title area holds the pointer. The selected cell becomes the one at the intersection of the selected row and selected field.	See Description.
selection mode	A, M, T, V, ▭	Controls whether only one row can be selected at a time (`` ` ``*`single`*) or several (`` ` ``*`multiple`*). When changed to `` ` ``*`single`*, the index attribute is set to `ι0` and all rows but the one (if any) indicated in the selected attribute are "deselected"; when changed to `` ` ``*`multiple`*, index is set to the row (if any) indicated in selected.	`` ` ``*`multiple`* for array and view; `` ` ``*`single`* for table.
selectrow	M, □, ▭	This action controls a select row event.	0
sensitive	NFT	If 0, all user interactions with this object and its descendents are blocked.	1

set	ALL, p	Specify a set callback on this object.	
setcol	A, M, T, V, cb	Callback has been disabled; use the select callback instead. Here is the definition anyway: If 0 (the default), no callback when col is set. If 1, when a column is selected by user or program, but not by a setcol callback, the default callback is executed (see next column). If `(f;s)`, when a column is selected by user or program, but not by a setcol callback, `f{s;c;v}` is called.	Set col to the same value in the objects named in hscrollwith.
setfirstcol	A, M, T, V, cb	If 0 (the default), no callback when firstcol is set. If 1, when firstcol is set by user or program, but not by a setfirstcol callback, the default callback is executed (see next column). If `(f;s)`, when firstcol is set by user or program, but not by a setfirstcol callback, `f{s;c;v}` is called. Note that a setfirstcol callback can be triggered when firstcol is set as a result of the selection (by program) of a column which is not currently being shown.	Set firstcol to the same value in the objects named in hscrollwith.
setfirstrow	A, M, T, V, cb	Like setfirstcol, but for vertical scrolling and vscrollwith.	Like setfirstcol
setrow	A, M, T, V, cb	Like setcol, but for row selection and vscrollwith. Callback has been disabled; use the select callback instead.	Like setcol
settings	ALL, r	``(`settings;v) of `object`` is, for a symbol scalar or vector *v*, a slotfiller holding, generally speaking, the attributes and values of the attributes named in *v* for `` `object`` that have nondefault values; if *v* is absent, as in ``\`settings of `object``, the result is a slotfiller holding, generally speaking, the attributes and values of all attributes for `` `object`` that have nondefault values.	
shadow thickness	NFT, ▭	The width in pixels of the shadow area around this object.	0; 2 for graph.
shell	ALL, r	The top-level object containing this object.	
shelltitle	TOP	The title of this object in the window manager's title area. The default is: for the first top-level object shown, or the object specified in ⎕WS, either the contents of	See Description.

		`s.WSNAME`, or if `_argv` is not empty, `0⊃_argv`; for all others, the name of the object.	
show	NFT, a	If 1, show this object; if 0, hide it. I.e., show and hide a top-level object on the screen, and any other object in its container; see the `show` and `hide` functions. Cf. the realize and syncshow attributes.	
show binding	O	Show a spiral binding on the notebook (1) or not (0).	1
show popup	O	When the right mouse button is pressed with the pointer on the notebook, show a popup menu allowing selection of a page (1), or do not respond to the button press (0).	1
showtabs	O	Show tabs at edges of pages of notebook (1) or not (0).	1
size	A, M, T, V, a, s	Setting size eliminates partial rows and columns.	
sliderbg	GS	The color of the slider, the moving part in a gauge or scale.	grey
slider height	GS	The vertical dimension in pixels of the slider, the moving part in a gauge or scale. A setting given for a vgauge is ignored.	vscale: 30; hscale and hgauge: 14.
slider width	GS	The horizontal dimension in pixels of the slider, the moving part in a gauge or scale. A setting given for an hgauge is ignored.	vscale and vgauge: 14; hscale: 30.
space	ALL (A, C, M, Q, S, T, tF), p	Width in characters of the data display. If the space originally needed to display the data, computed using the out attribute, exceeds the space attribute, then this attribute is increased to the space needed. If data are changed in an object already being displayed and the space becomes insufficient, then the display is governed by the respace and stars attributes. Setting the title or out attribute increases the space attribute if necessary to fit title or data.	depends on the data; cf. `s.WP` and `s.DATASPACE`
stars	ALL (A, M, Q, S, T, tF), p	If 1, fill a cell or row with `*`'s when there is insufficient room for the value; if 0, truncate the value display.	1
state	ALL, Rp, r	A character vector holding the s definition of this	

		object, but not the value of this variable. In a layout, only one definition is shown for any name that appears in it several times, for economy. For tableFields and graphTraces, the class does not appear in the result of state, to keep it executable.	
stateself	ALL, Rp, r	Like state, but for the object only, without recursion.	
structure	CNT, r	The value is a slotfiller object with the settings of the at attribute for all children of this object.	
style	gT(▭), T, tF, □	The type of graphic representation for a trace set. Also, the style of a column, one or more of `` `left `right `top `bottom``. If it is not set for a tableField, the value for its table is used, if set.	`` `line `` for a trace set. Null (`` `left ``) for tableField.
subtitle	G(▭), GS	The subtitle text.	
subtitlefg	G(▭), GS	The color in which the subtitle appears.	`` `axiscolor `` for graph.
subtitle font	G(▭), GS	The font in which the subtitle is type set.	lucidasans type writer-12 for graph.
subtitle justify	G(▭), GS	Justify the subtitle.	`` `center `` for graph.
suppress duplicate	T	Suppress (show as blank) any tableField entry that duplicates its immediate predecessor (1) or show it (0).	0
symbol	gT, □, ▭	The symbol marking the data points in a scatter trace.	`` `cross ``
symbolsize	gT, □, ▭	The size in pixels of the symbols in a scatter trace.	10
syncshow	NFT, a	Like show, except that ``(`syncshow;1)`` waits, unlike ``(`show;1)``, processing X-Events until its own MapNotify event is found. Processing then continues normally. This avoids the problem that occurs when labels and views in layouts are used as progress indicators (status reports on program initialization, for instance) but are not refreshed until after the program has finished loading and finally hits the mainloop.	

t	CNFT	If 1, top justify this object in its layout cell.	0
tabfrom	NFT	The object preceding this object in traversal order; if this object has focus, then press **Meta-Shift-Tab** or, on IBM keyboards, **Alt-Shift-Tab** to give that object focus.	
tablist	TOP	Vector of objects within this object in their order of traversal; pressing **Meta-Tab** or, on IBM keyboards, **Alt-Tab** repeatedly will give focus to these objects in the stated order. Each object can appear only once in the list.	
tabto	NFT	The object following this object in traversal order; if this object has focus, then **Meta-Tab** or, on IBM keyboards, **Alt-Tab** to give that object focus.	
textactivate	gT, ☐, ▭	The action taken when this text trace is interactively modified.	
textfg	gT	Color of text trace; symbol or character string (converted to symbol by s).	`` `white ``
textfont	gT	Font of text trace; symbol or character string (converted to character by s).	lucidasans type writer-12
3down	P, ☐	A 3down event occurs when the right mouse button is pressed.	
3up	P, ☐	A 3up event occurs when the right mouse button is released.	
title	ALL(except B, hM, hR, vM, vR, X), p, ▭	The text in the title area of this object. The value can be: for one row, a character vector or scalar, or a symbol; for one or more rows, a character matrix, a nested vector of character vectors or scalars, or a simple symbol vector. There are various other possibilities, whose acceptance and interpretation depend upon the display class involved; avoid them! For graphTrace, meaningful for text traces only.	See "Default and Coincident Titles".
titlefg	ALL(except B, hM, hR, vM, vR, X), p	The color in which the text of the title appears. For graphTrace, meaningful for text and pie traces only. A tableField inherits its value from its table, and when the value for tableField is reset to Null it uses the current value for table.	black
titlefont	ALL(except B, hM, hR	The font in which the title is set. For graphTrace, meaningful for text and pie traces only. A	kaplgallant

	vM, vR, X), p	tableField inherits its value from its table, and when the value for tableField is reset to Null it uses the current value for table.	
titlejustify	G(□), GS, NFT from Rel. 4.10	Justify the title: `` `left `right `bottom `top `center `none``.	`` `center``
top	G, □	The distance from the top of the graph window to the X-axis rule (visible or not), as a percentage of the window height.	0
topmargin	Rp	The top margin of the report, in inches.	1
2down	P, □	A 2down event occurs when the middle mouse button is pressed.	
2up	P, □	A 2up event occurs when the middle mouse button is released.	
underline	P, □	Boolean mask indicating underlined characters.	
uniform scaling	Rp	Scale proportionally on x and y axes (1) or not (0).	1
upto	NFT	Object to which **Shift-up-arrow** moves keyboard focus. Connected to arrowlist. [Future use]	Null
valid	D, r	1 if the text entered in the value area matches the password for the user name in the global variable; 0 otherwise.	0
validate	D, □	The function that is called when keyboard entry is completed (by an **Enter**).	
valuefg	GS	The color of the value label, which shows the present value of the underlying variable.	black
valuefont	GS	The font of the value label, which shows the present value of the underlying variable.	lucidasans typewriter bold-12
value justify	GS	The position of the value label, which shows the present value of the underlying variable. Values: `` `top `bottom `left `right `center `none``. The last deletes the number.	`` `center `top``
vcol	CNFT	See the at attribute.	
vcols	CNFT	See the at attribute.	

vcolspace	hP, L, O, vP	Space between the virtual columns of a layout, in pixels.	Null (0)
verify	ALL	Most display classes require a variable to satisfy certain rank and type restrictions. If 0, the A+ interpreter verifies them; when a bound variable is assigned a value inappropriate to its display class, an " ←: `invalid`" message is issued and execution is suspended. If 1, s context functions verify them; an inappropriate value elicits a more meaningful message, followed by a stop. If Null, the value of `s.VERIFY` is used. [Future use]	0
vertical space	R	The minimum distance between vertically aligned nodes, in pixels.	5
vrow	CNFT	See the at attribute.	
vrows	CNFT	See the at attribute.	
vrow space	hP, L, O, vP	Space between virtual rows of a layout, in pixels.	Null (0)
vscroll size	A, M, T, V	The width in pixels of a vertical scrollbar on this object.	15
vscroll with	A, M, T, V, ▭	Like hscrollwith, but vertical scrolling, setfirstrow, setrow.	0 ρ `
w	CNFT	If 1, do not change the width of the row containing this object on resize.	1 for vmenu
W	CNFT	If 1, do not change the width of this object upon resize.	1 for vmenu
ws	ALL (NFT), P, ▭	When an object is (initially) shown and `s.WS` is Null, if ws for the object is 1 it becomes the screen workspace and if ws is 0 it does not; when ws is Null, the value of `s.AUTOWS` is used for it.	Null (1)
x	NFT	The x coordinate, in pixels, of this object relative to its parent, or relative to the screen in case this is a top-level object.	0 (upper left corner)
X	NFT, r	The x coordinate, in pixels, of this object relative to the screen.	0 if toplevel or popup

xaxis	gT, ☐, ▭	Governs whether a trace is graphed relative to the x axis (`` `x ``) or X axis (`` `X ``).	`` `x ``
xextent	G, o, r, ▭	A three-element vector from which the length, in pixels, of the x axis can be computed. Can be used to detect zooming.	
xfg	G, o, ▭	The color of the indicated axis, axis labels, and tick marks.	black
xinc	G, o, ▭	The coordinate increment between major tick marks.	0 (s chooses)
xlabel	G, o, ☐, ▭	Various explicit settings of x-axis label text and tick mark positions and sizes.	
xlabelfont	G, o, ▭	The font in which the x-axis labels are type set.	lucida sans type writer-12
xlabel height	G, o, r, ▭	The height in pixels of any x-axis label.	
xlabel justify	G, o, ▭	Justify the x-axis labels: `` `left ``, `` `right ``, `` `center ``.	`` `center ``
xlabelout	G, o, ☐, ▭	Controls formatting of x-axis labels. These labels may have any values (integer or not) between xmin and xmax or, if xmin is not less than xmax, *any* floating-point values. Like out, xlabelout takes a function, a symbol, or a character string; unlike out, it does not take a number.	
xlabel width	G, o, r, ▭	The width in pixels of a specified x-axis label.	
xleft	G, x, ▭	The distance from the rule of the y axis to the bounding rectangle that holds all graph traces whose xaxis setting is `` `x ``, as a percentage of the distance between the rules of the y axis and Y axis (visible or not).	0
xlegend	G, ▭	The x-axis coordinate of the upper left corner of the legend box (when legend is not set after xlegend is set).	0
xmajor ticksize	G, o, ▭	The length in pixels of the major tick marks on the x axis.	10
xmax	G, o, ▭	The maximum value on the x axis. For xmax to be honored, it must be greater than xmin.	Computed value.

		Setting them all to Null also causes values computed from the trace sets to be used, but these attributes are reported as having the computed values. Any zooming that the user has done on the screen can be detected in a program by the values of xmax, etc. To undo zooming (in order, for example, to ensure that a new point is shown promptly), reset xmin, xmax, ymin, and ymax to valid values.	
xmin	G, o, ▭	The minimum value on the x axis. For xmin to be honored, it must be less than xmax. Can be used to undo zooming. See xmax.	like xmax
xminor ticks	G, o, ▭	Number of minor ticks between major ticks on the x axis.	1
xminor ticksize	G, o, ▭	The length in pixels of the minor tick marks on the x axis.	6
xright	G, x, ▭	The distance from the rule of the Y axis to the bounding rectangle that holds all graph traces whose xaxis setting is `` `x ``, as a percentage of the distance between the rules of the y axis and Y axis (visible or not).	0
xs	NFT(should be set only for TOP)	The horizontal size of this object, in pixels. If the object is not displayed when this attribute is set, the realize attribute for the object must have value 1.	
xsublabel	G, x, ☐, ▭	Various explicit settings of x-axis sublabel text and positions.	
xsublabel justify	G, x, ▭	Justify the x-axis sublabels: `` `left ``, `` `right ``, `` `center ``.	`` `center ``
xsublabel out	G, x, ☐, ▭	Controls formatting of x-axis sublabels. See xlabelout.	
xtickstyle	G, o, ▭	E.g., set the x-axis tick marks to point into the plot area.	`` `out ``
xtitle	G, o, ▭	The text of the title on the x axis.	
xtitlefg	G, o, ▭	The color of the text on the x-axis title.	black
xtitlefont	G, o, ▭	The font in which the text of the x-axis title is type set.	lucida sans type writer - 12

xtitle justify	G, o, ▭	Justify the x-axis title: `left, `right, `center.	`center
y	NFT	The y coordinate, in pixels, of this object relative to its parent, or relative to the screen in case this is a top-level object.	0 (upper left corner)
Y	NFT, r	The y coordinate (pixels) of the object relative to the screen.	0 if toplevel or popup
yaxis	gT, ☐, ▭	Specify whether a trace is graphed relative to the y axis or Y axis.	`y
ybottom	G, y, ▭	The distance from the rule of the x axis to the bounding rectangle that holds all graph traces whose yaxis setting is `y, as a percentage of the distance between the rules of the x axis and X axis (visible or not).	0
ylegend	G, ▭	The y-axis coordinate of the upper left corner of the legend box (when legend is not set after ylegend is set).	0
ymode	G, y, ▭	Controls which way the values run on the y axis and Y axis: `ascend or `descend.	`ascend
ys	NFT(should be set only for TOP)	The vertical size of this object, in pixels. If the object is not displayed when this attribute is set, the realize attribute for the object must have value 1.	
ytitle style	G, y, ▭	Set the y-axis title vertically or horizontally.	`hor
ytop	G, y, ▭	The distance from the rule of the X axis to the bounding rectangle that holds all graph traces whose yaxis setting is `y, as a percentage of the distance between the rules of the x axis and X axis (visible or not).	0
yx	NFT	The (y,x) coordinates, in pixels, of this object relative to its parent, or relative to the screen in case this is a top-level object.	0, 0 (upper left corner)
YX	NFT, r	The (y,x) coordinates, in pixels, of this object relative to the screen.	0, 0 if toplevel or popup
yxs	NFT(should be set only for TOP)	The (ys,xs) size, in pixels, of this object. If the object is not displayed when this attribute is set, the realize attribute for the object must have value 1.	

zero	G, ☐	Controls whether - for which axes - zero axes appear: `` `none ``, `` `x ``, `` `X ``, `` `y ``, `` `Y ``, `` `xy ``, `` `xY ``, `` `Xy ``, or `` `XY ``.	`` `xy ``
zerofg	G, ☐	Controls the color of the zero axes.	slategray
zerostyle	G, ☐	Controls the style of the zero axes: `` `dot1 `` through `` `dot5 ``, `` `dotdash1 `` through `` `dotdash5 ``, `` `dash1 `` through `` `dash5 ``, or `` `solid ``.	`` `dot1 ``
zerowidth	G, ☐	Controls the width of the zero axes: 0 through 10.	0 (very thin)

Formats for the out Attribute

Errors in dates return `0`'s except that an erroneous `` `y2 `` is set to "`**`", and the length of the result returned for a bad date is the same as for a good date.

The intraday formats in the fourth, fifth, and sixth rows of the "out Attribute Format Samples" table take an argument in seconds and use it mod 86400, the number of seconds in a day. For example:

```
     ↓t←sys.ts{}
 1994 10 12 13 22 19 550

     `hrminsec _sfmt 60⊥3 4 5#t
01:22:19pm
```

The formats in the last four rows of the table take an argument of the form yyyymmdd. For each of them, another format with the same name except that `_u` is appended handles the format for values given in Unix seconds (in the epoch). Unix seconds should be used for graphs with dates that are not contained in a single month, because the yyyymmdd form will cause the last day of one month and the first of the next to be widely separated in the graph.

Formats containing `y2` are restricted to the years 1950 through 2049.

out Attribute and _sfmt Format Samples

See the notes in the preceding section.

Format	Example		Format	Example

`float [(`float;n) accepted, but n ignored; of returns just `float].	97.7512	`fixed or (`fixed;n) where n is the number of decimal digits; of always returns the latter form.	97.751250
`frac	97 601/800	`32nd	97/24
`64th	97\48	`128th	97=096
`320th	97'241	`328th	97-241
`none	97.751250	`hr	2pm
`hr24	14	`min	32
`sec	15	`hrmin	2:32pm
`hrmin24	14:32	`minsec	32:15
`hrminsec	2:32:15pm	`hrminsec24	14:32:15
`day7	Thu	`day365	291
`day31	21	`m12	10
`m	Oct	`y2	93
`y4	1993	`dmy2	21/10/93
`dmy4	21/10/1993	`mdy2	10/21/93
`mdy4	10/21/1993	`mdy	Thu Oct 21, 1993

Default and Coincident Titles

Default titles are the names of the displayed global variables and are sensitive to their contexts. Fully qualified names are used for global variables outside the root context, except when a child object is in the same context as its parent, in which case the name of the child will be displayed without a context name. When a variable is reparented, if its title and qualified or unqualified name coincide, its title is set to Null, and so the parent then decides upon the appropriate title. (Therefore, if the title and the unqualified name coincide and the contexts of the parent and child are different, you must set the title just *after* the variable is placed in the parent, in order to avoid having the variable's context appear in its title.)

Colors

There are 292 named colors available. Many of the following color names have the suffix "4", which means that the four colors with suffixes "1", "2", "3", and "4" are available, in addition to the color without the suffix. For example, azure4 means there are colors azure, azure1, azure2, azure3, and azure4. The numbers indicate varying degrees of grayness in the color, with 1 signifying the least gray and 4 the most. Similarly but oppositely, there is a color grey100 denoting colors grey0 through grey100, where 100 indicates the least gray (white) and 0 the most gray (black).

Spaces in color names are for readability and need not be included in color specifications. Uppercase letters are not allowed.

The s-context variable `s.COLOR_NAMES` lists the permitted colors.

A+ finds the closest match to a requested color when the window-manager colormap is full. `of` returns the color that was requested, which allows the displayed color to be duplicated.

aquamarine4, azure4, beige, bisque4, black, blue4, brown4, burlywood4, chartreuse4, chocolate4, coral4, cornsilk4, cyan4, firebrick4, gainsboro, gold4, goldenrod4, green4, grey100, honeydew4, ivory4, khaki4, lavender, linen, magenta4, maroon4, moccasin, navy, orange4, orchid4, peru, pink4, plum4, purple4, red4, salmon4, seashell4, sienna4, snow4, tan4, thistle4, tomato4, turquoise4, violet, wheat4, white, yellow4.

Click here for a table showing a sample of each color.

Preferred Fonts

Since it is possible to obtain hardcopy printout for screen displays, the recommended fonts are the standard 35 PostScript fonts and APL. In the current A+ environment all the standard PostScript fonts can be scaled, and can therefore be used at any pixel size. To specify one of the screen management font attributes, use the name, followed by "-", followed by the pixel size: e.g., Courier-10. Use a character vector, not a symbol, to specify a font with a hyphen in its name. The standard PostScript fonts are:

avantgarde-book; avantgarde-bookoblique; avantgarde-demi; avantgarde-demioblique; bookman-demi; bookman-demiitalic; bookman-light; bookman-lightitalic; courier; courier-bold; courier-boldoblique; courier-oblique; Helvetica; helvetica-bold; helvetica-boldoblique; helvetica-narrow; helvetica-narrow-bold; helvetica-narrow-boldoblique; helvetica-narrow-oblique; helvetica-oblique; newcenturyschlbk-bold; newcenturyschlbk-bolditalic; newcenturyschlbk-italic; newcenturyschlbk-roman; palatino-bold; palatino-bolditalic; palatino-italic; palatino-roman; symbol; times-bold; times-bolditalic; times-italic; times-roman; zapfchancery-mediumitalic; zapfdingbats;

Kanji font is available on the screen, by setting the font attribute to

```
"-misc-fixed-medium-r-normal--14-130-75-75-c-140-jisx0208.1983-
0"
```
and using hex codes.

The APL font for printers can be scaled, but no scalable APL screen font is available. There are four APL screen fonts at various pixel sizes:

- `kaplcour (14 pixel font)`
- `kaplgallant (19 pixel font, aka bigapl)`
- `kaplscreen (11 pixel font)`
- `kaplscreen-bold (14 pixel font).`

The default for all font attributes is kaplgallant.

Tasks vs. Attributes

The next table classifies all the screen management attributes, except those specifically for reports. Some of them appear in several categories, but, in order to keep the table to a more helpful size, generally each is listed in only the most appropriate category, although there may be other categories that are not inappropriate.

Where (xX) or (xXyY) or the like appears in an attribute listing, the meaning is all attributes formed by prefixing each of the parenthesized letters to the unparenthesized letters.

Classification of Attributes

Category		Relevant Attributes
Classes	Status	bound, class
	Binding	is, reparent, verify
Attributes	Listing	has, settings, state, stateself
	Using	evaluate, freeze, recursively
Presence of features	Text	foot, head, labelinc, legendstyle
	Traces	linestyle, style, symbol
	Axes and grids	axis, grid, gridstyle, rule, xaxis, (xXyY)minorticks, (xXyY)tickstyle, yaxis, ymode, zero, zerostyle
	Other lines	backpages, box, collabelrows, colsep, line, minortickcount, rowsep, showbinding
	Printing	breakcriteriafunc, breakon, breakprocessfunc, breakprocesson, computepagebreakcb, currentbreakcolumn, framestyle, pagebreakcb, pagebreakon, pagecontrol, pagenumbering, pagenumbers, printdisstyle, reporttotalon, rowcontrol, suppressduplicate
Values		def, eval, footnote, icontitle, label, legend, max, min, preset, set, shelltitle, subtitle, title, titles, (xXyY)label, (xX)sublabel, (xXyY)title
	Printing	banner, breaktext, computationmode, disclaimer, disclaimerfile, footer, header, maxtitle, mintitle, pagenumbertext, printdisfile
Size	Objects	constraints, dynamic, extent, extents, fullscreen, H, locksize, naturalsize, resize, resizeable, W, xs, ys, yxs

	Non-graph features or parts	backpagethickness, bindingwidth, box, C, colspace, editspace, framethickness, h, hlthickness, hscrollsize, linewidth, majorticksize, margin, minorticksize, R, respace, scrollsize, shadowthickness, size, sliderheight, sliderwidth, space, vscrollsize, w
	Graph features	barwidth, gridwidth, legendhlthickness, legendshadowthickness, rulewidth, symbolsize, (xXyY)extent, (xXyY)inc, (xXyY)label, (xXyY)labelheight, (xXyY)labelwidth, (xXyY)majorticksize, (xXyY)minorticksize, zerowidth
	Printing	computesizecb, disclaimerrulewidth, framelinewidth, pagecounttotal, pagesize, printdisrulewidth, printsize, scalefont, scalefontcb, uniformscaling
Colors and blinking	**Setting**	backpagebg, backpagefg, bg, blink, blinkrate, boxcolor, colindexbg, color, colormap, colors, cycle, cyclemode, editbg, editfg, fg, framebg, hl, labelfg, maxtitlefg, mintitlefg, nodebg, nodefg, rowbg, rowindexbg, scrollbg, selectbg, selectednodebg, selectednodefg, selectedpagebg, selectedpagefg, sliderbg, titlefg, valuefg
	Setting, graphs only	fillcolor, gradient, gridfg, legendbg, legendfg, linecolor, subtitlefg, textfg, (xXyY)fg, (xXyY)titlefg, zerofg
	Printing	bggrayscale, breakbggrayscale, breakfggrayscale, fggrayscale, headingbggrayscale, headingfggrayscale, printmode
Font		bold, font, labelfont, maxtitlefont, mintitlefont, titlefont, underline, valuefont
	Graphs only	footnotefont, legendfont, subtitlefont, textfont, (xXyY)labelfont, (xXyY)titlefont
	Printing	breakfont, printdefaultfont, printdisfont, reportfont, reportheadingfont, reporttotalfont
Containment		ancestors, children, descendents, familytree, parent, shell
Position	**In field or object**	columnalignment, geometry, horizontalspace, justify, labeljustify, margin, maxtitlejustify, mintitlejustify, subtitlejustify, titlejustify, valuejustify, vcolspace, verticalspace, vrowspace
	In graphs	bottom, footnotejustify, left, legend, legendstyle, right, top, (xX)left, (xX)right, (xX)sublabel, (xX)sublabeljustify, (xXyY)label, (xXyY)labeljustify, (xXyY)titlejustify, (xy)legend, (yY)bottom, (yY)labeljustify, (yY)titlestyle, (yY)top

	In container	at, b, build, extent, l, lockposition, marginheight, marginwidth, orientation, position, r, structure, t, vcol, vcols, vrow, vrows, x, y, yx
	In screen	atsector, extents, x, X, y, Y, yx, YX
	In printed page	bottommargin, breakleading, breakoffset, breakstyle, columncontrol, columnpagespan, columnspacing, disclaimerbottommargin, disclaimerleftmargin, disclaimerrightmargin, disclaimertopmargin, disorientation, footeroffset, frameoffset, headeroffset, headingstyle, leading, leftmargin, margins, newspapercolumn, orientation, outputstyle, pagecount, printbottom, printdisbottommargin, printdisleftmargin, printdisorientation, printdisrightmargin, printdistopmargin, printleft, printorientation, printright, printstyle, printtop, reporttotalleading, reporttotalstyle, rightmargin, rowpagespan, topmargin
Display	**Group**	followers, followertree, leader, newshow, reparent, reshow, ws
	Toplevel	iconic, pin, ws
	Show, hide	currentpage, exit, hide, icon, lower, mapped, newshow, pin, raise, realize, show, syncshow
	Portion shown	col, cols, fields, firstcol, firstfield, firstrow, fixedfields, naturalsize, row, rows, size, (xXyY)max, (xXyY)min
	Changes	active, freeze, refresh
User interaction	**Traversal, focus, selection**	arrowdown, arrowkeys, arrowleft, arrowlist, arrowright, arrowup, col, colindex, coordinate, Coordinate, cornerindex, downto, field, focus, hscrollwith, index, leftto, mnemonics, preset, refer, referpoint, rightto, row, rowindex, select, selectcol, selectcorner, selected, selectedfield, selectfield, selectionmode, selectrow, set, setcol, setfirstcol, setfirstrow, setrow, tabfrom, tablist, tabto, upto, vscrollwith
	Sensitivity	acceptfocus, edit, protect, protected, selectable, selectdistance, sensitive
	Input	blank, buffer, clear, copy, cursor, delete, done, execute, exit, f1-f12, fill, fkeys, in, inc, insertabove, insertbelow, key, keysym, na, notify, pageinc, preset, primary, rband, rbandbox, request, save, set, stars, 3down, 3up, 2down, 2up, valid, validate
	Input, graphs	addtexttrace, addtrace, copytexttrace, copytrace, mode, movelimit, textactivate
	Size and position	columnresize, dragdrop

	Zooming	(xXyY)extent, (xXyY)max, (xXyY)min
	Output	blank, clear, done, fill, literal, na, notify, out, primary, request, stars, (xX)sublabelout, (xXyY)labelout
	Printing	cancel, print, report
Data conversion		formatbreakfunc, in, labelout, out, (xX)sublabelout, (xXyY)labelout
Saving and restoring; output files		file, outputmode, printfile, printoutputmode, script, state, stateself
Documenting		doc

4. Functional Attributes

Certain attributes are *functional*: the attribute can be assigned a function instead of a value, and the function will be called any time a value for that attribute is needed. Even though the values of nonfunctional attributes can be reset at any time, functional attributes are more dynamic, because the current state of the application can be tested and attributes set accordingly at the precise time the setting is needed. For example, whenever the value in a cell changes the foreground color of the cell can be set to red if the value is negative and black otherwise (the function producing the attribute value is called *after* the variable associated with the cell is modified).

An attribute function is one kind of callback function. Its syntax is the same as that of an ordinary variable callback function, and so are the meanings of the arguments (see "Set Callback"). That is, an attribute function can have from zero to six arguments, and the meanings of the arguments, no matter how many are present, are (from left to right in the function header): static data; data value; index; path; context name as symbol; variable name as symbol. These arguments will be denoted by `s`, `d`, `i`, `p`, `c`, and `v`, respectively. Just as ordinary variable callback functions are called when a variable changes, attribute functions are called when values of the variable's attribute are needed, and the function arguments indicate which part of which object needs the value. Unlike an ordinary variable callback function, which need not produce a value, but like a preset variable callback function, an attribute function must produce a value, which must be appropriate to the attribute.

The value of a functional attribute is either the function, say `f`, or a pair `(f;)`, or a pair `(f;s)`, where `s` is an array. The form `(f;s)` specifies the function and its static data `s`. Both `f` and `(f;)` are taken to be equivalent to `(f;())`, indicating that the static data is Null.

Some of the attribute function arguments differ in form for the various display classes and attributes. "Display Classes vs. Arguments to Attribute Functions" and "Display Attributes vs. Attribute Function Arguments and Rules" give descriptions of these arguments and some comments. The information in any attribute-class pair describes the particular characteristics of an attribute function for that class. The large table in the previous chapter, "Display Attributes", tells to which classes each attribute applies.

Display Classes vs. Arguments to Attribute Functions

Display Class	**Path and Index Arguments, `p` and `i`**
action	See slot.
array	The path `p` is always Null and the index `i` is of the form `(`*`row_index`*`;`*`column_index`*`)` for a matrix, `(`*`index`*`;)` for a vector, and `(;)` for a scalar.
button	Both the path `p` and the index `i` are always Null.
check, choice	See slot.
command, graph	Both the path argument `p` and the index argument `i` are always Null.
graphTrace	The path `p` is always Null and the index `i` is always of the form `(;`*`column_index`*`)`, where *`column_index`* indicates the visible trace within the trace set being affected. The function for an attribute specific to a graphTrace is called for every visible trace where the attribute has meaning. For example, the function for the fillcolor attribute will not be called for a scatter plot. Note that because of the form of the index `i`, these attributes can be specified separately for the visible traces within a trace set.
hgrid, hmenu, hpane, label, layout	Both the path `p` and the index `i` are always Null, except that when a label has a multiline value `i` is the line index for each call to the fg functional attribute. [Future use]
matrix	The path argument `p` is always Null. If the index argument `i` refers to a matrix entry, it is always of the form `(`*`row_index`*`;`*`column_index`*`)`; if it refers to a row label, it is always of the form `(`*`row_index`*`;)`; if it refers to a column label, it is always of the form `(;`*`column_index`*`)`; if it refers to the corner label, it is always `Й0`; and, finally, if it refers to all entries it is Null.
notebook, page	Both the path `p` and the index `i` are always Null. For page, the function must return an array of the same shape as the underlying variable.
password	No functional attribute is meaningful for password.
radio	See slot.
scalar	Both the path `p` and the index `i` are always Null.
slot	The path `p` is always a symbolic index of the slotfiller variable, or Null, whether the attribute refers to the label area or value area of the slot. The index `i` is always

	Null.
table	Both the path `p` and the index `i` are always Null, except that `i` is the break number for report generation.
tableField	The path `p` is always Null. The index `i` is of the form `(row_index;)`, or just a row index or pair of them, or a report break number. For formatting report break text, `d` is the unformatted string.
text, tree, vgrid	Both the path `p` and the index `i` are always Null.
view	The path `p` is always Null and the index `i` is of the form `(row_index;)`.
vmenu, vpane, window	Both the path `p` and the index `i` are always Null.

Each row of the following table names one or more display attributes and gives arguments `{s;d;i;p;c;v}` and rules for them.

The preset and set attributes, although functional, are not included in this table because their arguments are dependent on the changes to the variables for which they are set, rather than the characteristics of display classes.

Display Attributes vs. Attribute Function Arguments and Rules

Attribute	Arguments `{s;d;i;p;c;v}` and Rules
be	Both the path `p` and the index argument `i` are Null. The data argument `d` is the name of the child to be bound, in symbol form.
bg	These attributes are functional for data cells in the array, matrix, tableField, and view classes; see the descriptions of the arguments for these classes in the previous table.
blink bold color underline	These attributes are for the page display class only, so the arguments are as described in the previous table. The function must return an array of the same shape as the underlying variable.
bggrayscale breakcriteriafunc fggrayscale scalefontcb	The index argument `i` specifies a row.
breakbggrayscale	The index argument `i` specifies a break number.

breakfggrayscale breakfont breakleading breakoffset breakstyle pagebreakcb	
breakprocessfunc	Argument *i* is the beginning and ending row indices for the calculation.
colors	The function is called once for each element in an indexed specification (perhaps caused by screen editing), so the arguments specify a cell. The form they take is described in the previous table; see the appropriate one of the nine classes for which this attribute is meaningful.
colspace	Both the path *p* and the index argument *i* are Null. The function is called whenever the colspace attribute value is established for an object and whenever *show* is invoked to display that object or an object containing it.
fillcolor gradient legend linecolor linestyle linewidth, movelimit selectable style symbol symbolsize	These attributes are for the graphTrace display class only, so the arguments are as described in the previous table. For performance reasons, fillcolor, gradient, legend, linecolor, linewidth, style, symbols, and symbolsize are not fully functional. They are invoked once during assignment and when traces are dynamically added, except that linecolor and fillcolor are fully functional when gradient is true and the style is bar.
fg	In the case of an array, matrix, tableField or view, the attribute function is called once for every visible cell, row, or slot value area in the various display classes, whenever the attribute is specified or whenever the object is bound, displayed, or redisplayed. It is also called for any visible cell, row, or value area of a bound object that changes value. When a label has a multiline value, it is called once per line and *i* is the line number. Otherwise, it is called once for the initial display or any redisplay of the object. The function should return a single color. [Future use]
font	The rules for calling this attribute function are the same as for the fg attribute. The function should return a single font.
formatbreakfunc	The data argument *d* is the unformatted string.
geometry	The path *p* and index *i* are both equal to Null. The function should return the complete geometry specification for the object.
in	Attribute functions for the in attribute parse data taken from the screen, which is always in character vector form. The argument *d* holds a formatted character vector, and the function must produce a value suitable for the object in the workspace. The arguments *i* and *p* are of the standard form for

	each class, except that they are Nulls for attributed data. If the character vector `d` represents an invalid value, return the Null from this function in order to keep the input area active for another try.
label	The function for this attribute is called once for every slot area or matrix label area when the attribute is specified or the object is bound. It is not called when the object is redisplayed or changes occur to the slot value areas or matrix value. The function should return one character vector.
labelfg	The function for this attribute is called once for every label area, whenever the attribute is specified or whenever the object is bound, displayed, or redisplayed. The function should return one color.
labelfont	See label.
out	A function for this attribute formats data as character vectors to be written on the screen. The data is in the argument `d`, and the function must produce a character vector of the correct length for the display. The arguments `i` and `p` are of the standard form for each class (see previous table), except that they are Nulls for attributed data.
protect	The function for this attribute is called whenever a cell, row, slot value area, or visible trace is about to be modified on the screen, to determine whether or not it is protected. To determine the protection on a table cell the attribute function for the table is called first, and if its value is 0, the value or function of the protect attribute for the field holding that cell is used or called.
xaxis x/Xlabel x/Xlabelout x/Xsublabel x/Xsublabelout yaxis y/Ylabel y/Ylabelout	These attributes are for the graph display class only, so the arguments are as described in the previous table. For performance reasons, xaxis and yaxis are not fully functional. They are invoked once during assignment and when traces are dynamically added.

5. Attributes with Callbacks

An attribute associated with certain button presses or certain key presses or other events can be assigned an event callback function, which will be invoked whenever an associated button or key is pressed or event takes place. All these functions, except as noted, can have from zero to three arguments, the basic syntax being

```
ecfn{s;c;v}
```

where

- `s` is static data (see below);
- `c` is a symbol naming the context of the object in which the cursor was positioned when the key or button press took place or for which the event occurred; and

- `v` is a symbol giving the unqualified name of this object.

The arguments always have these meanings, in this order, no matter how many (or few) are used in the definition of the function.

The value of an attribute with callback is given as either the function name, say `f`, or a pair `(f;)`, or a pair `(f;s)`, where `s` is an array. The form `(f;s)` specifies the function and its static data `s`. Both `f` and `(f;)` are equivalent to `(f;())`, indicating that the static data is Null. Note that the first element of the value of the attribute is a function, not the name of a function; if you use the name `f` in specifying an attribute, change the definition of `f`, and want the callback to use the new definition, you must specify the attribute again.

The actions of some of these attributes depend on the row or cell that is currently selected; for example, the effect of the delete attribute is to delete the selected row. Whenever a function associated with one of these attributes is called because a key or button was pressed, some row or cell or trace in a displayed object must have been selected. Usually it is important to determine within the function which row or cell it is; the attributes row and col hold the row index and, when it is meaningful, the column index of the selected row or cell. The values of two other attributes are relevant here: the value of selected is the pair (row attribute, col attribute), and the value of selectedfield, which applies only to tables, is the nested array pair (row attribute; symbol holding the name of the selected field).

Some of these attributes have default behavior. For example, the default behavior of the delete attribute is to delete the selected row. These attributes can be assigned functional values to change the default behavior, or simply be assigned 0 to turn the default behavior off. If off, the default behavior can be turned back on by setting the value to 1. The default behavior can also be turned off with `` `false `` or `` `off `` in place of 0, or turned back on with `` `true `` or `` `on `` in place of 1.

When an attribute attr has a default behavior for the class to which an object `obj` is bound, that behavior can be invoked by ``s.call{`obj;`attr}``. Thus, a callback function can be assigned to an attribute of an object that will, when called, make a decision whether or not to invoke the default action, to do some other work, or to do both or neither. `s.call` *should* work for all attributes with default callback behaviors; it *will* work for all attributes listed in `0Шs.EVENT_CALLBACKS`.

Attributes with Callbacks (event callbacks - up to three arguments -, except as noted)

Attribute	**Description**	**Default**
addtexttrace addtrace copytexttrace copytrace refer referpoint textactivate	See the table "Attributes for Interactions with Graphs". See also "refer" in this table.	

clear	If an object was the last one in its A+ process to set the primary selection buffer, and that buffer is subsequently set by another process, a clear event occurs for that object. See "Accessing the Primary Selection Buffer".	
deiconized iconized	These event callbacks occur when the widget is deiconized or iconized.	
delete	A delete event occurs when **Meta-Delete** or, on an IBM keyboard, **Alt-Delete** is pressed. The default action is to delete the selected row or cell. If the attribute is 1, the default action is taken; if 0, it is not. If the last row is deleted, changing the row attribute, a select event (below) is also triggered.	0
done	A done callback function is called at the end of a screen entry and refresh cycle, which consists of some or all of the following actions, in the following order: in, preset callback, set variable, ordinary callback, out, done. **Note**: this is a ***variable*** callback function, taking up to six arguments, `{`*`s`*`;` *`data`*`;` *`index`*`;` *`path`*`;` *`c`*`;` *`v`*`}`.	
editbegincb editendcb	These events occur when a user begins or stops editing the widget, i.e., when the widget editor is invoked or unmapped, respectively. For array, matrix, slot, and table, the cell involved is given by selected.	Null (none)
exit	An exit event occurs when the right button is pressed while the mouse pointer is in the shelltitle area of a window, and **Quit**, or **Dismiss**, or **Close** is selected from the menu. The default action is to remove a top-level object from the screen. If the attribute is 1, it is taken; if 0, it is not.	1
f1 through f12	There is no default action for function key presses. f1 controls the action when **F1** is pressed, f2 when **F2**, and so on.	
increment decrement	These events occur when, respectively, an up or down arrow button (shown when arrowbuttons is 1) is pressed. A slot may have several such buttons; the selected attribute gives the symbol for the entry whose button was pressed.	
incurrent workspace outofcurrent workspace	In a CDE window, these callbacks occur when the window: • receives presence in the current workspace (i.e., when the presence of a window is changed to include the current workspace or when the current workspace is changed to a workspace which includes this window	

	from a workspace which did not include it) or • loses presence in the current workspace (i.e., when the presence of a window is changed to exclude the current workspace or when the current workspace is changed to a workspace which does not include this window from a workspace which included it). See the caveats for deiconized. Ignored outside CDE.	
insertabove	An insertabove event occurs when **Shift-Meta-Insert** or, on an IBM keyboard, **Shift-Alt-Insert** is pressed. If 1, the default action is to insert a new row or cell above the currently selected one. If 0, the default is not taken.	0
insertbelow	An insertbelow event occurs when **Meta-Insert** or, on an IBM keyboard, **Alt-Insert** is pressed. If 1, the default action is to insert a new row or cell below the currently selected one. If 0, the default is not taken.	0
is	There is no default action associated with this attribute. The function is called when an object is bound to a display class.	
key	There is no default action. The function is called when a key press occurs in the value area during editing.	
notify	The request attribute is set in order to reference the last entry written to the primary buffer. A notify event occurs when that entry is available for referencing. See "Accessing the Primary Selection Buffer".	
pagechangecb	When a user clicks on the tab of a notebook page which is not the current page, thereby making it the current page, this callback is fired. The new page name is, of course, available from the currentpage attribute.	Null (none)
rband	If the mouse pointer is on a page object and the left mouse button is pressed and held, a rubberband box is created. The box is anchored in the upper left corner of the character on which the pointer rested when the mouse button was pressed, and the diagonally opposite corner of the box moves with the pointer. The rubberband box as drawn disappears and an rband event occurs when the mouse button is released. If rband is 0, the box attribute is unchanged and the page object appears unchanged. If rband is 1, the location and extent of the box that triggered the event is appended to the box attribute and the box is shown in the page object in the color indicated in boxcolor.	0
refer	In an array, matrix, table, or view object, a refer event occurs when the pointer is in a selected row or cell that has focus and	0 for array, matrix, table

	the left mouse button or the **Enter** key is pressed. Hence it occurs as the result of a "double click" with the left mouse button in a row or cell. There is a default action for array, matrix, and table only, taken if refer is 1: highlight the referred row in the indexbg color and append the new value of the row attribute to the value of the index attribute. If refer is 0, the default action is not taken. Pressing the left mouse button on a referred row removes the reference. In a password or scalar object, a refer event occurs when the mouse pointer is in either the title or value area, and the left mouse button is pressed. In a slot, the pointer can be in any label or value area. In a graph, the pointer can be anywhere on the graph; if the pointer is on a line trace, the event occurs for that trace. In all these cases, a refer event occurs only if the area has focus; there is no default action.	and view; Null otherwise.
save	A save event occurs when **Ctrl-s** is pressed while an object bound to the text display class has focus. If save is 1, the edited text on the screen is saved in the workspace variable. If it is 0, this default action is not taken.	1
select	There is no default action for mouse button presses or **arrow** key presses. This attribute can be set for an object of class array, matrix, table, or view, and the function to which it is set will be called whenever the mouse pointer is on a row or cell of the object, that row or cell is not currently selected, and the left mouse button is pressed. The selected cell appears raised and in the color specified by the selectbg attribute, while the rest of the selected row appears in the color specified by the rowbg attribute. Once a row or cell is selected the user can move from row to row or cell to cell with the **arrow** keys. With each press of an **arrow** key the row or cell in the indicated direction becomes selected, and the function is called. Orogrammatically setting a row or column to -1 causes a select callback.	
selectcol	If 1, the default action is to highlight the selected column labels in the rowindexbg color and update the value of the colindex attribute with the row index of each selected column label. If 0, the default action is not taken. This attribute can be set for a matrix, and the function to which it is set will be called whenever the mouse pointer is on the label area of a column and any mouse button is pressed. If there is a selected row at the time a column label is selected, it remains unchanged, but the selected cell becomes the one at the intersection of the selected row and the column beneath the newly selected column label. In particular, if a new cell is selected, a selected event occurs. Press any mouse button on a referred column to remove the reference.	0
selectcorner	If 1, the default action is to highlight the selected corner label	0

	in the cornerindexbg color and change the value of the cornerindex attribute to 1. If 0, the default action is not taken. This attribute can be set for a matrix, and the function to which it is set will be called whenever the mouse pointer is on the label area of the upper left corner and any mouse button is pressed. Press any mouse button on the referred corner to remove the reference.	
selectfield	This attribute can be set for a table, and the function to which it is set will be called whenever the mouse pointer is on the title area of a field and any mouse button is pressed. If there is a selected row at the time a field title is selected, it remains unchanged, but the selected cell becomes the one at the intersection of the selected row and the field beneath the newly selected field title. In particular, if a new cell is selected, a selected event occurs.	
selectrow	If 1, the default action is to highlight the selected row labels in the rowindexbg color and update the value of the rowindex attribute with the column index of each selected row label. If 0, the default action is not taken. This attribute can be set for a matrix, and the function to which it is set will be called whenever the mouse pointer is on the label area of a row and any mouse button is pressed. The row with the selected label becomes the selected row. If no row was selected at the time the row label is selected, the selected cell is the one in the first column of the selected row. If there was a selected cell at the time the row label was selected, the newly selected cell is the one in the same column, but in the newly selected row. In particular, if a new cell is selected, a selected event occurs. Press any mouse button on a referred row to remove the reference.	0
3down	Pressing the right mouse button causes a 3down event.	
3up	A 3up event occurs when the right mouse button is released.	
2down	Pressing the middle mouse button causes a 2down event.	
2up	A 2up event occurs when the middle mouse button is released.	
validate	There is no default action associated with this attribute. The function is called when the keyboard entry for a password object is completed (by an **Enter**).	

Accessing the Primary Selection Buffer

When text is clipped in an Emacs or XTerm window, say by dragging the pointer over the area while the left button is depressed, the text is written in the *primary selection buffer*, and when text is pasted into an Emacs or XTerm window, it is retrieved from the primary selection buffer. A+ provides four attributes for using this buffer: primary, request, notify, and clear.

To write in the primary selection buffer, give the primary attribute a character value. (Attempted settings with other types of data are quietly ignored; the value of primary remains unchanged.) Since there is only the one buffer, the value of the attribute changes for all objects.

To read the buffer, specify the request attribute for a bound object. If and when primary has been given a proper value, a notify event occurs and the buffer can be read by referencing the primary attribute.

If the buffer is overwritten by another process (not by an object in this A+ process), the A+ object, if any, whose specification of the primary attribute was overwritten will undergo a clear event. Dragging the pointer over text in an Emacs, Xterm, or FrameMaker window, for example, will cause a clear event. The text that is highlighted can then be retrieved through primary, but text that Emacs demarcated by bouncing the cursor back and forth cannot - but can, however, after you press the "0 to PRIMARY" button in an xcutsel window.

See the tutorial scripts, which are available under the "Tutorials" item on the "On-Line Documents" section of the home page. The entry "s.tutorials/data" gives an example, for the view class. It is an ASCII text file; use Emacs to access it.

6. An Overview of the Display Classes

Display Class	**Description**
Action	A set of buttons that allow certain actions to be invoked.
Array	A simple display of a scalar, vector, or matrix, with scrollbars as needed.
Button	A single button, with which a function is associated, being called when the button is pressed.
Check	Displays a slotfiller as a set of buttons, some on, some off, which can be toggled.
Choice	A pulldown menu to allow selection of one from a set of alternatives.
Command	For entry of a command; the keystrokes can be monitored as they are entered.
Graph, ...Trace	Graphs of many kinds; line, scatter, barchart, piechart, and so on.
Hguage	Shows a scalar in graphical and (optionally) digital form.
Hgrid	A container: a simple horizontal arrangement of child objects.
Hmenu	Represents a slotfiller as a horizontal cascade menu; submenus are vertical.
Hpane	A container: a layout of child objects in which a viewer can move vertical dividers between children.
Hscale	Shows a scalar in graphical and (optionally) digital form and allows the user to alter it.
Label	A simple widget to show text; it never has scrollbars.
Layout	A container: an arrangement of child objects, with many options.
Matrix	Displays a matrix (that is not a simple character matrix), with row, column, and corner labels.

Notebook	A container looking like a notebook; its children have tabs for selecting the one to be shown.
Page	Character matrices for data services info; no editing, scrolling, but individual character color, keystrokes, boxes.
Password	For password entry and authentication (includes Kerberos support).
Radio	For slotfillers; displays buttons to choose 1 of n; automatically invalidates previous choice.
Report	Prints all of object or objects (continuation pages as needed); breaks, subtotals, much else.
Scalar	For any A+ global variable; shows it as an A+ expression whose value is that of the variable.
Slot	For slotfillers; displays slot areas: label (symbolic index by default), value (editable); no scrolling.
Table, tableField	Table (symbol vector) contains labelled columns (fields) displaying (vector or character matrix) tableFields.
Text	For character vectors; general means for text entry; can be used as note pad.
Tree	Shows nested slotfillers as trees.
Vgauge	Shows a scalar in graphical and (optionally) digital form.
Vgrid	A container: a simple vertical arrangement of child objects.
View	For simple character matrices larger than labels; can be scrolled.
Vmenu	Represents a slotfiller as a vertical cascade menu.
Vpane	A container: a layout of child objects in which a viewer can move horizontal dividers between children.
Vscale	Shows a scalar in graphical and (optionally) digital form and allows the user to alter it.
Window	For putting scrollbars on a single object.

7. The Action Display Class

The action display class is for the display of any slotfiller variables. The display consists of a title area above a collection of *label areas*, one for each symbolic index. When the slotfiller has a callback function and the left button is clicked with the mouse cursor on a label, the callback function is called with the path argument *p* equal to the symbolic index of the pressed label, and the new-value argument *d* equal to the slotfiller itself. The value of the slotfiller does not actually change; the callback occurs because an assignment of the form:

```
(p⊃d)←p⊃d
```

is automatically generated when an action item is selected.

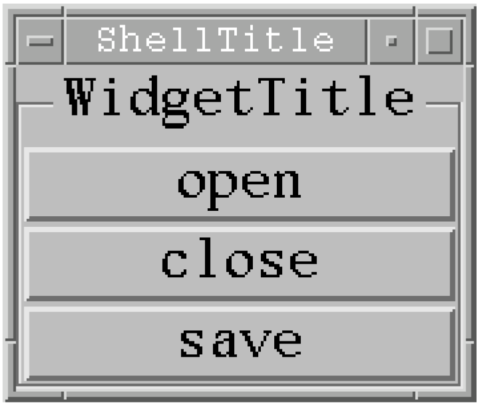
The slotfiller values do not affect the display of an action object, but can be used by the programmer to identify the actions to be taken when the labels are pressed.

The label areas in action objects are treated in exactly the same way as label areas in objects bound to the slot display class. The label areas cannot be edited. There are no value areas.

The label areas that appear in the display and their arrangement can be controlled by the setting of the **geometry** attribute; see "The Geometry Attribute". All label areas specified for display by this attribute are completely displayed; action displays never have scrollbars.

Visual Representation

```
ac←(`open `close `save;(;;))
`ac has (`class;      `action;
         `shelltitle;"ShellTitle";
         `title;      "WidgetTitle")
show `ac
```



For information about a widget element, click the left mouse button with tip of the pointer index finger on it. Point inside the frame but outside the widget for attributes governing overall size, position, etc.

The element you are pointing at is named in the status message area at the bottom of the window.

(Clicking the middle mouse button instead displays this information in a new browser.)

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the action display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are N, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the action display class (other than the print... attributes) are:

acceptfocus, active, ancestors, arrowdown, arrowkeys, arrowleft, arrowlist, arrowright, arrowup, at, atsector, b, bg, bound, C, class, clear, deiconized, doc, downto, dynamic, eval, evaluate, exit extent, f1-f12 , fg, fkeys, focus, followers, followertree, foot, font, freeze, fullscreen, geometry, h, H, has, head, hide, hl, hlthickness, icon, iconic, iconized, icontitle, incurrentworkspace, is, justify, l, label ,labelfg ,labelfont leader leftto literal lower mapped naturalsize notify outofcurrent workspace parent pin preset primary protect protected r R raise realize refresh request resize resizeable rightto script selected sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs *(Point the mouse cursor to an attribute to display a short description at the bottom of the screen)*

8. The Array Display Class

Any A+ scalar, vector, or matrix can be displayed as an object of class array. A display of a variable in this class has two parts, a title area and a value area. The value area appears below the title area, and consists of delineated cells. The display is arranged as follows:

- a scalar or character vector is displayed in one cell;

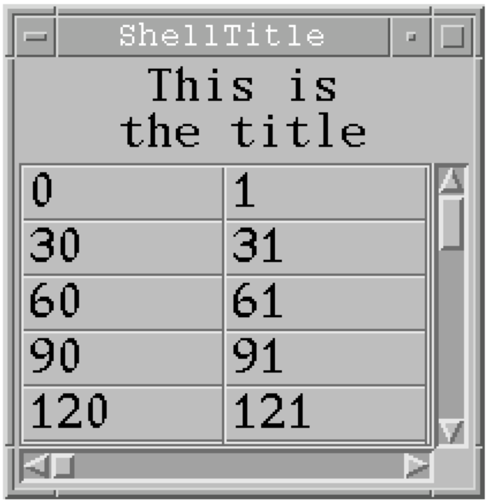
- all other vectors are displayed in a column of cells, one for each element in the vector; by default, the first element is in the top cell;
- a character matrix is also displayed in a column of cells, one for each row, and, by default, the first row is in the top cell;
- all other matrices are displayed in a two-dimensional arrangement of cells, one for each element; the rows of the matrix appear as rows in the display and, by default, element `(0;0)` of the matrix in the upper left cell.

If an array requires more than a few cells then, by default, only a subarray is presented on the screen, and the value area is provided with scrollbars. Users control the particular subarray that appears in the value area with the scrollbars; programmers set the attributes **firstrow**, **firstcol**, **rows**, and **cols**.

The contents of each cell in the value area can be edited.

Visual Representation

```
x←ι 20 30
`x has (`class;      `array;   ⍝ The default, actually.
        `shelltitle; "ShellTitle";
        `title;      ("This is";"the title"));
show `x
```



0	1
30	31
60	61
90	91
120	121

For information about a widget element, click the left mouse button with the tip of the pointer index finger on it. Point inside the frame but outside the widget for attributes governing overall size, position, etc.

The element you are pointing at is named in the status message area at the bottom of the window.

(Clicking the middle mouse button instead displays this information in a new browser.)

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the array display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are A, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the array display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg blank bound class clear col colors cols colsep copy cycle deiconized delete doc done downto dynamic edit editbg editfg editspace eval evaluate execute exit extent fg f1-f12 firstcol firstrow fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness hscrollsize hscrollwith icon iconic iconized icontitle in incurrentworkspace index insertabove insertbelow is l leader leftto literal lower mapped na naturalsize notify out outofcurrentworkspace parent pin preset primary protect protected r raise realize refer refresh request resize resizeable respace rightto row rowbg rows rowsep script scrollbg scrollsize select selectbg selected selectionmode sensitive set setcol setfirstcol setfirstrow setrow settings shadowthickness shell shelltitle show size space stars state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows vscrollsize vscrollwith w W ws x X xs y Y ys yx YX yxs *(Point the mouse cursor to an attribute to display a short description at the bottom of the screen)*

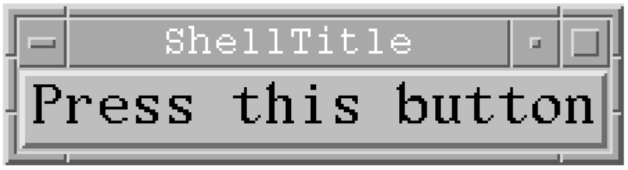
9. The Button Display Class

This is the display class for functions. The display is a labelled button, and when the button is pressed the associated function is executed. (To press a button, move the mouse cursor onto it and click the left button on the mouse.) The function must be monadic or niladic. Specifically, let *fn* be a monadic function and define *a*←(*fn*;*data*). Then the object *a* can be bound to the class button, and whenever the button is pressed, *fn*{*data*} is executed. A second form for defining a button is *a*←<{*fn*}, which is equivalent to *a*←(*fn*;) for monadic functions; when the button is pressed, the expression *fn*{()} or, for a niladic function, *fn*{} is executed. A dependency of the form *a*:<{*fn*} can also be bound to the button class; the value of this indirection is that the button will always invoke the latest definition of *fn*, since a change to *fn* will invalidate *a* and cause <{*fn*} to be evaluated again.

Note that the value of a variable bound to the button class is never a function, but rather is or includes a function array.

Visual Representation

```
fn{s}:↓s                      ⍝ fn prints its argument
b←(fn;"Pressed!")             ⍝ Pressing button prints "Pressed!" in log
show `b has (`class;          `button;
             `shelltitle;     "ShellTitle";
             `title;          "Press this button")
```



For information about a widget element, click the left mouse button with the tip of the pointer index finger on it. Point inside the frame but outside the widget for attributes governing overall size, position, etc.

The element you are pointing at is named in the status message area at the bottom of the window. (Clicking the middle mouse button instead displays this information in a new browser.)

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the button display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are U, ALL, CNFT, NFT, and TOP. The attributes that are meaningful for the button display class (other than the print... attributes) are: acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate exit extent f1-f12 fg font fkeys focus followers followertree foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is justify l leader leftto literal lower mapped margin naturalsize notify outofcurrentworkspace parent pin preset primary protect protected r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto upto title titlefg titlefont titlejustify vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

10. The Check Display Class

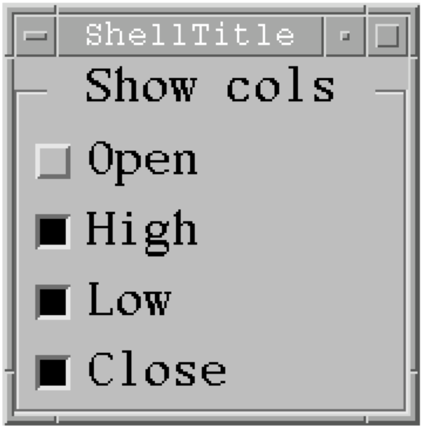
The check class is for check boxes, which are represented by slotfiller variables with boolean values. The display is like that of the slot class, except that the value areas are all square buttons and precede the labels. A button is marked "on" if the corresponding value is 1 and "off" if it is 0. When a button is "on" it has a sunken appearance and is the color specified by the **fg** attribute, and when "off" it has a raised appearance and is the color specified by the **bg** attribute. A click with the left mouse button on a check button toggles the appearance of the button between "on" and "off", and at the same time the corresponding value in the A+ variable toggles between 1 and 0. Therefore, if a callback function is defined for the slotfiller variable, mouse clicks cause the callback function to be executed, with the path argument *p* being the symbolic index of the changed value.

The label areas cannot be edited.

The slot areas that appear in the display and their arrangement can be controlled by the setting of the **geometry** attribute; see "The Geometry Attribute". All slot areas specified for display by this attribute are completely displayed; check displays never have scrollbars.

Visual Representation

```
ck←(`a`b`c`d;(0;1;1;1))
show `ck has (`class;`check;
              `shelltitle;'ShellTitle';
              `title;' Show cols ';
              `label;('Open';'High';'Low';'Close'))
```



For information about a widget element, click the left mouse button with the tip of the pointer index finger on it. Point inside the frame but outside the widget for attributes governing overall size, position, etc.

The element you are pointing at is named in the status message area at the bottom of the window.

(Clicking the middle mouse button instead displays this information in a new browser.)

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the check display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are K, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the check display class (other than the print... attributes) are:

acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound C class clear deiconized doc downto dynamic eval evaluate exit extent f1-12 fg fkeys focus followers followertree font foot freeze fullscreen geometry h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l label labelfg labelfont leader leftto literal lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary protect protected r R raise realize refresh request resize resizeable rightto script selected sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

11. The Choice Display Class

The choice class, like the radio class, is for boolean-valued slotfiller variables where one and only one of the values is 1. The value area and the button are one. The value area always holds the symbolic index of the only slotfiller value that is equal to 1. The items in the pulldown menu are called the label areas.

In effect, the choice display class is a one-item cascade menu with one submenu. The items of the submenu are the symbolic indices of the slotfiller. The menu is navigated and an item is selected in any one of the ways defined for the menu display classes; see "Selection using the

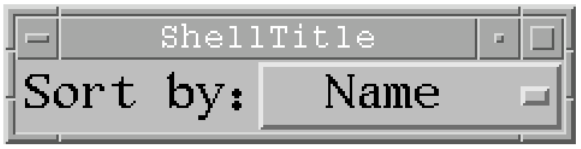
Left Mouse Button", and "Selection and Traversal using the Tab, Arrow and Page Keys". Selecting an item in the submenu amounts to selecting a symbolic index of the slotfiller. When an item is selected, if the value at that symbolic index is 0, then it becomes 1 and the value that was 1 becomes 0. In addition, the submenu disappears and the selected symbolic index - the choice of the selection - is displayed in the value area.

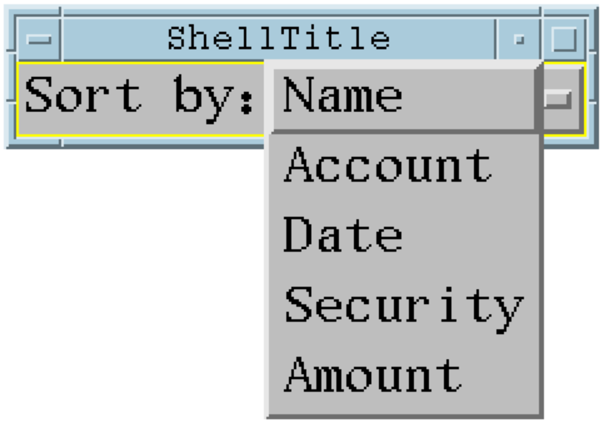
Selecting an item causes a value change in the underlying variable. Therefore, if a callback function is defined for the variable, it is called whenever a new item is selected; the path argument *p* is the symbolic index to the changed value. Note that even though there are two value changes, any callback function associated with the underlying global variable is called only once, and the path argument of the callback function refers to the value that changed from 0 to 1. If an item in the submenu that already has the value 1 is selected, no callback occurs.

The label areas of an object in the choice display class correspond to the items in the pulldown submenu. They cannot be edited. The text in the value area also cannot be edited.

Visual Representation

```
ch←(`Name`Account`Date`Security`Amount;(1;0;0;0;0))
`ch is `choice
`ch has (`shelltitle;'ShellTitle'; `title;'Sort by:')
show `ch
```





The second figure shows the appearance and position of the drop down menu after the user has pressed the left mouse button with the pointer anywhere on the choice button (not just the raised rectangle). It would look the same except that the yellow outline would not appear if **Enter** had been pressed with the pointer anywhere on the widget.

For information about a widget element, **click** the left mouse button with the tip of the pointer index finger on it. Point inside the frame but outside the widget for attributes governing overall size, position, etc.

The element you are pointing at is named in the status message area at the bottom of the window.

(Clicking the middle mouse button instead displays this information in a new browser.)

Attributes
See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the choice display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are E, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the choice display class (other than the print... attributes) are:

acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear cols deiconized doc downto dynamic eval evaluate exit extent fl-12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l label labelfg labelfont leader leftto literal lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary protect protected r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto upto title titlefg titlefont titlejustify vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

12. The Command Display Class

The command display class is for monitoring the character by character input of a command line. The command representation has two parts, a title area on the left and a value area on the right. Text can be entered in the value area, and the entry can be monitored one character at a time, even though the value in the workspace does not change until the input is complete.

Visual Representation

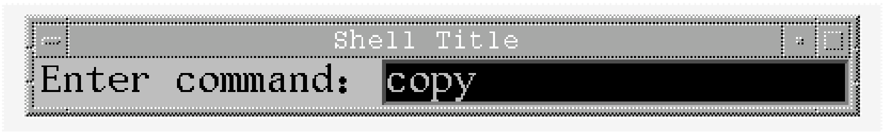
```
     f{}:{
        ↓(`buffer .of `com),
        (ρ¨`buffer .of `com),
        `cursor .of `com
        }
     com←0ρ' '
     `com has (`class;       `command;
               `shelltitle;  "Shell Title";
               `title;       "Enter command: ";
               `space;       20;  ⍝ This is the default space, actually.
        ⍝ Call the callback function f for each keystroke:
               `key;         (f;))
     show `com
```

The display in the figure shows the object `` `com `` in edit mode. The cursor is positioned just to the right of the character y. At this point the following evaluations can be made:

```
     `buffer .of `com
<  copy
     ρ¨`buffer .of `com
<   4
     `cursor .of `com
<   4
```

If the user now presses **t**, say, then *f* is called, and in *f* these evaluations are made:

```
     `buffer .of `com
< copyt
     ρ¨`buffer .of `com
<   5
     `cursor .of `com
<   5
```



```
Enter command: copy
```

For information about a widget element, click the left mouse button with the tip of the pointer index finger on it. Point inside the frame but outside the widget for attributes governing overall size, position, etc.

The element you are pointing at is named in the status message area at the bottom of the window.

(Clicking the middle mouse button instead displays this information in a new browser.)

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the command display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are C, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the command display class (other than the print... attributes) are:

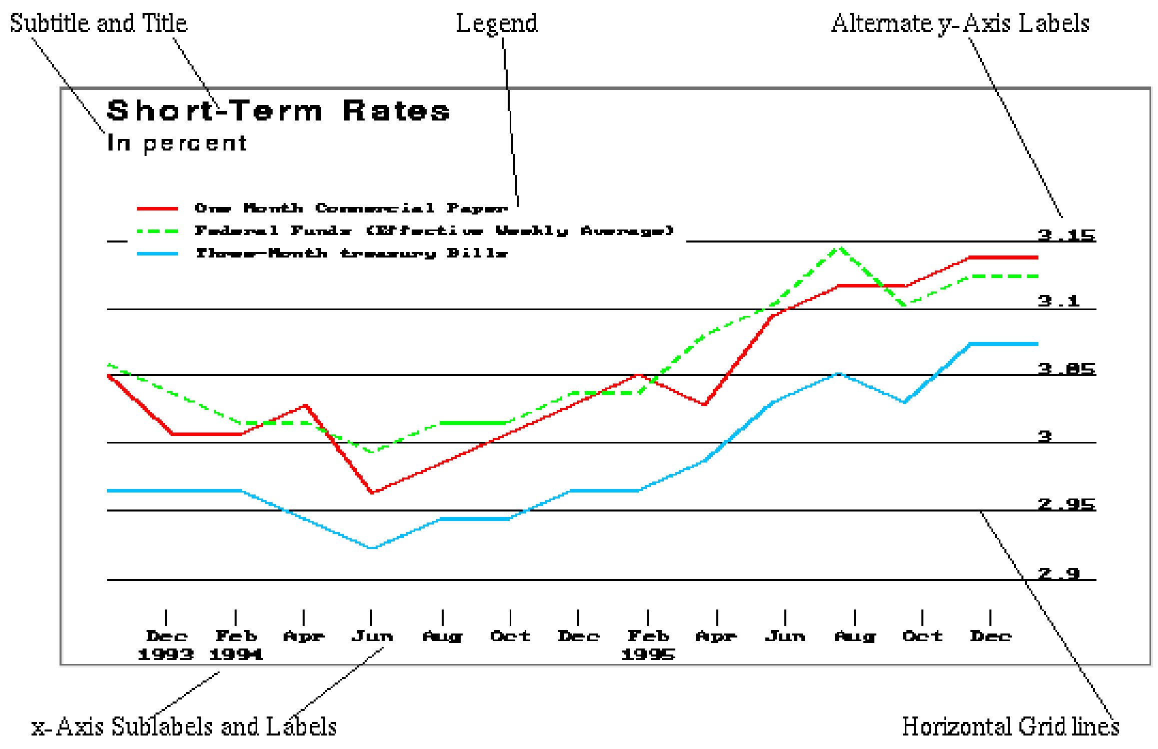
active ancestors arrowbuttons arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound buffer class clear colors cursor cycle decrement deiconized doc downto dynamic edit editbg editfg editspace eval evaluate exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle in increment incurrentworkspace is key l leader leftto literal lower mapped naturalsize notify out outofcurrentworkspace parent pin preset primary protect protected r raise realize refer refresh request resize resizeable respace rightto script sensitive set settings shadowthickness shell shelltitle show space state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

13. The Graph Display Class

The graph display class enables graphical representation of data in a variety of styles and formats, and interactive manipulation of that data. This class is a container class. A graph is represented by a global variable whose value is a scalar or vector of symbols. These symbols hold the names of other global variables, which are the children of the graph. Graph children have display class graphTrace. This class is automatically given to all children of a graph, and cannot otherwise be set.

The children of a graph are called graph trace sets. Each graph trace set can contribute one or more visible traces on a graph. The visible traces will simply be called traces. Some attributes are defined for graph trace sets, such as the colors of the traces, while others, such as the legend layout, are defined for the graph itself.

Some major features of a graph are shown in the first figure. Included are three traces, a title, subtitle, axes without rule lines, axis labels and x-axis sublabels, and a legend. The overall arrangement of a graph is fixed, but otherwise what appears in the figure can be set by means of the appropriate attributes.



There are various styles for the visible traces, including bar graph, pie chart, candlestick, and high-low, and a variety of ways users can interact with graphs, including zooming, annotating, and adjusting data points. The attributes available to a programmer for monitoring and controlling user interactions are described in this chapter, in the table "Attributes for Interactions with Graphs"; the user interactions themselves are described in the chapter "User Interactions with Displays", and in particular in "Graph Objects".

This chapter begins with a brief description of the organization of the data to be graphed and the attributes that apply directly to this data. Among these attributes is **style**, by which the appearance of the graph is specified. Descriptions of the major components of a graph and their associated attributes follow. The chapter concludes with sections on attributes associated with user interactions and miscellaneous attributes.

Trace Set Data

A graph trace set is either a numeric vector or a numeric matrix with at least two columns. The vector form, for a vector *v*, produces a single trace with x-coordinates Й#*v* and corresponding y-coordinates *v* or a pie chart with areas proportional to the elements of *v*. The matrix form produces one or more traces, with the first column containing the x-coordinates for all traces in the set. The remaining columns can represent one or more individual traces, such as lines or scatter charts, or one trace that requires more than one y-coordinate for its definition, such as a candlestick. A one-row, two-column matrix is used for a text trace, specifying the position of the lower left corner of the text. The text itself is specified as the **title** attribute of the trace set.

For performance reasons, when dates of the form yyyymmdd or mmdd appear in the data, they are considered as numbers, not parsed as dates, resulting in a distorted graph, unless the dates are all in the same month. If you convert the dates to Unix seconds, they will be properly distributed.

Trace Set Attributes

Trace set attributes are defined for trace sets, as opposed to the graphs themselves. They are nonpersistent: they disappear when a trace set is freed. They are all functional (see "Functional Attributes"). Each can be set to either a scalar value that applies to all traces, or a vector of values, one for each trace. A summary can be found in the table below. A detailed discussion of each attribute except **barwidth** follows the table.

Attributes Governing Appearance of Trace Sets in General

These attributes are all for graphTrace objects, except barwidth, which is an attribute of graph objects. See the next table for the trace styles to which the colors, widths, and linestyle apply.

Attribute (all functional)	**Summary Description**	**Data Format**	**Default Value**
barwidth	The maximum width in pixels of the bars in a bar or stack graph. (Graph attribute.)	numeric	10
fillcolor	Fill color for bar, pie, stack, area graphs; symbol color for scatter plots.	string or symbol	s.FILLCOLORS: one per trace set; if only one trace set, one per trace.
gradient	If 1, color cycling is used for some styles.	numeric	0
legend	Text in the legend box that annotates the traces.	string or symbol	decorated trace set variable name
linecolor	Line color for simple traces like scatter and line plots; component colors for compound traces like high-low and candlestick; outlines for bar, pie, etc.	string or symbol	s.LINECOLORS: one per trace set; if only one trace set, one per trace.
linestyle	The basic styles are *\`solid*, *\`dot*, *\`dash*, and *\`dotdash*.	symbol	s.LINESTYLES: used for variation in each trace set when more than one set.
linewidth	The line width in pixels for line style	numeric	1

	traces and pie outlines; for high-low style traces, the maximum display width. Not used for candlestick and various high-low graphs if it would cause individual figures to overlap.		See remark about overlap in description.
style	The visual appearance of a trace.	symbol	`` `line ``
symbol	The symbol used in scatter plots.	symbol	`` `cross ``
symbolsize	The size of the scatter plot symbol.	numeric	10
textfg	The color a text trace appears in.	symbol	same as its titlefg
textfont	The font a text trace appears in.	character string	same as its titlefont
xaxis	If `` `x ``, the trace is plotted against an x-axis on the bottom; if `` `X ``, it is plotted against an x-axis on the top.	symbol	`` `x ``
yaxis	If `` `y ``, the trace is plotted against a y-axis on the left; if `` `Y ``, it is plotted against a y-axis on the right.	symbol	`` `y ``

fillcolor

The **fillcolor** attribute specifies the color used to fill the regions of a bar, pie, stack or area trace, and the color of the symbols of a scatter plot. The default fillcolor value is the value of the linecolor attribute; thus, in the absence of a fillcolor setting the bars appear as a solid rectangle. With a fillcolor different from that of linecolor, the bars appear with a border drawn in the linecolor value. The default behavior for fill coloring is described in the table above.

gradient

If the **gradient** attribute has the value 1, then the colors of symbols in a scatter plot and bars in a bar graph are obtained by cycling through the colors listed in *s.FILLCOLORS*. If the attribute has been assigned a function, that function is called for every scatter point or bar.

legend

The **legend** attribute controls the text that annotates the trace segments in the legend. The default is the name of global variable that contains the trace set. If the variable for a trace set is a matrix, then the name is supplemented by ":n", where n is 0, 1, ... For example, if the global variable name is *x*, then the annotations are "x:0", "x:1", and so on.

linecolor

The **linecolor** attribute specifies the color of the line segments in line, line-scatter, step, step-scatter, and segment traces, and the color of the borders of area, fill, pie, and stack traces. The default behavior of linecolor for these trace styles is described in the table above.

Up to four colors are meaningful for candlesticks and open-high-low-close graphs. In the case of a candlestick graph, the first color specifies the fill color of the candle when the open exceeds the close, the second is the color of the vertical lines segments, or wicks, the third is the candle border, and the fourth is the fill color of the candle when the close exceeds the open. When there is only one linecolor for a candlestick graph, it is used to fill the candle when the open exceeds the close, and the candles are hollow when the close exceeds the open. See the table "Attributes for Text Areas of Graphs".

For open-high-low-close traces, the first color specifies the color of the open ticks, the second is the color of the vertical line segments, the third is unused, and the fourth is the color of the close ticks. The default is to use the line color for all three colors. See the table just mentioned. If fewer than four colors are given they are used cyclically, in the manner of the primitive function Reshape. There are several cases for each of the styles high-low-close, high-low, and close, due to the fact that the data for these styles can have several different formats (i.e., number of columns). See the table "High-low Family; Data and Line Colors".

linestyle

The basic linestyles are `` `solid ``, `` `dash ``, `` `dot ``, and `` `dotdash ``. In addition, `` `dot1 `` through `` `dot5 `` can be used for various intervals between dots, with `` `dot1 `` the smallest interval, `` `dot5 `` the largest, and `` `dot3 `` the same as `` `dot ``. Analogously, there are `` `dash1 `` through `` `dash5 ``, where the dash length varies and `` `dash `` is between `` `dash2 `` and `` `dash3 ``, and `` `dotdash1 `` through `` `dotdash5 ``. When there is more than one trace set, the default assigns the basic styles in the order given above to the individual traces in each trace set. If there are more than four traces in a trace set, these styles are repeated as needed. When there is only one trace set, all the traces are `` `solid ``. This attribute applies to line, linescatter, and pie traces, as well as to trace sets with styles open-high-low-close, high-low-close, high-low, and close.

linewidth

The **linewidth** attribute specifies (in pixels) the width of the line segments in line and linescatter traces, and the high-low style family of traces, and the outlines in pie charts. Line width is limited to 35, even if a higher value is set. Note: a significant performance penalty occurs on Sun Sparc workstations for line and linescatter traces with linewidths greater than 1 and more than 100 data points, due to system limitations. However, the various high-low style graphs look better with line segments wider than the default. See the figure "Various Line and Scatter Trace Styles", where all the graphs have line width 3.

Note: the linewidth setting for candlestick and the various high-low graphs is not used if it would cause the individual figures to overlap.

symbol

The **symbol** attribute is used to specify the symbol used in a scatter plot. The values for the various symbols are:

```
`circle,    `circlefilled,    `cross,
`diamond,   `diamondfilled,
`square,    `squarefilled,    `star,
`triangle,  `trianglefilled,  `xsym.
```

If the value is any other symbol or a character vector, the text of that value is used for the symbol; the font in which a text symbol appears is specified by the font attribute of the trace set variable.

symbolsize

The size, in pixels, of the symbol in a scatter plot.

style

The style of the traces in a trace set, specified as one of the symbols in the table "Trace Styles, Data Format, and Attribute Applicability". See "Trace Styles".

xaxis, yaxis

The **xaxis** attribute specifies whether a trace is plotted against the default x-axis at the bottom of the plot area or the alternate x-axis at the top; analogously, **yaxis** specifies whether the default y-axis on the left is used, or the alternate y-axis on the right. The alternate axes do not normally appear unless at least one trace is associated with them; see, however, the **axis** attribute.

Trace Styles

The graph display class supports a variety of trace styles that are summarized in the following table. Included in this table is a cross reference indicating the applicability of the trace attributes discussed above. In addition to the table summary, a discussion and sample graphs of each style follow.

Trace Styles, Data Format, and Attribute Applicability

The attributes pertain to graphTrace objects. An x indicates that the column's attribute is meaningful for the row's trace style.

Trace Style	**Value of style Attribute**	**Data Format**	**fill-color**	**line-color**	**line-style**	**line-width**
Area	`` `area ``	vector, matrix	x	x	x	x
bar	`` `bar ``	vector, matrix	x	x		
candlestick	`` `candle ``	n x 5 matrix		x more	x	x
close	`` `c `` or `` `close ``	vector, n x 2 - n x 5 matrix		x more	x	x
color profile	`` `colorprofile ``	vector, n x 2 - n x 5 matrix	x	x more	x	x
fill	`` `fill ``	vector, matrix	x	x	x	x
high-low	`` `hl ``	vector, n x 3 - n x 5 matrix		x more	x	x
high-low-close	`` `hlc ``	vector, n x 4 - n x 5 matrix		x more	x	x
line	`` `line ``	vector, matrix		x	x	x

linescatter	`` `linescatter ``	vector, matrix	x	x	x	x
market profile	`` `marketprofile ``	vector, n x 2 - n x 5 matrix		x more	x	x
none	`` ` `` or `` `none ``	any valid trace				
open-high- low-close	`` `ohlc ``	n x 5 matrix		x more	x	x
outline	`` `outline ``	vector, matrix	x	x	x	x
pie chart	`` `pie ``	vector	x	x	x	x
scatter	`` `scatter ``	vector, matrix	x	x		
segment	`` `segment ``	vector, matrix		x	x	x
stack	`` `stack ``	vector, matrix	x	x	x	x
step	`` `step ``	vector, matrix		x	x	x
stepscatter	`` `stepscatter ``	vector, matrix	x	x	x	x
text	`` `text ``	1 x 2 matrix				

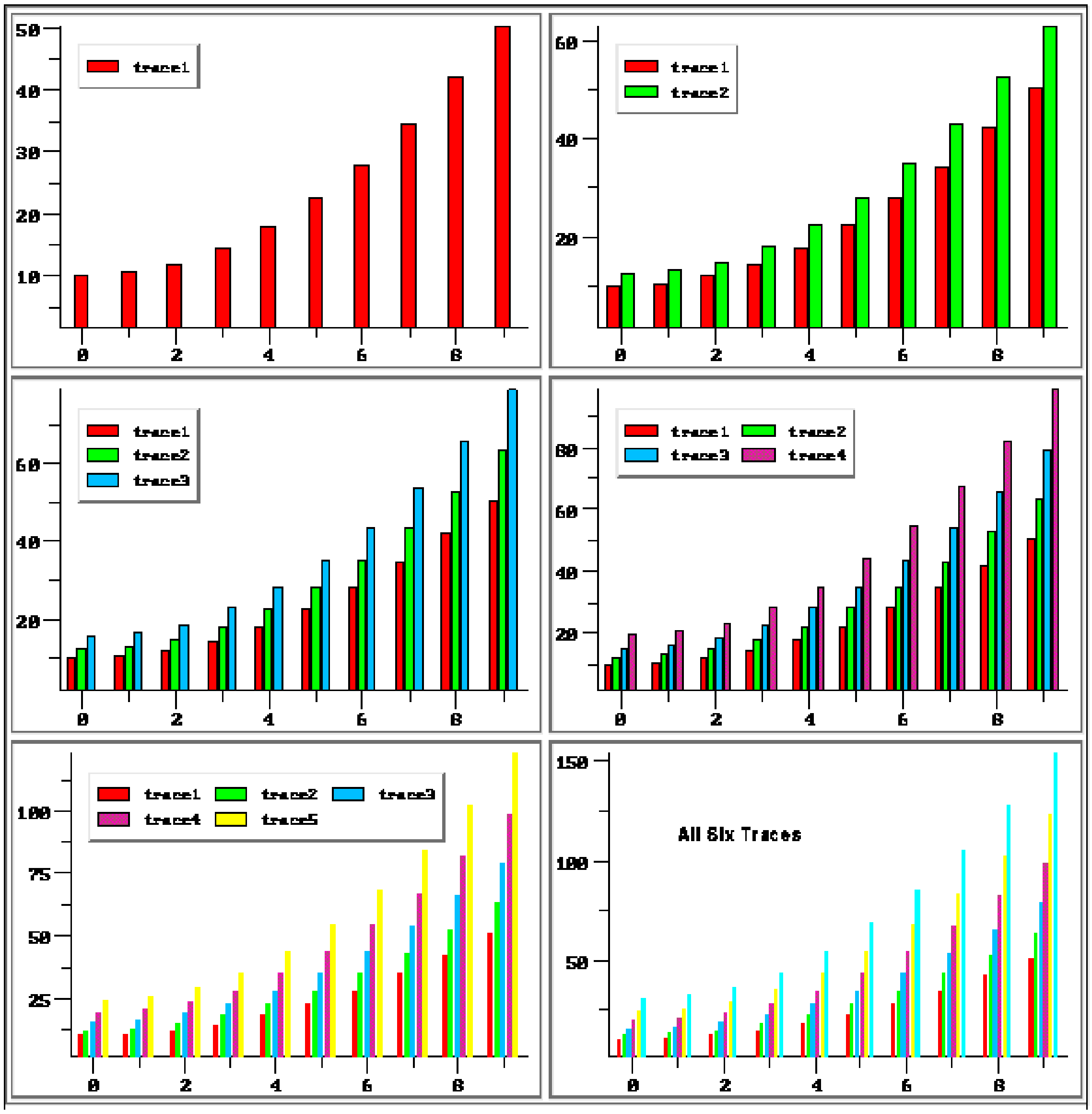
bar

The **bar** trace style displays trace data as a bar graph. The barwidth attribute (see the table "Miscellaneous Attributes for Graphs") specifies the maximum width for each bar. If a space limitation exists, the actual bar width is adjusted to fit within the available space. When there is a space limitation and the adjusted bar width is less than a threshold value, the border is dropped from the bar display and only the fill color appears.

The **fillcolor** and **linecolor** attributes are used to set the colors inside the bars, and the bar border colors, respectively. The linestyle and linewidth attributes have no effect on bar graphs.

The second figure shows a series of graphs with an increasing number of bar style traces in each successive graph. These graphs illustrate both the positioning of the bars on the x-axis relative to the x-value, and the effects of space limitation on the bar width. With each additional bar, the bar width is adjusted to fit within the available space while maintaining the same relative spacing. Notice that the borders around the bars eventually disappear.

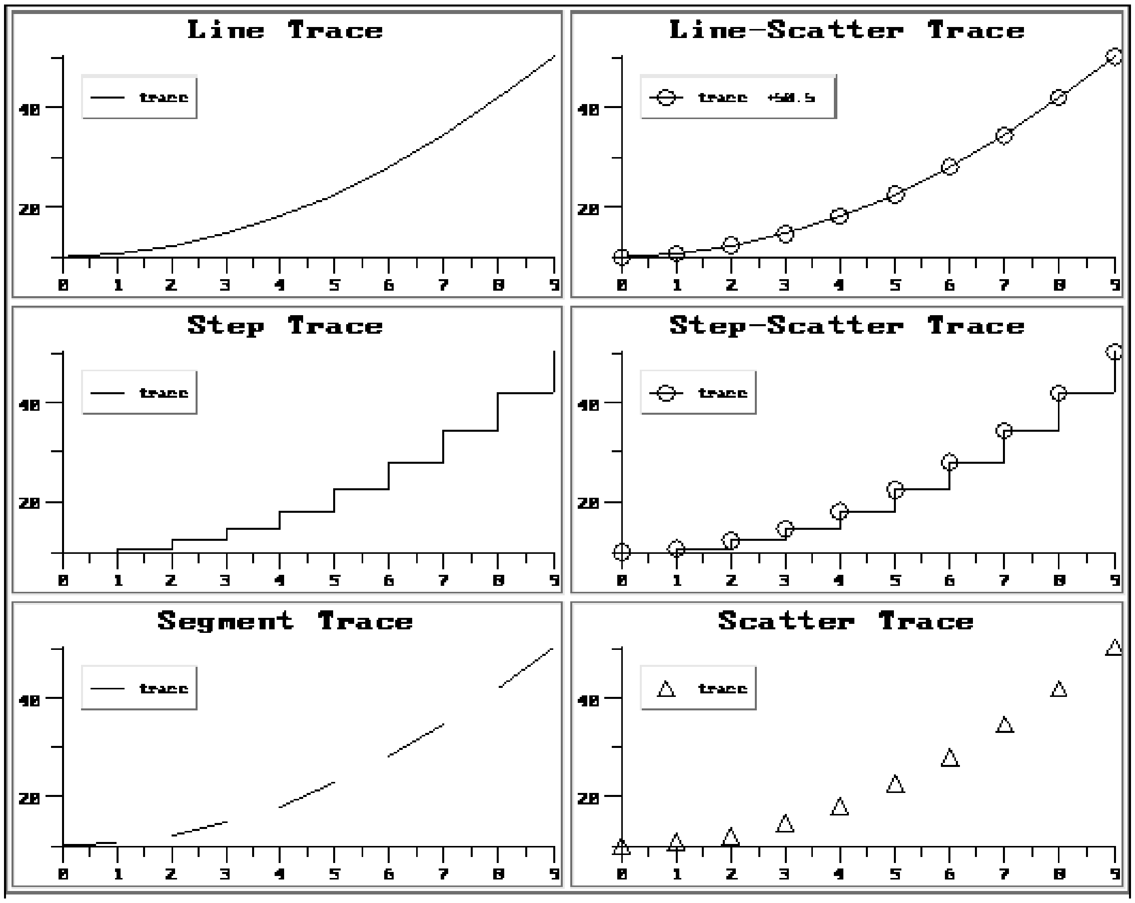
A Series of One to Six Bar Traces



line

The **line** trace style is the default trace style and consists of line segments connecting consecutive data points. See the figure immediately below.

Various Line and Scatter Trace Styles



scatter

The **scatter** trace style uses a symbol (the default is a cross) to indicate the location of each data point in a trace. To produce scatter plots with small dots, set the trace symbol to *`circlefilled* and the *`symbolsize* to 3 (a smaller size will not be shown). See "fillcolor" for a discussion of fill colors and this trace style. See the previous figure.

linescatter

The **linescatter** trace style is a combination of the **line** and **scatter** style traces; the trace is an ordinary line graph, but in addition scatter symbols mark each data point. See the previous figure.

step and stepscatter

The **step** trace style is a variation of the line style that replaces each line segment with a step consisting of a horizontal and a vertical segment. The **stepscatter** trace style is the analogous variation of the **linescatter** style. See the previous figure.

segment

The **segment** trace style is a variation of the line style that permits a series of disconnected line segments to be specified in one trace. The line segments are every other one of those in a line graph. See the previous figure.

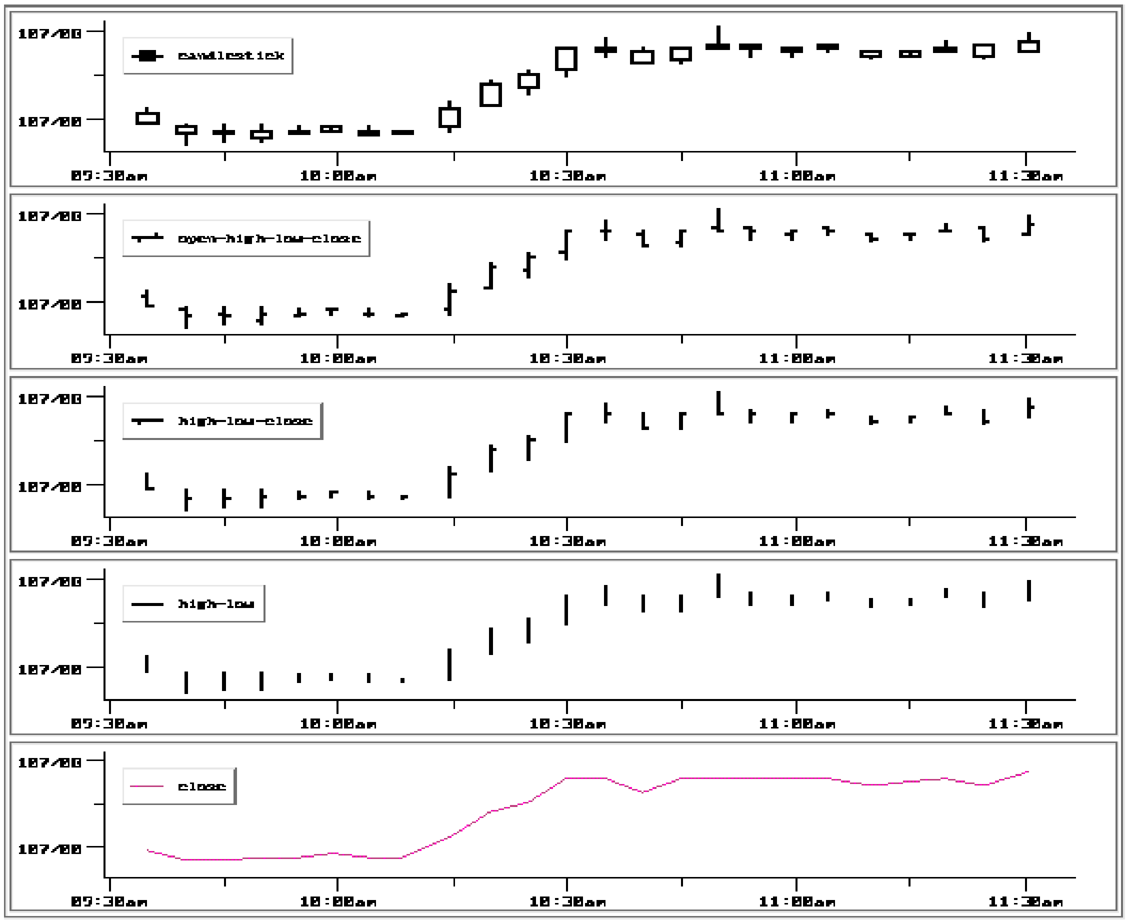
candle, ohlc, hlc, hl, c, marketprofile, colorprofile

The high-low family of trace styles can accept data in a variety of formats, which are summarized in the table "High-low Family; Data and Line Colors". With these flexible trace formats, it is possible to display any of the following graph styles without changing the underlying data: open-high-low-close, candlestick, high-low-close, high-low, close, market profile, and color profile. All these styles can be assigned to five-column trace sets.

Analogously, the following subset can be assigned to four-column trace sets: high-low-close, high-low, and close; to three-column trace sets: high-low and close; to two-column and vector trace sets: close, market profile, and color profile.

For these traces the linewidth attribute specifies the maximum width of line segments, if sufficient space is available. See the following figure.

Various Trace Styles Based on the Same Data



High-low Family; Data and Line Colors

Keys: x: x coordinate; u: unused; v: vertical line segment color; c: close tick color; t: trace color.

Trace Style	**Trace Set Data**	**Column Index**					**linecolor Index**			
		0	**1**	**2**	**3**	**4**	**0**	**1**	**2**	**3**
ohlc, candle	n x 5 matrix	x	open	high	low	close	See "linecolor"			
hlc	n x 5 matrix	x	open	high	low	close	u	v	u	c
	n x 4 matrix	x	high	low	close		v	u	c	
hl	n x 5 matrix	x	open	high	low	close	u	v	u	u
	n x 4 matrix	x	high	low	close		v	u	u	
	n x 3 matrix	x	high	low			v	u		
color profile, close, market profile	n x 5 matrix	x	open	high	low	close	u	u	u	t
	n x 4 matrix	x	high	low	close		u	u	t	
	n x 3 matrix	x	high	low			u	t		
	n x 2 matrix	x	close				t			
	vector	close					t			

text

Text traces are positioned on a graph by specifying the coordinates of the lower left corner of the bounding box of the text. The trace set for a text trace is a 1 by 2 matrix holding these coordinates as an (x, y) pair. The text that appears on the graph is the value of the **title** attribute of the trace set variable; the trace's **titlefg** attribute applies to the text unless its **textfg** attribute has been set, and likewise for **titlefont** and **textfont**. The text "All Six Traces" on the lower right graph of the bar trace figure is a text trace.

Text traces do not appear in the graph legend and are superimposed on top of the other traces.

none

A trace on a graph can be hidden from view by setting its style attribute to `` `none `` or to the empty symbol (`), and restored by resetting this attribute to any other appropriate value.

area

All traces with this style are accumulated in one visible trace. If there is only one trace with this style, its graph is the filled-in region bordered by the x axis and the line graph which this trace would form if its style were line. The lower region in the lower right graph in the figure "Various Stack and Area Graphs" is such a graph. If there is more than one trace with this style, the graph is a set of filled-in regions, stacked one above the other. See the filled-in regions in the upper right graph of the figure just mentioned, and compare that graph to the one on the lower left; they differ only in that both traces with style stack in the lower left have style area in the upper right.

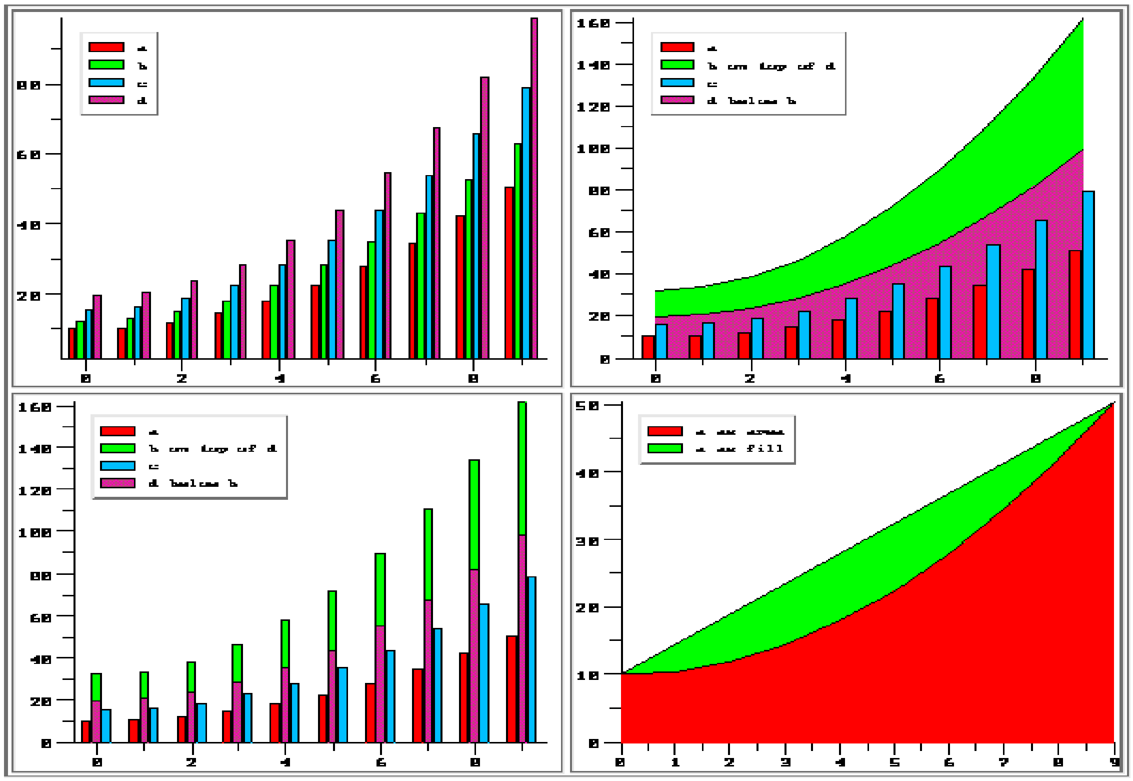
fill

If a trace of this style is drawn as a line trace and the first and last points are connected, the result is the border of a closed region. If the style is **fill** that region is filled-in with the **fillcolor**. See the lower right graph in the stack and area figure, where the top region has style **fill**, the bottom region has style **area**, and their trace sets are equal.

stack

All traces with this style are accumulated in one visible trace, which appears to be a set of bar graphs, stacked one above the other. In fact, those individual bar graphs can be seen by setting the style of all traces with style stack to style bar. See the lower left graph in the stack figure, and compare it to the upper left.

Various Stack and Area Graphs



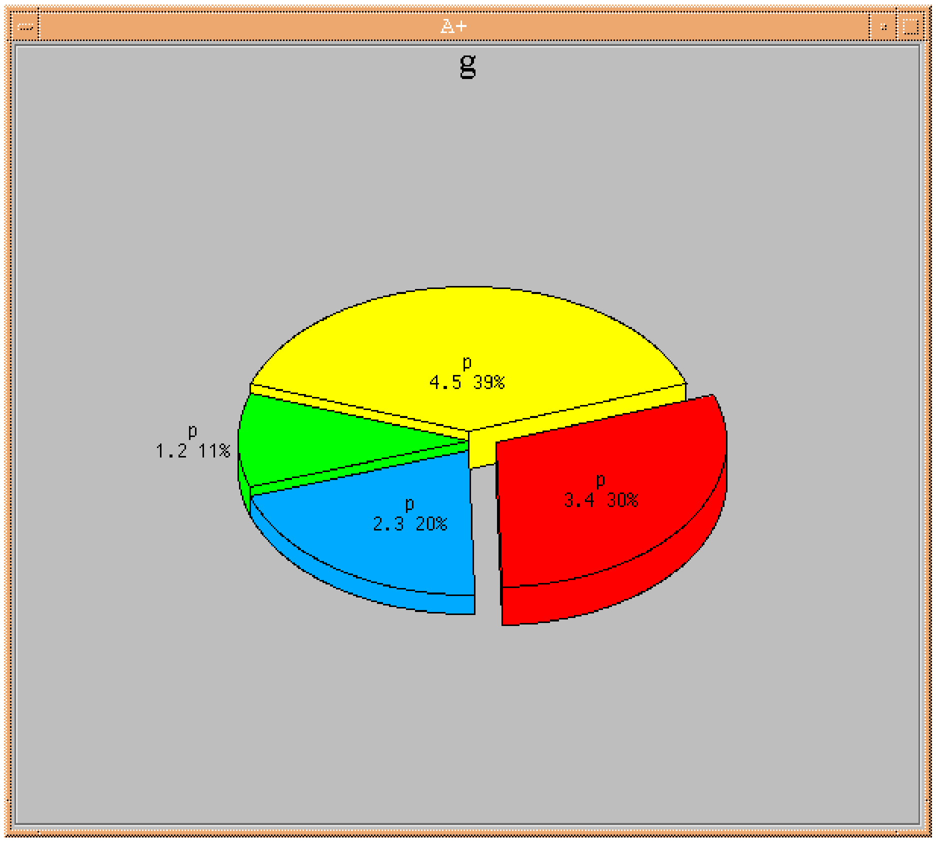
pie

Only y values are significant for **pie** charts, which are not graphed against a coordinate. Hence there are no x values, and only a vector is used for a `` `pie `` trace. In the absence of other traces in the graph, you might want to set the `` `axis `` attribute to `` `none ``, as has been done in the pie-chart figure. The attributes specifically for pie charts are shown in the following table.

```
ры1.2 2.3 3.4 4.5                ⍝ Set points for the graph
gы`p                             ⍝ Set up graph

`g is `graph                     ⍝ Establish class
`p has (`style;`pie)             ⍝ Establish style
```

```
     `g has (`axis;`none;            ⍝ Turn off x-y axis display
             `xs;602;`ys;496;        ⍝ Set size of window (x-size and y-size,
in pixels)
             `pieoffsetmargin;0.35)   ⍝ Set minimum space between text and
edges of pie

     `p has (`pievaluealign;`bottom;
             `piepercentalign;`bottom)
     `p has (`linecolor;`black;`textfg;`black)                   ⍝ Set colors
for pie borders and text
     `p has (`pieoffsets;  0       0             0.15 0;         ⍝ Pull out one
slice
             `fillcolor;   `green `deepskyblue `red `yellow;  ⍝ Slice colors
             `pieprofiles; 0.75    0.5            1     1)            ⍝ Slice
thicknesses

     show `g
```

A Pie Chart



Other Attributes Governing Appearance of pie Trace Sets

All attributes are for graphTrace except pieoffsetmargin, which is for graph.

Attribute	**Description**	**Data Format**	**Default Value**
pieangle	If pieprimaryslicealign is `none, pieangle gives the orientation of the primary slice of a pie chart in counterclockwise degrees from the three o'clock position.	integer scalar	0
pieaspectratio	The extent to which a pie chart is tilted towards the viewer, ranging from 0.0 to 1.0. 1.0 is a top view, or flat pie.	numeric scalar	0.6
piedepthfactor	The maximum height of the pie in a pie chart, as a fraction of the standard depth factor. Cf. pieprofiles.	numeric scalar	1.0
pielegendalign	Where the legend for each slice of a pie chart is placed. Values are null, `none, `inside, `outside, `left, `right, `top, `center, and `bottom. The last five control placement with regard to values and percentages and will be ignored if not consistent with the values of pievaluealign and piepercentalign. Several values may be concatenated, as in `inside`top`left. The color and font are controlled by the trace's titlefg and titlefont attributes.	scalar or vector `sym	`center
pieoffsetmargin	This attribute is for graph, not graphTrace. The margin around pie charts, as a fraction of the available space. The value ranges between 0 (inclusive) and 1 (exclusive). 0 makes pie charts as large as possible in the graph. 0.99 makes them dots. Values close to 1 cause calculation overflows, which appear as A+ domain errors.	numeric scalar	approx. 0.3
pieoffsets	The extent to which each slice is separated (pulled out) from the center of a pie chart. Values between 0.0 and 1.0 (inclusive) are treated as fractions of the pie's radius. Values greater than 1.0 and equal to or less than 100.0 are treated as percentages of the pie's radius. The value or values given are used cyclically for the slices.	numeric scalar or vector	0
piepercentalign	Where to place the display of the percentage that each slice is of a pie. Values are null	scalar or vector	Null

	`` `none ``, `` `inside ``, `` `outside ``, `` `left ``, `` `right ``, `` `top ``, `` `center ``, and `` `bottom ``. The last five control placement with regard to values and legends and and will be ignored if not consistent with the values of pievaluealign and pielegendalign. Several values may be concatenated, as in `` `inside`top`left ``. The color and font are controlled by the trace's titlefg and titlefont attributes.	`` `sym ``	
pieprimaryslice	The trace index of the slice to be considered the "primary" slice of a pie chart, the slice whose display is controlled by pieprimaryslicealign or pieangle. The value -1 means that the largest slice will be considered the primary slice. For a brief time, it was called primaryslice.	integer scalar	-1
pieprimaryslicealign	Where to place the primary slice of a pie chart (by rotating the pie). The possible values are null, `` `none `` (see pieangle), `` `left ``, `` `right ``, `` `top ``, `` `bottom ``. For a brief time, it was called primaryslicealign.	scalar `` `sym ``	Null (`` `top ``)
pieprofiles	The height of each slice of a pie, as compared to the maximum height of the pie. Values between 0.0 and 1.0 (inclusive) are treated as fractions of the pie's maximum height. Values greater than 1.0 and equal to or less than 100.0 are treated as percentages of the pie's maximum height. The value or values given are used cyclically for the slices.	numeric scalar or vector	1.0
pievaluealign	Where to place the display of the value of each slice of a pie chart. Values are null, `` `none ``, `` `inside ``, `` `outside ``, `` `left ``, `` `right ``, `` `top ``, `` `center ``, and `` `bottom ``. The last five control placement with regard to percentages and legends and will be ignored if not consistent with the values of piepercentalign and pielegendalign. Several values may be concatenated, as in `` `inside`top`left ``. The format of the value display is controlled by the out attribute, and the color and font are controlled by the trace's titlefg and titlefont attributes.	scalar or vector `` `sym ``	Null
primaryslice	Called this for a brief time, but now called pieprimaryslice.		
primaryslicealign	Called this for a brief time, but now called pieprimaryslicealign.		

The Order in Which Traces Are Drawn

The basic order in which traces are drawn is the index order of the graph trace variables in the graph variable, and within each graph trace variable, the index order of the columns. (This order may be changed without freeing the trace variables.) The traces appear in the legend in this order, starting at the top in vertical legends. When multiple bar graphs have common x-coordinates, they are drawn from left to right in this order, clustered near that common value. Stack and area graphs are drawn from top to bottom in this order. All area and fill traces are drawn in this order before traces of any other style are drawn; all bar and stack graphs are then drawn in this order; all remaining traces other than text traces are then drawn in this order; finally, all text traces are drawn in this order. Trace search order is the same as the order in which traces are drawn.

The Text Areas of a Graph

The text areas of a graph are its title, subtitle, footnote, and axis titles. The title and subtitle areas are illustrated in the first figure. The title appears at the top of the graph. The subtitle appears below the title, and the footnote appears below the x-axis at the bottom of the graph. Any of the three can have one or more lines, their text can be justified, and their color and font can be specified.

The x-axis title appears under the x-axis labels and sublabels; the y-axis title appears at the top of the y axis. See the table "Axis Attributes for Graphs".

Attributes for Text Areas of Graphs

These are all attributes of graph objects

Attribute	**Description**	**Data Format**	**Default Value**
footnote	The text of the footnote on the graph.	string, symbol; a nested vector of strings or symbols is accepted for multiline text.	null string
footnotefg	The color of the footnote text.	string, symbol	`` `axiscolor `` (inherits the axis color until the footnotefg attribute is set to a different color)
footnotefont	The font of the footnote text.	string	lucidasans typewriter-12
footnotejustify	`` `center ``, `` `left ``, or `` `right `` justify the text.	symbol	`` `center ``
subtitle	The text of the subtitle of the graph.	string, symbol; a nested vector of strings or symbols is accepted for multiline text.	null string

subtitlefg	The color of the subtitle text.	string, symbol	`` `axiscolor `` (inherits the axis color until the subtitlefg attribute is set to a different color)
subtitlefont	The font of the subtitle text.	string	lucidasans typewriter-12
subtitlejustify	`` `center ``, `` `left ``, or `` `right `` justify the text.	symbol	`` `center ``
title	The text of the title of the graph; trace set titles are meaningful for text traces only.	string, symbol; a nested vector of strings or symbols is accepted for multiline text.	variable name
titlefg	The color of the title text. (The trace's titlefg governs the color of text traces and of each pie trace slice's legend, value, and percent text.)	string, symbol	black
titlefont	The font of the title text. (The trace's titlefont governs the font of text traces and of each pie trace slice's legend, value, and percent text.)	string	kaplgallant
titlejustify	`` `center ``, `` `left ``, or `` `right `` justify the text.	symbol	`` `center ``

The Legend of a Graph

The **legend** appears automatically on a graph. Its default arrangement is vertical, but it can be also be aligned horizontally; see the **legendstyle** attribute in the table "Legend Attributes for Graphs", and see the bar trace figure for examples. The text that annotates each trace sample is specified using the *trace set* attribute named **legend**. The position, or justification, of the legend is specified using the *graph* attribute named **legend**. This position can be within the plot area, i.e., the area within the axes, or it can also be outside the plot area. The legend can be removed by setting the graph legend attribute to `` `none ``.

Legend Attributes for Graphs

These are all attributes of graph objects

Attribute	**Description**	**Data Format**	**Default Value**

legend	Position of the legend in the plot area or outside it. Values are:
	
	`` `bc ``, `` `bl ``, `` `br ``, `` `tc ``, `` `tl ``, `` `tr ``; `` `none `` (removes the legend).
	
	Inside: `` `bottom ``, `` `center ``, `` `left ``, `` `horizontal ``, `` `inside `` (centered both ways), `` `right ``, `` `top ``, `` `vertical ``; Outside: `` `outside ``, `` `outsidehorizontal ``, `` `outsidevertical ``.
	(xlegend and ylegend, if set, *always* take precedence over legend, but setting legend clears - unsets - them.)
		symbol	vector
			`` `left ``
			`` `top ``
			`` `inside ``

legendbg	The background color of the legend.	symbol or string	gray
legendfg	The foreground color of the legend.	symbol or string	black
legendfont	The font of the text in the legend. The legend symbol size depends upon the font size.	string	lucidasans typewriter-10
legendhlthickness	The thickness of the legend highlight area, in pixels.	numeric	1
legend shadowthickness	The thickness of the legend shadow area, in pixels.	numeric	1
legendstyle	The style of the legend: vertical (`` `ver ``), horizontal (`` `hor ``), horizontal with the last trace values (`` `lastvalue ``), or not at all (`` `none ``) (see the bar trace figure). The last values are those associated with the element or row with the largest index for each trace set.	symbol	`` `ver ``
xlegend, ylegend	The x, y coordinates of the upper left corner of the legend when the legend attribute of the graph is set and has not been cleared by a later setting of legend.	numeric	0, 0

The Axes of a Graph

The axes of a graph have the most attributes of the various components. For convenience, the descriptions appear in three tables, one for axes in general ("Axis Attributes for Graphs") and the other two for tick marks and tick labels specifically ("Axis Label and Tick Mark Attributes for Graphs", and "Attributes That Assist in Specifying Tick Marks and Labels"). Many of the axis-related attributes have more than one version, corresponding to more than one axis; the convention for their names is presented next.

Default and Alternate Axes

Graphs have a default set of axes, with the x axis bordering the bottom of the plot area and the y axis bordering on the left. In addition, an alternate x axis can be drawn on top of the plot area, and an alternate y axis on the right. These alternate axes are referred to as the X axis and Y axis, as opposed to the default x axis and y axis.

These additional axes can be used purely for appearance, either supplementing or replacing the default axes, or they can be used to plot traces with different scales on the same graph (see the trace attributes xaxis and yaxis).

These axes have independent attributes, such as color, and the names of the attributes are prefixed with "x" to indicate the default x axis, "X" the alternate x axis, i.e., the X axis, "y", and "Y". For example, in the case of foreground color, the four attributes are **xfg**, **Xfg**, **yfg**, and **Yfg**. To save space in the tables, the four attributes may appear in shorthand notation, like x/X/y/Yfg for the four color attributes.

The axis, grid, and rule Attributes

These attributes determine the appearance of the axes and grid lines.

The standard axis style is specified by the value ``` `std``` for the axis attribute; see the bar trace figure. The value ``` `std``` for the axis attribute has the property that when any of the xaxis or yaxis attributes of any trace is set to an alternate axis, that axis automatically appears. The boxed style, specified by the value ``` `box```, additionally has a box around the plot area. Axes are removed by setting the axis attribute to ``` `none```. Any combination of default and alternate axes can be used by setting the axis attribute to a symbol value containing the appropriate letters `x`, `X`, `y`, and `Y` (in any order).

The rules on the axes, which are the straight lines that run lengthwise, can be specified separately. For example, the boxed style can be obtained by setting the axis attribute to ``` `xy``` and the rule attribute to ``` `xXyY```. When the rule attribute is set to ``` `axis```, its value is effectively the same as that of the axis attribute.

In addition, grid lines can be drawn at the major tick marks on axes by setting the grid attribute.

The xlabel, Xlabel, ylabel, and Ylabel Attributes

These attributes provide complete control over the placement of tick marks, the text of their labels, their lengths, and the widths of their associated grid lines. A complete specification for any of them is a nested vector of length four, of the form `(ticks;labels;sizes;widths)`, where:

- The i-th element of `ticks` specifies the location of the i-th tick mark.
- `labels` is either a nested vector of character vectors or a character matrix whose i-th element or row is the label on the i-th tick mark. (A simple scalar or vector is treated as a matrix with one row.)

  Warning! A `` `null `` or `` `symbol `` item is handled as if it were character and is interpreted as indicating that the default labels, the tick values, should be used! Therefore (see the warning following these bullets) a Null or symbol item can override a character item!

  Warning! If fewer labels than ticks are specified, the last of the specified labels is used for them!

- The i-th element of `sizes` is a number between 0 and 1 which, when multiplied by the value of the majorticksize attribute, gives the length of the i-th tick mark. A scalar is treated as a vector with one element.
- The i-th element of `widths` is an integer specifying the width, in pixels, of the gridline that intersects the axis at the i-th tick mark. A scalar is treated as a vector with one element.

  Warning! A 0 in `widths` produces a 1-pixel width grid line (although in printing the line may be narrower than one specified by a 1)!

Warning! The first component, `ticks`, is positional, as you would expect. The other three components, however, are actually interpreted by type, a `` `char `` or nested character item being used for `labels`, a `` `float `` item being used for `sizes`, and an `` `int `` item being used for `widths`! Their order does not matter! In fact, if two or more of them have the same type, then the last is used and the rest (of the same type) are quietly ignored!

Only the ticks are required, and if they alone are specified the `ticks` vector does not need to be nested. For any of the other three components, if the number of its elements exceeds the number of ticks, the extra elements are ignored. For sizes and widths, if there are fewer elements than ticks, the elements are used cyclically, in the manner of the A+ Reshape primitive function.

The label attributes are functional, giving a way to set the tick marks and their associated values under a variety of dynamic situations, such as scrolling. There are several attributes whose values can be queried but not set which give information that helps in laying out the tick marks and labels; see the table "Attributes That Assist in Specifying Tick Marks and Labels".

Examples of Correct `` `xlabel `` Specifications

```
(`xlabel;)    ⍝ Set all to defaults. Labels: values; ticksizes: full (1.); no
gridlines.
(`xlabel; 2 3 4)                  ⍝ Ticks at 2 3 4. Defaults for others.
(`xlabel;(2 3 4; 3 2⍴"abcdef"))  ⍝ Ticks, labels. Defaults for others.
(`xlabel;(2 3 4; .4 .6 .4))      ⍝ Ticks and ticksizes.
(`xlabel;(2 3 4; 1 2 1))         ⍝ Ticks and linewidths.
(`xlabel;(2 3 4; 1 2))           ⍝ Same effect as preceding.
(`xlabel;(2 3 4; .5; 2))         ⍝ Ticks, ticksizes, linewidths.
(`xlabel;(2 3 4; 3 2⍴"abcdef"; 1))  ⍝ Ticks, labels, linewidths.
(`xlabel;(2 3 4;("ab";"cd";"ef"); .7; 1)) ⍝ All.
```

Zero axes

Zero axes can also appear in a graph. For which axes they are shown, their style, their color, and their thickness are governed by the attributes zero, zerostyle, zerofg, and zerowidth. The table "Axis Attributes for Graphs" gives their ranges of values and their defaults.

Axis Attributes for Graphs

These are all attributes of graph objects

Attribute	**Description**	**Data Format**	**Default Value**
axis	`` `std ``, `` `box ``, `` `none ``, or a symbol containing one or more of *x*, *y*, *X*, *Y* (see "The axis, grid, and rule Attributes")	symbol	`` `std ``
grid	`` `none ``, or a symbol containing one or more of *x*, *y*, *X*, *Y* that does not contain both *x* and *X*, or *y* and *Y* (see "The axis, grid, and rule Attributes"). Grid lines can occur only at major tick marks, and only at all of them. To have irregularly spaced grid lines, instead of using the grid attribute specify traces that each have two points with the same x value and minimum and maximum y values and attributes such as ``(`legend;`;  `linestyle;`dash; `linecolor;`black)``. These "grid" traces can be dependencies whose values depend upon those of the true traces.	symbol	`` `none ``
gridfg	The color of the grid lines.	string or symbol	black
gridstyle	The style of the grid lines; one of `` `solid ``, `` `dash ``, `` `dotdash ``, `` `dot ``.	symbol	`` `dash ``
gridwidth	The line width of the grid lines.	numeric	0 (very thin; like 1 for display, but finer for printing)
rule	`` `std ``, `` `box ``, `` `none ``, `` `rule ``, or a symbol containing one or more of *x*, *y*, *X*, *Y* (see "The axis, grid, and rule Attributes").	symbol	`` `std ``
rulewidth	The line width of axis rules and tick marks.	numeric	0 (very thin; like 1 for

			finer for printing)
xfg, Xfg, yfg, Yfg	The color of the indicated axis, axis labels, and tick marks.	string or symbol	black
x/X/y/Ymin, x/X/y/Ymax	The minimum and maximum values of the indicated axes. For these attributes to be effective, min must be less than max (if both are set); otherwise, the minimum and maximum values of the indicated axes are derived from the trace data. Use values in the same domain as the data to which the pair applies, e.g., dates in the same form. See xmax in the table listing all attributes.	numeric	Computed values used.
xtitle, Xtitle, ytitle, Ytitle	The text of the indicated axis title.	string	null string
x/X/y/Ytitlefg	The color of the text in the indicated axis title.	string	black
x/X/y/Ytitlefont	The font in which the text of the indicated axis title is set.	string	lucidasans typewriter-12
x/X/y/Ytitlejustify	`` `left ``, `` `center ``, `` `right `` for horizontal titles; `` `top ``, `` `center ``, `` `bottom `` for vertical titles.	string	`` `center ``
ymode	Controls which way the values run on the y-axis and Y-axis: `` `ascend `` or `` `descend ``.	numeric	`` `ascend ``
ytitlestyle, Ytitlestyle	The y-axis and alternate y-axis titles can each appear above the axes (`` `hor ``) or stacked vertically alongside (`` `ver ``). In both cases the titles are outside the plot area of the graph.	symbol	`` `hor ``
zero	Which zero axes appear: `` `none ``, `` `x ``, `` `X ``, `` `y ``, `` `Y ``, `` `xy ``, `` `xY ``, `` `Xy ``, or `` `XY ``.	symbol	`` `xy ``
zerofg	Controls the color of the zero axes.	string or symbol	slategray
zerostyle	The style of the zero axes: `` `dot1 `` through `` `dot5 ``, `` `dotdash1 `` through `` `dotdash5 ``, `` `dash1 `` through `` `dash5 ``, or `` `solid ``.	symbol	`` `dot1 ``
zerowidth	Controls the width of the zero axes (0 - 10).	numeric	0 (very thin; like 1 for display, but finer for printing)

Axis Label and Tick Mark Attributes for Graphs

These are all attributes of graph objects

Attribute	**Description**	**Data Format**	**Default Value**
x/X/y/Yinc	The increment in axis coordinates between adjacent major tick marks on the indicated axis. When it is set to nothing, (`` `xlabelout; ``), A+ determines this value based on trace data and axis label size.	numeric	0
xlabel, Xlabel, ylabel, Ylabel (functional)	The tick mark locations, and any of the following: the text of the labels, the lengths of the tick marks as fractions of the majorticksize setting, and the widths (in pixels) of the grid lines. For details see "The xlabel, Xlabel, ylabel, and Ylabel Attributes".	numeric or nested vector	
xsublabel, Xsublabel (functional)	Sublabels provide a second row of axis labels below the label row. The interpretation of these attributes is the same as for label attributes (see above in this table).		
x/X/y/Ylabelfont	The font in which the labels on the indicated axis are set.	symbol, string	lucidasans typewriter-12
x/Xlabeljustify, x/Xsublabeljustify	Justify the labels or sublabels of the indicated axis: `` `left ``, `` `center ``, `` `right ``.	symbol	`` `center ``
y/Ylabeljustify	Justify the labels of the indicated axis: `` `bottom ``, `` `center ``, `` `top ``.		
x/X/y/Ylabelout, x/Xsublabelout (functional)	Specification of tick label and sublabel formatting, in the same way that data formats are specified for the screen with the out attribute. Furthermore, set to ` or	symbol, function, or character	

	`` `none `` to remove the labels. If x/Xsublabelout is given a value when no sublabels are present, they appear automatically.	string	
x/X/y/Ymajorticksize	The size of the major ticks on the indicated axis, in pixels.	numeric	10
x/X/y/Yminorticks	The number of minor tick marks that appear between consecutive major tick marks on the indicated axis.	numeric	1, 1
x/X/y/Yminorticksize	The size of the minor ticks on the indicated axis, in pixels.	numeric	6
x/X/y/Ytickstyle	The tick style for the indicated axis: if `` `out `` or `` `in ``, the tick marks point out of or into the plot area; if `` `inout ``, the tick marks straddle the axis.	symbol	`` `out ``

Attributes That Assist in Specifying Tick Marks and Labels These are all attributes of graph objects

Attribute	**Description**
x/X/y/Yextent	A three-element vector *v* giving the minimum value for the indicated axis (`v[0]`), the maximum value (`v[1]`), and a scale factor (`v[2]`) such that `(v[1]-v[0])×v[2]` is the length of the axis in pixels.
x/X/y/Ylabelheight	The height, in pixels, of any label on the indicated axis, based on the current font.
x/X/y/Ylabelwidth	The width, in pixels, of a specified label for the indicated axis. The text of the label is given as a character vector, as for example: ``(`xlabelwidth;'label text') of `g`` where `` `g `` is the graph. A vector of character vectors or a character matrix can also be given for the text, and a vector of pixel lengths is returned. In the case of a matrix, the rows are taken as the labels.

Interactions with Graphs

There are a variety of interactions with graphs to customize the display, edit and manipulate trace data, perform detailed examination of traces, and navigate through graphs that are only partially displayed. These interactions are described in "User Interactions with Displays", and particularly in "Graph Objects". The following attributes provide the means for programmers to monitor user interactions and control the responses:

Attributes for Interactions with Graphs

Attribute	Description
addtexttrace	Like addtrace, but applies to text traces. See "Example" regarding automatic creation of variable names.
addtrace	An attribute with callback (see "Attributes with Callbacks"). If it is set to a value, which must be a function or a `(function; static_data)` pair, that function is called whenever a numeric trace is created interactively; the value of the coordinate attribute is the trace set of the created trace. The default behavior is to create a global variable containing the trace set and to append the name of this trace-set object (in symbol form) to the graph object; the new trace set is a two-column matrix and its style is *`line*. See "Example" regarding automatic creation of variable names.
coordinate	This is a reference-only attribute whose value is the two-element vector (x-axis coordinate, y-axis coordinate) of the pointer position in the case of a refer event on a graph, or the entire trace in the cases of a refer event on a trace and an interactive creation of a new trace.
Coordinate	Like coordinate, but the value is the alternate axis coordinates of the pointer position in the case of a refer event on a graph.
copytexttrace	Like addtexttrace, but applies when text traces are copied interactively. See "Example" regarding automatic creation of variable names.
copytrace	Like addtrace, but applies when a numeric trace is copied interactively. A numeric trace to be copied interactively must be selectable; see the selectable and refer attributes. See "Example" regarding automatic creation of variable names.
delete	Like addtrace, but applies when a trace is deleted interactively. The default value of this attribute is 0, and with this value traces cannot be deleted interactively. If it is 1, traces can be deleted interactively, and the variable of the deleted trace is automatically removed from the graph object. Only text traces and selectable numeric traces can be deleted interactively; see the selectable attribute.
mode	Set up interactive entry (*`addtrace*) or text entry (*`addtexttrace*) at *`coordinate*, or terminate entry, like **Enter** or double click (*`normal*).
movelimit	Restrains the movement of a data point being modified interactively; one of *`x* (the default), *`y*, and *`none*.
refer	A refer event occurs on the graph object when the left button is clicked with the pointer anywhere on the graph object except on a data point of a numeric trace. A refer event on a selectable trace occurs when the pointer is on the trace and the left button is double-clicked; see the selectable and coordinate attributes, and "Attributes with Callbacks".

referpoint	A referpoint event occurs when the pointer is on any data point of any numeric trace and the left button is clicked; see the selected attribute, and "Attributes with Callbacks".
selectable	Effective with two-column line traces only; that is, only two-column line traces are selectable. If the value is 1, the trace can be selected, i.e., the user can interactively manipulate the trace. The default value is 0.
selected	The value is the two element vector of row, column indices of the selected data point when a referpoint event occurs.
textactivate	A textactivate event occurs a text trace is interactively modified. The default action is to replace the value of the title attribute of the text trace with the new text. Note that there is no counterpart for the interactive modification of a numeric trace, but that action can be monitored with a callback function.
x/X/y/Yextent	These attributes can be used to detect zooming, since each one contains the least and greatest values currently shown on its axis, as well as a scale factor.
x/X/y/Ymin, x/X/y/Ymax	Resetting these attributes, even to all zeros, is a way of undoing in a program any zooming a user has done on the screen, in order, for example, to ensure that a new point is shown promptly.

Data points can be moved interactively (see "Data-Point Move"). Their movement can be restrained to vertical only, so that x coordinates don't change, by setting the movelimit attribute to `` `x``, or to horizontal only by setting it to `` `y``. There are no restraints when it is set to `` `none``.

Miscellaneous Graph Attributes

Miscellaneous Attributes for Graphs

These are all attributes of graph objects.

Attribute	**Description**	**Data Format**	**Default Value**
barwidth	The maximum width in pixels of bars in bar and stack graphs.	numeric	10
bg	The background color of the graph.	string or symbol	gray
bottom, top, left, right	The margin width or height of the area consisting of the plot area (the region within the four axis rules, even when some of the rules are absent), axes and labels, and axis titles, as a percentage of the default width or height of that area (a whole number or a fraction, e.g., 20 or	numeric	0

	0.20, but only a whole number if in excess of 100%).		
debug	If 1, the A+ processor displays a message every time a graph is redrawn, updated (i.e., the last point on a trace is changed), or printed.	integer	0
fg	The color of the axes, axis labels, legend titles and the legend border, subtitle, and footnote when their colors have not been explicitly specified.	string or symbol	black
selectdistance	The distance in pixels from which a trace data point is selectable for interactive manipulation of its value.	numeric	10
shadowthickness	The width in pixels of the shadow area (the border) around the graph. Set to 0 for no border.	numeric	2
xleft, xright, Xleft, Xright, ytop, ybottom, Ytop, Ybottom	xleft is the left margin of the xy trace area and xY trace area as a percentage of the axis range (a whole number or a fraction, e.g., 20 or 0.20, but only a whole number if in excess of 100%). Analogously, Ytop is the top margin of the xY and XY trace areas, and all the others are similar. The xy trace area is the region holding traces plotted against the x and y axes. By default it coincides with the plot area. There is also an xY trace area holding traces plotted against the x and alternate y axis, as well as Xy and XY trace areas.	numeric	0
xs, ys, extent	xs, ys: the dimensions of the graph in pixels. extent: the vector x, y, xs, ys, where x and y are the graph's coordinates in pixels relative to its parent, or to the screen if it is top-level.	numeric	xs and ys: 200. extent: 0 0 200 200

Lists of All Graph and GraphTrace Attributes

The keys in the "Table of All Display Attributes" in the "Display Attributes" chapter that pertain to the graph display class are G, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the graph display class (other than the print... attributes) are:

active addtexttrace addtrace ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector axis b barwidth bg bottom bound children class clear coordinate Coordinate copytexttrace copytrace deiconized delete descendents doc downto dynamic eval evaluate exit extent f1-f12 familytree fg fkeys focus followers followertree font foot footnote footnotefont footnotejustify freeze fullscreen grid gridfg gridstyle gridwidth h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader left leftto legend legendbg legendfg legendfont legendhlthickness legendshadowthickness legendstyle literal lower mapped mode naturalsize newshow notify outofcurrentworkspace parent pieoffsetmargin pin preset primary r raise realize recursively refer refresh reparent request reshow resize resizeable right rightto rule rulewidth script selectdistance selected sensitive set settings shadowthickness shell shelltitle show state stateself subtitle subtitlefg subtitlefont subtitlejustify syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify top upto vcol vcols verify vrow vrows w W ws x X xextent xfg xinc xlabel xlabelfont xlabelheight xlabeljustify xlabelout xlabelwidth xleft xlegend xmajorticksize xmax xmin xminorticks xminorticksize xright xs xsublabel xsublabeljustify xsublabelout xtickstyle xtitle xtitlefg xtitlefont xtitlejustify y Y ybottom ylegend ymode ys ytitlestyle ytop yx YX yxs zero zerofg zerostyle zerowidth

The keys in the "Table of All Display Attributes" that pertain to the graphTrace display class are gT and ALL.

The attributes that are meaningful for the graphTrace display class are:

active ancestors bg bound class coordinate doc f1-f12 eval evaluate fillcolor fkeys gradient has is legend linecolor linestyle linewidth movelimit parent pieangle pieaspectratio piedepthfactor pielegendalign pieoffsets piepercentalign pieprimaryslice pieprimaryslicealign pieprofiles pievaluealign preset protect protected refer referpoint script selectable selected set settings shell state stateself style symbol symbolsize textactivate textfg textfont title titlefg titlefont titlejustify verify xaxis yaxis

14. The Hgauge Display Class

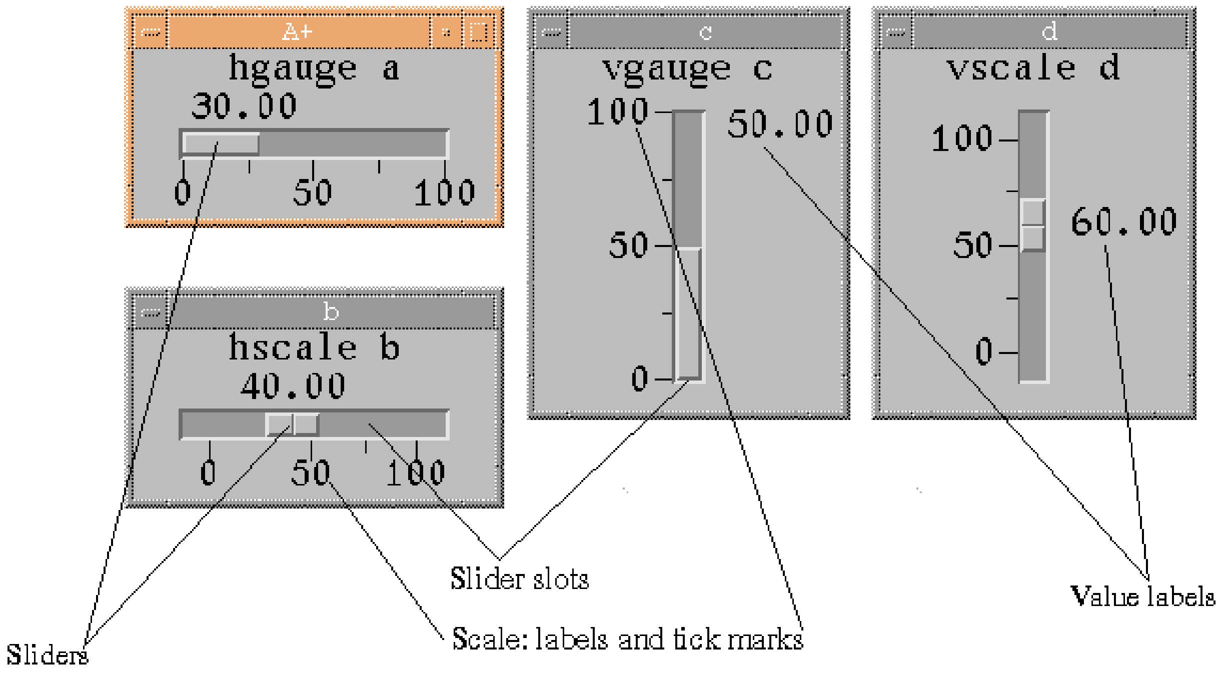
The hgauge display class displays a scalar number in graphical and also (optionally) digital form. The graphical representation shows a slider extending from the value of the **min** attribute to the value of the variable; the scale continues to the value of the **max** attribute. If the variable value is out of the **min-max** range, the slider occupies all or none of the slider slot, but the value label shows the correct number.

Labels for the minimum and maximum values can be specified in the **mintitle** and **maxtitle** attributes; their location, font, and color can be set using **mintitlejustify**, **mintitlefont**, **mintitlefg**, etc. Tick size is controlled by **majorticksize** and **minorticksize**, and the number of ticks between major ticks by **minortickcount**. Parallel sets of attributes, **label...** and **value...**, govern aspects of the scale and the value label, although **labelout** and **out** do not conform to this parallel naming convention. There are **sliderbg**, **sliderheight**, and **sliderwidth** attributes; for an hgauge, **sliderwidth** is ignored because the width depends upon the value. Other properties are covered by the usual attributes: **title**, **subtitle**, **bg**, etc.

For comparison, the three closely related classes vgauge, hscale, and vscale are also shown in the figure.

Visual Representation

```
(a;b;c;d)←30 40 50 60
`a`b`c`d is¨ `hgauge `hscale `vgauge `vscale
`a`b`c`d has¨ `title,¨<¨("hgauge a";"hscale b";"vgauge c";"vscale d")
show¨ `a`b`c`d
`a`b`c`d has¨ ((`x;3);(`x;3;`y;155);(`x;225);(`x;416))
```



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the hgauge display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are hG, ALL, CNFT, GS, NFT, and TOP.

The attributes that are meaningful for the hgauge display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate execute exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle inc incurrentworkspace is l labelfg labelfont labelinc labeljustify labelout[1] leader leftto literal lower majorticksize mapped max maxtitle maxtitlefg maxtitlefont maxtitlejustify min minortickcount minorticksize mintitle mintitlefg mintitlefont mintitlejustify naturalsize notify out[1] outofcurrentworkspace pageinc parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show sliderbg sliderheight sliderwidth state stateself subtitle subtitlefg subtitlefont subtitlejustify syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto valuefg valuefont valuejustify vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

1. Note that labelout is used to format the scale labels and out to format the value label.

15. The Hgrid Display Class

There are two grid classes, hgrid and vgrid. They are container classes like the layout class, but their children have their natural sizes, rather than being sized to fill the layout as far as possible, any placement requests by the children (right, bottom, etc.) are ignored, and the grid objects have no titles **-** the value of the **title** attribute is ignored.

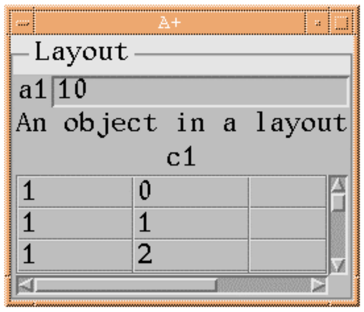
The hgrid class, unlike the vgrid and layout classes, has a horizontal default arrangement. Objects of the three classes, specified in the same way except for class and the geometry of the horizontal layouts, are shown in the figure below for comparison. The layout and grid backgrounds are shown in a lighter grey to make clear what space the objects occupy. Normally, they would appear in the same shade of grey as the rest of the objects, to present a homogeneous appearance, like the vertical layout with no title.

Points to notice in the figure are:

- The widths of the vertical layouts and grid are determined by the length of the label. The heights of the horizontal layouts and grid are determined by the height of the array.
- In the layouts, objects are increased from their natural sizes to fill the space available for them, causing the array to have a partial column in the vertical layout. The scalar and label objects, however, are constrained vertically, by default, so they are centered (by default) and the extra space is considered part of the layout background.
- In the grids, each object is placed (top left) in the next available space. The grid retains the leftover scraps of space. If `c5` and `c6` had been bound and their **rows** attribute set to 3, and *then* were reparented to the grids, there would not be the two rows of leftover space at the bottom.

Visual Representation

```
    a1←a2←a3←a4←a5←a6←10
    b1←b2←b3←b4←'An object in a layout'
    b5←'An object  in a vgrid'
    b6←'An object in an hgrid'
    c1←c2←c3←c4←c5←c6←(ι10)∘.*ι3
    l1←     `a1`b1`c1; l2←1 3 ρ`a2`b2`c2; l3←`a3`b3`c3
    l4←1 3 ρ`a4`b4`c4; l5←     `a5`b5`c5; l6←`a6`b6`c6
    ⍝ To show layout background:
    `l1`l2`l3`l4`l5`l6 has¨ <`bg `gray90
    `l1`l2`l3`l4`l5`l6 has¨ `title,¨ <¨ ('Layout';'Layout, horizontal'; '';
''; 'Vgrid'; 'Hgrid')
    {`l1`l2`l3`l4`l5`l6 `b1`b2`b3`b4`b5`b6 s.are
        (4ρ`layout),`vgrid,`hgrid, 6ρ`label;}
    `c1`c2`c3`c4`c5`c6 has¨ <(`rows;3)
    show¨ `l1`l2`l3`l4`l5`l6
    `l2`l4`l6 has¨<(`x;280)
    `l3`l4`l5`l6 has¨((`y;240);(`y;240);(`y;450);(`y;450))
```


A+

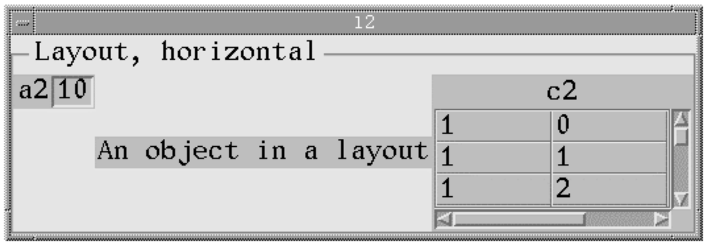
Layout

a1 10

An object in a layout

c1

1	0	
1	1	
1	2	


12

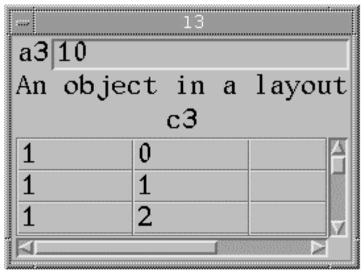
Layout, horizontal

a2 10

An object in a layout

c2

1	0
1	1
1	2

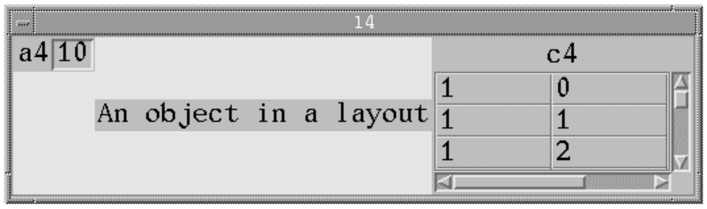

13

a3 10

An object in a layout

c3

1	0	
1	1	
1	2	

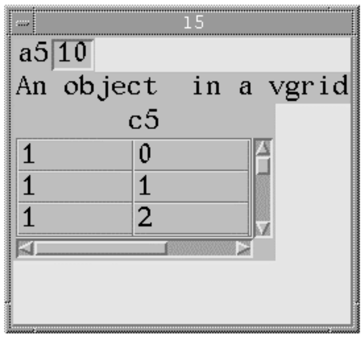

14

a4 10

An object in a layout

c4

1	0
1	1
1	2

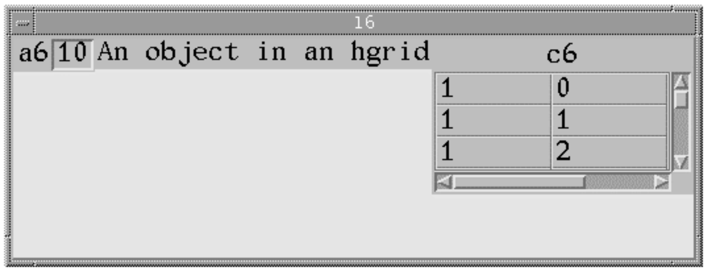

15

a5 10

An object in a vgrid

c5

1	0
1	1
1	2


16

a6 10 An object in an hgrid

c6

1	0
1	1
1	2

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the hgrid display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are hR, ALL, CNFT, CNT, NFT, and TOP.

The attributes that are meaningful for the hgrid display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b be bg bound build children class clear deiconized descendents doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fg fkeys focus followers followertree font foot freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lower mapped naturalsize newshow notify outofcurrentworkspace parent pin position preset primary r raise realize recursively refresh reparent request reshow resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself structure syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

16. The Hmenu Display Class

The hmenu and vmenu display classes are for representing nested slotfillers as cascade menus. When hmenu is used the top-level menu is laid out horizontally, whereas vmenu gives a vertical layout. All submenus are vertical in both cases. Menus cannot be edited.

A slotfiller consists of two parts, the symbolic indices and the values. A nested slotfiller is one whose values are also slotfillers. The values within the slotfiller values can also be slotfillers, and so on; slotfillers can be nested to any level. In terms of the menu display classes, the symbols of a slotfiller bound to either menu class correspond to the items in the top level of a menu. If a value is also a slotfiller, then its symbols correspond to a submenu, and so on.

A callback function on the slotfiller is required for menu actions; in the `d` position, only `p⊃d` is given when the function is called. A callback occurs whenever a menu item is selected. Selecting an item does not cause a preset callback to fire.

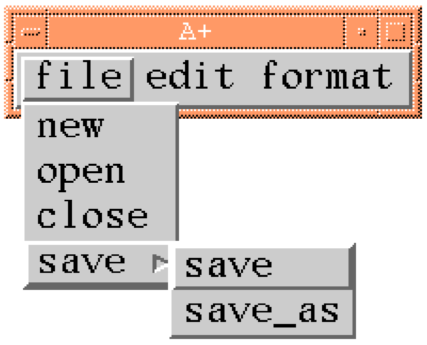
See "The Vmenu Display Class" for details of the vertical menu.

Visual Representation

```
menu←s.rsf{
   ⍝ s.rsf converts a recursive association list to a recursive slotfiller
       (`file;(
                  `new;        newfn;
                  `open;       openfn;
                  `close;      closefn;
                  `save;(
                               `save;      savefn;
                               `save_as;   save_asfn));
       `edit;(
                  `undo;       undofn;
                  `cut;        cutfn;
                  `copy;       copyfn;
                  `paste;      pastefn);
       `format;(
                  `font;(
                               `kaplgallant;;
                               `courier;);
                  `size;       sizelist;
                  `style;      stylelist)
  )
      }
```

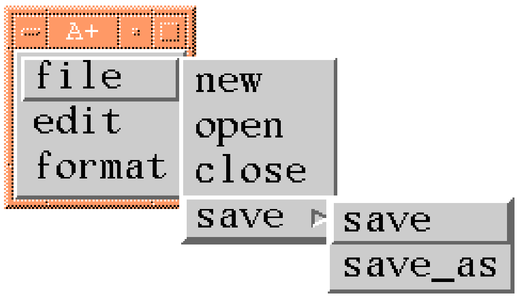
hmenu: Horizontal Cascade Menu:

```
show    `menu    is    `hmenu
```



vmenu: Vertical Cascade Menu:

```
show    `menu    is    `vmenu
```



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the hmenu display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are hM, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the hmenu display class (other than the print... attributes) are:

acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lower mapped mnemonics naturalsize notify outofcurrentworkspace parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings script shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

17. The Hpane Display Class

The hpane and vpane display classes are special layout classes that have movable dividers, or sashes, between the layout children. The user can control the proportion of the layout that each child occupies simply by moving the dividers. Various layout-constraint attributes, such as **position** and **build**, are disregarded. The children of an hpane layout should be arranged horizontally, and will have vertical dividers between them.

To move a divider, simply place the mouse pointer on it, press and hold the left mouse button, and then slide the divider left or right. A small button is provided on the bottom of each divider as a convenient place to locate the pointer.

The dividers of an hpane layout always extend from the top to the bottom of the layout, so care must be taken when using hpane layouts with more than one row. For example, in an hpane layout of the form ``(`a`b;`c)``, the divider between objects *a* and *b* will cut through object *c*. If several objects line up on a divider boundary, then movement of the divider controls the display sizes of these objects. If all the objects line up on divider boundaries, then movement of the dividers controls the display sizes of entire columns of objects.

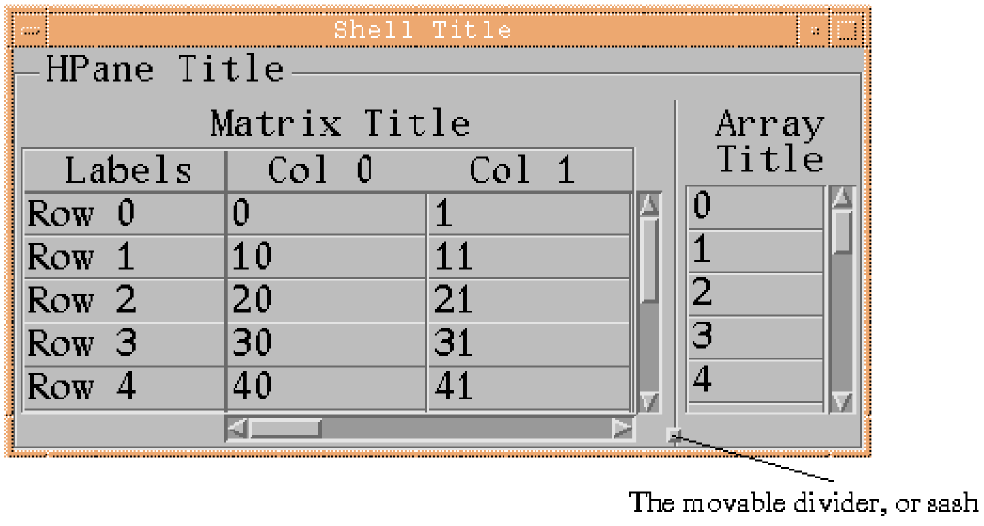
There is no direct way for a program to determine where a user has positioned a divider in an hpane layout. Sometimes there are indirect ways to approximate the position, such as checking the values of the **rows** and **cols** attributes of a matrix to determine how many rows and columns are currently shown.

Compare HPane with "The Vpane Display Class" for a visual representation of a similar object.

Visual Representation

```
m←ι10 10
`m has (`class;`matrix;  `title;"Matrix Title";
        `label;("Labels";
                (<"Row"),¨⍕¨ι10;
                (<"Col"),¨⍕¨ι10));
a←ι20
`a has (`class;`array;  `title;("Array";"Title"));
h←1 2ρ`m`a
```

```
`h has (`class;`hpane;  `title;"HPane Title";
        `shelltitle;"Shell Title"; `show;1);
```



Shell Title

HPane Title

Matrix Title

Labels	Col 0	Col 1
Row 0	0	1
Row 1	10	11
Row 2	20	21
Row 3	30	31
Row 4	40	41

Array Title

0
1
2
3
4

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the hpane display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are hP, ALL, CNFT, CNT, NFT, and TOP.

The attributes that are meaningful for the hpane display class (other than the print... attributes) are:

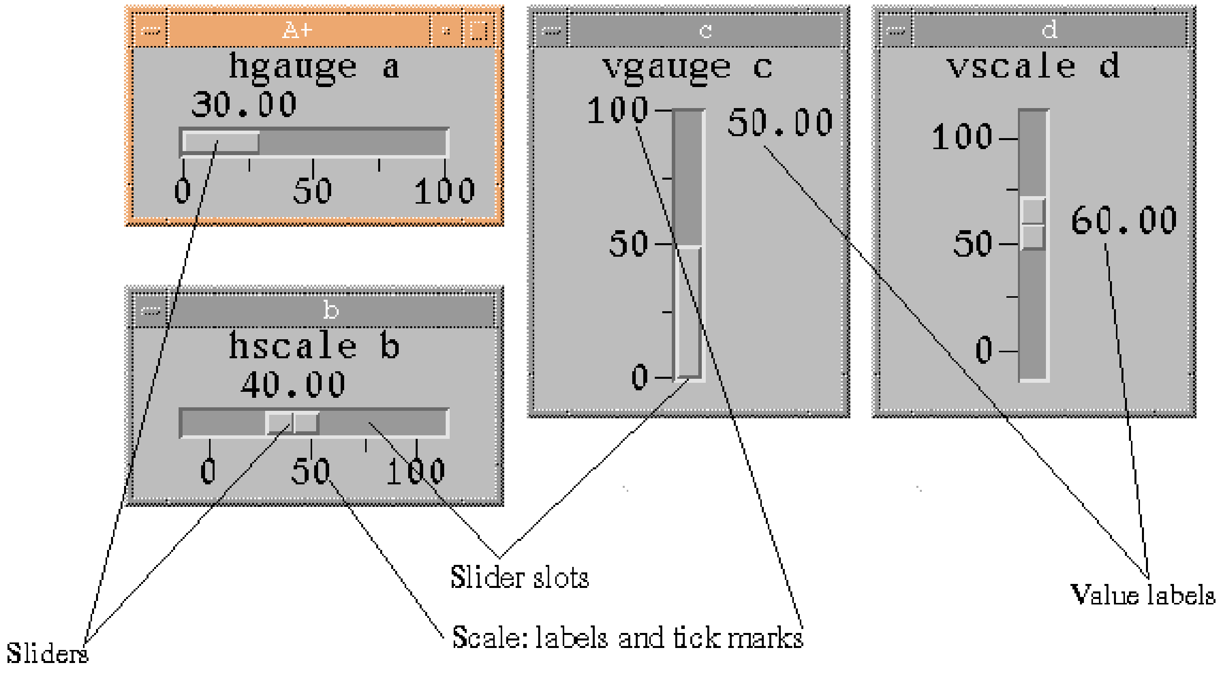
active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b be bg bound C children class clear constraints deiconized descendents doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fg font fkeys focus followers followertree foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lockposition locksize lower mapped naturalsize newshow notify outofcurrentworkspace parent pin preset primary r R raise realize recursively refresh reparent request reshow resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself structure syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols vcolspace verify vrow vrows vrowspace w W ws x X xs y Y ys yx YX yxs

18. The Hscale Display Class

The hscale class is like the hgauge class except for minor differences in appearance - principally that the slider is fixed in length and its middle points to the object's value - and one major difference: a user can set the value interactively. See the description of the hgauge class and the figure there illustrating all four slider classes.

A user can change the setting of an hscale object (and of course its underlying variable) by editing the value label or moving the slider in several ways. The **pageinc** attribute, whose default setting is 10, determines the amount of change when **Page Up** or **Page Down** is pressed, and **inc**, whose default setting is 1, the amount of change when an **arrow** key is pressed. Four attributes control editing behavior and appearance: **edit**, **editbg**, **editfg**, and **editspace**; currently, the default values of **editbg** and **editfg** are both black.

See "Hscale and Vscale Objects", in "User Interactions with Displays".



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the hscale display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are hS, ALL, CNFT, GS, NFT, and TOP.

The attributes that are meaningful for the hscale display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic edit editbg editfg editspace eval evaluate execute exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle in inc incurrentworkspace is l labelfg labelfont labelinc labeljustify labelout[1] leader leftto literal lower majorticksize mapped max maxtitle maxtitlefg maxtitlefont maxtitlejustify min minortickcount minorticksize mintitle mintitlefg mintitlefont mintitlejustify naturalsize notify out[1] outofcurrentworkspace pageinc parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show sliderbg sliderheight sliderwidth state stateself subtitle subtitlefg subtitlefont subtitlejustify syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto valuefg valuefont valuejustify vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

1. Note that labelout is used to format the scale labels and out to format the value label.

19. The Label Display Class

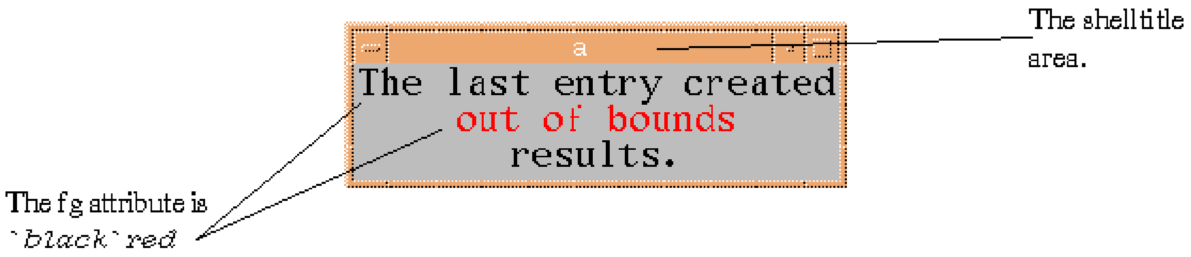
The label class is for displaying text that is in either of the following forms:

- a simple character scalar, vector, or matrix;
- a nested scalar or vector whose items are simple character scalars or vectors.

Labels cannot be edited. Unlike views, labels are completely displayed (up to some reasonable limit of the screen capacity) and never have scrollbars.

Visual Representation

```
a←('The last entry created';
   'out of bounds';
   'results.')
`a is `label
`a has (`fg`shelltitle;(`black`red;'a'))
show `a
```



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the label display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are B, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the label display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is justify l leader leftto literal lower mapped margin naturalsize notify outofcurrentworkspace parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

20. The Layout Display Class

A layout is a container whose children can be of any classes. The arrangement of the children in a variable *x* to be bound to this class usually determines the geometry of the display. A+ arranges the display, however, when the variable is a vector. If the value is a boxed vector of vectors, then each item of the value composes a row in the layout. For example, if *x* is a layout defined by (`a `b;`c), then the display of *x* as a layout shows the displays of *a* and *b* side by side, and the display of *c* below these two. If *x* is a matrix and all its elements are distinct, then the display of the child x[i;j] occupies the (i;j)th position in the display of the layout. If there are duplicates in *x*, then they must occupy contiguous positions in a subrectangle, in which case the display of the child fills all these "positions" in the layout. The (i;j)th position can be left vacant by assigning x[i;j]←` (the empty symbol). The arrangement of a layout can also be specified by setting the **at** attribute on each of its children. Cf. "Layouts, Geometry, Constraints".

A layout can be built from objects already displayed on the screen, and the relative arrangement of those objects on the screen can be maintained in the layout. To do this, set the **build** attribute or the s-context variable *s.AUTOBUILD* to 1 - its default is 0 - before creating the layout. (If the attribute is 0 the layout is still built, but the children are arranged in the order implied in the specification of the layout, and thus perhaps more compactly.) Then form the layout variable, as a simple vector of symbols naming the objects that are to go in the layout; any objects that are named but not currently displayed will be put in a pile in the top left corner of the layout - i.e., the default position within the layout is 0, 0. Finally, show the layout variable.

The manner in which a child behaves when a layout is resized depends in part on its display class. A child of class button, scalar, label, slot, check, or radio is called *sticky*, while all others are called *nonsticky*. If a layout has at least one row with only nonsticky children, then any row containing sticky children retains its vertical size when the layout is resized. See the **resize** attribute for other aspects of resizing layouts.

When a child is placed in a layout, the context of its name is assumed to be that of the layout. There is no need to qualify the child's name unless its context differs from the layout's.

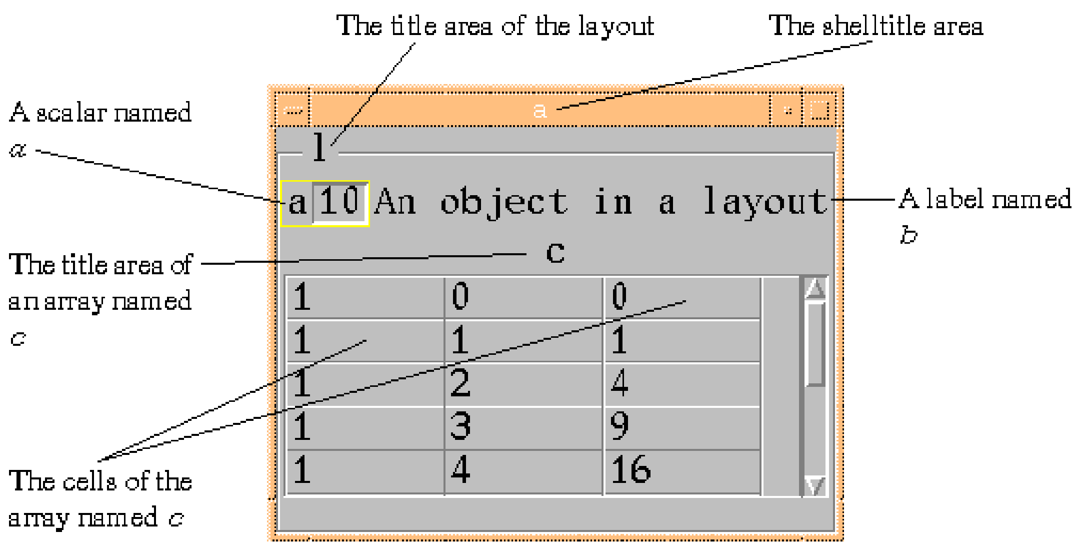
Modifying a Layout

If the **newshow** attribute is 0 (the default), objects appended to an existing layout do not automatically appear; each must be shown with the function *show* or its mapped or show attribute must be set to 1 or hide to 0.

Visual Representation

A line does not appear between the title area and the rest of the layout. Instead, the frame or outline is brought up higher, into the title area, and the title is set into it. Moreover, the title is left justified rather than centered. Note the change for scalar display also.

```
a←10
b←'An object in a layout'; `b is `label
c←(⍳10)∘.*⍳3
l←(`a`b;`c)
`l is `layout
`l has (`shelltitle;'a')
show `l
```


The title area of the layout
The shelltitle area
A scalar named *a*
a
l
a 10 An object in a layout
A label named *b*
The title area of an array named *c*
c
The cells of the array named *c*

1	0	0
1	1	1
1	2	4
1	3	9
1	4	16

Attributes Specific to Layouts

Here "layouts" includes five display classes: layout, hpane, and vpane, and, except for the **R** and **C** attributes, hgrid and vgrid.

at

The **at** attribute is a four-element integer vector representing the position and extent of an object in a layout:

(virtual row, virtual column, number of virtual rows, number of virtual columns)

Conceptually, a layout can be placed over an imaginary grid so that the border of each object in the layout lies on grid lines. It is then possible to describe the position and size of each object by the row index and column index of the grid block containing the upper left corner of the object, and the number of grid rows and columns that the object overlaps. If the grid is the minimal one, i.e., the one with the fewest possible grid lines, then these four numbers are the value of this attribute. The grid intervals need not be equal in either direction.

The origin for the grid coordinates is the upper left corner of the layout. Note that the four elements of an **at** attribute value can be set individually using the attributes **vrow**, **vcol**, **vrows**, and **vcols**.

For example, the arrangement shown in the figures could be obtained by:

```
     l←`a`b`c
     `a has (`at;0 0 1 1); `b has (`at;0 1 1 1);
     `c has (`at;1 0 1 2)
```

resize and h, H, t, b, l, r, w, and W

The **resize** attribute controls the detailed placement of objects in a layout and their behavior when it is resized. The objects in a layout do not necessarily fill in all the background area. For example, the label in a layout consisting of a label next to an array is vertically centered in an area the same height as the array. When an object does not fill its allotted background area it can be positioned in that area by setting this attribute: `'t'` for top; `'b'` for bottom; `'l'` for left; and `'r'` for right.

The size of an object in a layout, which is normally affected by other objects in its same row and column, or by resizing the layout, can also be controlled: `'H'` and `'W'` mean do not resize the object's height and width, respectively. If the newly allotted space for the object within the layout is larger than the object, its position within that space is controlled by the **justify** attribute. Finally, even stronger resize restrictions are available: `'h'` and `'w'` mean do not resize the entire row and column in which the object is positioned. This attribute can also be set to a character vector holding more than one of the above values.

Each setting takes effect in addition to existing ones unless there is a period in the setting. All settings are removed by setting the **resize** attribute to the empty character vector or the character period: `''` or `'.'`. The default settings are restored by setting it to the Null.

The cumulative settings of the resize attribute can also be accomplished with the specific attributes for each one:

`` `layout has (`resize;'t') `` is equivalent to `` `layout has (`t;1) ``

`` `layout has (`resize;'l') `` is equivalent to `` `layout has (`l;1) ``

Justifications can be removed by specifying the values of these attributes to be 0.

R and C

The virtual rows or columns (see the description of the **at** attribute above) can be constrained to be all the same size by setting the **R** or **C** attribute to 1, respectively.

position

Several objects can occupy a layout in such a way that only one is visible at a time, taking up the entire area of the layout, and any other one can be brought into view simply by setting the value of its raise attribute to 1. Since the objects within the layout can themselves be layouts, or other complex objects, this provides a simple, efficient, and general way of displaying multiple objects when only one must be visible at a time. For example, here is the procedure to follow:

```
lo←()                   ⍝ Initialize the layout to the Null.
`lo is `layout
`lo has (`position;0)   ⍝ This establishes the
                        ⍝ overlapping behavior.
lo←`a`b`c`d`e`f`g       ⍝ Now specify the layout using
                        ⍝ the simple vector form; other
                        ⍝ objects can be appended later.
show `lo
`d has (`raise;1)       ⍝ The object d will now be
                        ⍝ brought into view.
```

List of All Attributes for the Layout Display Class

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the layout display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are L, ALL, CNFT, CNT, NFT, and TOP.

The attributes that are meaningful for the layout display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b be bg bound build C children class clear constraints deiconized descendents doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lockposition locksize lower mapped naturalsize newshow notify outofcurrentworkspace parent pin position preset primary r R raise realize recursively refresh reparent request reshow resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself structure syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols vcolspace verify vrow vrows vrowspace w W ws x X xs y Y ys yx YX yxs

21. The Matrix Display Class

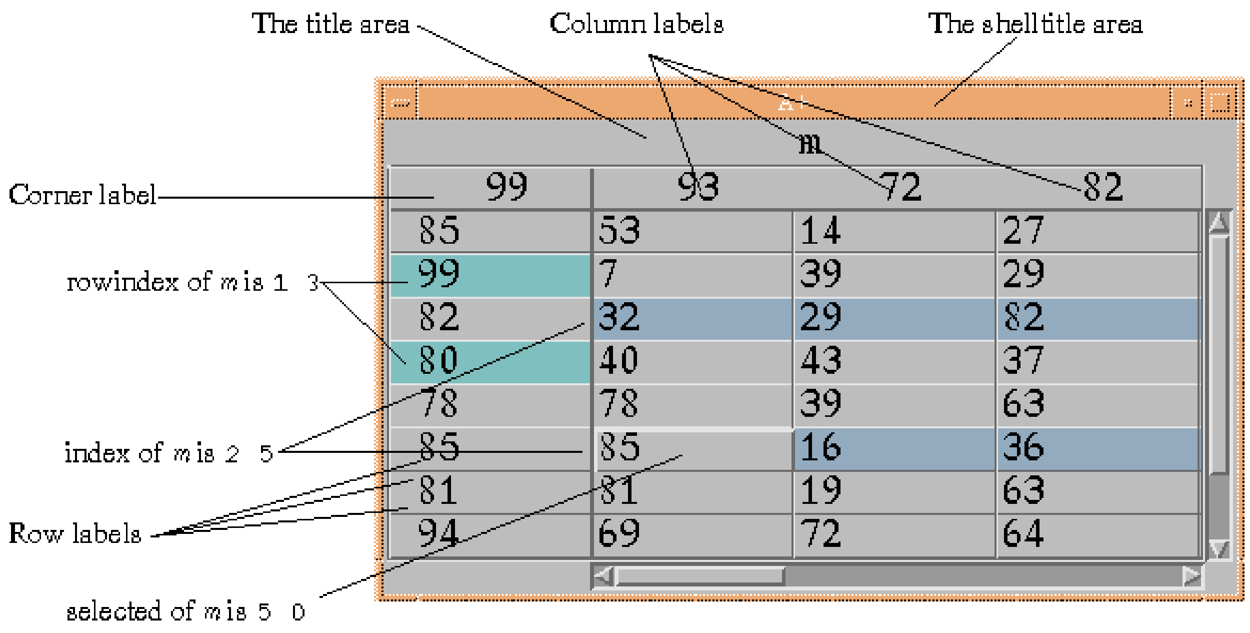
Any A+ matrix that is not a simple character matrix can be displayed as an object of class matrix. A display of a variable in this class has five parts: a title area, three label areas, and a value area. The label area above the value area is called the column label area, the one to the left is called the row label area, and the one in the upper left corner is called the corner label area, and the title area is above them all. The value area appears below the title and column label areas, and consists of delineated cells.

If a matrix requires more than just a few cells, then by default, only a submatrix is presented on the screen, and the value area is provided with scrollbars. The particular subarray that appears in the value area is controlled in two ways: by the scrollbars, manipulated by users, and by the attributes **firstrow**, **firstcol**, **rows**, and **cols**, set by programmers.

The contents of each cell in the value area can be edited.

Visual Representation

```
     m←?10 10ρ100
     labs←⍕¨(⌈/⌈/m;⌈/@1 m;⌈/ m)   ⍝ Corner label; row labels;
column labels.
     `m has (`class;`matrix; `label;labs)
     `m has (`selectionmode;`multiple)    ⍝ Allows multiple rows
to be selected simultaneously
     `m has (`rowindex;1 3)
     `m has (`index;2 5)
     show `m
```



m

99	93	72	82
85	53	14	27
99	7	39	29
82	32	29	82
80	40	43	37
78	78	39	63
85	85	16	36
81	81	19	63
94	69	72	64

To select a single row manually, simply click on the row desired. To select multiple rows manually, first enable multiple selections by entering:

```
    `m has (`selectionmode;`multiple)
```

Then, click on the first of the rows that you want to select. If the other rows are contiguous with the row that you selected, you can simply drag the mouse over the other rows while the left button remains depressed. To select non-contiguous rows, click on the first of the rows that you want to select, and then hold the **Ctrl** key down while you click on additional rows. To see which rows have been selected:

```
     `index of `m
<    2 5
```

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the matrix display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are M, ALL, CNFT, NFT, and TOP. The attributes that are meaningful for the matrix display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg blank bound class clear col colindex colindexbg collabelrows colors cols colsep colspace copy cornerindex cornerindexbg cycle deiconized delete doc done downto dynamic edit editbg editfg editspace eval evaluate execute extent exit f1-f12 fg firstcol firstrow fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness hscrollsize hscrollwith icon iconic iconized icontitle in incurrentworkspace index insertabove insertbelow is l label labelfg labelfont leader leftto literal lower mapped na naturalsize notify out outofcurrentworkspace parent pin preset primary protect protected r raise realize refer refresh request resize resizeable respace rightto row rowbg rowindex rowindexbg rows rowsep script scrollbg scrollsize select selectbg selectcol selectcorner selected selectionmode selectrow sensitive set setcol setfirstcol setfirstrow setrow settings shadowthickness shell shelltitle show size space stars state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows vscrollsize vscrollwith w W ws x X xs y Y ys yx YX yxs

22. The Notebook Display Class

The notebook is a special layout class for displaying several pages of objects in a compact and easily accessible manner. It has somewhat the appearance of a notebook with tabs, showing one page fully and just tabs and edges for other pages. Each child has its own page, and can, of course, be of any class, including a layout or notebook. The notebook does not honor any constraints on a child, configuring it for the available space.

The user can choose an object to be shown by clicking on the tab of its page, which causes the **currentpage** attribute to be set to the name of the object and thus that page to be shown. This causes a `` `pagechangecb `` callback.

The tabs are actually arranged in a line (rather than apparently attached to the pages), and as many are shown as will fit in the window. When not all are shown, scrolling arrows are provided. The tab of the page being shown is absent if it is outside the range of tabs being shown. Otherwise, it is shown attached to its page and aligned with the position in which it would appear if its page were not being shown.

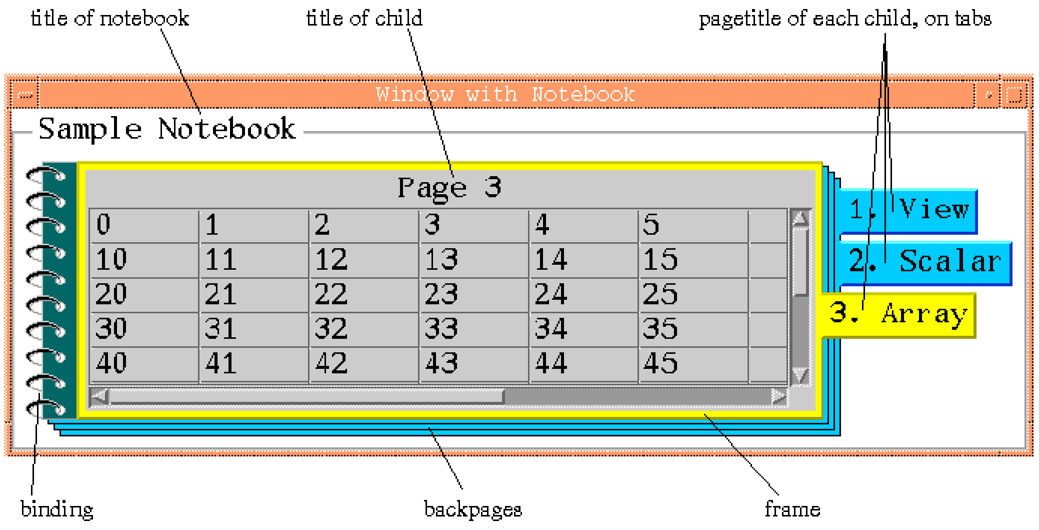
The **backpages** attribute controls the apparent thickness of the notebook, i.e., the number of page edges depicted. It is not related to either the number of pages in the notebook or the number of tabs that are shown.

The **backpagefg** and **backpagebg** attributes control the colors of the tabs and page edges, but not of the titles on the tabs, which obey the **fg** attribute (and use the various **title...** attributes for their text).

Notebooks can be shown in horizontal or vertical orientation, that is, with binding left and tabs right or binding top and tabs bottom. The spiral binding can be shown or not, and its width (diameter) can be specified. The current page (with its tab, if shown) is outlined by a frame, half highlighted and half in shadow, which, especially when thickened, gives it a strange raised and beveled appearance. The background color (**framebg**) and thickness (**framethickness**) of that frame can be specified; 0 thickness makes it vanish. There are margins around an object on its page. The top and bottom margins are controlled by the **marginheight** attribute, the left and right by **marginwidth**.

Visual Representation

```
`page1 is `view   ⊣ page1←80 40⍴" view "
`page2 is `scalar ⊣ page2←"scalar"
page3←⍳10 10
`page3 has (`class;`array; `title;'Page 3')
nb←`page1 `page2 `page3
`nb is `notebook
`nb has (`title;'Sample Notebook';
         `shelltitle;'Window with Notebook')
`page1 has (`pagetitle;"1. View")   ⍝ Insert titles for the tabs,
`page2 has (`pagetitle;"2. Scalar") ⍝ after making the objects
`page3 has (`pagetitle;"3. Array")  ⍝ the children of notebook.
⍝ To make the tabs readable, lighten their background.
`nb has `backpagebg `deepskyblue
⍝ Place the third page at the front.
`nb has (`currentpage;`page3)
`nb has (`bg;`white)
`nb has (`selectedpagebg;`yellow)
show `nb
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the notebook display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are O, ALL, CNFT, CNT, NFT, and TOP.

The attributes that are meaningful for the notebook display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b backpagebg backpagefg backpages backpagethickness be bg bindingwidth blank borderheight borderwidth bound children class clear copy currentpage deiconized descendents doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fg fkeys focus followers followertree font foot framebg framethickness freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal locksize lower mapped marginheight marginwidth naturalsize newshow notify orientation outofcurrentworkspace pagechangecb parent pin preset primary r raise realize recursively refresh reparent request reshow resize resizeable rightto script selectedpagebg selectedpagefg sensitive set settings shadowthickness shell shelltitle show showbinding showpopup showtabs state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

23. The Page Display Class

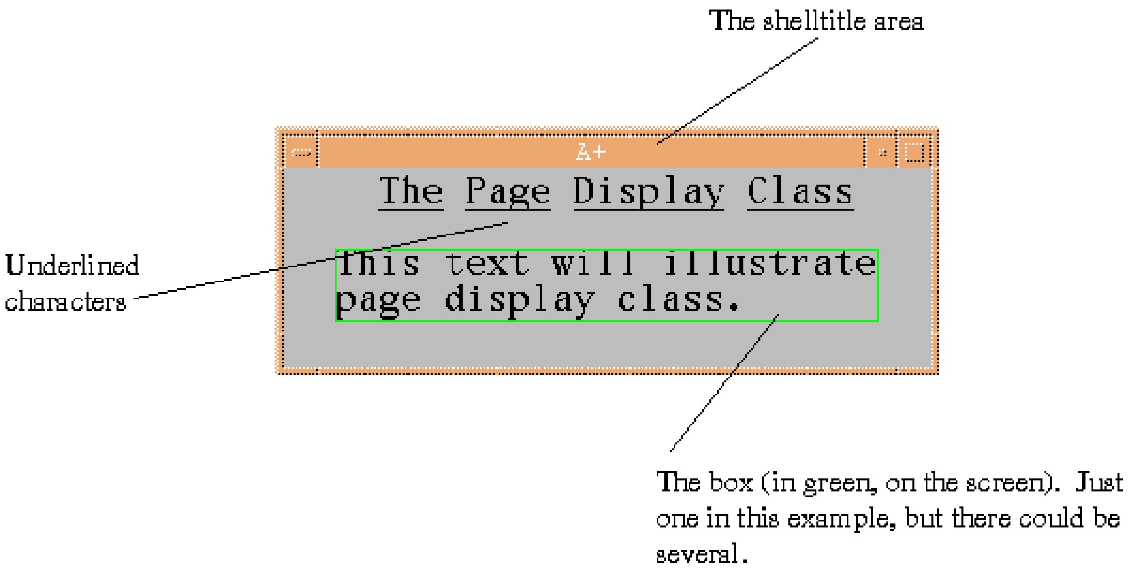
The page display class is designed for displaying page-based information from data services. Objects displayed in this format are character matrices. Normally the matrix is completely displayed; there are no scrollbars, and the **firstrow**, **firstcol**, **cols**, and **rows** attributes do not apply to page objects. This display class is not designed for large objects that require scrollbars, even though scrolling can be provided by putting a page object inside a window (see "The Window Display Class").

Page objects cannot be edited directly. However, keyboard entry can be captured with the **key** and **keysym** attributes, and therefore entry and edit modes can be programmed. For best behavior, the font in which the page is set should be fixed width (monospaced).

Boxes can be drawn at the screen. See "User Interactions with Displays", and specifically "Page Objects".

Visual Representation

```
     r
    The Page Display Class

  This text will illustrate
  page display class.

     ρr
 5  29
     `r is `page
     `r has (`underline;5↑r[,0]≠' ')
     `r has (`box;1 4ρ2 2 2 25)
     `r has (`fg;`green)
     show `r
```


The shelltitle area
A+
Underlined characters
The box (in green, on the screen). Just one in this example, but there could be several.

The Page Display Class

This text will illustrate
page display class.

Colors

The **fg** and **bg** attributes together determine the color of any box the user draws. The foreground and background colors are not set directly, but rather, in the simplest case, by setting the **colormap** attribute to be a 1 by 2 matrix whose elements are those colors (and retaining the default setting of the **color** attribute, Null).

The foreground and background colors can be set separately for each cell. If you want the cells to be shown in k distinct pairs of colors, set the **colormap** attribute to a k by 2 matrix whose rows are the pairs (in symbol form), and set the **color** attribute to an array of row indices of **colormap**. A nonfunctional setting of **color** is treated as if it were reshaped to the shape of the underlying character matrix, and each element determines the colors of the corresponding cell. A functional setting must always be a value with the same shape as the underlying character matrix.

For example, to have stripes, alternating rows of black on grey, blue on peru (brown), and red on tan:

```
c←(ρPg)[1]     ⍝ Row length
Pg .has(`colormap;3 2ρ`black`grey `blue`peru `red`tan;
`color;   (cp0),(cp1),cp2);
```

Page Updates

The cells of a page are redrawn when a change to the corresponding elements of the global variable occurs, and automatically reflect the current settings of the **bold**, **color**, and **underline** attributes for those cells. For any particular cell, whenever one of these attributes is a function, the effect of that attribute on the cell is not reflected until the corresponding element of the global variable is changed.

The blink attribute is somewhat different from the other functional attributes, **bold**, **color**, and **underline**. When it is a function, and when any change to the global variable occurs, the blink function is evaluated for the entire array, not just the part that was changed. Consequently, blinking may be turned on for characters that do not change value.

Key Press Events

The value of the **keysym** attribute is a nested pair of the form `(k;s)`, where the integer `k` is the ASCII code of the character pressed, and `s` is a boolean vector of eight integers indicating which modifier keys were pressed when the key press event occurred. These are:

(**Shift**, **Caps Lock**, **Control**, **mod1**, **mod2**, mod3, mod4, mod5)

where **mod1** is **Meta** and **mod2** is **NumLock**, using the standard A+ profiles. mod3 through mod5 are unused.

There is no cursor shown for the page display class. There is a **cursor** attribute, which is set to the pointer position (row and column) whenever any mouse button is pressed while the pointer is on the page. The programmer can account for the usual cursor with the **blink** attribute. It is up to the programmer to move this "cursor" - i.e., select an appropriate character to blink - based on the key press and mouse button events.

Attributes

A nonfunctional setting of a functional attribute is treated as if it were reshaped to the shape of the underlying character matrix. A functional setting should always be a value with the same shape as the underlying character matrix.

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the page display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are P, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the page display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg blink blinkrate bold bound box boxcolor class clear color colormap cursor deiconized doc done downto dynamic eval evaluate exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is key keysym l leader leftto line linewidth literal lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary r raise rband rbandbox realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto 3down 3up title titlefg titlefont titlejustify 2down 2up underline upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

24. The Password Display Class

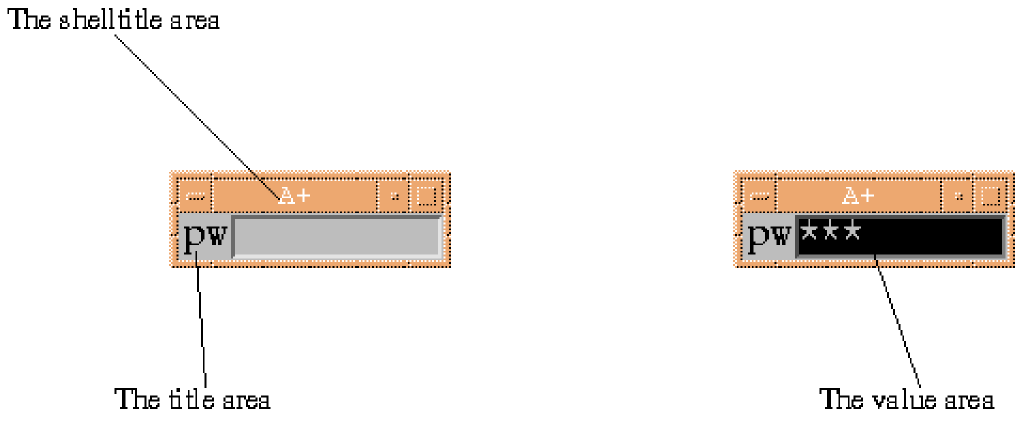
The password display class provides password protection to applications. The password representation has two parts, a title area on the left and a value area on the right. The underlying variable holds a user name. Text is entered in the value area and compared to the login password of that user. Kerberos authentication is supported.

Visual Representation

```
pw←'mahler'
`pw is `password
validate{s;c;v}:↓0⊃`valid of c,v
`pw has (`validate;validate)
show `pw
```

The figure shows this object as it initially appears and what it looks like as a password is being entered. Here, three characters have been entered, masked by asterisks (the value of the **fill** attribute), and the cursor is positioned to the right of the third asterisk. When the user completes entry, the function `validate` is called by the validate event; it displays 1 if the password for mahler has been entered, and 0 otherwise.

The initial state is shown at the left, and the appearance during input is shown at the right:



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the password display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are D, ALL, CNFT, NFT, and TOP. The attributes that are meaningful for the password display class (other than the print... attributes) are:

active ancestors arrowbuttons arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear colors cycle decrement deiconized doc downto dynamic edit editbg editfg editspace eval evaluate exit extent f1-f12 fill fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle increment incurrentworkspace is l leader leftto lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary r raise realize refer refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto valid validate vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

25. The Radio Display Class

The radio class is for radio buttons, which, like the check class, are represented by boolean valued slotfiller variables, but with the additional stipulation that one and only one of the values is 1. A button is marked "on" if the corresponding value is 1 and "off" if it is 0. When a button is "on" it has a sunken appearance and is the color specified by the **fg** attribute, and when "off" it has a raised appearance and is the color specified by the **bg** attribute. A click with the left mouse button on a radio button that is "off" turns the button "on" and causes the button that had been "on" to be turned "off"; the values of the A+ variables are changed accordingly, to 1 for "on" and 0 for "off". Therefore, if a callback function is defined for the underlying variable, it is called whenever a button is selected, with the path argument `p` being equal to the symbolic index of the selected button. Note that even though there are two value changes, a callback function associated with the underlying global variable of a radio button is called only once, and the path argument of the callback function refers to the value that changed from 0 to 1.

The underlying slotfiller variable can of course be changed by program, as long as its new values are boolean and exactly one of them is 1. Thus if `rd` is a radio and

```
    rd←(`a`b`c`d;(1;0;0;0))
```

its setting can be changed by the statement

```
    (1⊃rd)[0 2]←(0;1).
```

An aid is provided by s: if one value is set to 1 by a symbolic-pick assignment, then s sets any other value that is 1 to 0. Hence the change in setting just instanced could also be accomplished by the statement

```
    (`c⊃rd)←1
```

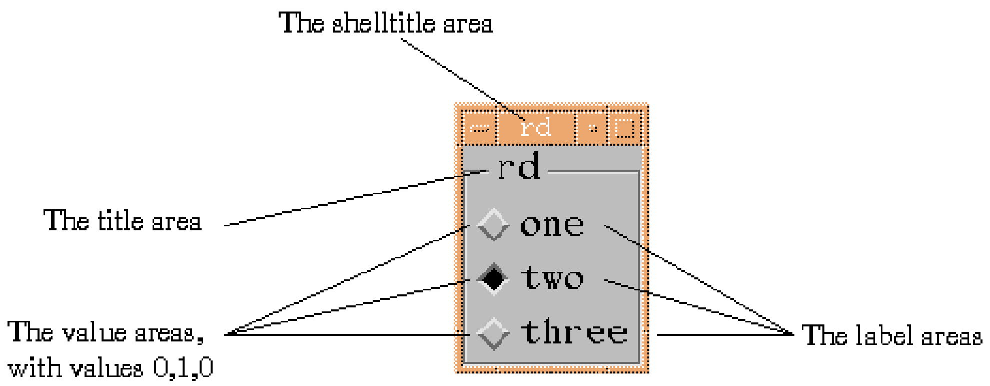
with s providing the necessary `(`a⊃rd)←0` action.

The label areas cannot be edited.

The slot areas that appear in the display and their arrangement can be controlled by the setting of the **geometry** attribute; see "The Geometry Attribute", in "The Slot Display Class". All slot areas specified for display by this attribute are completely displayed; radio displays never have scrollbars.

Visual Representation

```
    rd←(`one`two`three;(0;1;0))
    `rd is `radio
    `rd has (`shelltitle;'rd')
    show `rd
```



Attributes
See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the radio display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are I, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the radio display class (other than the print... attributes) are:

acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound C class clear deiconized doc downto dynamic eval evaluate exit extent f1-12 fg fkeys focus followers followertree font foot freeze geometry fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l label labelfg labelfont leader leftto literal lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary protect protected r R raise realize refresh request resize resizeable rightto script selected sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

26. The Report Display Class

When the **print** attribute is specified for an object, the part of the object that is currently shown on the screen is reproduced in a PostScript (or Encapsulated PostScript) file. The report display class, on the other hand, allows a table that is too large for the screen or too large for the printer paper to be reproduced in a PostScript (or EPS) file, so that it can be printed in its entirety. It also allows breaks within the table, with computations such as subtotals; banners (such as "Internal Use Only"), headers, footers, boxes, frames, and other decorators; monitoring of the printing progress by program; and so on.

The value of a variable bound to the report class is the name of a table object, in symbol form. To produce a print file, specify the report attribute for the report object, as in the example below.

Although *show* accepts the name of a report object without complaint, it ignores the request: reports cannot be shown on the screen. Use Ghostview, Acrobat, or a similar program to preview report output.

Classes That Attributes Apply To
Some of the attributes that control the production of reports apply to the report class, some to tables, and some to table columns.

To reports:

See the list at the end of this chapter.

To tables:

breakfont breakstyle columncontrol columnpagespan columnspacing computepagebreakcb computesizecb currentbreakcolumn fixedreportcolumns framelinewidth frameoffset framestyle headingstyle leading newspapercolumn outputstyle pagebreakcb reportfont reportheadingfont rowcontrol rowpagespan scalefontcb style

To tableFields:

bggrayscale breakcriteriafunc breakfont breakleading breakoffset breakon breakprocessfunc breakprocesson breakstyle breaktext computationmode fggrayscale formatbreakfunc headingbggrayscale headingfggrayscale headingstyle pagebreakon reportfont reportheadingfont style suppressduplicate

Visual Representation

Set up the table:

```
deptы{(3T<"A00"),(3T<"C01"),(4T<"D11"),
          (4T<"D21"),3T<"D31"};
namesы("Haas";"Lucchesi";"Thompson";"Kwan";"Nichols";
       "Quintana";"Adams";"Stern";"Walker";"Yoshimura";
       "Henderson";"Jefferson";"Marino";"Pulaski";
       "Setright";"Smith";"Turner")
amtы{52750 38170 41250 40175 35420 33800,
     30280 36250 20450 28680 35750 222180,
     33760 36170 19100 17750 20120};
`t is `table Э tы`dept`names`amt;
outFunc{s;d;i;p;c;v}:{0Шd}
`dept has (`title;"Department"; `out;outFunc);
`names has (`title;"Name"; `out;outFunc; `space;9);
`amt has (`title;"Salary");
show `t
```

Set up the report:

```
`rep is `report Э repы`t;
func{s;d;i;p;c;v}:{(>dept[0Шi])," Total"}
`dept has (`style;`boxl`boxr`boxt`boxb; `breakon;1;
           `breakstyle;`center`box;
           `breakprocessfunc;(func;);
           `suppressduplicate;1);
`names has (`style;`boxl`boxr`boxt`boxb);
`amt has (`style;`boxl`boxr`boxt`boxb`right;
          `breakprocesson;1; `computationmode; `sum);
`rep has (`margins;1.5; `framestyle;`box;
          `framelinewidth;5; `orientation;`portrait;
          `pagenumbering;0; `file;"simpleReport.ps";
          `header;(`text;
                   ("HEADER 1";"Header 2";"Header 3");
                `fggrayscale;0.25 0.5 0.75;
                `row;0 0 1; `column;0 2 1;
                `justify;`left`right`center;
                   `style;(`outline;;`outline);
                   `leading;10;
                `font;
                      ("Times-Roman-24";"Helvetica-24";
                       "NewCenturySchlbk-Bold-36"));
          `footer;(`text;("Footer 1";"Footer 2");
                   `justify;`right;);
          `banner;(`text;
                   ("For Internal Distribution Only");
                  `mode;`diagonal;
                   `font;"Times-Roman-72";
```

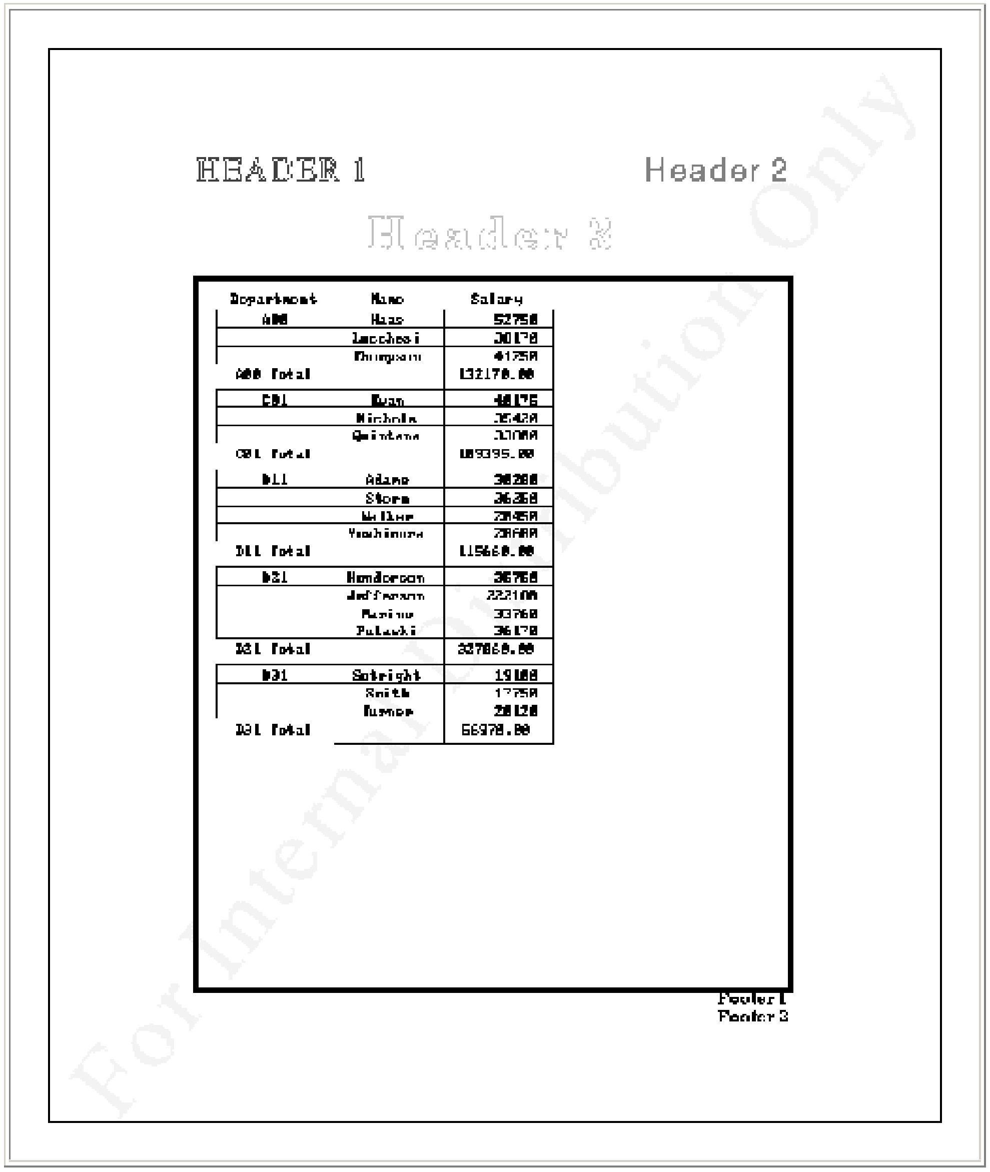
```
                    `xorigin;0; `yorigin;0;
                    `fggrayscale;0.95;
                    `justify;`center));
```

Print it to file:

```
`rep has `report;
```

The result as seen in Ghostview is shown in the figure.

A Sample Report, As Seen Through Ghostview:



HEADER 1 Header 2

Header 3

Department	Name	Salary
A00	Haas	52750
	Lucchessi	38170
	Thompson	41250
A00 Total		132170.00
C01	Kwan	40175
	Nicholls	35420
	Quintana	33800
C01 Total		109395.00
D11	Adams	38280
	Stern	36250
	Walker	23450
	Yoshimura	23680
D11 Total		115660.00
D21	Henderson	36750
	Jefferson	22180
	Marino	33760
	Pulaski	36170
D21 Total		227860.00
D31	Setright	19180
	Smith	17750
	Turner	28120
D31 Total		66970.00

Footer 1
Footer 3

For Internal Distribution Only

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the report display class, as well as lists of colors and fonts. The key in the Table of All Display Attributes that pertains to this class is Rp.

The attributes that are meaningful for the report display class are:

banner bottommargin cancel computepagebreakcb computesizecb delimiter disclaimer disclaimerbottommargin disclaimerfile disclaimerleftmargin disclaimerrightmargin disclaimerrulewidth disclaimertopmargin disclaimerorientation filename footer footeroffset framelinewidth frameoffset framestyle header headeroffset leftmargin margins orientation outputmode pagebreakcb pagecontrol pagecount pagecounttotal pagenumbering pagenumbers pagenumbertext pagesize print rightmargin scalefont scalefontcb state stateself topmargin uniformscaling

27. The Scalar Display Class

Any A+ global variable can be displayed as an object of class scalar. The representation has two parts side-by-side: a title area on the left and a value area on the right. The value area contains an A+ expression that represents the value of the variable, in the sense that the evaluation of the expression is identical to the value of the variable.[1]

The A+ expression in the value area can be edited like text.

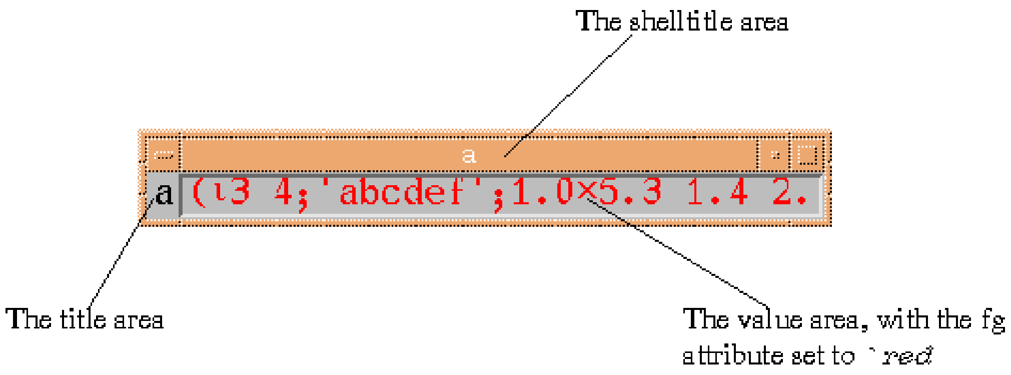
The title area contains the name of the global variable. The width of the title area is always the minimum required to display its text. In particular, all the additional size goes to the value area when the display is resized to be larger. The title can be specified by the **title** attribute. No title area appears when this attribute is specified to be the empty vector.

The title area cannot be edited.

Visual Representation

```
a←(ι3 4;'abcdef'; 5.3 1.4 2.7 0.863)
`a is `scalar
`a has (`shelltitle;'a')
`a has (`fg;'red')
`a has (`space`stars;(29;0))
show `a
```

A Scalar Display:


The shelltitle area
a
a
(ι3 4;'abcdef';1.0×5.3 1.4 2.
The title area
The value area, with the fg attribute set to `red

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the scalar display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are Q, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the scalar display class (other than the print... attributes) are:

active ancestors arrowbuttons arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear colors cycle decrement deiconized doc done downto dynamic edit editbg editfg editspace eval evaluate execute exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle in increment incurrentworkspace is l leader leftto literal lower mapped naturalsize notify out outofcurrentworkspace parent pin preset primary protect protected r raise realize refer refresh request resize resizeable respace rightto script sensitive set settings shadowthickness shell shelltitle show space stars state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

28. The Slot Display Class

The slot class is for the display of any slotfiller variables. The display consists of a title area above a collection of *slot areas*, one for each symbolic index. Each slot area has two areas side-by-side. The one on the left is called the *label area* and, by default, contains the symbolic index of the slot. The one on the right is a *value area* for the corresponding value. The display of each slot area is like that of a scalar object. For example, if

```
sl←(`one `two `three;
    (0 0 2; "abc"; 3 4))
`sl is `slot
`sl has (`shelltitle;'sl';
        `title;'The above example')
show `sl
```

then the display has three *slot areas:* one with label `one` and value `0 0 2;` a second with label `two` and value `abc`; and a third with label `three` and value `3 4`. For convenience, 0 is said to be the *index* of the first slot area, i.e., the one containing `one` and `0 0 2`, 1 the index of the second, and so on.

The value areas can be edited, but the label areas cannot.

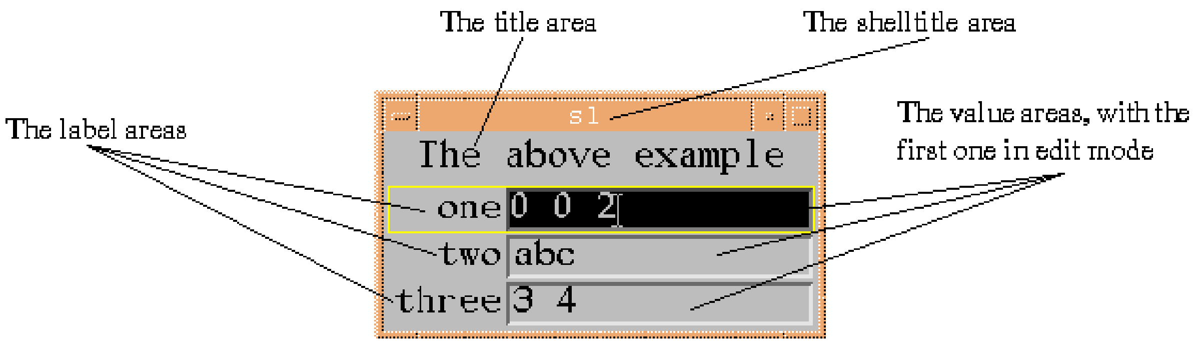
The slot areas that appear in the display and their arrangement can be controlled by the setting of the **geometry** attribute; see "The Geometry Attribute". All slot areas specified for display by this attribute are completely displayed; slotfiller displays never have scrollbars.

The width of the label area in a slot with only one slot area is determined in the same way as the title area of a scalar object. More generally, to determine the width of a particular label area, consider all label areas that line up with it on the left: they all have the same width, which is just large enough to display the longest label among them, except that there is *no* label area for an empty label.

The labels can be specified using the **label** attribute. No label area appears when the label for that area is specified to be an empty vector. A vector of n blanks produces a label area filled with n blanks. As with a scalar object, all additional size goes to the value areas when the display is resized larger.

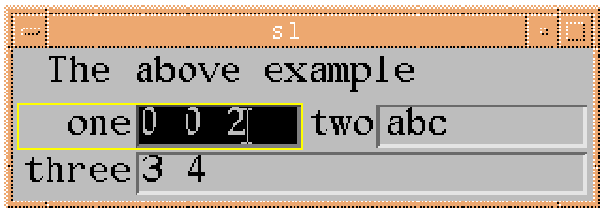
Visual Representation

`s1` **With Default Geometry:**



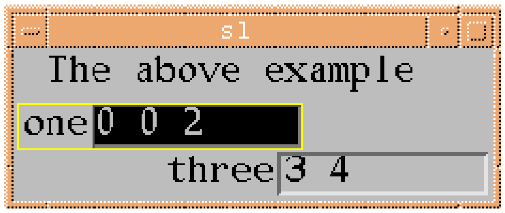
`s1` **With Geometry Changed To** `2 1`**:**

```
`s1 has (`geometry;2 1)
```



`s1` **With Matrix Geometry:**

```
`s1 has (`geometry;2 3⍴0 0 ¯1 ¯1 2 2)
```



Attributes

The Geometry Attribute

The **geometry** attribute controls the arrangement of an object of class slot on the screen. The value can be an integer, a vector, or a matrix.

Geometry Specified as an Integer

An integer specifies the number of columns in the arrangement, with the convention that -2 means a horizontal arrangement (and so is equivalent to the number of slot areas), and -1 means vertical (and so is equivalent to 1). Horizontal and vertical arrangements can also be specified by `` `horizontal `` and `` `vertical ``, respectively.

Geometry Specified as a Vector

The elements of a vector specify the number of slotfiller elements in the individual rows, the length of the vector being the number of rows.

In both the integer and vector cases, the slot areas fill the first row in order, then the second, and so on, until the elements are exhausted.

Geometry Specified as a Matrix

In the case of a matrix the elements must be from either (but not both) of:

- the list of integers `¯1`, `ιn`, where *n* is the number of slots;
- the list of symbols `` ` ``, `0⊃s`, where *s* is the slotfiller array (and `0⊃s` is therefore its list of symbols).

The display of the object is subdivided conceptually into blocks with the `(i;j)`th element of the matrix corresponding to the `(i;j)`th block; the upper left corner of the display is the origin. A -1 or empty symbol (`` ` ``) in the matrix indicates that the corresponding block is blank. A nonnegative integer is the index of the slot area (as described above) occupying that block, while a nonempty symbol is the symbolic index associated with that slot area. Repeated integers or symbols must form a submatrix. Not all integers or symbols from their respective lists need be used. See "Functional Attributes".

The R and C Attributes

The arrangement of the slot areas can be understood in terms of virtual rows and virtual columns, in the same way as objects in a layout. It is possible to constrain the virtual rows or columns to be all the same size by setting the **R** or **C** attribute to 1, respectively.

List of Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the slot display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are S, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the slot display class (other than the print... attributes) are:

active ancestors arrowbuttons arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound C class clear colors cycle decrement deiconized doc done downto dynamic edit editbg editfg editspace eval evaluate execute exit extent fg f1-f12 fkeys focus followers followertree font foot freeze fullscreen geometry h H has head hide hl hlthickness icon iconic iconized icontitle in increment incurrentworkspace is l label labelfg labelfont leader leftto literal lower mapped naturalsize notify out outofcurrentworkspace parent pin preset primary protect protected r R raise realize refer refresh request resize resizeable respace rightto script selected sensitive set settings shadowthickness shell shelltitle show space stars state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

29. The Table Display Class

This container is somewhat different from a window or layout, in that all children are of the class tableField and, if necessary, are re-bound to that class when a table is bound. If a table object is altered so as to omit some of its original children, the usual specification rule applies: those children remain on the screen, but not as part of the table object. Instead, they have independent displays, in the formats of their default display classes. In particular, they do not remain bound to the class tableField. An object can be bound to the tableField class only by A+ and only as a child of a table object. The children of a table object must be unique.

A variable can be bound to the table display class if it is a scalar or vector consisting of object symbols. The child objects, which are also called *fields* of the table, can be vectors or simple character matrices; they cannot appear more than once in the table variable. The display consists of a title area and a value area. The value area has a title bar on top that holds the names of the fields, while the area below consists of delineated cells, one column for each field. The fields appear on the screen in the same left to right order as their symbols appear in the variable that is bound to the table class. The top cell of each field is the first element of the corresponding variable; in the case of a character matrix field, the cells correspond to rows of the matrix and the top cell is the first row.

If a table requires more than just a few cells, then by default only a subtable is presented on the screen, and the display is provided with scrollbars. If the table fields have different numbers of elements, then whether or not a vertical scrollbar is used depends on the number of elements in the first field (i.e., `t[0]` for a table `t`). The particular subarray that appears in the value area can be controlled by the scrollbars and settings of the attributes **firstrow**, **firstcol**, **rows** and **cols**. Neither by scrollbars nor by **firstrow** is it possible to display rows for which the first field has no value: the number of elements in the first field completely controls the rows that can be displayed. If the table fields have different numbers of elements and you want to allow the display of all elements, the first field must have at least as many elements as any other field. For a table, the attribute **firstcol** refers to the leftmost field shown, and **cols** refers to the number of fields.

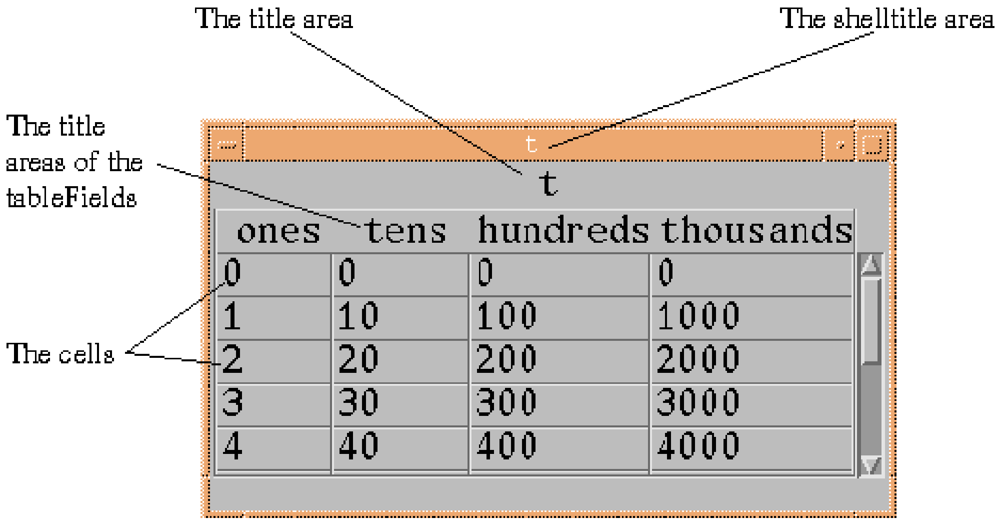
Be aware of one situation that can occur. Suppose a visible field is dependent upon the first field and another field, `c:a+b`, say. If a strand assignment changing their lengths is made to `a` and `b`, what happens depends upon the order in which the individual assignments are made. If the assignment to the first field is made last, there is no problem. Otherwise, the assignment to the first field causes the table to use its new length to determine the range of the vertical scrollbar and that involves materializing the field "too early", when `a` and `b` have unequal lengths, causing a length error in the evaluation of `c`. One workaround, as implied above, is to see that the first field is set last. Another is to unmap the table, perform the assignment, and then map it again.

Visual Representation

```
thousandsы10×hundredsы10×tensы10×onesыЙ10
tы`ones`tens`hundreds`thousands
```

```
`t is `table
`t has (`shelltitle;'t')
show `t
```

A Table of Numeric Fields:



ones	tens	hundreds	thousands
0	0	0	0
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000

Attributes
See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the table and tableField display classes, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to the table class are T, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the table display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg blank bound breakfont breakstyle children class clear col colors cols colsep columncontrol columnpagespan columnresize columnspacing copy currentbreakcolumn cycle deiconized delete descendents doc downto dragdrop dynamic edit editbg editfg editspace eval evaluate execute exit extent f1-f12 familytree fg field fields firstcol firstfield firstrow fixedfields fixedreportcolumns fkeys focus followers followertree font foot framelinewidth frameoffset framestyle freeze fullscreen h H has head headingstyle hide hl hlthickness hscrollsize hscrollwith icon iconic iconized icontitle in incurrentworkspace index insertabove insertbelow is l leader leading leftto literal lower mapped na naturalsize newshow newspapercolumn notify out outofcurrentworkspace outputstyle parent pin preset primary protect protected r raise realize recursively refer refresh reparent reportfont reportheadingfont reporttotalfont reporttotalleading reporttotalon reporttotalstyle request reshow resize resizeable respace rightto row rowbg rowcontrol rowpagespan rows rowsep script scrollbg scrollsize select selectbg selected selectedfield selectfield selectionmode sensitive set setcol setfirstcol setfirstrow setrow settings shadowthickness shell shelltitle show size space stars state stateself style syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows vscrollsize vscrollwith w W ws x X xs y Y ys yx YX yxs

The keys in the Table of All Display Attributes that pertain to the tableField display class are tF and ALL.

The attributes that are meaningful for the tableField display class are:

active ancestors arrowdown arrowkeys arrowleft arrowright arrowup bg bggrayscale blank bound breakbggrayscale breakcriteriafunc breakfggrayscale breakfont breakleading breakoffset breakon breakprocessfunc breakprocesson breakstyle breaktext class colors columnalignment computationmode copy cycle doc done editspace eval evaluate execute f1-f12 fg fggrayscale fkeys font formatbreakfunc has headingbggrayscale headingfggrayscale headingstyle in is literal mapped na out pagebreakon parent preset protect protected reportfont reportheadingfont respace set settings shell space stars state stateself style suppressduplicate title titlefg titlefont verify

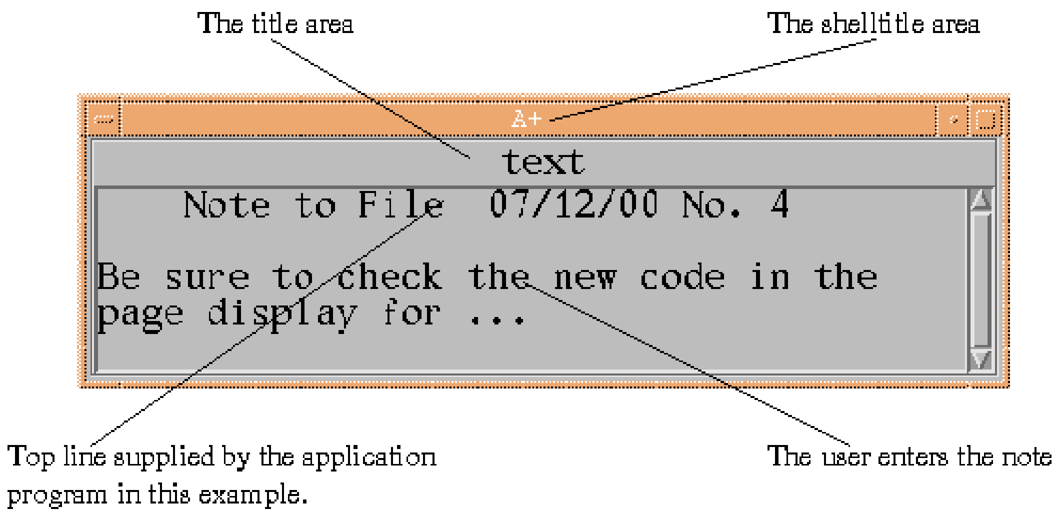
30. The Text Display Class

The text display class provides a general means for text entry. It can be used wherever a note pad is needed in applications. Any character vector can be displayed in this class. Once an object of this class has focus, editing can begin. New lines are created during text entry by pressing the **Enter** key, which then appears in the character vector as the character `` `char∨10 `` (newline); this is the only control character that should be used in the text. APL characters can be entered.

There are no user actions that begin and end edit mode; consequently, when the text is entered the underlying character vector may not be automatically updated (see the **save** attribute). However, even if the character vector is not updated, the current text on the screen can be obtained as the value of the **buffer** attribute.

Visual Representation

```
t←"     Note to File   07/12/00 No. 4"
                  ⍝ Could start with the empty vector
`t has (`shelltitle `title;('text';'text'))
show `t is `text
```


The title area
The shelltitle area
A+
Top line supplied by the application program in this example.
The user enters the note

text

```
    Note to File  07/12/00 No. 4

Be sure to check the new code in the
page display for ...
```

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the text display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are X, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the text display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound buffer class clear cols cursor deiconized doc downto dynamic eval evaluate exit extent f1-12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary r raise realize refresh request resize resizeable rightto rows save script sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

31. The Tree Display Class

The tree display class is for displaying nested slotfillers as treelike data structures. The symbolic indices are shown; the values at the leaves are omitted, since the primary intended use is to display layouts in tree form. The number of children per node is limited to 128.

If you want to have values at the leaves shown, you must doctor the nested slotfiller, if possible, to turn the values into symbolic indices in slotfillers with arbitrary artificial values. For example, if you want the numeric values shown for the slotfiller

```
t←(`a;<(`b`c`d;(7;(`e`f;(8;9));10)))
```

you could construct something like

```
t1←(`a;<
    (`b`c`d;
     ((`7;<0);(`e`f;((`8;<0);(`9;<0)));(`10;<0))))
```

and show `t1` as a tree representing `t`.

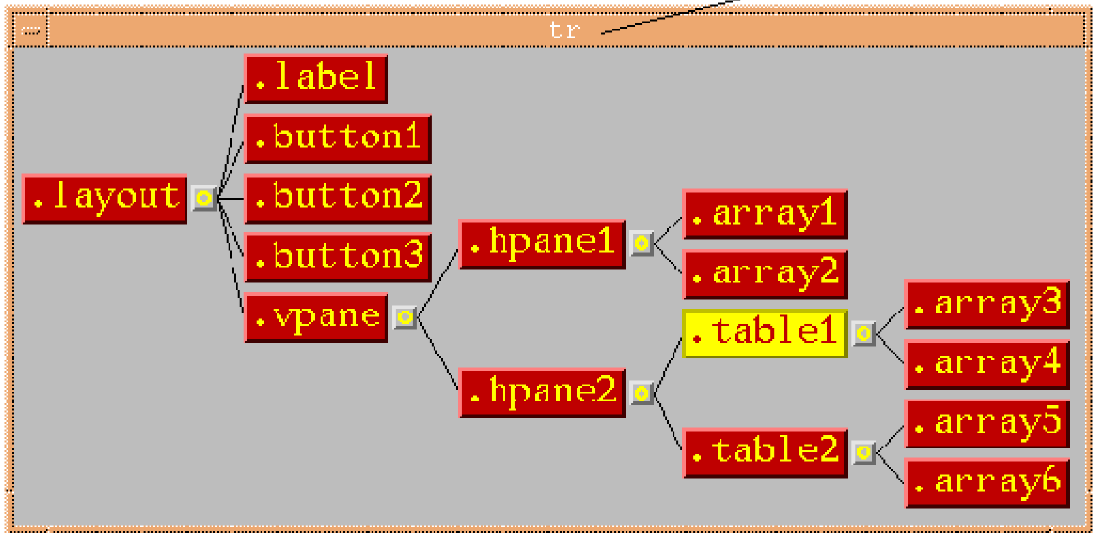
Visual Representation

A complicated layout can be represented as a nested slotfiller, and then displayed as a tree for a convenient view of its structure. In fact, the appropriate nested slotfiller representation is produced by the **familytree** attribute. For example, if `ly` is a layout, then its organization may look as follows:

```
     tr←0⊃`familytree of `ly
     show `tr is `tree
```

A Layout Represented as a Tree:

The shelltitle area with a default shelltitle



In this example, the ".table1" button has been selected by clicking on it.

User Interactions

If a callback function for the object is defined, it is called when:

- a selected item is double-clicked on; or,
- **Enter** is pressed when an item is selected (see the tree section in the "User Interactions with Displays" chapter).

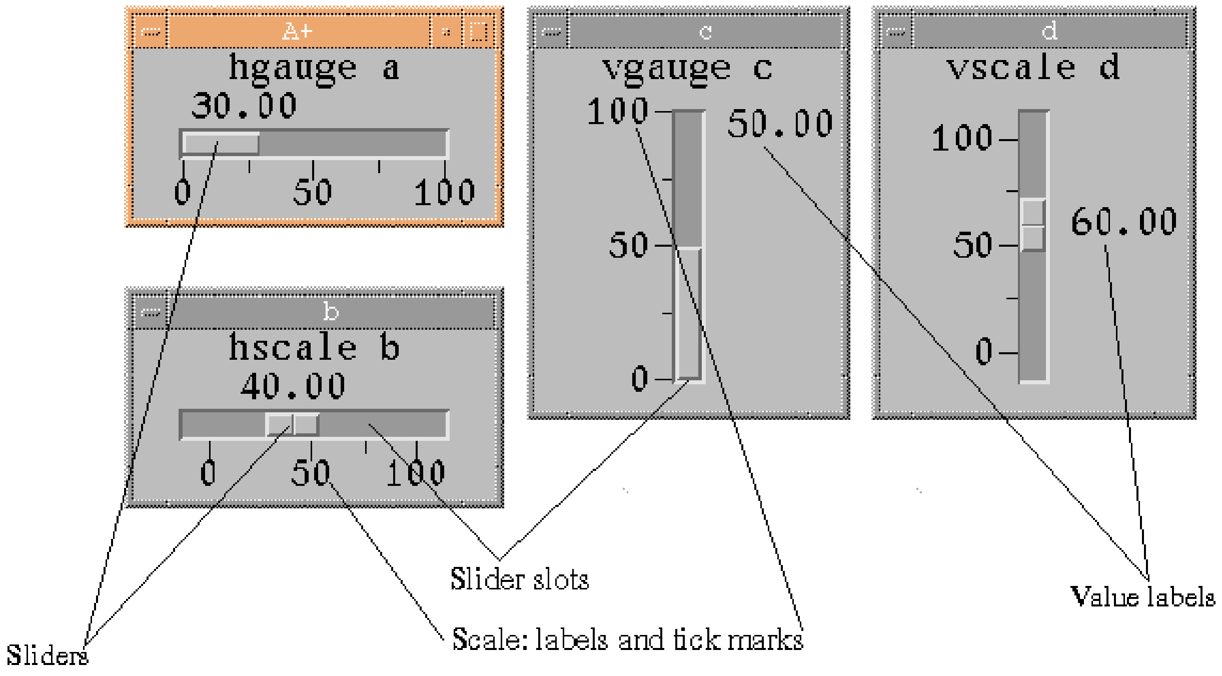
In such a call, the path variable (see "Callback Functions") is the symbol vector path to the node.
Attributes
See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the tree display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are R, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the tree display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness horizontalspace icon iconic iconized icontitle incurrentworkspace is l label labelfg labelfont leader leftto linecolor literal lower mapped naturalsize nodebg nodefg notify orientation outofcurrentworkspace parent pin preset primary r raise realize refer refresh request resize resizeable rightto script select selected selectednodebg selectednodefg sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify verticalspace vrow vrows w W ws x X xs y Y ys yx YX yxs

32. The Vgauge Display Class

The vgauge class is like the hgauge class, with the obvious differences: **sliderwidth** is respected and **sliderheight** is ignored, the justification attributes have different effects, and so on.



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the vgauge display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are vG, ALL, CNFT, GS, NFT, and TOP.

The attributes that are meaningful for the vgauge display class (other than the print... attributes) are:

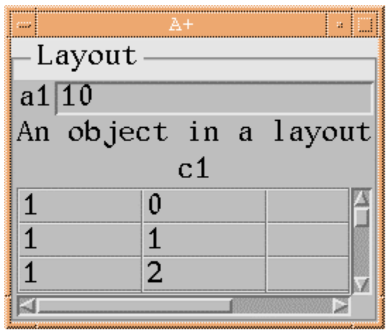
active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate execute exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle inc incurrentworkspace is l labelfg labelfont labelinc labeljustify labelout[1] leader leftto literal lower majorticksize mapped max maxtitle maxtitlefg maxtitlefont maxtitlejustify min minortickcount minorticksize mintitle mintitlefg mintitlefont mintitlejustify naturalsize notify out[1] outofcurrentworkspace pageinc parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show sliderbg sliderheight sliderwidth state stateself subtitle subtitlefg subtitlefont subtitlejustify syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto valuefg valuefont valuejustify vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

1. Note that labelout is used to format the scale labels and out to format the value label.

33. The Vgrid Display Class

The vgrid class is like the hgrid class, except that the default arrangement is vertical.

Examples of the Layout and Grid Classes:

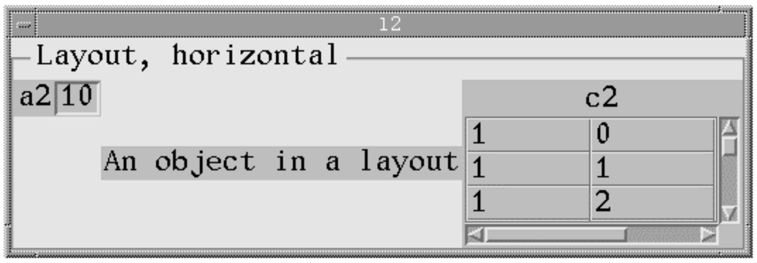


a1 10

An object in a layout

c1

1	0	
1	1	
1	2	

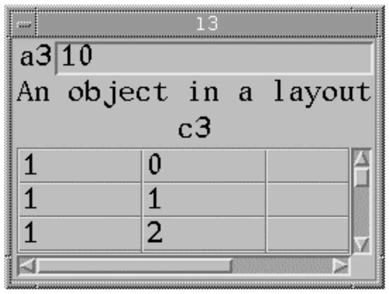


a2 10

An object in a layout

c2

1	0
1	1
1	2

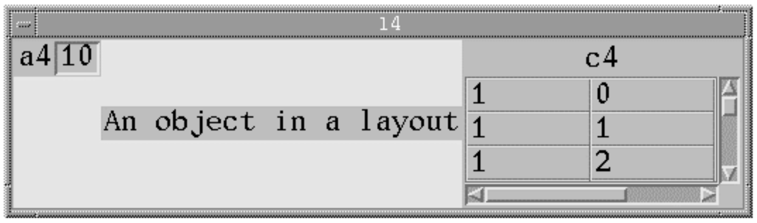


a3 10

An object in a layout

c3

1	0	
1	1	
1	2	

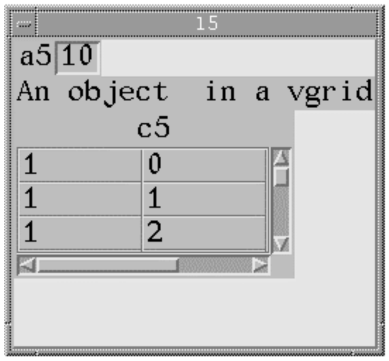


a4 10

An object in a layout

c4

1	0
1	1
1	2

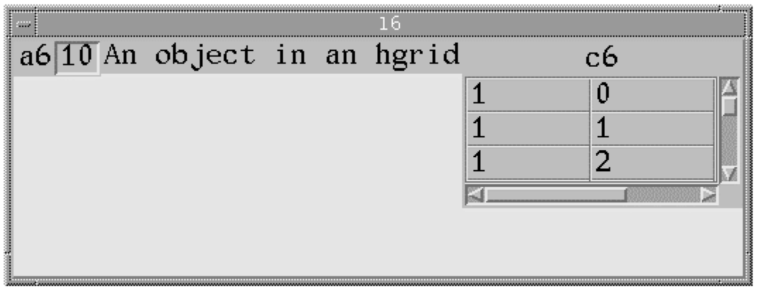


a5 10

An object in a vgrid

c5

1	0
1	1
1	2



a6 10 An object in an hgrid

c6

1	0
1	1
1	2

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the vgrid display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are vR, ALL, CNFT, CNT, NFT, and TOP. The attributes that are meaningful for the vgrid display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b be bg bound build children class clear descendents deiconized doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fg fkeys focus followers followertree font foot freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lower mapped naturalsize newshow notify outofcurrentworkspace parent pin preset primary r raise realize recursively refresh reparent request reshow resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself structure syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

34. The View Display Class

The view class is for simple character matrices that are somewhat larger than labels. It is used, for example, to display reports. If the matrix is large, then by default only a subarray is presented on the screen, and the display is provided with scrollbars. Users control the particular subarray that appears in the value area with the scrollbars; programmers use settings of the attributes **firstrow**, **firstcol**, **rows** and **cols**.

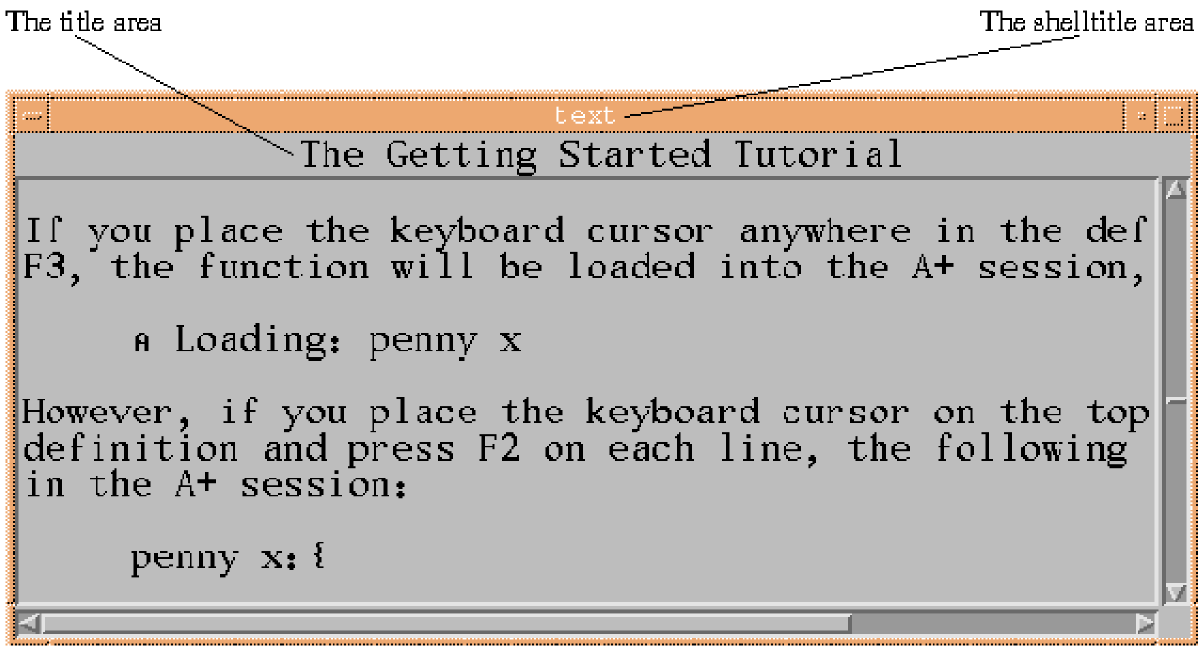
It is recommended that you avoid the use of matrices that contain tabs, backspaces, linefeeds, etc., as these can sometimes produce anomolous results. In general, avoid ASCII control-type characters in any array to be bound to the view class.

Views cannot be edited. Only fixed-width (monospaced) fonts should be used for horizontal scrolling.

Visual Representation

```
t←sys.readmat 'GettingStarted'
`t is `view
`t has (`shelltitle;'text')
`t has (`title;'The Getting Started Tutorial')
show `t
```

A Display of Two-Dimensional Text:



```
The Getting Started Tutorial

If you place the keyboard cursor anywhere in the def
F3, the function will be loaded into the A+ session,

     ⍝ Loading: penny x

However, if you place the keyboard cursor on the top
definition and press F2 on each line, the following
in the A+ session:

     penny x: {
```

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the view display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are V, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the view display class (other than the print... attributes) are:

acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear col colors cols colsep cycle deiconized doc downto dynamic eval evaluate exit extent f1-f12 fg firstcol firstrow fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness hscrollsize hscrollwith icon iconic iconized icontitle incurrentworkspace index is l leader leftto literal lower mapped naturalsize notify outofcurrentworkspace parent pin preset primary r raise realize refer refresh request resize resizeable rightto row rowbg rows rowsep script scrollbg scrollsize select selectbg selected selectionmode sensitive set setcol setfirstcol setfirstrow setrow settings shadowthickness shell shelltitle show size state stateself syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols verify vrow vrows vscrollsize vscrollwith w W ws x X xs y Y ys yx YX yxs

35. The Vmenu Display Class

The vmenu and hmenu display classes are for representing nested slotfillers as cascade menus. When vmenu is used, the top-level menu is laid out vertically, whereas hmenu gives a horizontal layout. All submenus are vertical in both cases.

A slotfiller consists of two parts, the symbolic indices and the values. A nested slotfiller is one whose values are also slotfillers. The values within the slotfiller values can also be slotfillers, and so on; slotfillers can be nested to any level. In terms of the menu display classes, the symbols of a slotfiller bound to either menu class correspond to the items in the top level of a menu. If a value is also a slotfiller, then its symbols correspond to a submenu, and so on.

Menus cannot be edited.

A callback function must be defined for the slotfiller if any actions are to be taken in response to selection of menu items. When a callback occurs, only `p⊃d` appears in the `d` position.

Selecting an item does not cause a preset callback to fire.

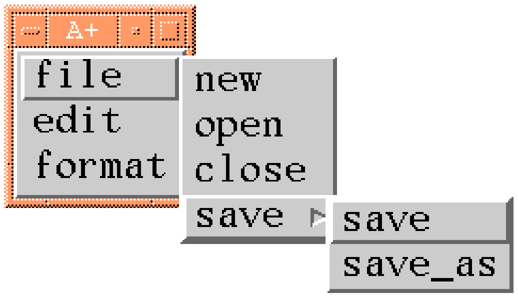
See "The Hmenu Display Class" for details of the horizontal menu.

Visual Representation

```
menu←s.rsf{
   ⍝ s.rsf converts a recursive association list to a recursive slotfiller
       (`file;(
                  `new;       newfn;
                  `open;      openfn;
                  `close;     closefn;
                  `save;(
                              `save;      savefn;
                              `save_as;   save_asfn));
       `edit;(
                  `undo;      undofn;
                  `cut;       cutfn;
                  `copy;      copyfn;
                  `paste;     pastefn);
       `format;(
                  `font;(
                              `kaplgallant;;
                              `courier;);
                  `size;      sizelist;
                  `style;     stylelist)
  )
      }
```

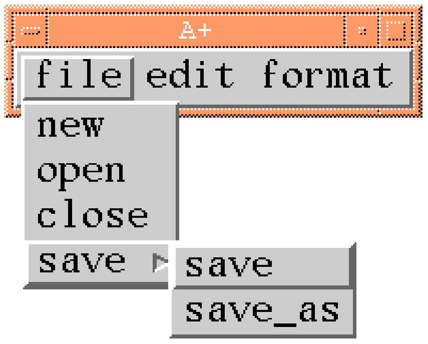
vmenu: Vertical Cascade Menu:

```
show `menu is `vmenu
```



hmenu: Horizontal Cascade Menu:

```
show `menu is `hmenu
```



Attributes
See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the vmenu display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are vM, ALL, CNFT, NFT, and TOP.

The attributes that are meaningful for the vmenu display class (other than the print... attributes) are:

acceptfocus active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic eval evaluate exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lower mapped mnemonics naturalsize notify outofcurrentworkspace parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings script shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto titlejustify upto vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

36. The Vpane Display Class

The vpane and hpane display classes are special layout classes that have movable dividers, or sashes, between the layout children. The user can control the proportion of the layout that each child occupies simply by moving the dividers. Various layout-constraint attributes, such as **position** and **build**, are disregarded. The children of a vpane layout should be arranged vertically, and will have horizontal dividers between them.

To move a divider, simply place the mouse cursor on it, press and hold the left mouse button, and then slide the divider up and down. A small button is provided on the right end of each divider as a convenient place to locate the cursor.

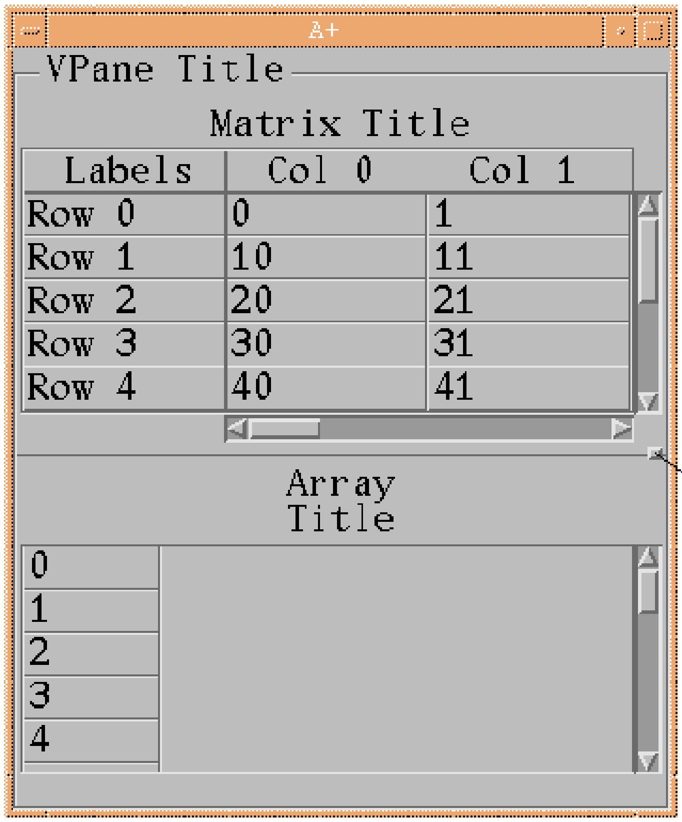
The dividers of a vpane layout always extend from one side of the layout to the other, so care must be taken when using vpane layouts with more than one column. In a vpane layout of the form `2 2ρ`a`b`a`c`, for example, the divider between objects *b* and *c* cuts through object *a*. If several objects line up on a divider boundary, then movement of the divider controls the display sizes of these objects. If all the objects line up on divider boundaries, then movement of the dividers controls the display sizes of entire rows of objects.

There is no direct way for a program to determine where a user has positioned a divider in a vpane layout. Sometimes there are indirect ways to approximate the position, such as by checking the values of the **rows** and **cols** attributes of a matrix to determine how many rows and columns are currently shown.

Compare VPane with "The Hpane Display Class" for a visual representation of a similar object.

Visual Representation

```
m←⍳10 10
`m has (`class;`matrix;  `title;"Matrix Title";
        `label;("Labels";
                (<"Row"),¨⍕¨⍳10;
                (<"Col"),¨⍕¨⍳10));
a←⍳20
`a has (`class;`array;  `title;("Array";"Title"));
h←1 2⍴`m`a
`h has (`class;`vpane;  `title;"VPane Title";
        `shelltitle;"Shell Title"; `show;1);
```



Labels	Col 0	Col 1
Row 0	0	1
Row 1	10	11
Row 2	20	21
Row 3	30	31
Row 4	40	41

0
1
2
3
4

Attributes

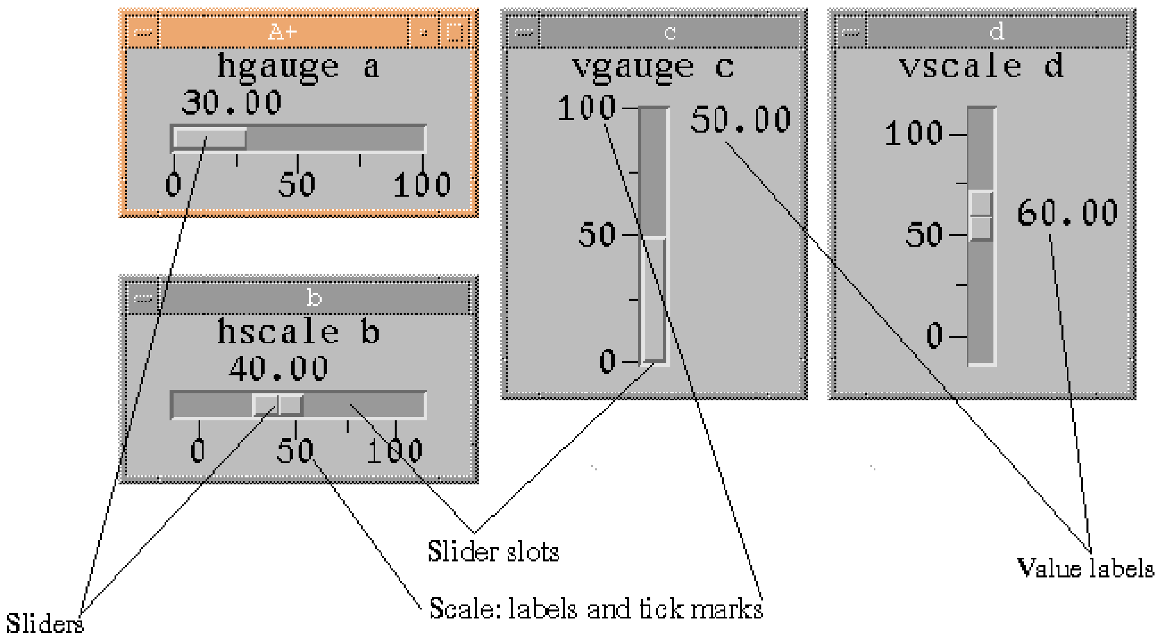
See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the vpane display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are vP, ALL, CNFT, CNT, NFT, and TOP.

The attributes that are meaningful for the vpane display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b be bg bound C children class clear constraints deiconized descendents doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fg font fkeys focus followers followertree foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto literal lockposition locksize lower mapped naturalsize newshow notify outofcurrentworkspace parent pin preset primary r R raise realize recursively refresh reparent request reshow resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show state stateself structure syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto vcol vcols vcolspace verify vrow vrows vrowspace w W ws x X xs y Y ys yx YX yxs

37. The Vscale Display Class

The vscale class is like the hscale class, with a few obvious differences, such as the effects of the justification attributes. See the description of the hgauge class. Also see "Hscale and Vscale Objects", in "User Interactions with Displays".



Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the vscale display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are vS, ALL, CNFT, GS, NFT, and TOP. The attributes that are meaningful for the vscale display class (other than the print... attributes) are: active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound class clear deiconized doc downto dynamic edit editbg editfg editspace eval evaluate execute exit extent f1-f12 fg fkeys focus followers followertree font foot freeze fullscreen H h has head hide hl hlthickness icon iconic iconized icontitle in inc incurrentworkspace is l labelfg labelfont labelinc labeljustify labelout[1] leader leftto literal lower majorticksize mapped max maxtitle maxtitlefg maxtitlefont maxtitlejustify min minortickcount minorticksize mintitle mintitlefg mintitlefont mintitlejustify naturalsize notify out[1] outofcurrentworkspace pageinc parent pin preset primary r raise realize refresh request resize resizeable rightto script sensitive set settings shadowthickness shell shelltitle show sliderbg sliderheight sliderwidth state stateself subtitle subtitlefg subtitlefont subtitlejustify syncshow t tabfrom tablist tabto title titlefg titlefont titlejustify upto valuefg valuefont valuejustify vcol vcols verify vrow vrows W w ws X x xs Y y ys YX yx yxs

1. Note that labelout is used to format the scale labels and out to format the value label.

38. The Window Display Class

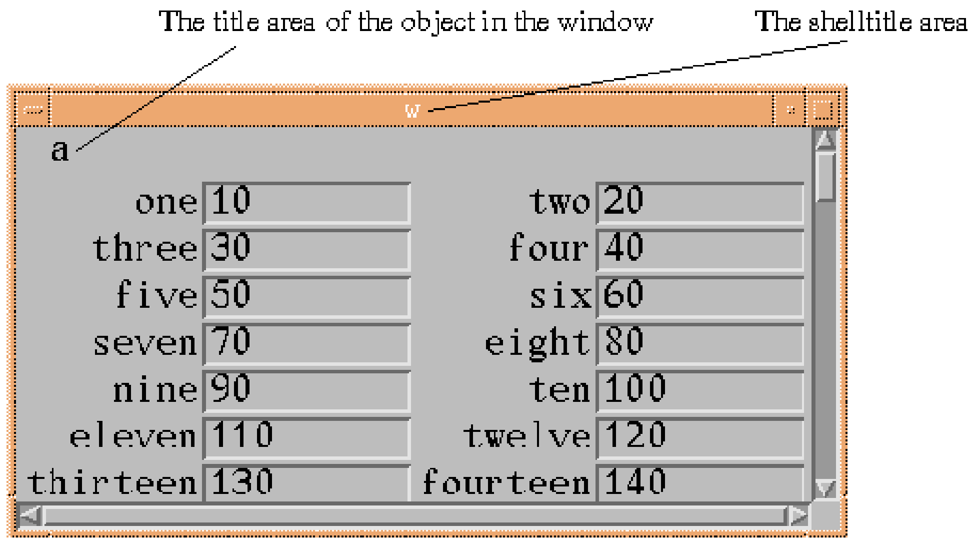
A window contains only one object, and has the effect of putting scrollbars on the display of that object. It is therefore useful in displaying large objects, such as a slotfiller array with many slots. The object in a window can be of any display class.

Note that both vertical and horizontal scrollbars will always be shown, even if one of them is not needed - or indeed if neither is needed.

Visual Representation

```
    sym←⊃⊥¨⍕¨⍳100
    val←(<@0) 10×1+⍳100
    a←(sym;val)
    `a is `slot
    labs←('one';'two';'three';...)
    `a has (`geometry `label;(2;labs))
    w←`a
    `w is `window
    `w has (`shelltitle;'w')
    show `w
```

A Large Slotfiller in a Window:



a

one	10	two	20
three	30	four	40
five	50	six	60
seven	70	eight	80
nine	90	ten	100
eleven	110	twelve	120
thirteen	130	fourteen	140

Attributes

See the "Display Attributes" chapter for details concerning all the attributes that apply to objects in the window display class, as well as lists of colors and fonts. The keys in the Table of All Display Attributes that pertain to this class are W, ALL, CNFT, NFT, and TOP. The attributes that are meaningful for the window display class (other than the print... attributes) are:

active ancestors arrowdown arrowkeys arrowleft arrowlist arrowright arrowup at atsector b bg bound children class clear deiconized descendents doc downto dynamic eval evaluate exit extent extents f1-f12 familytree fkeys focus followers followertree foot freeze fullscreen h H has head hide hl hlthickness icon iconic iconized icontitle incurrentworkspace is l leader leftto lower mapped naturalsize newshow notify outofcurrentworkspace parent pin preset primary r raise realize recursively refresh reparent request reshow resize resizeable rightto script scrollbg scrollsize sensitive set settings shadowthickness shell shelltitle show state stateself syncshow t tabfrom tablist tabto upto title titlefg titlefont titlejustify vcol vcols verify vrow vrows w W ws x X xs y Y ys yx YX yxs

39. User Interactions with Displays

This chapter describes screen interactions in A+ applications from the user's point of view. These interactions are covered from an application programmer's point of view in "Introduction to Screen Management", "Display Attributes", "Attributes with Callbacks", and some of the chapters devoted to individual display classes - e.g., in a section of the graph chapter. Much of the behavior discussed in this chapter is due to the window manager of the underlying system.

Many screen interactions - repositioning objects, for instance - are with OLVWM or OLWM, rather than with A+, so some aspects of this interaction are discussed here; see "The Window Manager", below. Windows for A+ applications contain buttons, menus, tables, etc., all with various forms of user interactions, which are discussed in "The A+ Screen Manager". Not all interactions can be addressed in a general document like this one. For example, applications may use any of the function keys **F1** through **F12** for any purpose, but there is no default use of these keys that can be discussed here. For readers familiar with window-based applications, a fast path through this chapter is the first two sections of "The Window Manager" and the sections of "The A+ Screen Manager" up to "Interactions Peculiar To Individual Display Classes".

The Mouse and Pointer

The *mouse* has three buttons, which in this manual are called simply the left button, middle button, and right button. (When the mouse has only two buttons, the middle button is simulated by using the other two buttons simultaneously.) Depressing a button and releasing it is called *clicking*. Clicking twice in rapid succession is called double clicking. For mouse buttons only, *pressing* a button always means holding the button down until you are told to release it. In this chapter, *button* means either a mouse button or a button object on the screen.

Associated with the mouse is a *pointer*, which can appear anywhere on a workstation screen. It may take several forms. When it is in the shape of an arrow, it refers to the spot on the screen that is at the point of the arrow. When it is in the shape of a target, you can resize a window. When it is in the shape of a clock or stopwatch, the system is busy. (See the `s.BUSY` and `s.CLOCK` parameters to change the appearance of the pointer.) Usually key and button actions that occur when the clock is shown are stored and acted upon as soon as the system is not busy, but sometimes they are ignored.

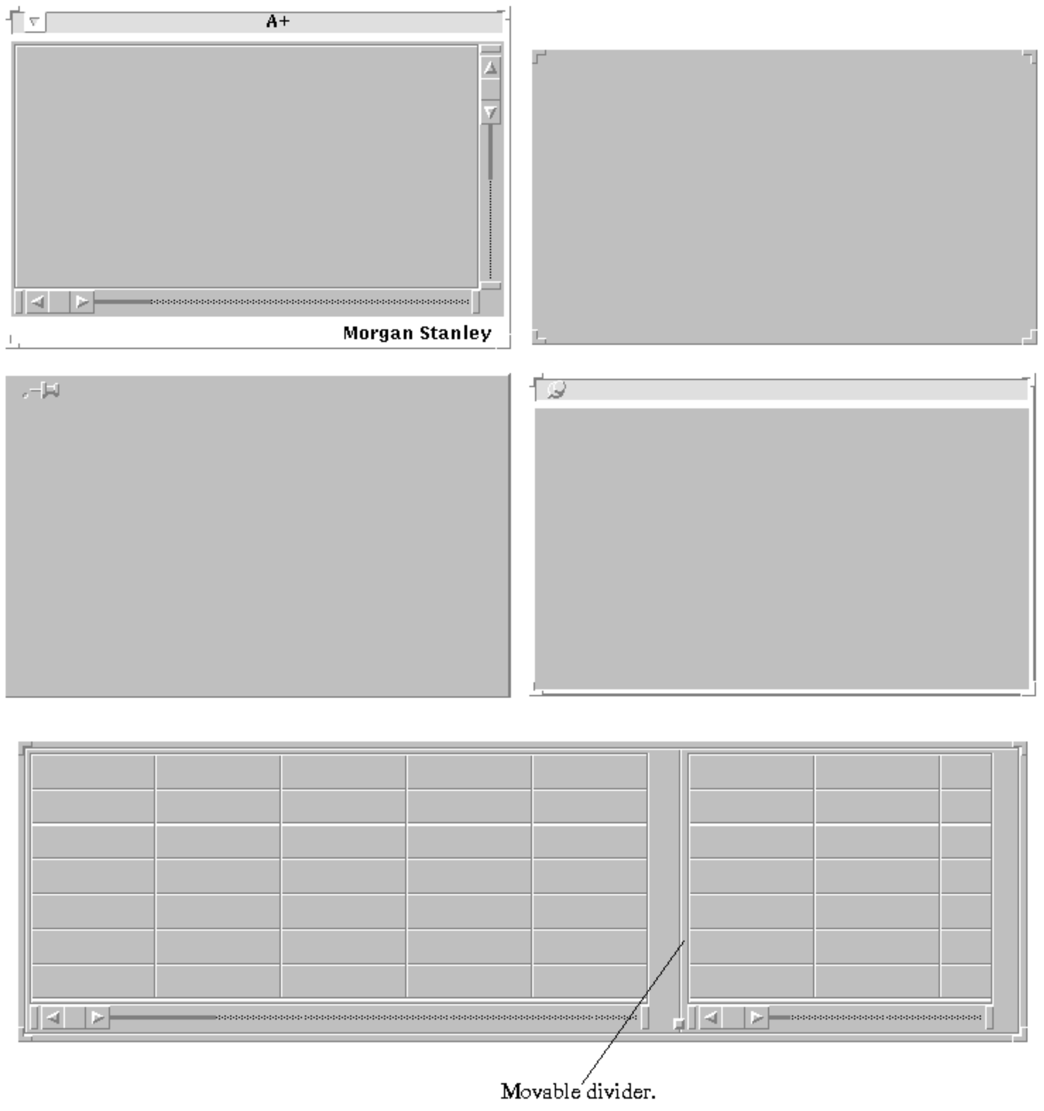
To click or press on an object, or when in some area, means to take that action when the pointer is in the designated place. In particular, to press a button pictured on the screen means to place the point of the pointer on the button and to click or press a mouse button - usually, to click the left button.

The Window Manager

Windows

The screen can contain several windows, which appear as delineated rectangular areas. The windows in the first figure illustrate many of the variations that you may see in A+ applications. In A+, one speaks of top-level windows and popups. Popup windows have pins, as illustrated by the middle two windows in the figure; all other windows in shown there are top-level windows.

Samples of Various A+ Windows:



Key and button actions apply to the window with *focus*, the window that the pointer is in. Usually you are given a visual clue to the window with focus, such as a brightened or different-colored outline, like the top left window in the figure. A window may be composed of several objects, such as buttons and tables, and if the window has focus then at most one of these objects has *keyboard focus*, meaning that any keys that are pressed will affect that object.

A window always has a *frame* around it, and the frame may have a *header* at the top or *footer* at the bottom, bands as wide as the window. The upper left window in the figure has both a header and a footer, the one on the upper right has neither, and the ones in the middle have headers only. The frame may also have raised corners at the frame corners called resize corners, asshown in these examples.

In addition, windows may have:

- a button in the header, on the left side, called the window menu button (not found on popups); see the upper left window in the figure;
- a pin, shown either on its side or as if pushed into the screen (found only on popups); see the middle windows in the figure;
- a pointer;
- scroll bars, which can appear on the right and on the bottom; see the upper left window in the figure;
- movable dividers, which can be either vertical or horizontal; see the bottom window in the figure.

The part of a window that is inside its frame is called the *body* of the window here. Note that the header is not part of the body. The body of a window may contain one or more objects, or panes. The positions of the panes are fixed, but the user can make more of one pane visible and less of another if the pair is separated by a movable divider.

Scroll bars actually belong to objects within windows - their behavior is not controlled by the OL[V]WM - and so it is common to see more than one vertical or more than one horizontal scrollbar in one window, as in the one at the bottom of the figure. Multiple dividers in a window, though less common, are also possible.

Select and Traverse

The visible sign that a window has been selected - given focus - is a brightening of its frame and frame outline; see the middle right window in the figure. You can select a window simply by moving the pointer into it. (It is possible that a window may come up with the mouse pointer in it but without keyboard focus; if this happens, you can move the pointer out of the window and back into it to get keyboard focus.) The keyboard may also be used to select a window, providing the means to step through the windows one at a time. For this purpose a traversal order is established for each application. When the pointer is in the body of one window of an application, you can select the next one in the traversal order by pressing **Control-Tab**. To select the previous one, press **Shift-Control-Tab**. When you use **Control-Tab** or **Shift-Control-Tab**, the pointer is automatically moved to the selected window.

Front, and Raise and Lower

Overlapping windows are presented as if in three dimensions, and the one that is fully visible is said to be in front, or to be raised. To lower a window is to place it furthest back, so that it is obscured by all windows that overlap it. There are several ways to raise and lower windows. They all act as toggles: if the window is not in front, it is moved there; if it is in front, it is moved to the back.

On a keyboard with a **Front** key, place the pointer anywhere in the visible part of the window and press **Front** to place it in front or in back, depending upon its present position. On a keyboard without such a key, position the pointer the same way, and press **Alt** and hold it while pressing **F5** (i.e., press **Alt-F5**). Double-clicking the left button in a visible part of the header but not on the window menu button or pin (if any) will usually accomplish the same result, and pressing the right button there will generally produce a window manager menu that offers you this action. Likewise, if there is a window menu button, pressing the right button on it will produce another menu that may offer you this action.

Quit, and Open and Close

You can remove a window from the screen entirely by choosing **Quit** from its application menu. Likewise, pressing the right button while in the header but not on the window menu button usually produces a window manager menu with **Quit** or, for popups, **Dismiss** as one of its options (although an application may choose not to honor a request to remove the window from the screen). A popup may have a displayed button labeled **Done** or **Cancel** or the like, clicking on which has the effect of a **Quit** in addition to performing some task for the application.

To leave a temporarily unused top-level window on the screen, but have it take up a lot less screen space, either choose **Close** from one of these menus or press the **Open** key (**Alt-F6** if you don't have **Open**) with the pointer anywhere on the window: the window is replaced by an icon, placed somewhere on the screen. To restore the window as it was, except that it will be in front whether or not it was before: with the pointer on the icon, press the right button for a menu with **Open**, double click the left button, or press the **Open** key (or **Alt-F6**).

For popup windows, **Close** is usually equivalent to **Dismiss**: the window is removed entirely from the screen; no icon appears for it.

In A+, a programmer can designate windows that are followers of a given window. When the window is closed, its followers will also be closed. If a window was closed in this manner, then it will be opened when its leader window is opened. So if you request that a window be open or closed, several windows may be affected.

Drag

A window can be dragged around on the screen by placing the pointer on the frame anywhere that does not have special significance - within the header is a convenient place -, pressing the left button, moving the pointer and thereby the window to the desired location, and releasing the button. During this operation, the window will continue to be displayed in its current location, but a rectangle, the outline of the frame, will follow the mouse, so that the potential locations can be seen. Also, a small label or tag may appear that says "location:" and gives the horizontal and vertical coordinates in pixels of the location to which the upper left corner of the outline has so far been dragged.

Resize

If a window has special corners, as do all but the middle left window in the figure, it can be made larger or smaller by the user. When the mouse pointer is moved to one of these corners, the pointer changes to a target. Then you can press the left button, drag the corner wherever you want, and release the button. The new size and shape of the window is established by the point to which you dragged that corner and the (original) location of the diagonally opposite corner. During this operation, as during dragging, the display of the window remains unchanged but a rectangle gives the outline of the potential new frame for the current mouse location. Also, a small label or tag may appear that says "size:" and gives the horizontal and vertical sizes corresponding to the current location, in characters and lines, respectively.

In one or both dimensions, you can drag the selected corner past the fixed corner, which remains catercorner to it.

Scroll

Sometimes not all the text or other contents of an object can be displayed in it. When that happens for a dimension, horizontal or vertical, a scroll bar is displayed, an object that looks somewhat like an elevator shaft (see the window on the top left of the figure and the bottom window). At the ends, the scroll bar has anchors. Connecting the anchors is a cable, with a scroll box somewhere along the cable, (Openlook) or a channel with a slab in it (Motif). Stops may appear on the cable that restrain the scroll box from moving all the way to an end, thereby preventing the view area of the object from becoming partially or completely vacant. The relative size of the stops also indicates the proportion of the displayed object that is actually in view; the smaller the stops, the more out of view.

Pressing the right mouse button with the pointer on a scroll bar produces a menu that lets the viewer select the Openlook or Motif look or the beginning or end of the object being viewed. Any change in appearance is only for the scroll bar pointed to, so an ugly combination of vertical and horizontal scroll bars is possible.

The left mouse button is used to select scrolling operations. The top and bottom thirds of the scroll box have arrows. Clicking on an arrow moves the display one element - e.g., line of text or row of data - in the indicated direction. Pressing on an arrow causes the display to move one element at a time until the button is released. Pressing on the middle of the scroll box and dragging it, by moving the pointer toward one anchor or the other, causes the appropriate part of the text to be displayed when the button is released. Clicking on the cable above or to the left of the box moves the display back one windowful (unless an extremity is arrived at), and clicking on the cable below or to the right of the box moves the display ahead one windowful. Pressing on the cable causes successive windowfuls to appear until the button is released. Finally, clicking on an anchor causes the displayed text to include the corresponding extremity.

There is also an action for the middle button: pressing on the cable moves the display so that the middle of the middle of the scroll box moves to the arrow position.

When a user scrolls an object, other objects may be automatically scrolled with it. When a user selects a row or column, the same row or column may be automatically selected in other objects, in which the first (displayed) row or column may consequently change. More elaborate coordination may occur, such as rows with columns, or title or label changes.

Reapportion

Some windows permit a user to reapportion the relative areas allotted to certain objects. Their special feature is movable sashes, or dividers, between objects, which allow you to control the proportion of the window that each object occupies, simply by moving the sashes. To move a sash, just place the mouse pointer on it, press the left button, drag the sash where you want it, and release the button. A small button is provided on the bottom of a vertical sash and on the right of a horizontal sash as a convenient place to locate the pointer.

The bottom window in the figure has a vertical divider, which has been used to make more of the table on the left visible at the expense of the one on the right. There are also horizontal dividers to separate panes that are stacked vertically.

Pin and Unpin

Pinning is a facility that allows application programmers to give users the ability to influence the removal of popups. For example, suppose an application has a single commit-or-cancel popup. Normally, such a popup appears when needed and disappears after the user clicks one of its two buttons. The application program can, however, include a pin, shown on its side, unpinned, as in the popups shown in this chapter. If you prefer to leave this popup on the screen instead of having it come and go all the time, you place the pointer on the pin and click the left button, causing the pin to be shown pressed in. The application program, before removing the popup, checks to see whether it was pinned, and if so, leaves it alone. The application can also originally include a pin that is shown pushed in, allowing you to click on it and thus unpin the popup, indicating that you would like it removed.

A+ popups are programmed so that when the pin is pulled out, the popup is removed from the screen. The action is the same no matter who, programmer or user, pushed the pin in. Thus clicking twice on an unpinned pin removes such a popup.

The A+ Screen Manager

A+ provides for multiple objects within one window, each with its own scroll bars, buttons, and entry fields as required (cf. the bottom window in the figure); for convenient movement between objects within a window, and within objects; for indicating actions to be taken through option selection and button presses; and for entering and editing data (even in graphical form).

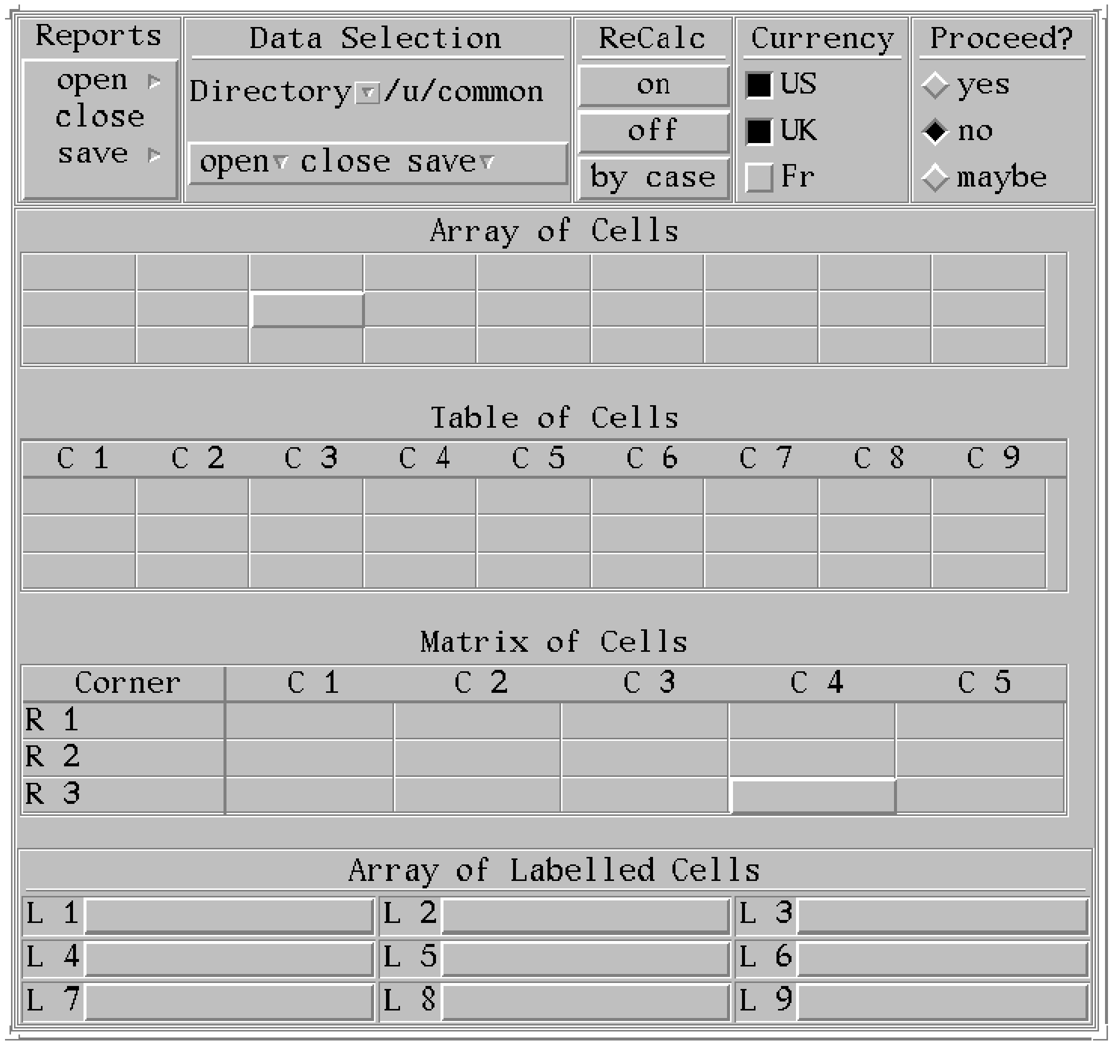
The interactions you can have with the display and the application program are:

- selection and traversal: selecting an object or cell for choice or editing;
- choice: choosing an action or actions, a row or rows of a table, or whatever;
- editing: including input.

All selections and choices can be made with the left mouse button, and the three forms of editing can be initiated with the three mouse buttons. In addition, selections can also be made by pressing keys. The behavior of the mouse buttons is presented first, followed by the forms of editing and the alternative forms of selection using keys. Finally, there is a series of sections on user interactions that are special to particular display objects like graphs and pages.

The visual form of some objects, such as menus, is distinctive and therefore easily recognized in applications. In other cases objects with very different behavior look alike, e.g., buttons and selected input cells. In practice, of course, confusion is unlikely because the actual behavior of an object will be clear from context, so a comparison of visual forms of the various objects is unnecessary. The basic objects for specifying actions and editing in cells appear in the next figure.

Various Input Objects



Selection using the Left Mouse Button

Along the top of the figure are various action objects. The object on the left with the title "Reports" is called a vertical cascade menu. The **open** and **save** items have small triangles to their right indicating submenus. Click on either item or triangle and a submenu will appear and remain visible. Any submenu item that itself has a submenu also behaves this way. Click on any item without a submenu and that action will be taken; any visible submenus will disappear as well. Click anywhere outside the currently displayed submenu and no action will be taken; once again, any visible submenus will disappear. You can also navigate the menu in the more usual fashion of putting the pointer somewhere on the menu, pressing the left button, and dragging the pointer up and down the menu items and over the submenu indicators. If things freeze up during a menu operation, try pressing the **Esc** key.

To the right of the vertical cascade menu are two other menus under the title "Data Selection". The top one is a simple menu, holding a set of choices but no further submenus. Click the left button on the triangle to the right of the title "Directory" and the submenu will appear. Then click on any item and it will be chosen, as well as displayed to the right of the triangle. The menu under the label "Directory" is a horizontal cascade menu; except for the top menu's being displayed horizontally, it is exactly like the vertical cascade menu.

The next object to the right is a vertical set of buttons entitled "ReCalc". Press any one of these buttons and the indicated action will be taken. The selected button may appear with a colored band around it, or, depending on the application, some other color indicator.

The next object to the right, entitled "Currency", is a check object, which allows you to select any number of items from the list. Simply click on the object or the square next to it to select it, and click again to "deselect" it. The squares appear depressed and (usually) in a different color to indicate selection.

The final object of the top row of the figure, the one all the way to the right, is a radio box. It is like a check box, except that one and only item can be chosen (as in a pushbutton radio). Check boxes always have square indicators, and radio boxes have diamond indicators.

Below the top row of action objects is an array of cells. Click on any one of them to select it, and it will appear raised. Depending on the application, it may also have a different color, and all the cells in its row may have still another color. Below that array of cells is another array of cells that differs from the first by having column labels C 1, C 2, etc. Note that no cell is selected in this table. A column can be selected by clicking on its label, and the default behavior is:

- if there is no currently selected cell, the top cell in that column is selected;
- if there is a currently selected cell, the cell at the intersection of the row of the currently selected cell and the selected column is selected.

The application may do something different, e.g., compute a column total.

Below the table of cells is a third variation that additionally has row labels and a corner label in the upper left. Any row or column can be selected, as well as the corner area, by clicking on its label. Selecting a column proceeds in the same way as above, and selecting a row in a similar way.

Finally, at the bottom there is an array of labelled, raised input cells. Click on a cell or its label to select it; the default behavior is for the label and cell to appear with a highlighted border.

Choosing using the Left Mouse Button; Multiple Selection

In the above examples of arrays of cells, a single click selects an individual cell or row. Choosing must be done by double-clicking. Choosing causes a "refer" event to occur and a callback to be executed if a refer callback function was set.

There are two selection modes, single and multiple.
In single selection mode, clicking the left mouse button on a cell, with the **Control** or **Shift** key or neither held down, selects that row and cell and eliminates any previous selection.
In multiple selection mode: **Control**-click changes the selection status of the row pointed to; **Shift**-click extends a selected block to the row pointed to if that row is unselected, and reduces the block otherwise, to "deselect" the row; and click alone selects the row pointed to and eliminates any other rows from the selection. Each of these three actions has a variant, in which the left mouse button is held down and the pointer dragged, involving all the rows the pointer passes over.

Selection using the Middle or Right Mouse Button

The cells in the arrays of the figure are input-output cells. Data can be displayed in them and users can enter new or modified data in them. An application can protect cells so that users cannot modify them. When a cell is protected, clicking on it with either the middle or right mouse button has the same effect as clicking on it with the left button, and also produces a beep. If the cell is not protected, then clicking on it with the middle or right button, besides producing the ordinary left button effect, causes one of the three input modes to be automatically initiated.

Input

There are two input modes for cells such as those in the figure:

- overwriting, or replacement, character by character, anywhere in the field;
- insertion, anywhere in the field; if the mode is entered by just typing a character or pressing **Backspace**, the entire previous contents of the cell are deleted.

To enter overwriting mode initially, select the cell using the middle mouse button, taking care that the pointer is at the position where the overwriting is to begin. The text cursor is a blinking block positioned over the character to be overwritten.

To enter insertion mode initially, either: select the cell using the right mouse button, again taking care to position the pointer; or, when the cell is already selected (perhaps by use of the left mouse button), either press the **Insert** key to begin at the end of the current contents, or press **Backspace** or just start to type to delete the previous contents. The text cursor is a blinking a vertical bar at the insertion point.

An attempt to enter either input mode when the selected cell is protected will elicit a beep.

For an object of the array, command, matrix, scalar, slot, or table class, the space available for editing, in characters, is either 256 (with scrolling on entry and by use of the arrow keys) or the number that fits in the visible space. An attempt to enter too many characters elicits a beep.

Once an input mode has been entered:

- clicking the left button when the pointer is within the input field moves the text cursor position to the pointer position, and makes the mode insertion;
- clicking the middle button when the pointer is within the input field moves the text cursor position to the pointer position, changing to overwriting mode if the mode was insertion;
- clicking the right button when the pointer is within the input field moves the text cursor position to the pointer position, changing to insertion mode if the mode was overwriting;
- pressing the **Insert** key toggles between overwriting and insertion modes without moving the text cursor position;
- typing **Control-e** moves the text cursor to the end of the cell and **Control-a** moves it to the beginning;
- pressing **Shift**-middle-mouse-button pastes the primary selection contents at the pointer position;
- pressing the **left** and **right arrow** keys or **Control-b** and **Control-f** moves the text cursor within the field;
- pressing the **Enter** key ends input, meaning that the present contents of the edit area become the contents of the cell;

- pressing any of the following keys ends input and selects a new cell as indicated: the **Tab** key (usually) selects the cell to the right if one is there; the **Shift-Tab** the cell to the left; and the **up** or **down arrow** key the cell above or below, respectively, if there is no appropriate line in the present cell;
- pressing the **Esc** key ends input mode and returns the contents of the cell to what it was before, but does not select another cell.

For the command and text classes, the cursor attribute describes the text cursor position, as the number of characters from the left. It is 0 when the object is not in input mode. Its vector value for the page display class is described in "Page Objects".

An attempt to enter erroneous input (by pressing **Enter**, **Tab**, etc. with an illicit entry in the cell) elicits an error message; the input mode and the cell remain as they were, to permit the entry to be corrected.

Clipping Text

A+ provides facilities that allow application programmers to permit users to employ the primary selection buffer. If so enabled, a user can insert Emacs or XTerm text in a displayed object, and can also clip text from a displayed object and insert it in an Emacs or XTerm window. If an application uses these facilities, its documentation should describe the procedure for copying or cutting and pasting text.

Selection and Traversal using the Meta-Tab Keys

The visible sign that an object has been selected is a highlight line around it, usually yellow. When the currently selected window was first selected, some object within the window may or may not have been selected. Pressing **Meta-Tab** or, on an IBM keyboard, **Alt-Tab** will select a new object if one has been selected, or the first selectable object if not. The selection order is determined by the application, and repeatedly pressing **Meta-Tab** selects the objects in that order. To select the previous object in that order, press **Shift-Meta-Tab** or, on an IBM keyboard, **Shift-Alt-Tab**. Note that the pointer is not moved to reflect the **Meta-Tab** selections.

Not all selectable areas can be selected with **Meta-Tab** or, on IBM, **Alt-Tab**, nor can you always know beforehand the ones that can. For example, in the figure, pressing **Meta-Tab** repeatedly will (most likely):

- select all the action items along the top row in some order;
- at some point select any of the arrays of cells entitled "Array of Cells", "Table of Cells", or "Matrix of Cells". When one of these arrays is selected, a raised cell within it has keyboard focus, meaning that any character key presses will be directed at it;
- at some point select just one of the labelled cells in the array entitled "Array of Labelled Cells", or, depending on how the application was designed, select them all, one by one.
-

Selection and Traversal using the Tab, Arrow and Page Keys

Once an array of cells like those in the figure has been selected using **Meta-Tab** and **Shift-Meta-Tab** (**Alt**, not **Meta**, on IBM keyboards), a cell within it is selected with the **Tab**, **Shift-Tab**, and **arrow** keys. Arrays of labelled cells like the one entitled "Array of Labelled Cells" in the figure, however, are an exception. If the application is designed so that **Meta-Tab** traverses all the labelled cells, then the **Tab**, **Shift-Tab**, and **arrow** keys have no effect here. However, if

Meta-Tab selects only one labelled cell, then when that cell is selected, the **Tab**, **Shift-Tab**, and **arrow** keys can be used to select one of the other labelled cells.

When keys are used to select a cell, any required scrolling is done automatically, and if no movement is possible in a given direction, the action of the corresponding key is null.

The **PgDn**, **PgUp**, **End**, and **Home** keys can be used in the usual ways to move the view window over a partially visible array of cells.

When a horizontal menu has focus, the **left** and **right arrow** keys move you across the menu. For a vertical menu, and therefore all submenus, the **up** and **down arrow** keys move you up and down from one item to another. Movement into an item with a submenu causes the submenu to appear; there is no need for a further **down arrow** or **right arrow**. For a set of choices like the one labeled "Directory" in the figure, the **down arrow** leads to the menu of choices. Press the **Enter** key when the selected item is an action item to have that action taken; any visible submenus will disappear.

Items on a menu or submenu can also be selected by pressing the keys for the underlined characters. If the item is an action item, the action will be taken and any visible submenus will disappear; if the item has a submenu, it will appear. If two or more items in the same menu or submenu have the same first letter or underlined letter, pressing the key for that letter references the topmost, or leftmost, item. The other items with the same first or underlined letter cannot be referenced in this way. This method can be intermixed with the use of arrow keys.

Deleting and Inserting Rows

If the action is enabled, **Meta-Delete** or, on IBM keyboards, **Alt-Delete** deletes the selected row from an array of cells.

If the action is enabled, **Shift-Meta-Insert** or, on IBM keyboards, **Shift-Alt-Insert** inserts a new row above the selected row. If the action is enabled, **Meta-Insert** (or **Alt-Insert**) inserts a new row below the selected one.

The row attribute is unchanged if possible. When a row is deleted, the next row becomes the selected row, except that if the last row is deleted the new last row becomes the selected row. A row inserted above the selected row becomes the selected row.

Interactions Peculiar To Individual Display Classes

Table Objects

If the application permits, a user can change the width of a column in a table. When the pointer is over a column separator it changes to a two-headed horizontal arrow. Pressing the left mouse button causes a green outline to appear around the column to the left. As the pointer is moved, the right side of the outline moves and becomes white whenever it is not at the current separator position. When the button is released, the column fills the outline and columns to the right of the one whose size was changed are moved accordingly.

If the application permits, a user can move columns within a table. When the pointer is over a tableField or its title area, pressing **Meta**-left-mouse-button (**Alt**-left-mouse-button on IBM keyboards) causes the pointer to change to a two-headed horizontal arrow and the column to be covered by an image of the column and title area, with a yellow outline. As the pointer is moved, the image moves horizontally. When the button is released, the column is moved past any columns between the image and its original location, and the table variable is respecified accordingly. If the new location is the same as the old, no respecification of the table variable takes place. (Actually, you can use any mouse button for this operation and you can press **Control** at the same time if you are so inclined; indeed, you can press several mouse buttons simultaneously and the drop will take place when the last button is released.)

Password Objects

These objects provide password protection to applications. They look like the labelled cells in the figure. Input is started only by selecting the object and pressing a key (insertion, although there is no initial text), and it is completed by pressing **Enter**. Input characters are masked, usually by *'s, and the pointer is positioned to the right of the last * unless you move it.

Hscale and Vscale Objects

A user can change the settings of hscale or vscale objects using the mouse buttons and the **Home**, **End**, **Page Up**, **Page Down**, and **arrow** keys.

When the pointer is anywhere in the object:

- Pressing **Home** moves the slider to the left or top and sets the value to min or max; pressing **End** moves the slider to the right or bottom and sets the value to max or min. (Because of the conflict between text conventions and axis and scale conventions, the same key changes the variable value oppositely in the horizontal and vertical cases.)
- Pressing **Page Up** increases the value by an amount that is controlled by the application; the default is ten. Pressing **Page Down** decreases it by the same amount.
- Pressing **up-arrow** or **right-arrow** increases the value by an amount that is controlled by the application; the default is one. (Either key works for both classes, horizontal and vertical.) Pressing **down-arrow** or **left-arrow** decreases it by the same amount. Holding an **arrow** key down causes its action to be rapidly repeated, after a slight initial delay.

When the pointer is in the slider slot but not on the slider:

- Pressing the middle mouse button moves the slider to the pointer location.
- Pressing the left button when the pointer is on the high side of the slider has the same effect as pressing **up-arrow** or **right-arrow**, and when on the low side as pressing **down-arrow** or **left-arrow**.

When the pointer is on the slider:

- Pressing the left mouse button and moving the pointer drags the slider along in the slot.

When the pointer is on the value label:

- Pressing the middle or right mouse button initiates editing, in overwriting and insertion mode, respectively.

Scalar Objects

Editing has two peculiarities in this class, both involving the left mouse button. Clicking on the value area causes editing to begin, in insertion mode. Clicking twice on the value area causes editing to begin if it was not under way, the value area to be shown in reverse video (double-clicking again will not undo reverse video), and insertion mode to be entered: the next keystroke (if not **Esc**) will delete the entire present contents.

Text Objects

These objects provide a general means for text entry; they can be used whenever a note pad is needed in applications. To enable input, you need only select the object and begin typing. The only input mode is insertion.

The **arrow** keys move the text cursor, and it can be placed anywhere in the text by moving the pointer to the desired location and pressing any mouse button. Text can be expunged using the **Delete** and **Backspace** keys. Press **Enter** or **Linefeed** to create new lines during text entry, which are reflected in the underlying character vector by newline characters.

The workspace value of the vector is not automatically modified as you edit the display. Unless the action is inhibited by the application program, pressing **Control-s** causes the vector to be updated to match the text that appears on the screen - or can be made to appear with the scrollbars.

Page Objects

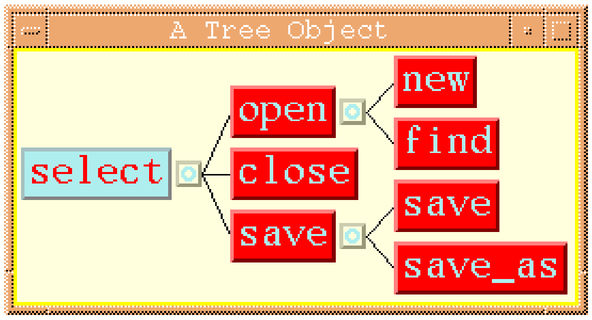
These objects provide formatting of page-based and record-based real-time market data. An application program can provide text entry and editing for these objects in any form. When you click any mouse button anywhere on the page, the position of the mouse pointer is recorded in the cursor attribute, as number of rows from the top and number of characters from the left.

You can create a rectangular outline of a page segment of particular interest, which is transmitted to the application for whatever use it may make of it. Press the left mouse button to establish one corner and move the pointer and release the button to establish the opposite corner. The original corner will start at the top left of the nearest character (which may be blank). During the operation the box that will be transmitted if you release the button is continuously shown.

Tree Objects

Hierarchical data can sometimes be conveniently displayed in a tree object. For example, the items in the cascade menus in the figure are shown as a tree in the next figure. One of the items in the display has been selected, as indicated by the reversed yellow and red. An item is selected by clicking on it with the left button. Once an item is selected, the **arrow** keys can be used to navigate through the items, changing the selected item as you go. To inform the application of your choice (by triggering a callback): double-click on a selected item or press **Enter** when an item is selected.

A Tree Object:



Notebook Objects

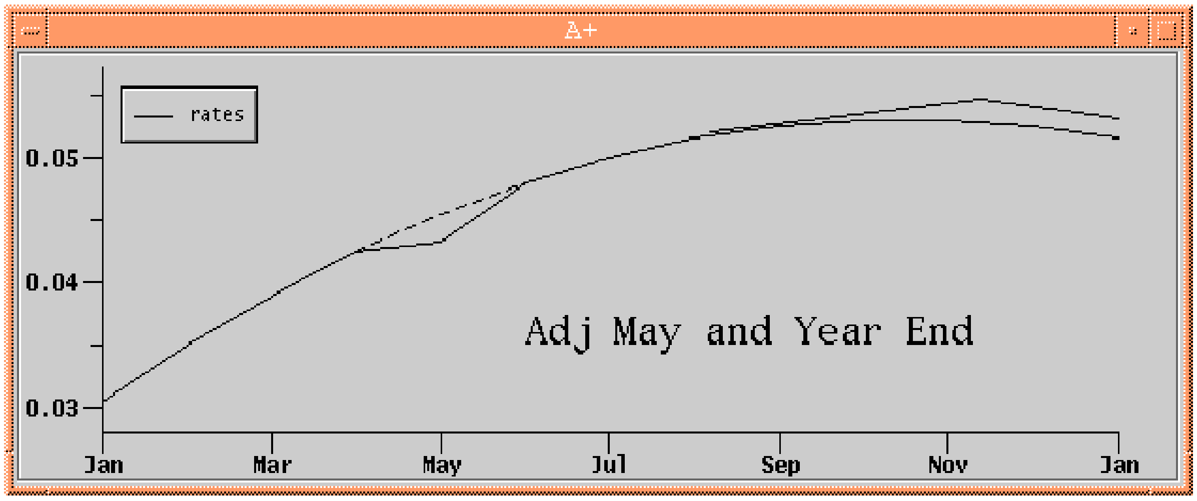
Clicking the left mouse button on a tab causes the associated page of the notebook to be shown and the current one to be hidden. If not all tabs are shown, the list of tabs can be scrolled up by clicking or pressing the left mouse button on the arrow above the list and down by clicking or pressing on the arrow below the list.

Graph Objects

There are a variety of interactions with graphs to customize the display, edit and manipulate trace data, perform detailed examination of traces, and navigate through graphs that are only partially displayed.

The graph in the next figure shows a smooth trace, a data point on that trace that has been modified by the user (the one for May), a line trace that has been added by the user (above the smooth trace for Jul-Sep to Jan), and a text annotation that has also been added by the user. Note that the dashed, curved line above the modified data point for May does not ordinarily appear after a data point is modified.

A Graph Object:



Text Traces

The labels on graphs may be text traces that can be manipulated by users, if allowed by the application:

- Select a text trace:

Place the pointer on the text trace and double-click the left button; the trace will appear in reverse video (the color will change) and nodes will be shown, and the pointer will become cross hairs.

- "Deselect" a text trace:

Double-click the left button anywhere on the graph background.

- Reposition a text trace:

Drag the selected text trace with the left button to the new position and release the button.

- Copy a text trace:

Press and hold the **Shift** key before dragging the selected text trace to the new position. Release the **Shift** key after releasing the mouse button.

- Delete a text trace:

Press **Meta-Delete** or, on IBM keyboards, **Alt-Delete** to delete the selected text trace.

- Create a text trace:

Press and hold the **Meta** key (**Alt** on IBM keyboards) and then click the middle button. A text area appears with its lower left corner located at the pointer. All normal text-editor keys (**arrow** keys, **Backspace**, **Delete**, etc.) are available. If the length of the entered text exceeds the editor width, the text is scrolled to the left. Press the **Enter** key to define the text trace. The trace will be positioned at the same lower left corner as during editing of the text. Press the **Esc** key to discard the entered text and the new text trace.

- Edit a text trace:

With the pointer positioned anywhere on a text trace, press and hold the **Meta** key (**Alt** on IBM keyboards) and then press the middle button. The text editor is invoked with the pointer located at the position of the pointer. The procedure is the same as for creating a text trace. Press the **Enter** key to replace the original text. Press the **Esc** key to discard the modifications and close the editor.

Legend Placement

The graph legend can be moved:

- Select the legend:

Position the pointer on the legend and double-click the left button. It will appear in reverse video.

- Move the legend:

Drag the selected legend with the left button to the new position and release the button. The legend may be outside the axis area and its movement may be constrained to be either horizontal or vertical only.

- Release the legend:

Position the pointer in the graph, but not on the legend, and double-click the left button.

Line Traces

The line segments on graphs may be numeric traces that can be manipulated by users, if allowed by the application:

- Select a line trace:

Place the pointer on a line segment of the trace (not a data point) and double-click the left button; the pointer changes color when on a selected line trace and takes the form of cross hairs away from it.

- "Deselect" a line trace:

Double-click the left button anywhere on the graph background.

- Reposition a line trace:

Drag the selected line trace with the left button to the new position and release the button.

- Copy a line trace:

Press and hold the **Shift** key before dragging the selected line trace to the new position. Release the **Shift** key after releasing the mouse button.

- Delete a line trace:

Press **Meta-Delete** or, on an IBM keyboard, **Alt-Delete** to delete the selected line trace.

- Create a line trace:

Press the **Control** key and then press the left button. As the pointer moves, a line appears with one end anchored to the point where the pointer was when the button was pressed and the other connected to the moving pointer. Release the button and the line segment will become fixed with small knobs at the two ends. To rotate the line, put the pointer on either knob, press the left button, and move the pointer. To move the line while keeping it parallel to its original orientation, put the pointer on the line between the knobs, press the left button, and move the pointer. To create another segment connected to the first, put the pointer on either knob, press the **Shift** key, and then press the left button and move the pointer. Once the pointer moves, the **Shift** key can be released. Double-click the left button to fix the new trace.

- Edit a line trace:

See Data-Point Move.

Data-Point Move

For line, scatter, linescatter, and bar graphs, the data points of a (not necessarily selected) trace can be interactively repositioned, if allowed by the application. A data point is moved by pressing the right button while the pointer is positioned on or near a trace-data point. If it is near enough, cross hairs appear (for line style traces, line segments appear between adjacent points), with a box to indicate the x- and y-coordinates. Moving the pointer, while continuing to depress the right button, repositions the data point at the location where the button is released. Depending on the application, the movement of the data point may be constrained to the vertical or horizontal direction.

Scan X

The scan-x feature enables detailed analysis of the trace data by displaying the numeric value of each trace point as the pointer is swept horizontally across the graph. Pressing the middle button draws a vertical line at the pointer location, and displays a y-value for each trace in the legend window and an x-value in a box which appears at the other end of the pointer. For high-low style traces, the legend window is further expanded to show component values. The xy-values appear in the same format as the corresponding axis labels.

As the line is moved across the screen, the displayed values change. When the line intersects the traces, the x-value and y-values are updated in a discrete fashion; for each trace, a y-value is shown for the data point directly to the left of the line; the x-value that is displayed is the one that corresponds to the displayed y-values for the last trace set in the definition of the graph variable. When the data for a trace gives values for x in an order that is not increasing, however, the trace is considered by s to be overlapping, and then the y-values are not shown and if it is the only trace the x-values vary continuously; even if the values for x for the trace are respecified, it continues to be considered overlapping.

When the pointer is in an area outside the range of the x-values, the displayed x-values vary continuously and y-values are not shown. If the legend is not shown when the middle mouse button is pressed, the scan-x box will be empty: it will not display x coordinates.

Scan XY

The scan-xy feature is a variation of scan-x. Pressing the **Control** key and then pressing the middle button draws cross hairs intersecting at the pointer location and shows the x, y values in a new box near the intersection of the cross hairs. The **Control** key can be released once the cross hairs appear. The displayed x- and y-coordinates vary continuously as the pointer moves.

Zoom

The zoom feature allows the user to focus on a particular region of the plot area by pressing the left button to outline a region of interest. A rectangle is drawn with the origin at the point where the button was initially pressed and the diagonally opposite corner at the pointer's present location. Upon the release of the button, the graph is redrawn to include just axis values falling within the rectangle coordinates.

If, when the button is released, the pointer is outside the graph object or very near the point where the button was initially pressed, zooming does not take place.

Further zooming is possible, in order to further refine the view.

When a graph is "zoomed," the user can scroll the unseen parts of the graph into view with the **arrow** keys.

Zooming is undone by double-clicking the left button on the background of the graph.

40. The s Context

Although the external functions of the s context are present when A+ is started, a `$load s` must be performed to bring in the rest of the s context. This command retrieves a packfile. (The script corresponding to the packfile is in `s.raw.+`.)

Many of the functions in the s context have to do with creating and managing display classes and their attributes. For examples, see "Display Classes".

Specification of attribute values can usually be given in one of two ways: as a slotfiller or an *association list*. In the case of a slotfiller `sf`, each of its symbolic indices holds an attribute name, and the value of the attribute `attr` is accessible as `attr⊃sfal`. An association list `al` is a vector with an even number of elements. The elements at the even index positions, 0, 2, 4, etc., are symbols holding attribute names. The elements at the odd index positions are the attribute values. For example, if `i` is even then `(i+1)⊃al` is the value of the attribute `i⊃al`.

Definitions of s-Context Functions

Array to Character Vector `s.box{a}`

Argument and Result

The argument is any A+ array. The result is a character vector.

Dependency

Printing Precision, `` `pp ``.

Definition

The result is a character vector that, with one exception, produces an A+ array identical to the argument `a` when executed. The exception is that the decimal parts of floating-point numbers may not be represented with sufficient precision to reproduce the original value.

This function is used to produce output representations for objects in the array and slot display classes.

Attribute Settings *s.domainOf{c;a}*

Arguments and Result

The left argument *c* is a one-element symbol array, and the right argument *a* is a symbol scalar or vector. The result is a nested vector of depth 1 whose count equals the count of *a*.

Definition

The symbol *c* names a display class, such as `` `graph``. The right argument is a list of attribute names for that class. The i-th element of the result is an enclosed vector of the possible values of the i-th attribute in the right argument, for the display class specified in *c*. For attributes whose values are not simply chosen from a list, like `` `yx``, the corresponding elements in the result are enclosed Nulls. Examples of attributes whose values are chosen from a list are `` `fg`` and `` `editspace``.

Attribute Sources *s.derivedFrom{c;a}*

Arguments and Result

The left argument is a one-element symbol array and the right argument, *a*, is a symbol scalar or vector. The result is a vector of symbols whose count equals the length of *a*.

Definition

The left argument, *c*, names a display class and the right argument one or more attributes. The result is a vector of class names indicating from what real or virtual class *c* inherits its definition of each attribute.

Bind Several *s.are{v;c}*

Arguments and Result

The arguments are lists of symbols, each argument a vector or scalar. The result is a vector of symbols whose length equals the count of *v*.

Definition

The left argument, *v*, names variables and the right argument names display classes. The explicit result is a vector of the names in *v*. Bind Several binds the variables listed in *v* to the classes listed in *c*. It is like *is*¨ but looks ahead in *c* to avoid the rebinding that a top down approach can involve. The variables in *v* should be listed in a top down order, to avoid the reparenting that a bottom up approach would entail. If *c* is not the same length as *v*, it is used cyclically or partially - i.e., `(⍴v)⍴c` is used as the list of classes.

Here is a comparison of four different ways of binding three objects:

```
     a←b←⍳30 40; l←`a`b;

     `a `b `l is¨ `matrix `matrix `layout;
⍝    .a: variable bound to matrix
⍝    .b: variable bound to matrix
⍝    .l: variable bound to layout     Because of the bottom up
⍝    .a: variable reparented to .l    order, a and b must be
⍝    .l: variable reparented to top level window
⍝    .l: workspace will be created    reparented and, since
⍝    .b: variable reparented to .l    a was top-level,
     free `l;                         l must also be reparented.

     `l `a `b is¨ `layout `matrix `matrix;
⍝    .l: variable bound to layout
⍝    .a: S will bind variable to array
⍝    .a: variable bound to array      Part of binding of l
⍝    .b: S will bind variable to array
⍝     .b: variable bound to array      Because of the use of is¨
⍝     .a: variable bound to matrix     and top down order, both
⍝     .b: variable bound to matrix     children must be re-bound.
      free `l;
                                       ⍝ Now the s.are ways.
      `a `b `l s.are `matrix `matrix `layout;
⍝     .a: variable bound to matrix
⍝     .b: variable bound to matrix
⍝     .l: variable bound to layout
⍝     .a: variable reparented to .l  No lookahead for reparenting:
⍝     .l: variable reparented to top level window  all three
⍝     .l: workspace will be created    objects must be reparented,
⍝     .b: variable reparented to .l    just as they were by is¨
      free `l;

      `l `a `b s.are `layout `matrix `matrix;
⍝     .l: variable bound to layout    Arguments are in top down
⍝     .a: S will bind variable to matrix order, so no reparenting
⍝     .a: variable bound to matrix     is required, and s.are
⍝     .b: S will bind variable to matrix  looks ahead to avoid
⍝     .b: variable bound to matrix        rebinding.
```

Call Default Callback Function *s.call{x;a}*

Arguments and Result

The left argument *x* is a one-element symbol array, and the right argument *a* is a symbol scalar. The result is null.

Definition

The symbol *x* names a displayed object, and *a* is an attribute name for that object; *a* must have a default action for objects of the class to which *x* is bound. The effect is to invoke that default action.

This function would typically be used within a user's callback function for the attribute *a*; it enables selective invocation of default behavior, as well as providing a way to add other function to the default behavior

The attributes currently supported are:

addtexttrace addtrace copytexttrace copytrace cornerselect delete exit insertabove insertbelow rband refer save selectcol selectrow textactivate

Capture the Screen Configuration *s.script{}*

Result

The result is a character vector.

Definition

The result contains A+ expressions to recreate the binding of objects currently bound when this function is executed, to set their attributes, and to set the global variables of these objects to their current values. In effect, the current screen management state of an application can be captured for future reproduction. Which objects actually appear in the script is controlled globally by the switch *s.AUTOSCRIPT*, and locally by the save and script attributes.

CDE Window Manager *s.CDE*

Result

Scalar 1 or 0.

Definition

s.CDE is a dependency. Its value is 1 if the window manager is a CDE one and 0 otherwise. It is evaluated the first time it is referenced; thereafter, the stored value is used. The value does not change within a process.

Character Vector to Array `s.execute{c}`

Argument and Result

The argument `c` is a character vector. The result is an A+ array.

Definition

The argument `c` represents an A+ array, in the same sense that results of the function `s.box` represent A+ arrays. The result is the A+ array represented by the argument `c`.

Class Can Have Attributes `s.canHave{c;a}`

Arguments and Result

The left argument is a symbol naming a class and the right argument is a symbol vector listing attributes. The result is a boolean scalar.

Definition

The result is 1 if all the attributes named in `a` can be set for an object of class `c`, and 0 otherwise.

Classes with Attributes `s.classesHaving{a}`

Argument and Result

The argument `a` is a symbol scalar or vector. The result is a symbol vector.

Definition

The result is a list of classes having the attributes listed in the argument `a`.

Example

```
s.classesHaving{`at`geometry}
```

Closest Named Color `s.closest{r}`

Argument and Result

The argument is a three-element integer vector. The result is a symbol.

Definition

The result holds the name of the color whose rgb value is closest to that represented by the argument.

Color Name to Pixel Conversion `s.color{x}`

Argument and Result

If the argument is a character vector or symbol, the result is an integer scalar; if the argument is an integer, the result is a symbol.

Definition

This function converts from color names to pixel representations of colors, and from pixel representations to names. The pixel representation is the most efficient way to specify colors in s. Consequently, for applications that set colors often, it may be best to compute their pixel values once and for all during application initialization, and use those values in color specifications. Do not store the pixel values in your A+ scripts, however, because they change from one session to another.

Color Shades `s.shade{b;c}`

Arguments and Result

The left argument is an integer between 2 and 19, and the right argument is a two-element vector of symbols. The result is a character matrix with `b` rows and seven columns.

Definition

The result `r` is such that `⍎r` is a vector of hex representations of colors that vary from the color named in `0⊃c` to that in `1⊃c`.

Color to Hex `s.ch{c}`

Argument and Result

The argument is a character vector. The result is a symbol of the form `` `#rrggbb``.

Definition

This function is the inverse of `hc`.

Color to RGB `s.cc3{p}`

Argument and Result

The argument is a symbol scalar. The result is a three-element integer vector.

Definition

The argument is a color name, the result its rgb representation. See `s.hc3` for a discussions of the rgb color representation.

Copy Attributes `s.copy{x;v}`

Arguments and Result

`x` is a symbol scalar and `v` is a list of symbols. The result is a vector of symbols.

Definition

Copy Attributes attempts to bind the variables named (in symbol form) in the vector or scalar `v` to the same class as that of the variable named in `x`. The other attributes of each one successfully bound are then given values copied from `x`. The result lists the variables which were successfully bound. **Warning:** `x` should have only one item; if you give a list for `x`, the first item is used for all the variables in `v` and the other items are ignored, *without any message or error report*.

If `x` names a table (or graph) and a variable named in `v` could be a tableField (or graphTrace) but not a table (or graph), then a parent variable is generated for it and bound to table (or graph), but the attribute values of `x` are nevertheless copied, where appropriate, to the variable named in `v`, not to the generated variable.

Attribute values are copied except as dictated by the variables, as in the following example, where a blank space appears in `t1` at the right because of the cols limitation:

```
     a←b←c←d←e←f←⍳10
     t←`a`b`c`d
     t1←`e`f
     `t has (`class;`table; `cols;4; `show;1)
     `t s.copy `t1
 `.t1
     >>`space of¨ `a`b`c`d`e`f
 5  5  5  5  5  5
     (<`yxs`cols) of¨ `t`t1
<  <   191  279
   <   4
<  <   191  279      ⍝ Same size as the four-column t
   <   2             ⍝ Only two columns because ⍴t1 is 2.
```

Define Attributes `s.classHas{c;a}`

Arguments and Result

The left argument `c` is a symbol. The right argument is a two-element nested vector, in which the first element holds a symbol and the second either holds a two-element function array or is the Null. The value is a symbol.

Definition

The effect of this function is to define a new attribute for the class named in `c`, or delete a defined attribute. If the right argument is of the form `(s;(g_f;s_f))`, then the attribute named in the symbol `s` will be defined. If the right argument is of the form `(s;)`, then the defined attribute named in `s` is deleted.

The syntax of the function `g_f` is `g_f{o}`; if the symbol `o` is an object of class `c`, then the result of `g_f{o}` is the value of the attribute `s` for that object. The value of the attribute should be obtained in the same way as for primitive attributes, namely as the result of `o has s`. The function `has` uses `g_f` to get the value.

The syntax of the function `s_f` is `s_f{o;a}`; if the symbol `o` is an object of class `c`, then the effect of `s_f{o;a}` is to set the value of the attribute `s` to `a` for that object.

Example

See "Display Classes".

Desktop Geometry `s.desktop{}`

Result

The result is a two-element vector.

Definition

The result is the number of rows and number of columns in the virtual desktop.

Electrified Attributes `s.used{x}`

Argument and Result

The argument is a singleton symbol and the result is a vector of symbols.

Definition

The result is a list of the attributes of the object named by `x` that are explicitly electrified. In addition to attributes connected to variables and functions, however, the result includes attributes connected to dependencies.

See the example for the next function, `s.usedBy`, especially concerning "explicitly".

Electrifying Objects `s.usedBy{y;x}`

Arguments and Result

The argument `y` is a scalar or vector of symbols; `x` is a singleton symbol; the result is a vector of enclosed symbols or Nulls.

Definition

The argument `y` lists attributes and `x` names an object. For each attribute, the result names the variable, function, or dependency which explicitly (see the example) supplies its value for the object. For an attribute using none of these, a Null appears in the result instead.

Example

```
     show `a ⊣ a←ι8 10
     s.BLACK←`blue          ⍝ The fg color in the display of a
     s.used `a              ⍝ immediately turns blue, but s.BLACK
     `fg s.usedBy `a        ⍝ is a default, not explicitly set,
<                           ⍝ so neither s.used nor s.usedBy
                            ⍝ shows the electrification.
     `a has (`fg;`s.BLACK)  ⍝ Explicitly connect the fg color to
     s.used `a              ⍝ s.BLACK.  Now both s.used and
```

```
`fg                           ⍝ s.usedBy show the connection of
     `fg s.usedBy `a          ⍝ `fg for `a with s.BLACK
<   `s.BLACK
```

Establish Screen Management Functions *s.functions{}*

Result

Null.

Definition

Establish the six root-context screen-management functions - *free*, *has*, *hide*, *is*, *of*, *show* - in the current context.

Font Name to Internal Form Conversion *s.font{x}*

Argument and Result

If the argument is a character vector or symbol, the result is the internal form, an integer scalar. If the argument is an integer, the result is a symbol.

Definition

Just as the pixel representation of colors is an efficient, internal form, there is an internal representation for fonts (see *s.color*). This function converts font names to the internal form, and also the internal form to font names. For applications that set fonts often, it may be best to compute their internal values once and for all during application initialization, and use those values in font specifications. Do not store the internal values in your A+ scripts, however, because they change from one session to another.

Free All Objects *s.reset{}*

Definition

The effect is to free all objects that would be named by *s.windows{}*. The result is a vector of symbols naming all the objects that were freed.

Get Attribute Defaults for a Class *s.defaultOf{a;c}*

Arguments and Result

The left argument a is a symbol scalar or vector. The right argument *c* is a one-element symbol array. The result is a boxed array of the same shape as the left argument *a*.

Definition

The i-th element of the result is the enclosed default value of the i-th attribute in *a* for the class *c*.

Example

```
    s.defaultOf{`bg`fg;`array}
```

Get Attribute Default Variables or Values *s.Of{a;c}*

Arguments and Result

The left argument is a scalar symbol or vector of symbols. The right argument is a symbol. The result is a nested vector with the same number of elements as the left argument.

Definition

The i-th element of the result is an enclosed name or value corresponding to the i-th attribute listed in *a* for the class *c*: the name of the attribute default variable if a dot was placed at the beginning of the attribute name in *a*, else the default value of the attribute. (See "Attribute Default Variables"). Cf. *s.Has*.

Examples

```
     `bg s.Of `array
<   `grey
     `.bg s.Of `array
<   `s.GREY
```

Get Attribute Values for Parent *s.ofParent{a;o}*

Arguments and Result

The left argument is a scalar symbol or vector of symbols. The right argument is a symbol. The result is a nested vector with the same number of elements as the left argument.

Definition

The i-th element of the result is the enclosed value of the i-th attribute listed in *a* for the parent of the object *o*.

Hex to Color *s.hc{h}*

Argument and Result

The argument is a symbol of the form `` `#rrggbb``. The result is a character vector.

Definition

The argument is a hex representation of a color, and the result is the color name. See *s.color* for a discussion of the pixel color representation.

Hex to RGB *s.hc3{h}*

Argument and Result

The argument is a symbol of the form `` `#rrggbb``. The result is a three-element integer vector.

Definition

Each pair of characters *rr*, *gg*, *bb* in the argument represents a two-digit hex, or base 16, integer. The base 10 representations of the pairs of hex integers are integers between 0 and 255. The result is of the form (n1, n2, n3), where n1 is the base 10 representation of *rr*, n2 is the base 10 representation of *gg*, and n3 is the base 10 representation of *bb*. The result is an rgb color representation. Both the argument and result are interpreted as the relative intensity of red, green, and blue for a screen color.

List of All Bound Objects *s.objects{}*

Result

The result is a nested array of symbols.

Definition

The result is a list of all objects currently bound to some display class.

List of All Objects Bound to a Class *s.boundTo{c}*

Argument and Result

The argument is a symbol scalar. The result is a vector of symbols.

Definition

The result is a list of the objects bound to the class *c*.

List of Attributes *s.attributes{c}*

Argument and Result

The argument *c* is a symbol. The result is a vector of symbols.

Definition

The result is a list of the attributes for the class named in the argument *c*. In addition, if *c* is `` `variable``, then the result is a list of the persistent attributes.

List of CDE Workspaces `s.wslist{}`

Result

A nested vector of character vectors.

Definition

The result is a list of the workspace names or, if the window manager is not a CDE one, null.

List of Classes `s.classes{}`

Result

The result is a vector of symbols.

Definition

The result is a list of symbols holding the names of all current classes.

List of metaClasses `s.metaClasses{}`

Result

The result is a vector of symbols.

Definition

The result is a list of symbols holding the names of the display classes that are not virtual and cannot be directly instantiated. In effect, these classes have no visual representations.

Example

See "Display Classes".

List of Popups `s.popups{}`

Result

The result is a vector of symbols.

Definition

The result is a list of all displayed objects that are popups - not contained in another object and not top level.

List of Real Classes `s.realClasses{}`

Result

The result is a vector of symbols.

Definition

The result is a list of symbols holding the names of the display classes that can be directly instantiated, such as array and table.

Example

See "Display Classes".

List of Shells `s.shells{}`

Result

The result is a vector of symbols.

Definition

The result is a list of all displayed objects that have window manager decorations, both top-level windows and popups.

List of Subclasses of a Class `s.subClasses{c}`

Argument and Result

The argument is a symbol scalar. The result is a nested array of symbols.

Definition

The result is a nested array of symbols whose values are the names of the subclasses of the class specified in the argument `c`. A class `b` is a subclass of the class `c` if the attribute set of `b` contains the attribute set of `c`. The structure of the result indicates the hierarchical structure of the subclasses, based on set membership of their attribute sets.

List of Superclasses of a Class *s.superClasses{c}*

Argument and Result

The argument is a symbol scalar. The result is a nested array of symbols.

Definition

The result is a nested array of symbols whose values are the names of the superclasses of the class specified in the argument `c`. A class `b` is a superclass of the class `c` if the attribute set of `b` is contained in the attribute set of `c`. The structure of the result indicates the hierarchical structure of the superclasses, based on set membership of their attribute sets.

Example

See "Display Classes".

List of Top-level Windows *s.toplevels{}*

Result

The result is a vector of symbols.

Definition

The result is a list of all top-level windows - those not contained in another object, with `` `iconic`` equal to 1.

List of Virtual Classes *s.virtualClasses{}*

Result

The result is a vector of symbols.

Definition

The result is a list of symbols holding the names of the display classes that cannot be directly instantiated, but nonetheless have visual representations, namely tableField and graphTrace.

Example

See "Display Classes".

Move Mouse Pointer *s.warpPointer{w}*

Releases 2.44 and 4.09 on.

Argument

The argument is a scalar symbol naming a workspace.

Definition

This function moves the mouse pointer to the window associated with the symbol `w`. This is useful after a window has been given focus and you want keyboard input to go immediately to that window. Example:

```
(a;b) ← (⍳10; ⍳3 3)
s.show¨ `a `b
`b s.has (`focus;1)
s.warpPointer `b
```

Nested Association List *s.ral{x}*

Argument and Result

The argument is a nested, i.e. recursive, slotfiller and the result is a nested association list.

Definition

This function converts a nested slotfiller to the equivalent nested association list.

No Longer Use Variables `s.doesNotUse{o;a}`

Arguments

Each argument is a one-element symbol array or a vector of symbols.

Definition

If `,o` and `,a` have the same number of elements, then for every valid index `i`, the attribute `i#,a` of the object `i#,o` will no longer use any variable it may have been using (see `s.uses`). Otherwise, one of them has only one element, and it is paired with every element of the other argument.

Objects with Specific Attribute Values `s.which{a}`

Argument and Result

The argument is either a slotfiller or an association list. The result is a symbol vector.

Definition

The argument represents a list of attributes and their values (see above). The result is a list of the objects which currently have those settings for those attributes.

Object with Keyboard Focus `s.this{}`

Result

The result is a symbol scalar, or the Null.

Definition

The result is the object on the screen that currently has keyboard focus, or the Null if there is none.

Primitive Attributes `s.primitiveTo{c}`

Argument and Result

The argument is a one-element symbol array and the result is a vector of symbols.

Definition

The result is a list of the attributes for the given class whose definitions it does not inherit, i.e., whose definitions are primitive to the class.

Recursive Association List to Nested Slotfiller `s.rsf{a}`

Argument and Result

The argument `a` is a recursive association list, i.e., an association list in which some of the values may themselves be association lists.

Definition

The result is a nested slotfiller equivalent to the argument `a`. That is, any value in `a` can be reached by a sequence of associations `` `x``, `` `y``, `` `z``, ... ; that same value can be reached in the result by the primitive function Pick with a left argument of `` `x`y`z`` ... . The individual changes are made using `_alsf`.

Refresh All Objects `s.refresh{}`

Definition

The effect of this function is to refresh the visual appearance of all objects to reflect recent settings of display-related attributes such as fg and out.

Restore the Captured Screen Configuration `s.load{f}`

Argument and Result

The argument is a character vector. The result is a two-element nested array, with each element a simple vector of symbols.

Definition

The effect is the same as `_load{f}`, except for the explicit result. The result is of the form `(b;f)`, where `b` is a vector of symbols listing the objects that are bound to a display class as a result of loading this file, and `f` is a vector of symbols listing the objects freed. Cf. Save the Captured Screen Configuration, `s.save`, below.

RGB to Color `s.c3c{r}`

Argument and Result

The argument is a three-element integer vector. The result is a symbol scalar.

Definition

This function is the inverse of Color to RGB, `s.cc3`.

RGB to Hex `s.c3h{r}`

Argument and Result

The argument is a three-element integer vector. The result is a symbol of the form `` `#rrggbb``.

Definition

This function is the inverse of Hex to RGB, `s.hc3`.

Save the Captured Screen Configuration `s.save{f}`

Argument

The argument is a character vector.

Definition

The effect of this function is to save the same result as that of the function `s.script{}` in the Unix file named in `f`. Cf. Restore the Captured Screen Configuration, `s.load`, above.

Screen Management Documentation `s.doc{f}`

Argument and Result

The argument is a character vector naming a Unix file.

Definition

The effect of this function is to write summary documentation for the screen management system to the file named by the argument `f`.

Scroll Together Horizontally `s.hScroll`, Vertically `s.vScroll`

Definition

Scroll Together Vertically and Scroll Together Horizontally are callback functions that manage scroll groups. When they are used, a group is represented by a symbol vector and the appropriate one of these functions is established as a preset callback function for this vector. They can add and delete members, setting appropriate values for new and deleted members' vscrollwith, setcol, setfirstcol, and hscrollwith, setrow, setfirstrow attributes. Note:

- The scroll group vector should be empty when the preset callback function is set on it, so that each object is handled in a callback when it is placed in the group.
- Any objects placed in a group should already be bound, since attributes will be set for them.

For example:

```
     `a`b`c`d`e`f is¨ `array ⊣ a←b←c←d←e←f←⍳20 20
     X←Y←()
     _spcb{`X;(s.hScroll;)}; _spcb{`Y;(s.vScroll;)};
     X←`a`b`c
```

`Now these three objects will scroll horizontally together.`

```
     show¨ `a`b`c`d`e`f
     Y←`a`b`e`f
```

`Now these objects will scroll vertically together, a and b both ways.`

`Make d scroll horizontally with the others (Append Assignment okay).`

```
     X[,]←`d
```

`Make c scroll with a, b, and e, and make f not scroll with any object.`

```
     Y[3]←`c
```

Sector of Desktop `s.beHere{o}`

Argument

The argument is a symbol scalar.

Definition

The top-level object `o` is displayed in the currently active sector of the virtual desktop. This is a cover function for `` `a has `atsector `here``.

Set Attribute Default Variables or Values `s.Has{c;s}`

Arguments and Result

The left argument is a symbol, the right argument a slotfiller or association list, and the result a symbol.

Definition

This function sets the default values or attribute default variables for those attributes for the class named by `c`. See Set Attribute Values for Parent regarding slotfillers and association lists. The right argument represents a list of associations between attributes and either default values or, for any attribute names that have been preceded by a dot, attribute default variables (see "Attribute Default Variables"). Cf. `s.Of`.

Example

```
     `array s.Has `bg `grey
<   `array
     `array s.Has `.bg s.GREY
<   `array
```

Set Attribute Defaults for a Class `s.hasDefault{c;s}`

Arguments and Result

The argument `c` is a symbol scalar. The argument `s` is a slotfiller or association list. The result is identical to `c`.

Definition

Either form of argument, slotfiller or association list, associates a list of attributes with a set of values in the manner described above. The effect of this function is make these values the default values of their attributes. After `s.hasDefaults{c;s}` is executed, all objects subsequently bound to the class specified by `c` will have default values as specified in `s`.

Note the following relation between this function and persistent attributes: suppose this function is used to set the default value of a persistent attribute for a class `c`, and that an object bound to another class has had the value of this attribute set explicitly. If that object is then rebound to the class `c`, the value of the persistent attribute will be the one that had been explicitly set, not the default that was set with this function.

Examples

After the following is executed:

```
`array s.hasDefault (`bg`fg;(`blue;`red))
```

all objects bound to class `` `array `` will have as defaults a blue background and red foreground color. For examples using derived classes, see "Display Classes".

Set Attribute Values for Parent `s.parentHas{o;s}`

Arguments and Result

The left argument is a symbol, and the right argument is either a slotfiller or an association list. The result is a symbol.

Definition

If `s` is a slotfiller, then each of its symbolic indices holds an attribute name `attr` and each `` `attr⊃s `` is a value. If `s` is an association list, each element at an even index `i` is a symbol holding an attribute name, and the corresponding element at index `i+1` is a value. In either case, the right argument represents a list of associations between attributes and values. This function sets those attributes to those values for the parent of the object named by `o`. An object `p` is the parent of an object `o` if `` `o `` explicitly appears in the value of the global variable `p`.

Set Current CDE Workspace `s.setcurrentws{`***n***`}`

Argument and Result

The argument is a character string and the result is null.

Definition

The workspace named *n* is made the current workspace (i.e., displayed) if the window manager is a CDE one. Otherwise, no action is taken. [Future use]

Tree Dependency `s.wstree{s}`

Argument

The argument `s` is a scalar symbol naming a global variable.

Definition

The value of the global variable named in the argument becomes a nested array representing the workspace-tree dependency. (The first top-level object created in an A+ session becomes the so-called screen workspace for that session; if that object is subsequently put in a container, that container becomes the screen workspace. Any other top-level objects do not appear in the result.)

Use Variables `s.uses{o;s}`

Arguments

The argument `o` is a one-element symbol array or a vector of symbols. The argument `s` is either a slotfiller or an association list.

Definition

In the simplest case `o` has one element naming an object, and `s` contains a pair of symbols `a`, `v` where `a` is an attribute and `v` names a global variable. The effect of this function is to identify the value of the attribute for the object named by `o` with the value of the global variable, so that whenever the value of the variable changes, the attribute (for the object named by `o`) automatically changes to the same new value.

In general, if `s` is a slotfiller, each of its symbolic indices names an attribute `attr` and each value `` `attr⊃s `` is a symbol naming a global variable. If `s` is an association list, each element at an even index `i` is a symbol naming an attribute and the corresponding element at index `i+1` is a symbol naming a global variable. In either case, the right argument represents a list of associations between attributes and global variables.

There are three possible relations between the number of such pairs, or associations, and the number of objects named in `o`. If `o` has one element, then every attribute of the object named by `o` is identified with its global-variable partner in the manner described above. If `o` is a list with the same number of object names as there are attribute, global-variable pairs on the right, then the `i`th attribute of the `i`th object is identified with the `i`th variable for every valid `i`. Finally, if there is only one attribute, global-variable pair on the right, then that attribute is identified with that variable for every object named in `o`.

Subsumed by `s.has`.

Users of a Variable `s.using{v}`

Argument and Result

The argument is a symbol naming a global variable. The result is a nested pair of symbol vectors.

Definition

The result is of the form `(o;a)`, where `o` and `a` are vectors of symbols, and for every valid index `i`, the attribute `a[i]` of the object `o[i]` currently uses the variable named in `v` (see `s.uses`, above).

Values of Attributes for a Class `s.ofClass{a;c}`

Arguments and Result

The left argument is a scalar symbol or vector of symbols. The right argument is a symbol. The result is a nested vector with the same number of elements as the left argument.

Definition

The i-th element of the result is the enclosed value of the i-th attribute listed in `a` for the class `c`.

Example

See "Display Classes".

Variable Can Be of Class `s.canBe{v;c}`

Arguments and Result

The left argument is a symbol naming a variable and the right argument a symbol naming a display class. The result is a boolean scalar.

Definition

If `v` can be bound to `c`, the result is 1; otherwise the result is 0.

s-Context Global Variables

The s_context global variables are classified as switches, parameters, system attribute default variables, and data variables. See the tables "s-Context Switches (Global Variables)", "s-Context Parameters (Global Variables)", "s-Context System Attribute Default Variables", and "s-Context Data Variables". When the value of an attribute is Null and there is a corresponding `s.AUTO...` variable, the value of that variable is used for the attribute.

s-Context Switches (Global Variables)

s-Context Variable	Description	Default
`s.AUTOBUILD`	If 1, layouts defined by simple vectors of symbols invoke the autobuilder when the objects for the layout are on the screen. Then the objects appear in the layout in the same relative positions they occupied on the screen before the layout was built.	0
`s.AUTOCOPY`	If 1, when a row is inserted values are copied from the row above or below which the new row is inserted.	0
`s.AUTOEDITSPACE`	If 1, the size of the edit buffer is the value of `s.EDITSPACE`.	1
`s.AUTOEVALUATE`	If 0, variables are not to be evaluated in order to determine appropriate attributes for display; they will be set by the programmer. Use with great caution. The evaluate attribute gives finer discrimination.	1
`s.AUTOEXECUTE`	If 1, input expressions are executed to obtain a value; if 0, they are not.	1
`s.AUTOHAS`	If 1, whenever an attribute value is changed, report the new setting in the A+ session log, as a comment.	0
`s.AUTONEWSHOW`	For an existing container: if 1, a reparented child (see `s.AUTOREPARENT`) is reshown and an unbound variable is shown; if 0, a reparented child is reshown only if mapped and an unbound variable is not shown.	0
`s.AUTOPOSITION`	In layouts defined by simple vectors of symbols: if `s.AUTOBUILD` is 1, layout objects already on the screen keep their relative positions and all others have at values of `0 0 1 1`; otherwise, all objects are stacked vertically if `s.AUTOPOSITION` is 1, and all have at of `0 0 1 1` if it is 0.	1
`s.AUTOPRINTABLE`	Default value for printable attribute. See /common/s/opt.doc.	1
`s.AUTOREPARENT`	If 1, an orphaned child (such as an object removed from a layout) is reparented to be a top-level object. If 0, an orphaned child is freed.	1

s.AUTORESHOW	If 1, a reparented child (see *s.AUTOREPARENT*) is reshown if it was shown in its former parent; if 0, it is not shown.	0
s.AUTORESPACE	If 1, the space attribute is increased when exceeded by the result of the *default* out function.	0
s.AUTOSCRIPT	If 0, no object definitions appear in the result of *s.script{}*; if 1, the definitions appear for all objects that have their script attributes set to 1.	1
s.AUTOWS	When *s.WS* is null and an object is (initially) shown whose ws attribute is null, *s.AUTOWS* controls whether the object becomes the screen workspace (and hence top-level; cf. *s.SHELL*): if 1, yes, if 0, no.	1
s.MENUDEFAULT MNEMONIC	If 1, when a menu is bound, the default accelerator for each button is the first letter of its symbolic index. If 0, the default is no accelerators. Cf. the mnemonics attribute.	0

s-Context Parameters (Global Variables)

Variable	Description	Default
s.ABORT	Controls whether severe errors (and perhaps also moderate ones - see *s.ERROR*) cause s functions to abort execution (when *s.ABORT* is 1) or to suspend it (when 0). Cf. *s.QUIET*.	1 (abort)
s.AUTOBLANK	The default value used for the display of a NA - e.g., where a row or cell has been inserted - in an object for which the blank attribute has not been set. Cf. *s.NA*.	Null
s.AUTODOC	Default for doc attribute, for documentation.	' '
s.BACKING STORE	Set to 0, turns off the use of backing store for any windows created thereafter, which radically reduces XServer memory consumption but impairs refreshing. If 1, backing store is used (as normal) for any windows created thereafter. Thus some windows can use backing store while others do not.	1
s.BEEP	Specify (any value) to cause the display to beep.	0

s.BUSY	Whenever s.BUSY is set to 1, an internal "busy count" is incremented. Whenever s.BUSY is set to 0, this count is decremented by 1 (but not below 0). When the count is greater than 0, the pointer is busy: it appears as a little clock or stopwatch whenever it is in an s window. This behavior is independent of the setting of s.CLOCK.	0
s.CLOCK	If 1, the pointer is busy - it appears as a little clock or stopwatch whenever it is in an s window - during callbacks for displayed objects and during assignments to variables being displayed on the screen, independent of s.BUSY. If 0, it is not, except as required by s.BUSY. s.CLOCK is set to 0 in applications requiring fast update rates, e.g., certain real-time updates.	1
s.CONNECTED	A slotfiller. The symbolic indices identify displays. An item of the second element is set to zero if the server for the display identified by its symbolic index fails. If the user has not set a callback on s.CONNECTED, then s calls _exit{1} in the event of a failure.	Items of the second element are all nonzero.
s.DATASPACE	The default for the space attribute.	9
s.DISCONNECT	When server disconnects, do _exit{s.DISCONNECT}.	0
s.DOUBLE CLICK INTERVAL	The maximum interval, in milliseconds, between two consecutive clicks for them to be considered a double click.	250
s.EDITSPACE	Default size of the edit buffer; see the editspace and space attributes.	256
s.ERROR	If 0, s functions proceed upon encountering a moderate error (but see s.QUIET); if 1, they act as dictated by s.ABORT.	0 (proceed)
s.EXIT	If s.EXIT is a scalar integer, then when the exit attribute is set to 1 the A+ session ends with _exit{s.EXIT}.	Null
s.FILLCOLORS	This symbol vector provides the default values for the fillcolor attribute of the graph display class. See "fillcolor".	The vector `red `green `deepskyblue

		`` `cyan `pink `white ``
`s.LINECOLORS`	This symbol vector provides the default values for the linecolor attribute of the graph display class. See "linecolor".	The vector `` `red `green `deepskyblue `violetred `yellow `cyan `pink `white ``
`s.LINESTYLES`	This vector provides the default values for the linestyle attribute of the graph display class. See "linestyle".	The vector `` `solid `dash `dotdash `dot ``
`s.MSG`	If `s.MSG` is `` `x `` or `` `c.x `` or `` `c`x ``, s messages are put in `` `x `` or `` `c.x ``, and if it is a suitable displayed object, they appear in the display. If `s.MSG` is null, the messages are displayed in the session log.	Null
`s.NA`	The default values used in the workspace for NA - e.g., in rows or cells created by insertabove and insertbelow -, arranged in a slotfiller. The value used is based on the type of the variable being expanded and is used if the na attribute is Null (e.g., has not been set). Cf. `s.AUTOBLANK`.	`` (`int `float `char `sym `box `func `null; (¯999999999; ¯999999999.0; ' '; `; <(); ⊣; ())) ``
`s.QUIET`	Controls the display of messages in the A+ session: if -1, none; if 2, only for severe errors (indicated by `⍝ !!!` in the message); if 1, also for moderate errors (indicated by `⍝ !!`); if 0, also for warnings (indicated by `⍝ !`) and information (indicated by `⍝`). See `s.ABORT` and `s.ERROR`.	0 (all)
`s.SCREEN`	A dependency giving the height and width in pixels of the current screen.	
`s.SHELL`	Determines the default window type. All windows that are created when it is 0 are top-level by default. All windows created when it is -1 are popups by default, with one possible exception: if `s.WS` is the Null, then the next window that is created for a variable whose	¯1

	ws attribute is 1 or, if `s.AUTOWS` is 1, null is top-level and becomes the screen workspace. But see "Reversing the Effect of the s.SHELL Setting". Also see `s.WS` in this table and `s.AUTOWS`.	
`s.TRACE SYMBOLS`	This vector provides the default values for the symbol attribute of the graph display class. See "symbol".	The vector `` `cross `circle `circlefilled `triangle `trianglefilled `square `squarefilled `diamond `diamondfilled `xsym `star ``
`s.VERIFY`	Most display classes require a variable to satisfy certain rank and type restrictions. If 0, verification takes place in the A+ interpreter, and when a variable bound to a display class is assigned a value inappropriate to that class, an `←: invalid` message is issued and execution is suspended. If 1, verification takes place within the s-context functions and a more meaningful message is issued.	0
`s.WP`	When an object with default formatting is bound, `s.WP` is used in calculating a tentative value for the object's space attribute: if `s.WP` is 0, the result is the value of `s.DATASPACE`; if it is -1, the result is the minimum width required for any cell in the entire object; otherwise, the result is the minimum width required for any cell in just the first `s.WP` rows or (for a vector) cells, to save computation of widths. The space attribute is then set to the greatest of its existing value, the value just calculated, and, if the object is a vector, the width of its title.	100
`s.WS`	The screen workspace: the first object bound in a session, and thereafter any container that becomes the parent of the current workspace. If `s.WS` is set to the Null, the next object bound becomes the screen workspace; see `s.SHELL`	Null (when no object has been bound)

	When `s.WS` is given a valid nonnull setting, then, no matter what form is given in the setting, its value is a two-element symbol vector `` ` ``*contextname* `` ` ``*unqualifiedvarname*.	
`s.WSNAME`	The default value of the shelltitle attribute for the object named in `s.WS`.	`0⊃1↑_argv`, or `'A+'` if `_argv` is the Null.

Reversing the Effect of the s.SHELL Setting

The effect of the setting of `s.SHELL` can be reversed in individual cases by prefixing an underscore (_) to the name of the class to which a top-level object is to be bound.

For example, if the object's name is `table1` and it is bound as follows:

```
    `table1 is `_table
```

and if `table1` is not the screen workspace (see the description of `s.WS`), then:

- if `s.SHELL` is -1, `` `table1 `` is a top-level window;
- if `s.SHELL` is 0, `` `table1 `` is a popup.

Attribute Default Variables

An attribute default variable for an attribute and a class is a variable that contains the default value for that attribute for that class. Several attributes may share such a variable, and so may several classes. A *system* attribute default variable for an attribute and a class is a variable that *by default* contains the default value for that attribute for that class. For example, `s.GREY` is the system attribute default variable for bg for all classes, editfg for all classes, and so on. Each system attribute default variable has a default value; for `s.GREY`, as its name might imply, that value is grey.

With three exceptions, when the value of an attribute default variable is changed, the value of each attribute for each class member for which that variable supplies the default is immediately changed: the attribute is "electrically connected" to the variable. For example, if the value of `s.GREY` is changed to rosybrown, then the backgrounds of all objects being displayed will change to rosybrown and objects subsequently shown will have rosybrown backgrounds.

There are, as said, three exceptions to this electrical connection. When the value of a *system* attribute default variable is changed, the value for a given attribute for a given object will *not* be changed under any of these circumstances:

- When it has a value different from the old value of the system attribute default variable (because, for example, `` `a has `bg `blue `` was executed). Note that if the value of the system attribute default variable is changed to agree with the value for this object, then if the value of the system attribute default variable is changed again, the value will change for this object also.
- When it is using an attribute default variable different from the default one for the object's class (for example, `` `a has `.bg `abg `` or `` `a s.uses `bg `abg `` having been executed).

- When a different variable has been made the attribute default variable for that attribute for the object's class, e.g. by executing `` `array s.Has `bg `arraybg``.

There are several ways to find the attribute default variable, if any, for an attribute for a class, such as:

```
     `.bg s.Of `array
`s.GREY
```

An attribute default variable can be removed for an attribute and class, as in:

```
     `array s.Has (`bg;)
```

The attribute default variable can be reset to the system attribute default variable in the same way it would be set to any other variable, e.g.:

```
     `array s.Has `.bg `s.GREY
```

There are ten system attribute default variables, as shown in the following table.

s-Context System Attribute Default Variables

Variable	Default Value	Application
s.BLACK	black	editbg, fg, labelfg, titlefg
s.BLUE	lightsteelblue3	rowbg
s.GREEN	mediumaquamarine	colindexbg, cornerindexbg, rowindexbg
s.GREY	grey	bg, editfg, selectbg
s.RED	lightsteelblue3	indexbg
s.YELLOW	gold	Hl
s.FONT	kaplgallant	font, labelfont, titlefont (all classes except graph)

S-Context Data Variables

s-Context Variable	Description
s.COLOR_NAMES[1]	A symbol vector holding color names. See "Colors" in the chapter on attributes.
s.COLOR_NUMBERS[1]	A three-column integer matrix holding rgb values for colors.

`s.FONT_NAMES`	Font names, returned as a character matrix.

1. For every index `i`, `s.COLOR_NAMES[i]` and `s.COLOR_NUMBERS[i]` refer to the same color.

Load s into your A+ session to see the values of these variables. A simple way to view `s.COLOR_NAMES` is

```
show ⊤s.COLOR_NAMES
```

You can also view `s.COLOR_NUMBERS` in this way, but it is better to show colors with `s.COLOR_NAMES`.

41. Display Classes

Objects and Classes

A+ applications are constructed from variables, bound and free. A bound variable, or *object*, is a global variable which has been tied to a display class through the `is{}` function:

```
a←⍳10
`a is `array
```

The display class specifies the general form of how the variable looks on the screen. Some of these characteristics, or *attributes*, can be changed through use of the `has{}` function:

```
`a has (`rows;10;`fg;`blue)
```

But even prior to binding, a variable has attributes. These include both the primitive A+ attributes of shape, type, and depth, which are properties of the data, and certain further attributes, such as foreground color, font, and callback on assignment. Together, these variable attributes are termed *persistent*, since they are attached to variables and survive re-binding to different display classes. By contrast, the attributes of any particular display class come and go with binding and freeing of global variables.

Subclasses

There are several kinds of hierarchical relation in A+:

- container and child;
- leader and follower;
- class and subclass.

The first two relations have been described in "Containers" and "Leaders and Followers". The topic of the present chapter is how to use the subclass relation.

A class is, minimally, a set of attributes. An object which is an instance of a particular class has those attributes. For example, a variable which "is" a table, "has" a certain number of fixed fields. The class `` `table `` includes the fixedfields attribute.

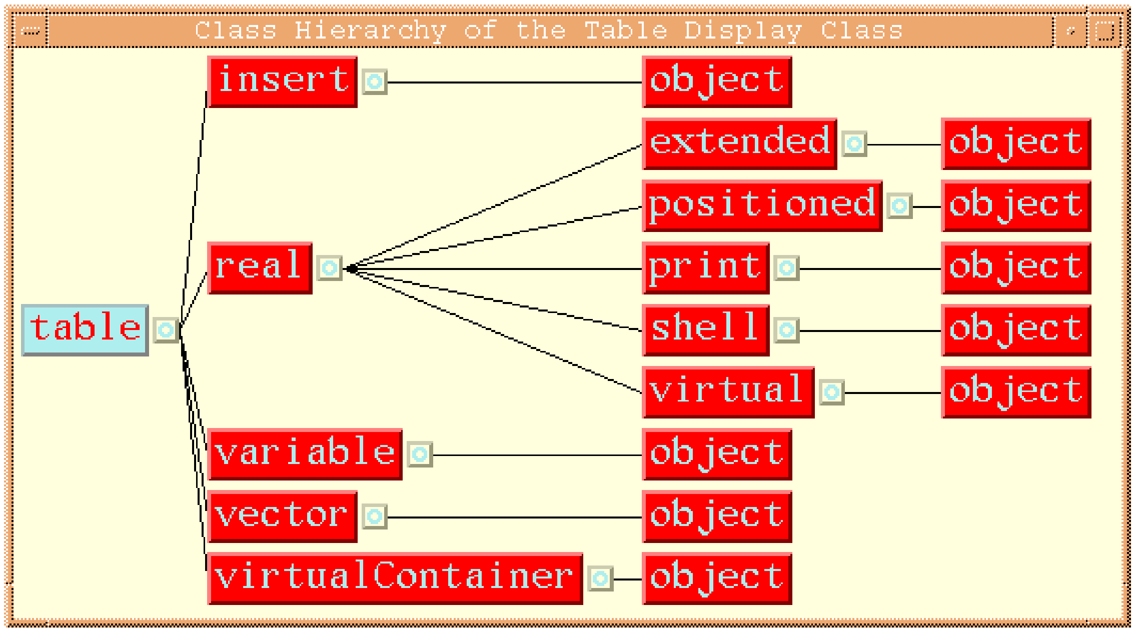
The attributes which belong exclusively to a class are called the *proper* attributes of that class. In general, a variable which has been bound to a class has many more attributes than are proper to that class, because the attributes of a class consist of the attributes proper to that class as well as the attributes of its superclasses, which it is said to *inherit*.

For example, a variable which has been bound to the `` `table`` class also has a background color `` `bg``, which is not a proper attribute of `` `table``. It acquires `` `bg`` by inheriting it from one of its superclasses, `` `real``.

The class hierarchy of `` `table`` is shown in the figure; compare this display with the result of the function `` s.superClasses{`table} ``.

A variable bound to `` `table`` has all the attributes of every class in this tree, since `` `table`` inherits all the attributes of its superclasses.

A Class Hierarchy:



Metaclasses

None of the superclasses of `` `table`` can be instantiated. That is, one cannot say:

```
     a←ι10
     `a is `real
```

The `` `real`` class is simply a convenient place to put certain attributes which many subclasses can access though inheritance. Classes such as this, which cannot be instantiated, are called *metaclasses*, and in A+ they significantly outnumber the classes which can be instantiated. The set of metaclasses of A+ is enumerated by the niladic function `s.metaClasses`:

```
     s.metaClasses{}
 `object `virtual `print ...
```

Virtual Classes

Not all of the instantiable classes in A+ can be directly instantiated. For example, one cannot say:

```
     a←ι10
     `a is `tableField
```

But one can say:

```
     t←`a
     `t is `table
     `class of `a
< `tableField
```

Classes such as `` `tableField `` can only be instantiated by binding a variable to some other class, in this case `` `table ``; that variable's children are then implicitly bound to the class in question. The `` `tableField `` class is called *virtual*, and A+ contains two such classes:

```
     s.virtualClasses{}
 `tableField `graphTrace
```

whose virtual parents are `` `table `` and `` `graph ``, respectively.

Real Classes

A class which can be directly instantiated is called "real". The set of real classes of A+ is enumerated by the niladic function `s.realClasses`:

```
     s.realClasses{}
 `slot `check `radio ...
```

Simple Subclassing

Suppose one has an application in which there are two kinds of arrays: A-arrays and B-arrays. And suppose that the application creates multiple instances of both kinds of array.

A-arrays should always have the following characteristics:
bg: green
fg: black
font: helvetica-bold-18

and B-arrays:
bg: black
fg: cyan
font: courier-bold-14

Define two subclasses of array, thus:

```
      `aArray s.isClass `array
      `bArray s.isClass `array
```

and set the appropriate class defaults:

```
      `aArray s.hasDefault (
            `bg;      `green;
            `fg;      `black;
            `font;    'helvetica-bold-18'
            )
```
and
```
      `bArray s.hasDefault (
            `bg;      `black;
            `fg;      `cyan;
            `font;    'courier-bold-14'
            )
```

So that:

```
      a←ι10
      `a is `aArray
      b←ι3 4
      `b is `bArray
      `class of `a
 < `aArray
      `class of `b
 < `bArray
```

In this case, subclassing allows the application to dispense with attribute bookkeeping.

Gadgets

As One of a Kind Objects

Consider the following "gadget":

```
      file{s;c;v}:
        {
        (c%`FILE)←(t≠' ')/t←Dir,'/',(0⊃`row of c,v)#c%v
        }
      files{dir}:
        {
        t←⌽@1 mat←sys.agetdents{dir};
        t←((t=' ')ι@1⊢0)⌽@0 1 t;
        (t[;ι2]≡@1 'm.')/mat
        }
      Dir←''
      Files:files{Dir}
      Filepick←(`Dir;`Files)
      `Files is `array
      `Dir is `scalar
      `Filepick is `layout
      `Files has (`protect;1; `refer;file)
      show `Filepick
```

Entering a directory name displays the `.m` files in that directory. Referring to an item in the display list stores that item in the global `FILE`. By setting the appropriate callback on `FILE`, we can use this gadget as a tool for displaying, printing, editing, or otherwise using the referenced file.

As Generator Functions

The single most important reason for recoding such a structure as a class is that we wish to be able to create multiple instances of it. Our first step in this direction should therefore be to define a function which generates such structures. Continuing the above example:

```
      filepicker{cx}:
        {
        (cx%`Dir)←'';
        (cx%`Files)←'';
        (cx,`Files) has (`def;'.files{',(⊤cx),'.Dir}');
        (cx%`Filepick)←(`Dir;`Files);
        (cx,`Files) is `array;
```

```
    (cx,`Dir) is `scalar;
    (cx,`Filepick) is `layout;
    (cx,`Files) has (`protect;1; `refer;file);
    cx,`Filepick
    }
```

Note that `filepicker{}` returns the symbol of the generated gadget, so that

```
  show filepicker{`my}
```

instantiates a filepicker in context `` `my``.

As Composite Classes

Here is an example of how the filepicker could be defined as a composite class, and then used as the control panel for a simple file editor. Continuing from above:

Filepicker instantiation function:

```
newFp{s;c;v}:
  {
  (d;f)←c%v;
  (c,f) has (`def;'.files{',(ᴛc),'.',(ᴛd),'}');
  (c%d)←'';
  (c,f) has (`protect;1);
  }
```

Composite class has layout structure and new function:

```
`fp s.isClass (`scalar;`array);
`fp s.hasDefault (`new;newFp);
```

Reference function for file list:

```
fileEdit{s;c;v}:
        {
        dir←(dir≠' ')/dir←c%0⊃%0⊃`parent of c,v;
        fil←(fil≠' ')/fil←(0⊃`row of c,v)#c%v;
        (c%`Edit)←1Inam←(if (0<#dir) dir,'/'),fil;
        (c,`Edit) has (`title;nam);
        show c,`Edit
        }

Edit←''
`Edit is `array
Directory←Files←''
EditFile←(`Directory;`Files)
`EditFile is `fp
`Files has (`refer;fileEdit)
show `EditFile
```

The right argument to `isClass{}` is a schema which obeys the semantics of (nested) layouts. The symbols in the schema are class names. So `` `fp`` will be a `` `layout``, with a `` `scalar`` above and an `` `array`` below.

The `` `new`` function `newFp{}` is called when a new instance of `` `fp`` is created. `newFp{}` retrieves the symbols of the children of the new `` `fp``, redefines the file list in terms of the directory variable, which it initializes, and sets protection and reference functions on the file list.

Rudimentary structural error checking is performed:

```
     u←v←w←''
     z←(`u `v;`w)
     show `z is `fp
⍝ !! .z: variable cannot be bound to fp
```

Extending the Class Hierarchy

The decision to use A+ subclasses rather than write functional gadget generators rests entirely with the programmer. Subclasses hide detail and guarantee uniformity of approach across applications, but can be tricky to design. Gadget generators fit more comfortably into the experienced A+ programmer's bag of tricks, but cannot exploit the inheritance and attribute-management mechanisms which subclasses provide. As an example, consider how the file editor application of the `` `fp `` class in the above example can itself be reconstructed as a subclass:

File editor instantiation function:

```
     newEfp{s;c;v}:
       {
       (d;f)←c%v;
       (c,f) has (`def;'.files{',(⍕c),'.',(⍕d),'}');
       (c%d)←'';
       (c,f) has (`protect;1; `refer;fileEdit);
       (c%`Edit)←'';
       (c,`Edit) is `array;
       }
```

File editor subclass and (overriding) new function:

```
     `efp s.isClass `fp
     `efp s.hasDefault (`new;newEfp)
     Directory←Files←''
     EditFile←(`Directory;`Files)
     `EditFile is `efp
     show `EditFile;
```

Synthetic Attributes

Making composite objects through the use of subclasses hides detail. In particular, we may wish to refer to attributes of the components through new synthetic attributes of the composite object. For example, in the composite layout of three arrays `` `array3 ``, we may wish to be able to set the background colors of the components without specifically addressing them through `has{}`:

```
      `array3 s.isClass (`array;`array;`array)
      g_bg{var}:⊃⊃`bg of¨(cx,¨⊃cx%var) ⊣ (cx;var)←var
      s_bg{var;x}:{
             (cx,¨⊃cx%var) has¨(`bg,¨x) ⊣ (cx;var)←var}
      `array3 s.classHas (`bg3;(g_bg;s_bg))

      a←b←c←⍳3
      d←(`a;`b;`c)
      show `d is `array3
```

```
     `d has (`bg3;`red `blue `green)

     `bg3 of `d
< `red `blue `green

     `bg3 s.ofClass `array3
< < g_bg
  < s_bg
```

To eliminate a synthetic attribute:

```
     `d has (`bg3;)
```

Subclassing from Virtual Containers

The children of `` `table `` and `` `graph `` are virtual objects, viz., `` `tableField `` and `` `graphTrace ``. Consider the case:

```
      `xTable s.isClass `table
      a←⍳10
      b←`a
      `b is `xTable
      `class of `b
< `xTable
      `class of `a
< `tableField
```

Since `` `xTable `` is a subclass of `` `table ``, its children are of class `` `tableField ``. Suppose, however, that we wish to define a new kind of table, whose children are themselves a subclass of

`` `tableField ``:

```
      `yTableField s.isClass `tableField
      `yTable `yTableField s.isClass `table
      a←⍳10
      b←`a
      `b is `yTable
      `class of `b
< `yTable
      `class of `a
< `yTableField
```

The left argument to `isClass{}` can be a simple vector of count 2, in which case the second symbol specifies the class of virtual children.

Metaclasses and Multiple Inheritance

A metaclass is a class defined strictly for the purpose of naming a set of attributes. Metaclasses cannot be instantiated.

For example, consider the case where one wishes to define subclasses `` `subtable ``, `` `subslot ``, `` `subarray ``, `` `subscalar `` of `` `table ``, `` `slot ``, `` `array ``, and `` `scalar `` (the input objects of A+) all of which make use of some set of attributes on input `` `r1 ``, `` `r2 ``, and `` `r3 ``. Suppose, moreover, that `` `s1 `` and `` `s2 `` are strictly attributes of `` `subtable `` and `` `subslot ``, `` `t1 `` and `` `t2 `` of `` `subtable `` and `` `subarray ``, and `` `u1 `` and `` `u2 `` of `` `subarray `` and `` `subscalar ``:

	r1	**r2**	**r3**	**s1**	**s2**	**t1**	**t2**	**u1**	**u2**

subtable	+	+	+	+	+	+	+		
subslot	+	+	+	+	+				
subarray	+	+	+			+	+	+	+
subscalar	+	+	+					+	+

Let us then define the metaclass `` `r `` as a subclass of `` `object ``, the most general metaclass in A+:

```
      `r s.isClass `object
```

and add to `` `r `` the attributes `` `r1 ``, `` `r2 ``, and `` `r3 ``:

```
      `r s.classHas (
            `r1;(g_r1;s_r1);
            `r2;(g_r2;s_r2);
            `r3;(g_r3;s_r3)
            )
```

Similarly, the metaclasses `` `s ``, `` `t ``, and `` `u `` to name their respective sets of attributes:

```
      `s s.isClass `object
      `s s.classHas (
            `s1;(g_s1;s_s1);
            `s2;(g_s2;s_s2);
            )
      `t s.isClass `object
      `t s.classHas (
            `t1;(g_t1;s_t1);
            `t2;(g_t2;s_t2);
            )
      `u s.isClass `object
      `u s.classHas (
            `u1;(g_u1;s_u1);
            `u2;(g_u2;s_u2);
            )
```

Finally, the subclass definitions:

```
      (`subtable;`r `s `t) s.isClass `table
      (`subslot;`r `s) s.isClass `slot
      (`subarray;`r `t `u) s.isClass `array
      (`subscalar;`r `u) s.isClass `scalar
```

If the left argument of `isClass{}` is nested, then the second item is a vector of metaclasses.

Also note that if `v` is a vector of metaclasses mentioned in the left argument, and `` `a`` is an attribute of several classes which appear in `v`, then the `g_` and `s_` functions which actually attach to the subclass being defined are the ones which attach to the class which appears *earliest* in the vector. More generally, if `` `a`` appears as an attribute in the class from which subclasses are being derived, then the `g_` and `s_` functions of the superclass override those of `` `a`` in the specified metaclass vector (if any).

42. Callback Functions

Variable callbacks are functions associated with global variables, including dependencies, (or, indeed, functions or names that do not yet have a value) that are called just before or after Assignments or certain Selective Assignments are made to those variables. Variable callback functions are the subject of this chapter and will usually be spoken of as just "callback functions". Callbacks are particularly important for asynchronous events, such as in screen management in the `s` context, for tracking user actions; see "Interprocess Communication: adap", and "Introduction to Screen Management" and the chapters that follow it.

A mapped file `mf` can have an associated callback, but that callback is associated with the variable `mf`, not the underlying file. A change to the file through another mapped variable, in the same process or not, or by any other means than an assignment to `mf` will not trigger a callback for `mf`.

In A+, variable callback functions (as distinct from event callback functions) have the same basic syntax (variations are discussed below):

```
    cfn{s;d;i;p;c;v}
```

For example, if the function `cfn` is associated with the global variable `price` defined in the context `comp`, then when `price` changes, `cfn` is automatically called with arguments supplied by A+:

1. `s` is static data (see below);
2. `d` is the new value, i.e., the value of the righthand side of the Assignment triggering the callback;
3. `i` is an index or set of indices (left argument to Choose - see `p`);
4. `p` is a path (left argument to Pick) such that `i#p⊃price` is the subarray that changed;
5. `c` is the name of the context, in the form of a symbol, `` `comp``; and
6. `v` is the name of the global variable in the form of a symbol, `` `price``.

When `cfn` is called because of an Assignment of a global variable, the change to that variable can be described in terms of the arguments to `cfn` as

`(i#p⊃c%v)←d`	if `i` is nested, or
`(i#,p⊃c%v)←d`	if `i` is simple, or
`(p⊃c%v)[,]←d`	if `i` is out of range for `c%v` before this assignment.

The last case involves the Append form of Selective Assignment and `i` is the set of indices in `p⊃c%v` (after the assignment) of the appended elements.

The first argument of a callback function is called the *static data*. Static data is specified when the association between a variable and a callback function is established. Typically, the static data is any constant extra information needed to process the change in the associated variable.

A callback function can have anywhere from zero to six arguments, but no matter how many it has, their meaning from left to right is the same as above. For example, if `f{a;b;c}` is a callback function, then `a` is static data, `b` is new data, and `c` is an index.

The other type of callback is the event callback. Event callback functions have the basic syntax `cfn{s;c;v}` where the names `s`, `c`, and `v` have the same significance as above. An event callback function may have from zero to three arguments; the arguments are positional. See the chapters mentioned in the first paragraph of this chapter and especially "Attributes with Callbacks".

Setting and Causing Callbacks

A+ provides the Set Callback (`_scb`) and Set Preset Callback (`_spcb`) system functions for establishing and removing the association between a global variable and a callback function. Set Callback is discussed here, and Set Preset Callback below. The left argument is a symbol naming the global variable. The right argument is a pair of the form `(fcn_name;static_data)`, where `static_data` is the static data (perhaps Null), and `fcn_name` is a function expression, a name entered without backquote or quotes. Thus the first element of this argument, with the enclosure by strand, is a function scalar and not a name, so this association is unaffected by any later changes to the meaning of the name of `fcn_name`. For example:

```
     cbf{s;d;i;p;c;v}:↓(s;d;i;p;c;v)    ⍝ A callback function.
     `a _scb (cbf;'-- a --')    ⍝ Call cbf whenever a is specified.


     a←92         ⍝ Specify a and see the arguments to the callback function.
< -- a --         ⍝ Static data s
<  92             ⍝ New data d
<                 ⍝ Index i is null.
<                 ⍝ Path p is null.
< `               ⍝ c names the root context.
< `a              ⍝ v names the variable a

     a←10 20 30 40    ⍝ Set a and see args to cbf
< -- a --
<  10 20 30 40
<
<
< `
< `a

     a[1]←200         ⍝ Indexed Assignment.
< -- a --
<  200
<  1
<
< `
< `a

     `a _scb (;)      ⍝ Remove the callback on a and
     a←999            ⍝ see no callback.
     a
 999
```

```
      $cx ctx          ⍝ Set new context.
      b←(`scalar`vector`matrix; (3.14; 'abcdef'; ⍳3 2))
      `b _scb (.cbf;'-- b --')  ⍝ Call cbf when b changes.

      (1;0)#`matrix⊃b       ⍝ Pick-Choose 2 from the matrix.
 2

      ((1;0)#`matrix⊃b)←22  ⍝ Change the 2 to a 22.
<  -- b --
<   22
<  < 1                    ⍝ Index (1;0)
   < 0
<   `matrix               ⍝ Path `matrix
<  `ctx
<  `b

      (1;0)#`matrix⊃b       ⍝ Pick-Choose 22 from the matrix.
 22
```

Selective Assignment Causes Callbacks

Any form of Selective Assignment can cause a callback function to be executed. Taking the last example from the previous section and continuing:

```
      (1 0/b)←<`Scalar`Vector`Matrix
<  -- b --
<  < `Scalar `Vector `Matrix
<  0               ⍝ Index is 0.
<                  ⍝ Path is Null.
<  `ctx
<  `b
      b            ⍝ And the change was made.
<   `Scalar `Vector `Matrix
<  <   3.14
   <   abcdef
   <     0  1
        22  3
         4  5
```

The next example shows an Append Selective Assignment triggering a callback:

```
      x←⍳10
      `x _scb (.cbf;'-- x --')
      x[,]←100 200
<  -- x --         ⍝ The callback was triggered.
<   100 200
<  <   10 11
<
<   `ctx
<   `x
```

Avoiding Callbacks

There may be times when you want to avoid callbacks - for example, when you are clearing a suspension and with normal execution many more suspensions would occur because of errors in callback functions. You can use the `$Sf` command for that purpose. See "Callback Flag" and note the warnings there.

When Callbacks Occur

It is important to know that callbacks established by `_scb` are called *after* the associated global variables have changed. This can be verified by defining a callback function that displays the value of the associated variable. When the function is called, the new value of the variable will be displayed, not the old. Note that the callback argument `d` is not necessarily this new value; `d` is, rather, the explicit result of the Assignment.

An ordinary callback function on a dependency is not called when the dependency definition is evaluated, although a preset callback function is. Similarly, during a callback on a dependency, the dependent variable's value is not marked invalid by any change the function makes in a variable on which the dependency depends.

Preset Callbacks

It is also possible to establish callbacks that are called just before the value of a variable is changed, so that the change can be validated. To do so, use the system function `_spcb`, which takes the same arguments as `_scb`.

The differences between callback functions established by `_spcb` and `_scb` are:

- preset callback functions on dependencies are called when either a dependent variable is assigned a value or the dependency is evaluated, whereas callback functions are called only when the dependent variable is assigned a value;
- `_spcb{y;x}` causes the callback function specified in `x` to be called *just before* the value of the global variable named in `y` is changed;
- preset callback functions are used to validate new values for global variables and they therefore must return meaningful results (namely, the validated data), while callback functions need not, their explicit results being ignored;
- for the Append form of Selective Assignment, the indices `i` are not valid for a preset callback, since the appending has not yet been done.

Since preset callbacks are used for validating new values of global variables, the following rules should be followed in their definitions:

- if a new value is valid, set the result of the preset callback function to that value;
- if the indices `i` are not null, do not change the shape of `d` or you will get a length error report;
- if a new value is invalid, signal an error (see "Signal"); the value of the global variable will remain unchanged.

Callbacks during Dependency Evaluations

The evaluation of a dependency can trigger a callback on the dependent variable, one that was established by the Set Preset Callback function - but not one that was established by the Set Callback function. The callback occurs after the definition body has been evaluated and before the dependent variable is set. The second argument to the callback is the result of evaluating the definition body, and the result of the callback is the value to which the dependent variable is set - or to which the items of the dependent variable are set, in the itemwise case.

Moreover, when a dependency is evaluated, callbacks of either kind can be triggered by assignments to other variables during the evaluation. These callbacks may mark saved values of other dependencies invalid.

A dependency that is currently being evaluated, however, or indeed a dependency whose callback function is being executed is marked as being in that state, and so its value will not be marked invalid while the evaluation or execution is going on.

Evaluation of a dependency is not complete until all callbacks, including any on the dependent variable, have been finished. See "Dependencies" for an example.

Callbacks during Protected Execute and Monadic Do

An error that occurs during the execution of a callback within the execution of a Protected Execute or Monadic do is reported in the result of the protected execution.