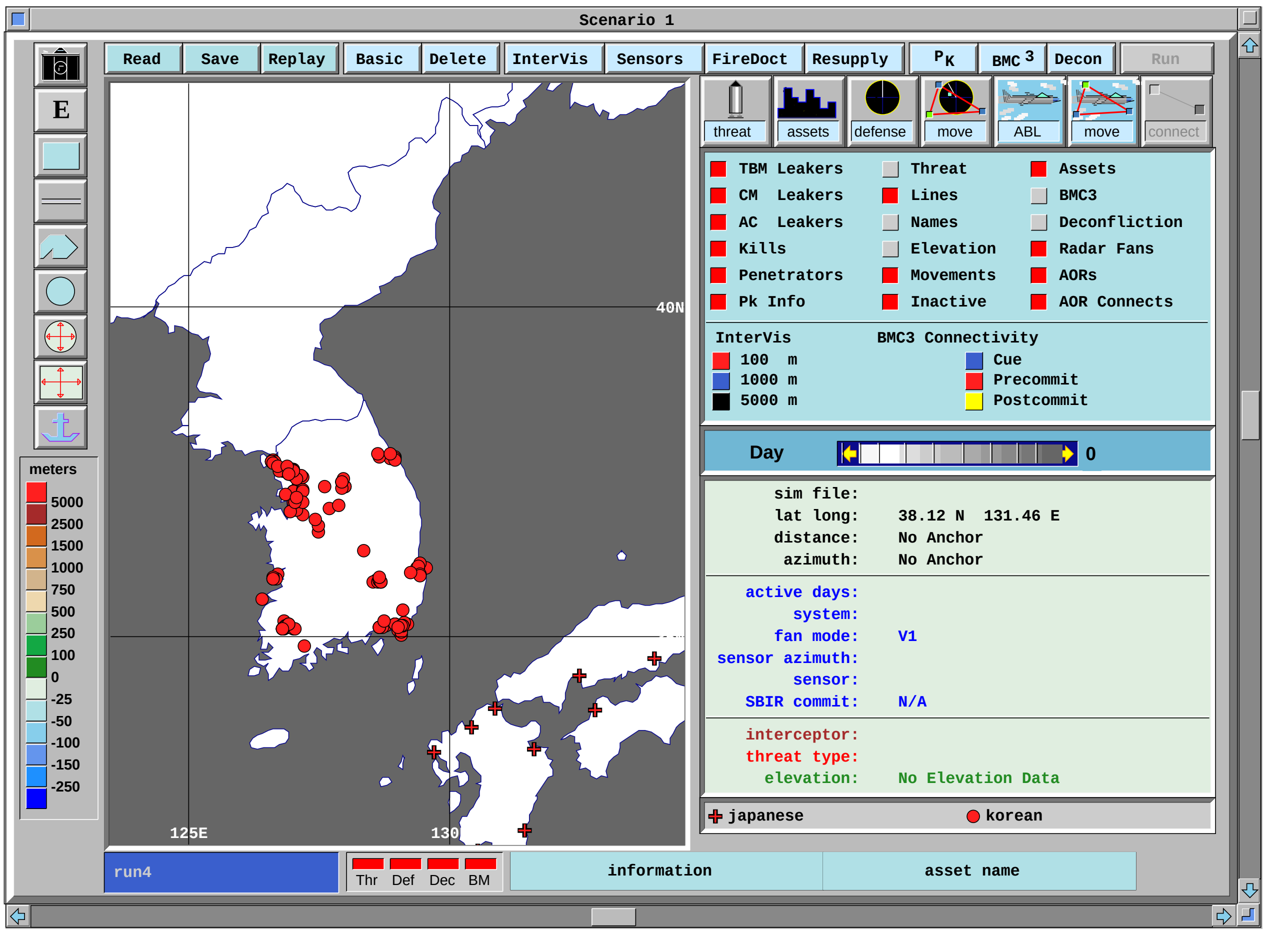
H^{v} Programming Manual

David P. Heddle
Christopher Newport University
and
The Thomas Jefferson National Accelerator Facility†



†. Formerly known as the Continuous Electron Beam Accelerator Facility (CEBAF)

Cover picture: An H$\underline{\mathbf{v}}$ view from *caps*, an application with a GUI built using the H$\underline{\mathbf{v}}$ library. *caps* analyzes missile defense technology and system architectures. Courtesy of Sparta, Inc. of McLean Virginia.

Document Date: May 31, 1996

UNIX is a registered trademark of AT&T.

The X Window System is a trademark of the Massachusetts Institute of Technology.

OSF/Motif and Motif are trademarks of Open Software Foundation, Inc.

Macintosh is a registered trademark of Apple Computer Inc.

Postscript is a registered trademark of Adobe Systems Inc.

Framemaker is a registered trademark of Frame Technology Corporation.

The Southeastern Universities Research Association (SURA) operates the Continuous Electron Beam Accelerator Facility (CEBAF) for the United States Department of Energy under contract DE-AC05-84ER40150.

Questions, comments and bug reports concerning $H^{\underline{v}}$ should be directed to:

heddle@cebaf.gov
(804) 249-7432

SECTION 1 Overview

SECTION 2 H$\underline{v}$ Programming

SECTION 3 Reference

SECTION 1 Overview

Hv
Chapter:

1 Introduction

In this chapter, we provide a general introduction to Hv. Details will follow in later chapters. Occasionally we will provide examples that make explicit reference to an Hv function. This is done with the understanding that, while the function has not yet been officially introduced, its purpose will be clear from the context. Details such as argument lists and return types should be ignored throughout this introduction.

1.1 What is Hv?

Hv (from Hot Views) is a library that greatly simplifies the development of applications with a sophisticated graphical user interface. It is layered on top of X, Xt (R4 or later), and OSF Motif (version 1.1or later); however, the developer is completely insulated from those libraries. Applications can be developed solely through calls to the Hv library. Hv is *not* an interface builder for creating Motif applications; it is a library whose use results in applications with a unique appearance and feature set. While Hv is in part a tool for simplifying Motif and X, it has extended capabilities not available in either package. In particular, Hv is useful for scientific and engineering applications including modeling and simulation.

Hv is also to be distinguished from data visualization packages, such as the AVS and Explorer systems. Hv is used, for the most part, to display and interact with representations of physical objects rather than complex data sets, although visualization at the level of scientific plotting (usually of the *results* of a simulation) is an integrated Hv feature. This will become clear as we discuss example Hv applications.

The specialization has a price. To simplify the programming, some of the flexibility of X, Xt and Motif was sacrificed. For example, Hv limits the application to a single display, with the benefit being that a display "variable" never appears in an argument list for an Hv function. Hv will be of use in modest developments where the manpower to create full-blown X/Motif applications is not available, and especially in scientific and engineering applications *where the developers are the users* and have little interest in facing the steep X/Motif learning curve.

Note!

The full power of the X, Xt and Motif libraries are still at your disposal. However, none of the examples described in this manual make any direct calls to these "lower-level" libraries.

As a simple but important example of why H$^{\underline{v}}$ is particulary useful for scientific and engineering applications, we note that H$^{\underline{v}}$ maintains a floating-point *world* coordinate system as well as the pixel based *local* system used by X and Motif. The world system is sized to accommodate the dimensions of what is actually being drawn. For example, a display of detectors in a nuclear physics laboratory might require a world system that is several meters in extent. With H$^{\underline{v}}$, there is no need to artificially pre-scale physical objects to match a screen-based pixel system. The graphical items carry along the actual dimensions and positions of the objects they represent.

Another example of an H$^{\underline{v}}$ feature that is especially useful for technical applications is *pointer-tracking* (which, from this point on, we shall usually refer to as *feedback*). As the user moves the pointer over graphical items, the application will display information about the item. Returning to the detector example, when the user points at a detector component, the application will display in real-time the ID and location of the detector component, as well as any data presently available that pertain to the item. H$^{\underline{v}}$ automates this as much as possible for the developer by continuously reporting both which item is being pointed at and the pointer's present pixel *and world* coordinates.

H$^{\underline{v}}$ also provides a mechanism for *registering* multiple simulations. One application can have many simulations running simultaneously, each being updated at a different (user controllable) rate. By registering a simulation, the developer is guaranteed that a private simulation process will be called at programmable time intervals. For example, the demo application *hvmap* included with the H$^{\underline{v}}$ distribution displays a rotating globe. The rotation is handled by registering a simulation, called once per second, that recalculates the map configurations based on the earth's rotation.

H$^{\underline{v}}$ was developed jointly at the Thomas Jefferson National Accelerator Facility (TJNAF)[†] and Christopher Newport University.

1.2 The H$^{\underline{v}}$ Paradigm

An H$^{\underline{v}}$ application looks something like a cross between a Motif and a Macintosh application. It contains a single X window, (henceforth called the H$^{\underline{v}}$ *main window*) with a single main menubar, much like the Macintosh desktop. However, unlike the Macintosh desktop the H$^{\underline{v}}$ main window can be moved, resized, and iconified.

Within the H$^{\underline{v}}$ main window are additional window-like objects called *views*. Again, these views are like the windows on the Macintosh desktop, except they have increased functionality: dragging, resizing, pointer tracking (feedback), scrolling, zooming, zoom-to-fill, controls, etc. The views have a 3D sculptured appearance that provides an elegant look-and-feel to H$^{\underline{v}}$ applications. An example of a H$^{\underline{v}}$ main window containing multiple views is shown in Fig. 1.

Note!

Some discussion of the figures in this document is appropriate. H$^{\underline{v}}$ provides for postscript and encapsulated postscript rendering of its views. In fact, that is how many figures in this document were produced. Thus, the representations of views are accurate, except that you probably printed this document in black and white, which does not do justice to their appearance. Dialog boxes, on the other

†. Formerly known as the Continuous Electron Beam Accelerator Facility (CEBAF).

hand, are created with *Motif*, and H^V doesn't know how to print them. Pictures of Motif dialog boxes in this manual were screen-captured. The window surrounding the views, when present in a figure, is purely notional. Mentally insert the style of your window manager of choice. Note that the notional window in Fig. 1. is drawn quite small in order to fit into this document; in a typical application on a workstation the views will have a lot more room to maneuver.

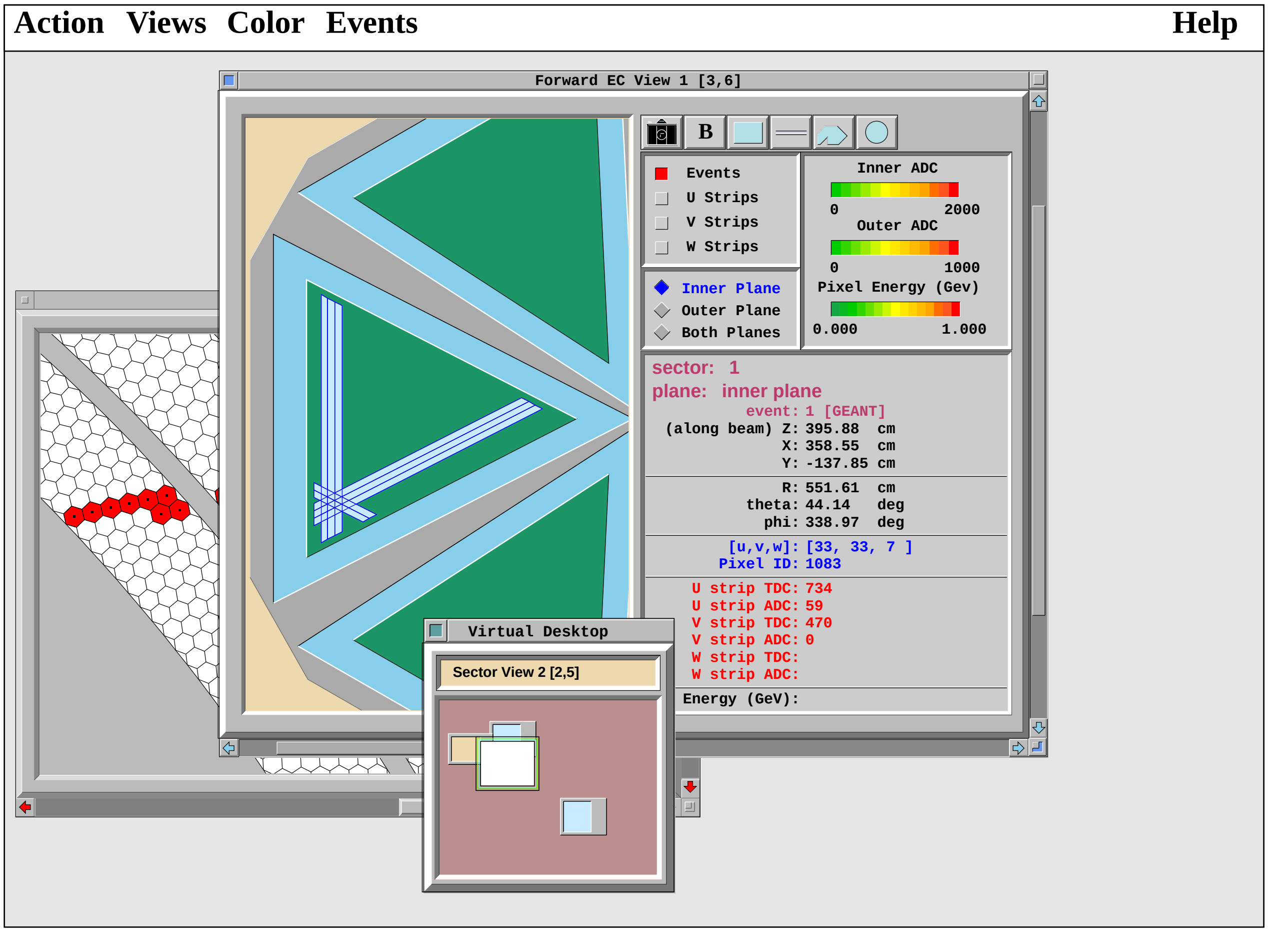


FIGURE 1. A example of a main window from an Hv application. Three over-lapping views are present. The main window and menubar are simple notional representations.

Contained in each H^V view are H^V *items*. These include but are not limited to sliders, buttons of various flavors, rainbow scales, wheels, text, and, most importantly, *user-defined* items that represent real objects, such as detectors. Each item has a rich suite of *attributes* that can be used to control its behavior. To reiterate: the H^V main window contains one or more H^V views; each H^V view contains many H^V items. The H^V items are *not* based on widgets, either Xt or Motif. They are completely "homegrown".

An H$^{\underline{V}}$ view has three distinct areas, each being optional but typically all three being present. The *HotRect* is the primary “drawing” area where the bulk of the graphical items will live. The *Control* area is where sliders, buttons (of various flavors), and other control items are found. Finally the *Feedback* area is where updates associated with tracking the pointer are displayed.

In Fig. 2 we display a typical view arrangement, designating the controls, HotRect, and feedback areas. Also in Fig. 2 we point out a familiar set of manipulators for dragging, resizing, hiding, and scrolling an H$^{\underline{V}}$ view.

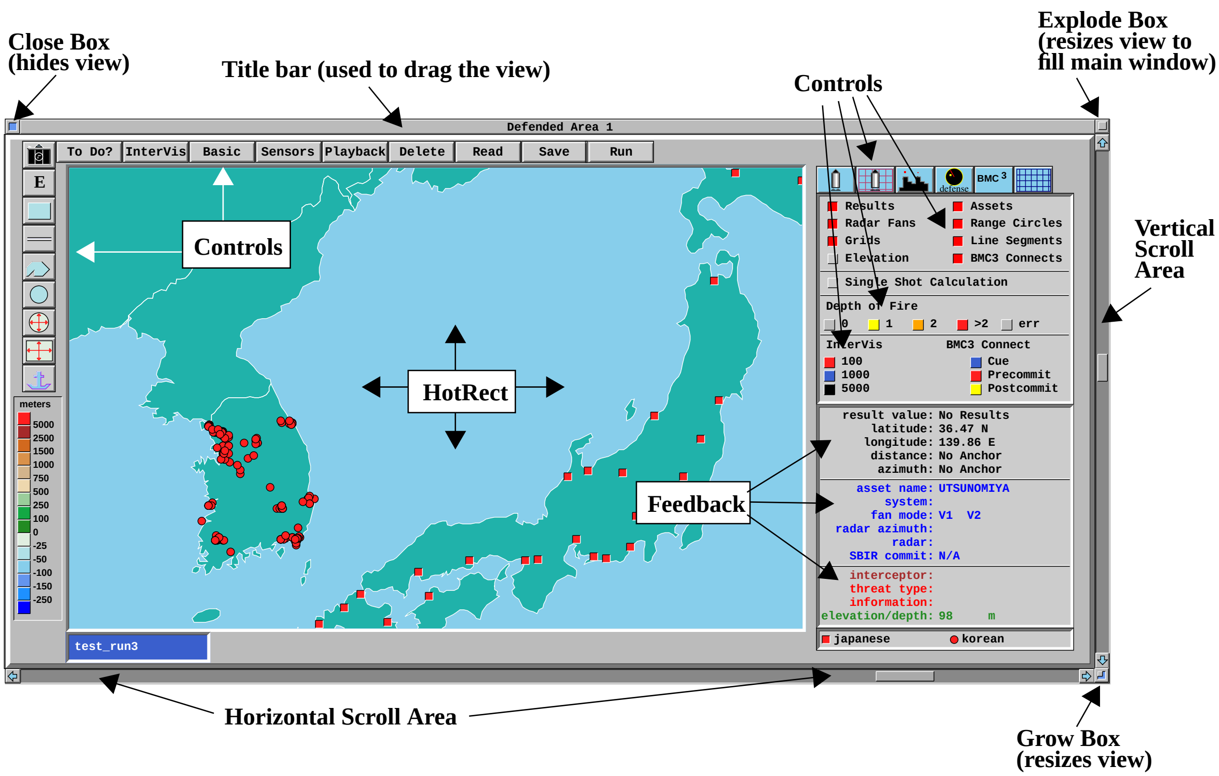


FIGURE 2. The anatomy of an Hv view. This view could be described as following the canonical Hv style: Drawing tools (if desired) to left of the HotRect, textual buttons along the top, picture buttons and other controls on the top right and the feedback area on the bottom right. (Courtesy of Sparta, Inc.)

It is essential to understand the purpose of each area. The HotRect is the “canvas” for the view. Items, usually very application-specific representations of actual objects, will be dragged, resized and rotated within the HotRect. Results may be represented graphically and displayed within the HotRect. It is the area affected by scrolling and zooming. In most applications it will change frequently.

The Control area is relatively static, and it is also the only of the three areas that may be comprised of disconnected regions of the view. It gives the view its personality in the sense that the controls are specific to that view, as opposed to the main menubar which applies to all views. The control area is not affected by scrolling or zooming. A typical application will have a small family of view types, each with a specialized set of controls. *In $H^{\underline{V}}$ applications the bulk of the power to control the state of an application resides in the views' controls, not in the main menubar, which tends to remain relatively simple.*

The Feedback area is where the results of pointer tracking are displayed. In most $H^{\underline{V}}$ applications, as the pointer is dragged through the HotRect, the text in the feedback area will be rapidly changing. The user stops at a point of interest and reads the information in the feedback area. Pointing at items in the HotRect and getting instant feedback is perhaps the most popular $H^{\underline{V}}$ feature. In any of the figures in this report that display $H^{\underline{V}}$ views you can find examples $H^{\underline{V}}$'s feedback mechanism.

The size and placement of these areas is at the discretion of the developer. For example, the view shown in Fig. 3 has a very different layout than the view in Fig. 2, since its parent application is less demanding in its need for controls and feedback. The arrangement of Fig. 3 minimizes wasted screen real estate.

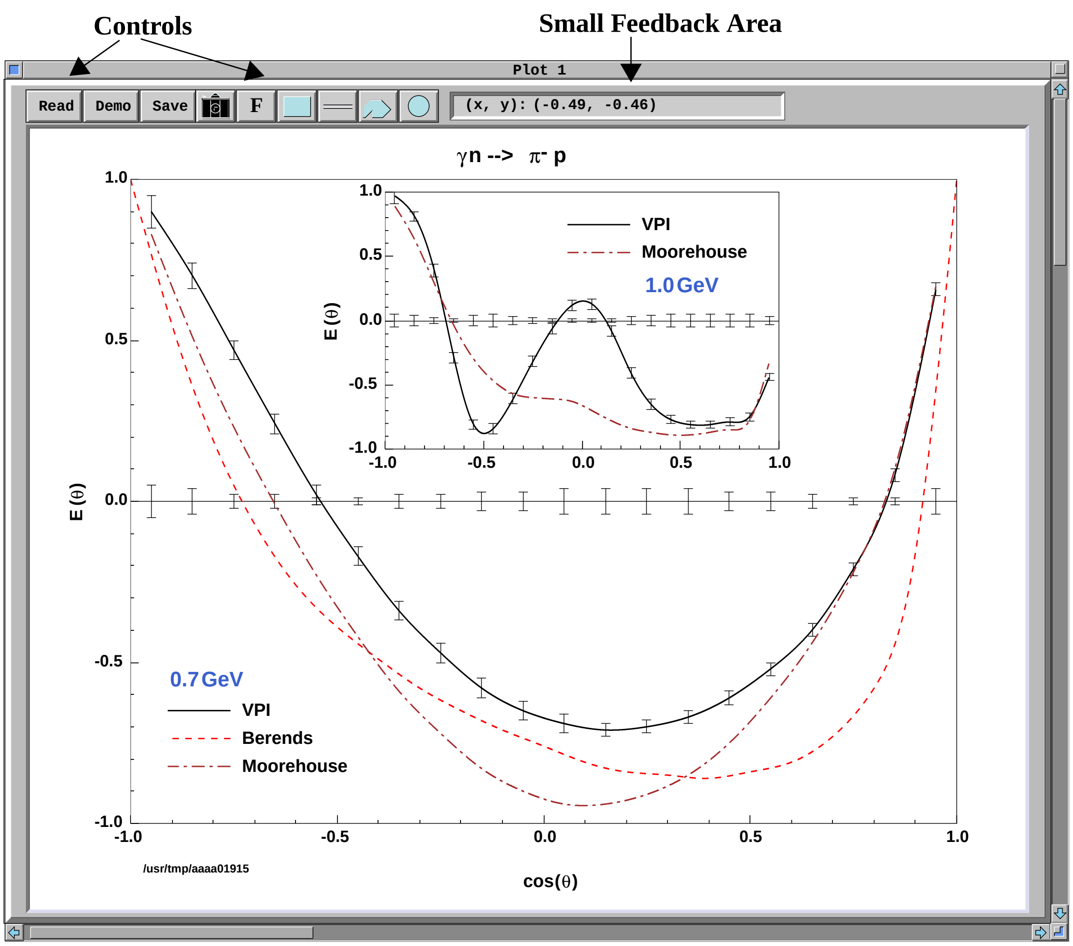


FIGURE 3. An example of an Hv view with very different layout from the one shown in Fig. 2. Here only minimal controls and feedback are required and are placed along the top. This allows the maximal allocation of screen real estate to the HotRect.

We developed H$^{\underline{v}}$ this way because we *like* the Macintosh paradigm: a single menubar controlling a number of windows (views). It is simpler than the X model, wherein either each document window of an application has a copy of the *same* menubar, or there is a separate "executive" window. Under H$^{\underline{v}}$, views do not get lost behind unrelated application windows. (At least not singly, it is of course possible to lose them all at once by occluding the H$^{\underline{v}}$ main window). Any special needs that a view has, which cannot be addressed by the generic main menubar, can be handled through use of either the customizable popup menu attached to each view (See 2.5.3, page 15) or by adding appropriate control items to the view.

Note!

Motif is used internally by H$^{\underline{v}}$ for three purposes: Menus , dialogs, and compound strings. The views and items, which are the heart and soul of an H$^{\underline{v}}$ application, make no use of Motif. They are drawn via calls to the low level *X* routines. The one exception: the H$^{\underline{v}}$ *TextEntryItem*, which places a Motif widget on top of an H$^{\underline{v}}$ item.

In effect, H$^{\underline{v}}$ is (among other things) a window manager. Its purview is the H$^{\underline{v}}$ main window (which is itself maintained by some other window manager, such as *mwm*.) The H$^{\underline{v}}$ views are its windows. However, a lot of the maintenance that is usually not provided by a window manager is provided by H$^{\underline{v}}$. Most importantly, H$^{\underline{v}}$ will handle all the *exposures*, freeing you from that tedious job.

Note!

When part of a window is occluded and then exposed, the windowing system (e.g. X) will send to the application an "exposure" event, telling it that it must redraw the previously hidden part. Most novice programmers are surprised to discover this, undoubtedly thinking that the "system" would handle this by remembering the contents of every window (which does occur to a limited extent on some platforms). The programmer faces the daunting task of redrawing the exposed part. For any serious graphics application, an enormous amount of effort is brought to bear in optimizing this process, so that the redrawing can be performed efficiently and seemlessly. In H$^{\underline{v}}$, it is handled automatically.

An H$^{\underline{v}}$ view by itself is little more than a rectangle (albeit a nice 3D sculptured one) that can be dragged around and resized. What makes the view useful is that it contains and manages lists of H$^{\underline{v}}$ *items*. There is a set of predefined items such as buttons (of several varieties), text, and sliders, and others that we will meet presently. More importantly, H$^{\underline{v}}$ allows you to define your own items, which may or may not have some of the standard items embedded within. At a minimum, all you have to do is write a procedure for drawing the item. H$^{\underline{v}}$ also helps here with a complete suite of drawing routines so that direct calls to low level X drawing routines should not be necessary.

Note!

One more point about the figures. In Figures 1-3, the view is shown as it appears on the screen, namely with three areas on a gray 3D sculptured background. Fig. 14, however, shows only the contents of a view's HotRect on a clean "white" background. This is usually what users want to print, just the *contents* of the HotRect. Printing the entire view is useful when producing a user's manual, so that one can refer to the controls when discussing their use. When a print request is made from an H$^{\underline{v}}$ application, the user can choose whether to print the entire view with all adornments or just the contents.

1.3 H$^{\underline{v}}$ Freebies

H$^{\underline{v}}$ provides the following services/resources for all applications.

- *Drag and Drop.* H$^{\underline{v}}$ will allow you to create items that can be dragged, resized and rotated, usually within the confines of the HotRect.
- *Drawing Tools.* H$^{\underline{v}}$ provides a set of drawing tools that can be optionally added to any view. Thus any view can be annotated with text, lines, rectangles, etc. prior to printing.
- *Scientific Plotting.* H$^{\underline{v}}$ provides a high powered scientific plotter that can be integrated into any H$^{\underline{v}}$ application. The plotter includes spline smoothing and curve fitting of various types.
- *Simulations.* Each H$^{\underline{v}}$ view in an application can register a simulation by providing a *simulation process* and a *simulation time step* in milliseconds. H$^{\underline{v}}$ will automatically call the simulation process at intervals of the time step, for example every 100 ms. Thus an application could have the same simulation running independently in multiple cloned H$^{\underline{v}}$ views, or a completely different simulation in each H$^{\underline{v}}$ view.
- *Postscript printing.* Any application will be able to print its views to a postscript printer or to create postscript (or encapsulated postscript) files for inclusion into other documents. This is true even for user-defined items providing the user supplied drawing routine (See 7.8, page 68) uses only H$^{\underline{v}}$ drawing functions (as opposed to X functions).
- *Balloon Help.* Any item created by an application can include a string that will appear in a balloon when a) the user has selected “Show Balloons” from the **`Help`** menu and b) the pointer is placed over the item.
- *On-line help.* When the user selects “Help” from the **`Help`** menu, a scrollable list of topics (automatically alphabetized) is presented. Clicking on an item brings up any available help. Developers only need to maintain a simple ASCII file with the help text (See 2.5.3.3, page 17).
- *Fonts and colors.* H$^{\underline{v}}$ provides 37 predefined fonts for use in H$^{\underline{v}}$ functions that take “font” as an argument, and 80 colors for use in H$^{\underline{v}}$ functions that take “color” as an argument. Some color manipulation routines are also provided.
- *Animated Items.* Any H$^{\underline{v}}$ item can be assigned a redraw time, for example it can be told to redraw itself every 500 ms, from which the developer can effect simple animation.
- *Function Key registration.* The `F1` to `F10` function keys can be registered to call user procedures. This is often done to provide diagnostics. H$^{\underline{v}}$ pre-registers several function keys (See 2.3, page 12) which can be overridden by the developer.
- *Virtual Desktop*. An application can (optionally) request a virtual desktop, which allows you to effecively increase the size of your main window. A viewport is provided to move about the virtal desktop.

1.4 Requirements

H$^{\underline{v}}$ requires:

- the Motif library (not just the Motif window manager, which in fact is not required), version 1.1 or higher.
- a color monitor that supports 256 colors or gray scales. H$^{\underline{v}}$ takes full advantage of color, and we lack the manpower to insure that it does something reasonable on monochrome monitors.

1.5 Limitations of the H$^{\underline{v}}$ library

As mentioned previously, a partial purpose of H$^{\underline{v}}$ is to avoid direct use of Motif and X. There is always a trade-off in such efforts. The impressive flexibility of Motif and X is sacrificed to permit simpler access to a subset of their capabilities. We hope that we have captured what most applications require most of the time. For those other cases, you can still make direct calls to Motif and X. Here is a list of the recognized H$^{\underline{v}}$ limitations:

- The single H$^{\underline{v}}$ main window. This is something that could be changed; we have not found it to be a hindrance. (Keep in mind that any number of views can appear on the single main window).
- The single display. Presently, you cannot create multi-headed applications with H$^{\underline{v}}$.
- The requirement of 8-bit color. Considering the proliferation of color displays and X terminals, this should not be a serious limitation. Part of the appeal of H$^{\underline{v}}$ is its elegant appearance. We do not have the manpower to try to simulate the effects with dithered monochrome patterns.
- Limited colors and fonts. The predefined colors and fonts free the developer from the details under which they are allocated and stored in X. However there is presently no mechanism for extending these sets. One cannot, for example, use the H$^{\underline{v}}$ drawing functions to fill a rectangle in LemonChiffon, one of the many colors in the X database.

1.6 The Big Picture: is H$^{\underline{v}}$ useful?

H$^{\underline{v}}$ was developed for scientific and engineering applications wherein part or even most of the purpose of the application is to produce *and interact with* some sort of graphical output. The interaction can occur in the usual way with controls (buttons, etc.) but also in a less familiar way: by placing the pointer over (and perhaps clicking) non-control items such as pieces of a physics detector or a map of the world.

This concept might be clarified by a synopsis of some existing H$^{\underline{v}}$ applications, given below. You will see pictures from these applications used as examples in various places in this manual.

***ced* (cLAS eVENT dISPLAY.)** This is the first H$^{\underline{v}}$ applications. *ced* displays multiple representations of the CEBAF Large Acceptance Spectrometer (CLAS), overlaying the raw or reconstructed (or both) data. The user can place the pointer on a detector component and get immediate feedback (in the *Feedback Area* of the view) about the data for that component. If it means anything to you, that would include such information as the raw electronic data, or perhaps reconstructed information such as the energy deposited.

hvplot. While seemingly relegated to a role as a demo, *hvplot* is actually the most widely distributed and used H$^{\underline{v}}$ application. Scientific plots are generated from ASCII data files. The user can then interact with the plots in a variety of ways. Individual plots (there may be many in a view) can be zoomed, the plot and its labels can be clicked on and edited, data can bet fit in a variety of models.

tigris. Another CLAS related application. Like *ced*, *tigris* provides a graphical representation of the detector. In this case, however, the user clicks on various components to program a “trigger”, which is a description of what type of events you want the data acquisition system to record in an ensuing experiment.

caps. This is the first commercial use of the H$^{\underline{v}}$ library. SPARTA, Inc. developed a theater missile defense simulation by combining a FORTRAN “engine” with an H$^{\underline{v}}$ graphical interface. Objects such as radars and interceptors are placed on maps and can then be dragged and rotated.

zplot. This is an particle accelerator related application under development at CEBAF. It will query a device server for real-time information about devices and their attributes (e.g. magnets and currents) and display selected attributes superimposed on a layout of the beamline.

These applications share the feature of requiring, or at the very least providing for, extensive user interaction with *complex, user-defined* graphical objects. Nuclear physics detector components and radar fans are not part of any standard widget set. With H$^{\underline{v}}$, the developer can turn such objects into dynamic graphical entities in a straightforward manner.

For applications similar to one of the four described above, where users click and in some sense probe and/or modify specialized graphical items, or interact with the graphical representation of the data resulting from an analysis or simulation, then H$^{\underline{v}}$ should be an appropriate tool. For applications requiring only normal controls, such as buttons, file-selection boxes, sliders, etc., well known control toolkits such as Tcl/Tk might better serve the developer.

1.7 Obtaining the H$^{\underline{v}}$ Library and demos

H$^{\underline{v}}$ has been tested on the following platforms:

- DECStation 5000 series (Ultrix)
- HP 9000 Series (HPUX)
- SUN sparc
- DEC Alpha (OSF)
- IBM 6000 Series (AIX)
- SGI Irix
- Linux (on a Pentium)

H$^{\underline{v}}$ is freely available without warranty. Developers may use H$^{\underline{v}}$ providing they give proper attribution and citation. A complete manual, source code, installation instructions, and demo applications are available via anonymous ftp from three sites: at TJNAF (CEBAF) at *ftp.cebaf.gov* in the directory *pub/heddle/Hv*, the Super Computing Research Institute at Florida State University: *ftp.scri.fsu.edu* in the directory */pub/CLASsoft/files/classw/hv*, and from *america.pcs.cnu.edu* in the directory *pub/heddle/Hv*. All necessary files are compressed into one tar file named *Hv.1XXX.tar.Z* where *XXX* is a number representing the latest version.†

1.8 Installing the H<u>v</u> Library

The obvious first two steps are to uncompress and untar the *Hv.1XXX.tar.Z* file. We recommend that you do this in your home directory, as this will minimize further customization of the installation procedure.

```
$ uncompress Hv.1XXX.tar.Z
$ tar xf Hv.1XXX.tar.Z
```

After these steps, the directory structure shown in Fig. 4 should exist.

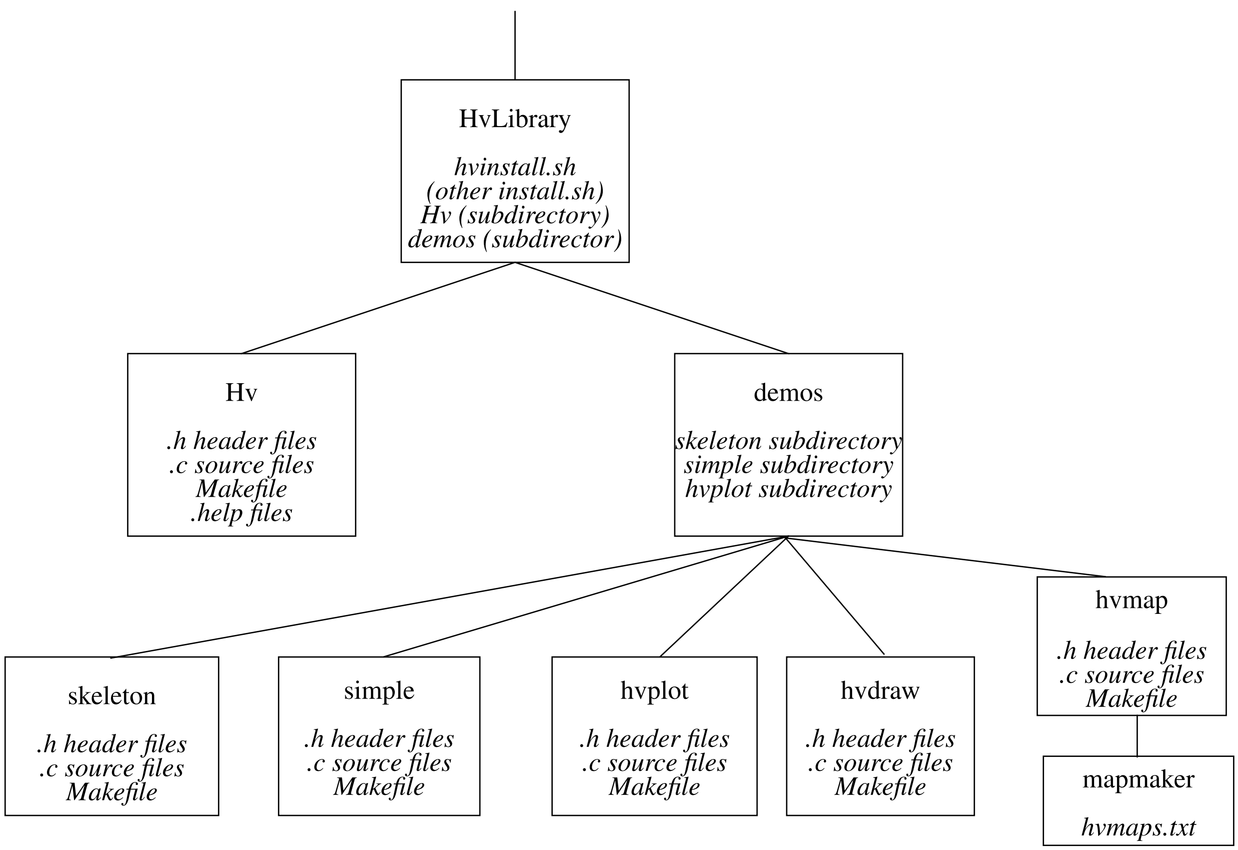


FIGURE 4. Directory stucture after the uncompress and (un)tar of the file Hv.1XXX.tar.Z

Move to the directory **`HvLibrary`**. Make a backup of the file *hvinstall.sh*, and follow the directions therein. Your work will involve changing the definitions of environment variables defined within *hvin-*

---

†. The TJNAF location (*ftp.cebaf.gov*) is the site guaranteed to have the latest version.

stall.sh to match your system. These include such definitions as the path to the Motif library, etc. (The other *.sh* files in the HvLibrary directory are for the sites where H$^{\underline{v}}$ was tested. They are included in the chance that they may be of some benefit.)

After modifying *hvinstall.sh*, “source” it (C shell):

```
$ source hvinstall.sh
```

Use

```
printenv | grep HV
```

to review each variable and make sure it is what you expected.

You are now ready to build the H$^{\underline{v}}$ library. Move to the Hv subdirectory and issue a *make* command

```
$ make
```

That should do it. Any violent complaining will be the result improper modification of the file *hvinstall.sh*. If this occurs, try to interpret the error message in terms of which eninvironment variable was not properly set.

If the procedure was successful, the library *libHv.a* should now exist. This procedure is repeated for each of the demo subdirectories. Move to the subdirectory and issue a solitary *make* command. The only exception is the *hvmap* demo, which requires an additional, non-graphical map-builder application to be created first. See the README file in the *hvmap* directory.

1.9 Attribution

As discussed, H$^{\underline{v}}$ is freely available. Any application that uses it should include in its documentation a proper attribution such as:

> The H$^{\underline{v}}$ library was developed at Christopher Newport University and the Continuous Electron Beam Accelerator Facility under contract to the U.S. Department of Energy.

1.10 Bug Reports

> Bug reports, as well as comments, suggestions and contributions should be directed to *heddle@cebaf.gov.*

1.11 Acknowledgments

The author is delighted to acknowledge the early users and critics of H$^{\underline{v}}$, including Dave Doughty, Mick Guidal, Etienne Burtin, Mikhail Kossov and Kevin Beard at CEBAF, Tim Smith at the University of New Hampshire, Larry Dennis, Greg Riccardi and Peter Dragovitsch at Florida State University, Barry Preedom at The University of South Carolina, and Martin Durbin, Rick Addelson, Wai Lee and Laura Lee at SPARTA. Special acknowledgments are reserved for Mike Finn at INRI and CNU who contributed much code to a precursor development, and to James Muguira of CEBAF and CNU who created H$^{\underline{v}}$'s TextEntryItem.

Hv
Chapter:

2 Using Hv Applications

Any application created with Hv will share some common features. In this chapter we explore these universal behaviors.

2.1 View Controls

These are probably familiar to anyone who has used any windowing system. The views can be closed, dragged, resized, exploded, and scrolled (see Fig. 2.)

2.2 Zooming

Zooming can be done using the view popup (See 2.5.3.4, page 18). However, the more common method is to “drag” a zoom area. This is done by placing the pointer in the HotRect, pressing and holding the MIDDLE mouse button, and rubber-banding a rectangle. When the button is released, the view will be redrawn with its world coordinate system modified to reflect the desired zoom.

Note!

For many views, you will want to set the **Hv_SQUARE** draw control bit at creation. This somewhat misnamed control will force the zoom to maintain the present aspect ratio. This prevents distortion, for example when zooming in on a map. The effect of this on the zoom operation itself is that the rubber-band rectangle will not follow the pointer precisely but will show the “nearest” rectangle that preserves the aspect ratio. This is a feature, not a bug.

2.3 Hidden Diagnostics

Three function keys: F2, F3, F5, F6, and F9 have the following diagnostic uses in developing Hv applications.

- F2 and F9 do the same thing: dump (to stderr) selected information on all items belonging to any *visible* view. This is useful, for example, if you think you have created an item but it doesn’t appear. You may find out that you did in fact create it, but its location is somewhere off the screen.
- F3 toggles “synchronization”, the effect of which is that any drawing will slow down appreciably. Using this you can sometimes follow the progress of a complex drawing operation to see where it goes wrong.

- F5 reports (to stderr) how many pointers are active as a result of using **`Hv_Malloc`**, **`Hv_Free`**, **`Hv_CreateString`**, **`Hv_DestroyString`**, **`Hv_CreateRegion`** and **`Hv_DestroyRegion`** as well as any $H^{\underline{v}}$ routines that create and return **`Hv_Region`**s such as **`Hv_IntersectRegion`**. This is a very crude way of detecting (but not finding) memory leaks. After initialization, the number reported should be fairly stable. In particular, if closing and opening a view causes an increase in the number of active pointers, it is a sign that a memory leak exists in a drawing function. (Most common culprit: not freeing an **`Hv_Region`**).

Note!

Any use of the standard **`malloc`** or **`free`** will "short-circuit" this diagnostic. Particularly nasty would be to have a mixed usage, such as **`Hv_Malloc`** to allocate the memory and **`free`** to release it. This would appear as an "artificial" leak.

- F6 reports (to stderr) informations on $H^{\underline{v}}$ items that are "connected" or "linked", which is an advanced feature (See "Item Connections" on page 87.)

The other fkeys can be used for any purpose, or the functions of fkeys described above can be overwritten. This is done (in either case) by "registering" a function to be called when the fkey is pressed. It will be discussed in the programming section of this manual (See "Using the Function Keys" on page 85.)

2.4 Other Hidden Features

Below we list some other "hidden" features. Some of these can be accomplished in other non-hidden ways, such as through the view popup.

- To bring a view to the front, so that it appears on top of other views, click anywhere on the view with the LEFT mouse button. To send a view to the back, so that it appears to be underneath all other views: While pressing SHIFT, click anywhere on the view using the LEFT mouse button.

Note!

Perhaps "click anywhere" requires some qualification. It really means click on any *innocuous* part of the view. If you click on an item, the item's callback will be called and the view will not move to the front. The title area of a view is always a safe place to click.

- To return a view to its original world coordinate system (i.e., to undo any and all all zooming): While pressing SHIFT, click anywhere on the view using the MIDDLE mouse button.
- To close ALL views at once: while pressing SHIFT, click the close box of any view.
- To get pixel updates while dragging or resizing a view, or while dragging or rotating an item, press and hold SHIFT while performing the operation.
- To "lock" the feedback, perhaps for purposes of freezing the information displayed in the feedback area prior to printing, position the pointer at the desired location, then press and hold SHIFT while dragging the pointer out of the HotRect. You can then select print without disturbing the feedback information (unless you let the pointer sneak back into the HotRect!).

- To drag a HotRect item, assuming its **`Hv_DRAGGABLE`** drawcontrol bit is set, click and hold the LEFT button while the pointer is on top of the item.
- For most HotRect items, a single click (with the LEFT mouse button) will bring the item to the front while a SHIFT-click will send it to the back.
- For HotRect **`Hv_TEXTITEM`**s, such as those resulting from the text annotator button, a double click (of the LEFT button) will bring up the text item editor. This is also true for any items drawn onto the HotRect via the $H^{\underline{v}}$ drawing tools. User items that include an item editor are encouraged to behave likewise, i.e. the editor should be invoked through a double click.

2.5 Standard $H^{\underline{v}}$ Features

Since the *skeleton* demo (See 4.1, page 33) contains virtually no user code, it effectively represents the baseline of all $H^{\underline{v}}$ applications. (At this point, it will be very useful if you have actually built the demos, at least *skeleton*.) In particular, all $H^{\underline{v}}$ applications will have at a minimum the four menus **`Action`**, **`Views`**, **`Color`** and **`Help`**. As we will see as we progress through the examples, applications can add items to these menus and/or create entirely new ones.

2.5.1 The Welcome View

The single view present in `skeleton` is called the *Welcome View*. It contains the name of the program, the version number, a “rainbow” color bar, and a working clock. A Welcome View or a Virtual Desktop View (below) will be present on most $H^{\underline{v}}$ applications. About all that you can do in `skeleton` is drag the Welcome View around the main window.

2.5.2 The Virtual Desktop View

An example of $H^{\underline{v}}$’s *Virtual Desktop View* is show in Fig. 5. The Virtual Desktop’s hotrect contains a viewport (the hollow green rectangle) and minerature representations (solid gray rectangles) of all open views (two in this case). In this manner, the screen can be expanded with many views without excessive clutter. There are three ways to see a presently off screen view:

- The viewport, which indicates what portion of the virtual screen is presently visible, can be dragged over the view.
- The small rectangle representing the view can be dragged to the viewport.
- A double click on the small rectangle representing the view will cause that view to be centered in the viewport.

When the pointer is positioned over one of the small gray rectangles, the Virtual Desktop View’s feedback displays the title of the corresponding view.

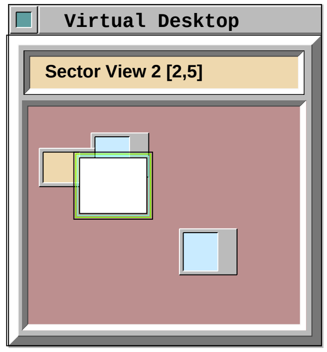


FIGURE 5. The "Virtual Desktop" view. The small hollow rectangle (green, if printed in color) is the view port, showing what portion of the virtual screen is cuuurently visible. It can be dragged. The small gray rectangles (two in this case) represent this application's open views.

2.5.3 Standard H$^{\underline{v}}$ Menus

The four "for-free" menus present on any H$^{\underline{v}}$ application are the **`Action`**, **`Views`**, **`Color`** and **`Help`** menus. In addition there are two popup menus, one asscociated with views and one with the main window. We discuss each of these in turn.

2.5.3.1 The Action menu

The default **`Action`** menu contains three items, **`Refresh Screen`**, **`Print Screen`**, and **`Quit`**.

- **`Refresh Screen`** is a reluctant addition; somewhat of an admission of defeat. It simply causes a redraw of all views. In should, in principle, be unnecessary, since all redrawing should either be "forced", for example when a new item is dynamically allocated, or as a response to an exposure event. This command was included to handle the unlikely occurrence in which the application window was corrupted.
- **`Print Screen...`** will bring up the printer selection dialog box, which will be discussed later. (See 2.5.4.1, page 21). In this case it refers to printing the entire main window. All visible views will be printed on a single page in their present configuration. If necessay, H$^{\underline{v}}$ will scale-to-fit rather than crop.
- **`Background Color...`** will bring up H$^{\underline{v}}$'s ubiquitous and self-explanatory color-picker dialog box shown in Fig. 6.
- **`Quit`** is self explanatory. It first brings up a dialog box that gives the user the opportunity to veto the selection, shown in Fig. 7.

Note!

Normally when your application adds an item to one of the standard menus it will appear at the bottom. However, in the case of the **`Action`** menu, H$^{\underline{v}}$ will reshuffle the items so that **`Quit`** is always last.

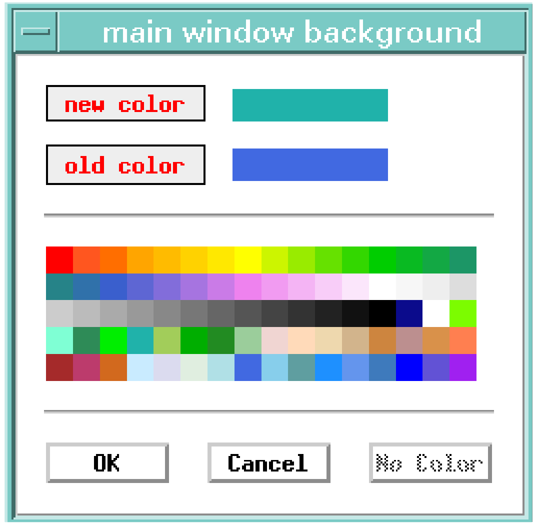


FIGURE 6. The color picker shows up at various places in H$^{\underline{V}}$, here to change the background color. User applications can easily call the color picker to change colors for their own items.

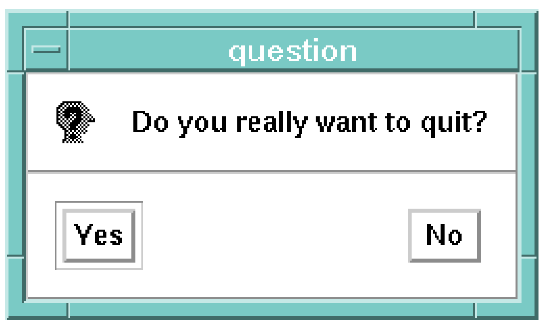


FIGURE 7. The H$^{\underline{V}}$ quit dialog box. There is another way to quit an H$^{\underline{V}}$ application, by using the main window popup (Figure 2.5.3.5 on page 21). The difference: the popup does not offer the user a chance to veto.

2.5.3.2 The Views menu

The **`Views`** menu will have one entry for each view that has been created. Selection of a given view from the menu will bring the view to the front, so that it appears on top of other views, opening the view if necessary. It is thus a convenient way to find hidden views. For *skeleton*, the only entry in this menu will read **`user@host`**, which is the "title" of the Welcome view. When an application adds a new view the title of the view will automatically be used to create a new entry in the **`Views`** menu.

We can think of no reason why any application would need to add an item to the **`Views`** menu. Its sole purpose is to "find" views, and all new views are added automatically.

2.5.3.3 The Help Menu

The **`Help`** menu has two default items.

- **`Show/Hide Balloons`** toggles whether or not help balloons are displayed.
- **`Help`** brings up the on-line help. This will look for a file named “appname.help” where, of course, “appname” is the real name of the application. It will look in the present working directory, and in any others added to the “path” via the **`Hv_AddPath`** function.

  The help file is a simple ASCII file. Help topics are delimited between a line consisting entirely of

  ```
  [HELP_My Topic]
  ```

  and another consisting enitely of

  ```
  [END]
  ```

The application will then include “My Topic” in its list of help topics. For example, the help on *Controlling Views* in the standard H$\underline{\text{V}}$ help file is written as:

```
[HELP_Controlling Views]

             Controlling Views

Views are very much like other windows, such as those  created
by  a  window  managerunder  Unix or on a Macintosh. We will
discuss in turn  the  various  ways  that  theviews  can  be
manipulated.

Hiding the view

    In the upper left corner of the view  is  a  smallsquare
    enclosing  an  even smaller square. Clicking here with the
    left (first) mousebuttonwill  cause  the  viewto  be
    hidden.  It  can  be  redisplayed by selecting it from the
    Views menu.

Moving the view

    The rectangular region around the title  of  the  view  is
    used  to  drag it around the main window. Depress and hold
    the left (first) mouse button anywhere in this  rectangle.
.
.
.

[END]
```

Thus it is completely trivial to add on-line help to any H$\underline{\text{V}}$ application!

The actual help that you will see is displayed in Fig. 8.

Note!

We suggest using the old work-horse *nroff* to make the help file look nice. Also, most developers will want to add a snazzy “About myapp” item to the help menu. Finally, if you add help to the help file while the application is running, but after it has already accessed help at least once, it won’t “see” the new topic until you restart.

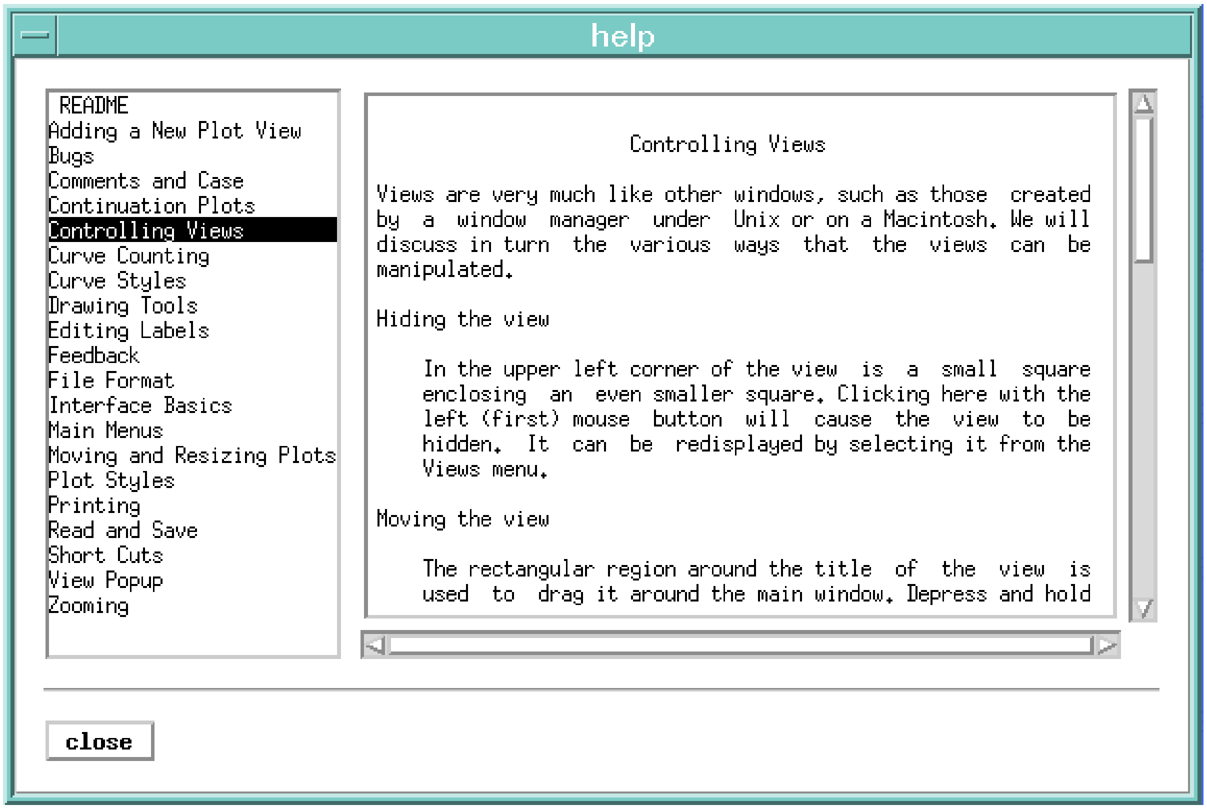


FIGURE 8. The on-line help dialog box. This figure was generated with a screen capture; the actual dialog box will look much nicer.

2.5.3.4 View popup menu

The view popup menu is accessed by placing the pointer anywhere on a view and pressing the RIGHT mouse button. A notional representation is provided in Fig. 9

.

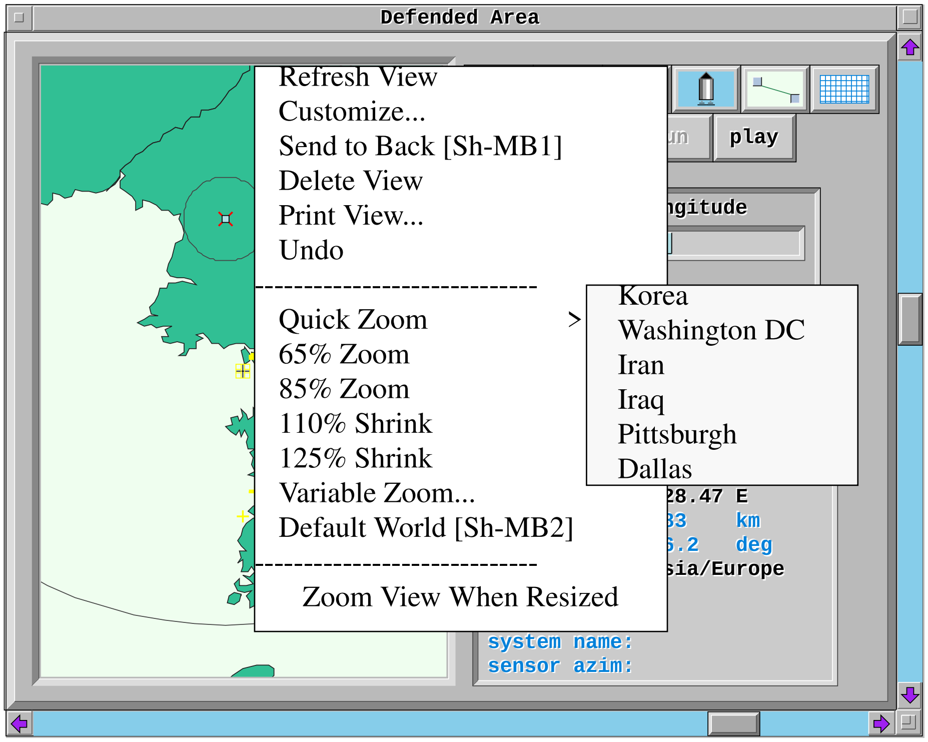


FIGURE 9. An example of a view popup. All the items on the big menu are standard. The menu on the right is a heirarchical menu, under user control, attached to the Quick Zoom item.

Note!

A political correctness alert. We use the term “RIGHT mouse button” when we should use “mouse button 3”, with full malice of forethought and the realization that some left-handers (helicity impaired?) software-reconfigure their mouse in such a way that button 3 is actually on the left.

The view popup contains 13 menu items. The items can be enabled or disabled as appropriate for a given view by the use of the **`Hv_POPUPCONTROL`** attribute during view creation. Each item is discussed below.

- **`Refresh View`** is like its cousin **`Refresh Screen`** on the **`Action`** menu, except that it applies only to the selected view.
- **`Customize`** will call a user provided customization routine for the view (See 5.4, page 48).
- **`Send To Back`** sends the view to the back, so that it appears underneath all other views.
- **`Delete View`** erases the view and frees the memory associated with the view and its item list.
- **`Print View`** is like its cousin **`Print Screen`** on the **`Action`** menu, except that it applies only to the selected view.

- **`Undo`** will undo the most recent dragging or rotation of an $H^{\underline{V}}$ item.

Note!

The following six items are related to zooming. Zooming can also be performed in a perhaps more straightforward manner by dragging a zoom rectangle with the MIDDLE mouse button pressed (See 2.2, page 12).

- **`Quick Zoom`** will permit a one-step zoom to predefined areas of special interest (See 5.5, page 50).
- **`Zoom in (65%)`** causes a zoom of 65%
- **`Zoom in (85%)`** causes a zoom of 85%.
- **`Zoom Out (110%)`** results in 10% more of the world being visible.
- **`Zoom Out (125%)`** results in 10% more of the world being visible.
- **`Variable Zoom`** allows the use to pick an arbitrary scale factor.
- **`Default World`** restores the original world system, in effect undoing any and all zooming.
- **`Zoom View When Resizing`** is a toggle button that changes the effect of a view resize. If it is set, then resizing, a view will also cause a zoom such that you see the same part of the world before and after. If it is not set, you see correspondingly more (or less) of the world after the resize. This is perhaps better understood by referring to Fig. 10.

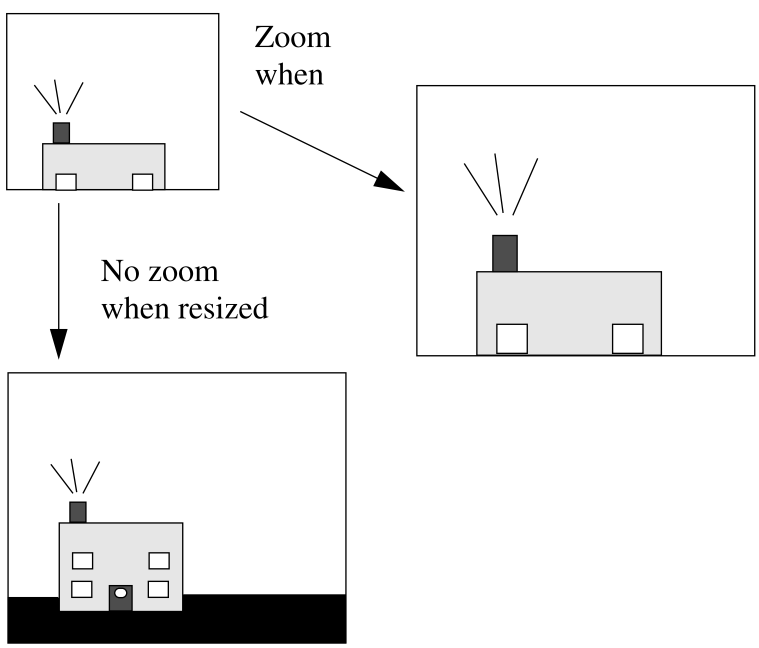


FIGURE 10. Comparison of the effects of a zoom with and without "Zoom When Resize" set.

Note!

If a HotRect item is pixel-based, it will not change its size as the result of a zoom, although they will move consistenly with the change in the world system. All the items created in the **`simple`** example are of this type. The **`hvplot`** and **`hvmap`** demo are better at showing-off zooming.

2.5.3.5 Main window popup menu

The main window pop-up menu is accessed by placing the pointer anywhere except on a view, that is to say where the background is visible, and pressing the RIGHT mouse button. The items available are **`Refresh Screen`**, **`Print Screen`**, and **`Quit`** and have exactly the same meaning as their counterparts in the Action menu (See 2.5.3.1, page 15).

Note!

The **`Quit`** selection from the main window popup works slightly differently from the **`Quit`** found on the **`Action`** menu in that it does not offer a chance to veto. It is thus the fatest way to exit an H$\underline{\text{V}}$ application.

2.5.4 Standard H$\underline{\text{V}}$ Dialog Boxes

There are several predefined dialog boxes that you might encounter, even with the humble *skeleton* application. One of these is the on-line help, which was already discussed (See 2.5.3.3, page 17). The others are discussed briefly below. It is our expectation that they will be self explanatory when encountered.

2.5.4.1 Printer selection

Typically four different actions can result in the appearance of the printer selection dialog box, which is shown in Fig. 11. These are: Selecting **`Print Screen`** from the **`Action`** menu, selecting **`Print Screen`** from the main window popup, selecting **`Print View`** from the view popup, and clicking on a view's camera button. The printer dialog permits you to the destination as either a printer or a file (the file will be in encapsulated postscript format, so that it can be imported into a word processor), add a footer comment to the bottom of the printout, and to choose between portrait and landscape. It also allows you to chose between "print contents only" and "print entire view". The former prints only the contents of the HotRect. The latter prints the enitre view including the Control and Feedback area. The "lp" or "lpr" option allows you to choose the print command appropriate for your flavor of UNIX.

The delay is useful if you want to print on transparencies, since it allows you to start the print and run to the printer to insert the clean view graphs.

The “contents only ” selection causes just the items present (and visible) in the view’s HotRect to be printed. This is almost certainly what you want. If “entire view” is selected, the view adornments, such as the drawing tools, are also printed.

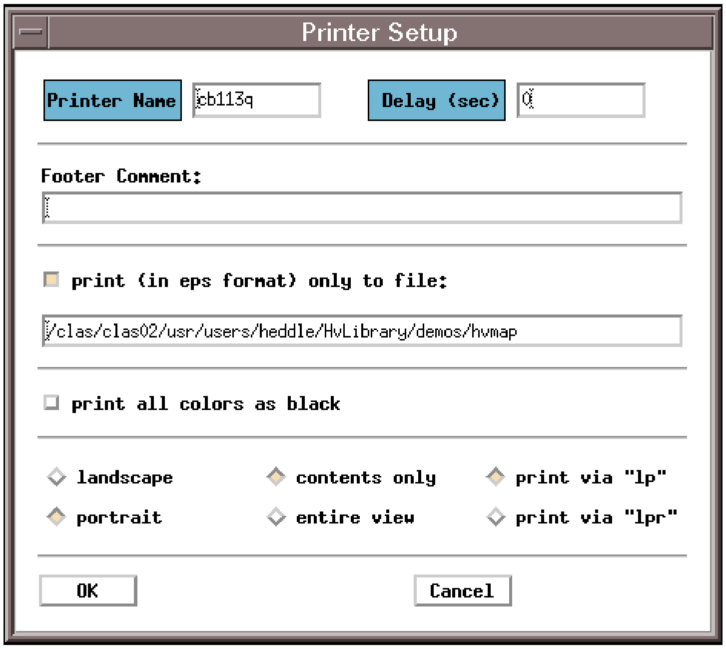


FIGURE 11. The printer setup dialog box. Use the “print all colors as black” option when printing to a black and white printer.

Note!

The “print all colors as black” is extremely useful. It tells $H^{\underline{V}}$ to change all colors (with a few exceptions) to black. This will make graphics look sharper when printed on a black and white printer. Otherwise, a less than pleasing dithering will occur. $H^{\underline{V}}$ will use the **PRINTER** envronment variable, if defined, to set the default printer name.

2.5.4.2 Questions, Warnings, Information

There are also simple yes/no question dialogs, warning dialogs, and information dialogs predefined in $H^{\underline{V}}$ (we have already seen a question dialog when Quit is selected (See 2.5.3.1, page 15). As an example of a warning, Fig. 12 displays what you would see an $H^{\underline{V}}$ application (in this case *hvplot*) with improper command line arguments.

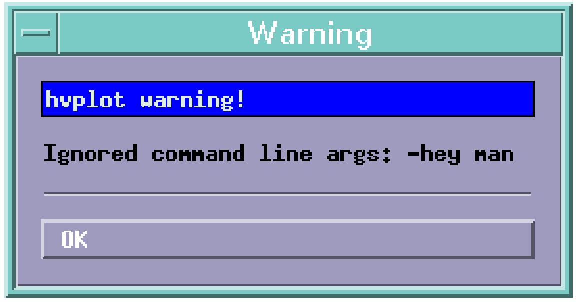


FIGURE 12. A warning dialog generated by starting *hvplot* via $ `hvplot -hey man`. If you use an H$\underline{\text{V}}$ warning dialog this in your application, will automatically prepend the application's name (in this case *hvplot*).

Note!

H$\underline{\text{V}}$ applications will accept all the standard command line arguments recognized by any X application. For example, try **`skeleton -bg red`**.

2.5.4.3 Item Editors

Many standard H$\underline{\text{V}}$ items have built in graphical editors, which are always invoked by double clicking on the item. Try double clicking on items, both of the HotRect and control variety, in any of the demos. For example, consider the editor for **`Hv_TEXTITEM`**s. Double click on any of the text in the HotRect of the *`simple`* application to get a dialog (Fig. 13) that allows you to edit the font, color, orientation, and text of the **`Hv_TEXTITEM`** as well as deciding whether or not it should have a 3D frame around it. You can also enter LATEX-like control sequences to mix fonts, create subscripts, etc.

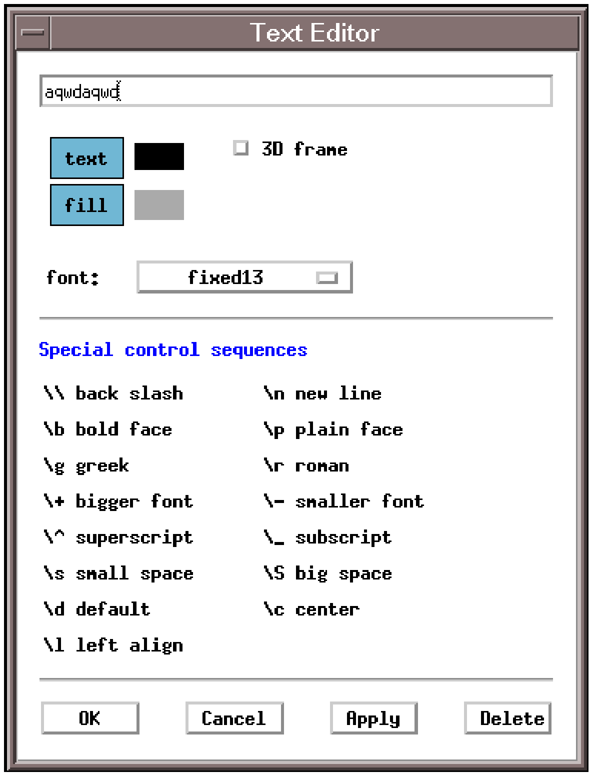


FIGURE 13. The Hv_TEXTITEM editor.

The special control sequences are initiated with a backslash: '\'. Here is a listing of presently recognized sequences:

sequence	effect
\\	the backslash character
\n	insert a newline
\b	change to boldface
\p	change to plainface
\g	change to greek (symbol)
\r	change to roman (helvetica)
\+	next bigger font
\-	next smaller font
\^	raise the baseline (superscript)
_	lower the baseline (subscript)
\s	small horizontal space
\S	large horizontal space

TABLE 1. Special control seqences that can be embedded in strings. The change of a format, for example to greek, lasts "forever", that is until the next control sequence undoes it. ***The control sequences are case sensitive***

sequence	effect
\c	center text
\d	restore default font

TABLE 1. Special control seqences that can be embedded in strings. The change of a format, for example to greek, lasts "forever", that is until the next control sequence undoes it. ***The control sequences are case sensitive***

As an example, the string

```
d\gr\s\d(q\^2\d)\s/\sdt\^2
```

would be formatted as

$d\rho\,(q^2)\,/\,dt^2$

Note!

The only virtual entry in the equally virtual $H^{\underline{V}}$ *Style Guide* is that double clicking on an item, if it does anything, should bring up the item's editor, Thus developers, if they want their user-items to have an editor, should attach it via the **`Hv_DOUBLECLICK`** attribute.

SECTION 2 Hv Programming

Hv
Chapter:

3 Top-level view of Hv Programming

3.1 Designing the Application

In this chapter we will give a top-level overview of how an Hv application is developed. No actual code is presented nor is any actual Hv procedure described. Such details will be provided in the following chapters. The purpose here is to outline the basic steps used in constructing an Hv application and in doing so to provide further elucidation of the nature of Hv.

The design that must be carried out prior to the development of an Hv application involves deciding how many *different types* of views are required and then deciding on a view-by-view basis what control items are needed, what HotRect (usually user-defined application-specific) items are needed, and what manner of pointer tracking (feedback) would be informative.

Note!

It is the number of *types* of views that should be considered during the design phase. In most applications, the number of actual views is unlimited; the user creating as many as necessary. For example, in *hvplot* there is only one type of view, but the user may have "instantiated" many clones. Most applications will want to provide a "New View" menu item so that users can create new views "on-the-fly".

Let us return to the CLAS event display (*ced*) and recreate part of the process of this preliminary design phase. A slice through the middle and showing the entire detector is presented in Fig. 14. It is not important for the reader to have any knowledge of detectors, only to recognize that we want to represent a large object (CLAS) that is comprised of independent components (the various detector packages).

We decided to construct one view that shows the entire detector -- the view shown in Fig. 14. We also recognized the need for several additional views, each showing only one component of CLAS, and a final view displaying the toroidal magnetic field that gives CLAS its spectroscopic capability. Some of these views can be found in Fig. 1. Thus we designed *ced* to contain a handful of *view types*, each with its own controls, user-defined items, and feedback but all behaving consistently in terms of the user interface. Finally we note that each view type can be cloned indefinitely: We could have any number of magnetic field views, each a carbon copy in regards to the controls and feedback, but independently positioned, sized, and zoomed.

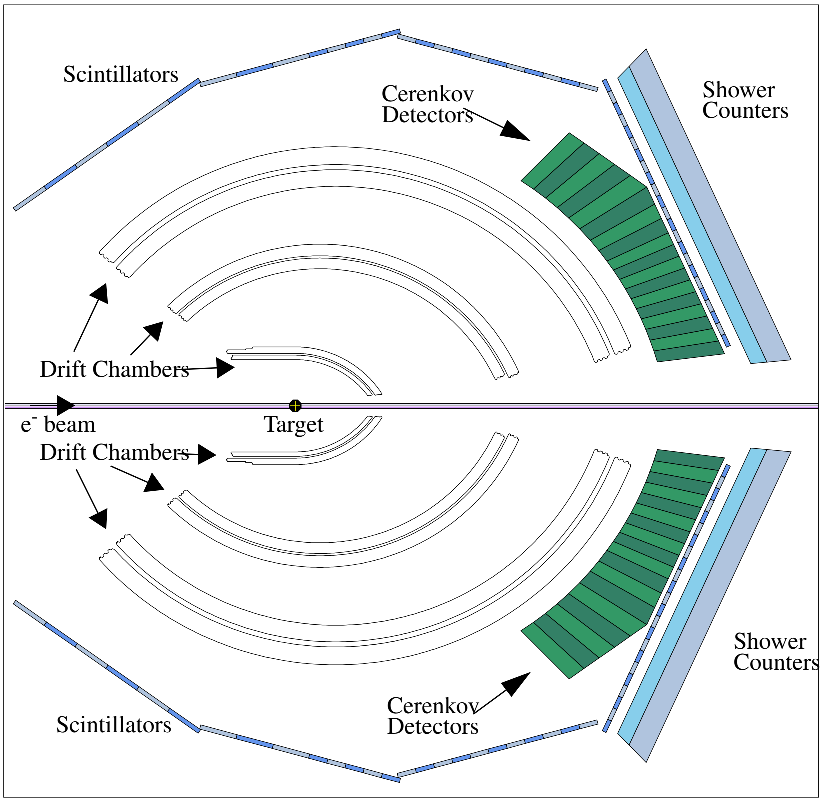


FIGURE 14. A slice of the CLAS detector, which is comprised of concentrically (with the target at the center) arranged component packages. A knowledge of detectors is not important here, only the notion that we are representing a large, composite object.

Having a design in mind, we are ready to begin coding. Our discussion is now completely generic. The main program in an H<u>v</u> application generally consists of three “real” lines: one to initialize H<u>v</u>, one to call the application’s private initialization, and a final call to enter an event dispatching loop:

```
void main(int    argc,
          char **argv)

{
  Hv_VaInitialize(argc, argv,
                  Hv_WELCOMETEXT, “version 1.0”,
                  Hv_USEWELCOMEVIEW, True,
                  NULL);
  PrivateInitialization();
  Hv_Go();
}
```

The first step, the H<u>v</u> initialization (**`Hv_VaInitialize()`**), performs all the underlying X initialization, creates the H<u>v</u> main window, allocates the fonts and colors, and creates the main menubar. (Don’t worry about the details of **`Hv_VaInitialize`** yet, just accept it as an initialization routine.) The third step, the event loop (**`Hv_Go()`**), is where X events such as pointer motion, pointer clicks and keystrokes are dispatched. *The developer does not need to know any of the details of the* H$^{\underline{v}}$ *initialization or how* H$^{\underline{v}}$ *processes X events.*

All the development goes into the second step, the private initialization. Here is where an application's initial views are created, along with the controls, user-defined items, and entries into the feedback area. Any routine initialization (such as initializing global variables) and additions to or modifications of the main menubar are also done here. This is represented schematically in Fig. 15.

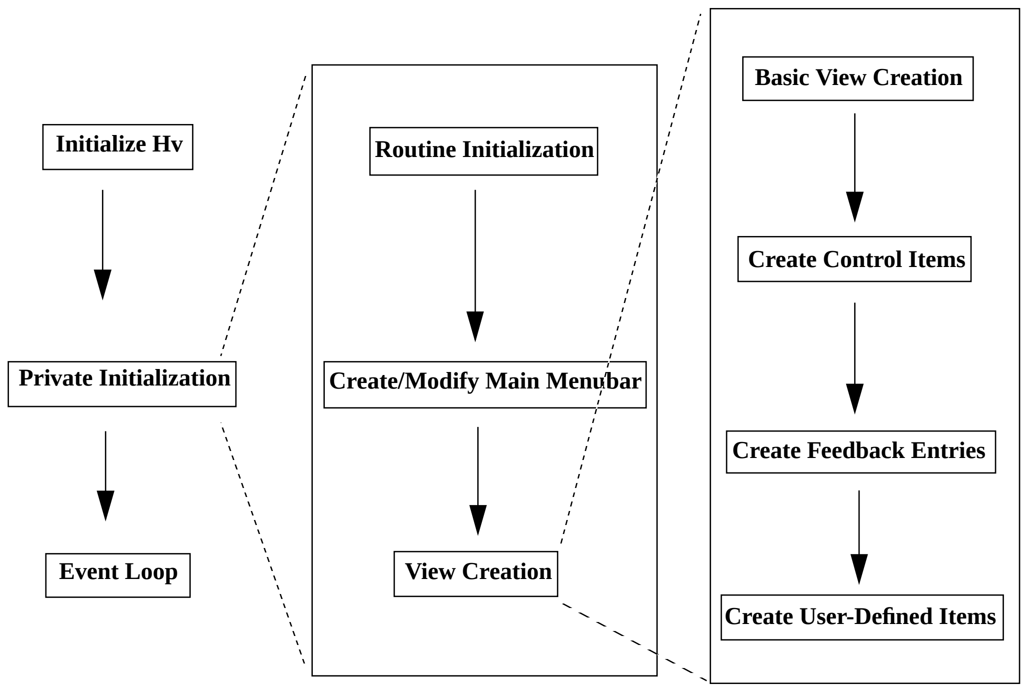


FIGURE 15. A top level overview of H$^{\underline{v}}$ programming. The user's main program typically consists of three lines: One to initialize H$^{\underline{v}}$, one to call the application specific initialization and a final call to an event processing loop. The user's initialization is further expanded into the standard sequence of steps the most important of which is the creation of the application specific views.

When creating the views and items, the developer will attach to them procedures that are called in response to various actions, entirely analogous (but simpler than) the "callbacks" of X and Motif. For example, when a view is created, a feedback procedure is (optionally) attached. This is the procedure that H$^{\underline{v}}$ will invoke as the pointer is moved within the view's HotRect. The user has to provide the code that will take the information passed to it by H$^{\underline{v}}$ (*i.e.*, the present pointer position) and convert it to a meaningful string which H$^{\underline{v}}$ will then display in the appropriate position in the feedback area.

As for the user-defined items, there are two methods available for the developer. One is to create *truly* user-defined items that H$^{\underline{v}}$ knows nothing about. Here the developer has to provide a procedure for drawing the item as well as various support procedures that tell H$^{\underline{v}}$ how the object should respond to scrolling, resizing, rotation, etc. A simpler method recommended whenever applicable is to "piggy-back" onto an existing H$^{\underline{V}}$ item (See 7.8.1, page 69). For example, many items (such as the detectors in CLAS) can often be represented by world-based polygons, *i.e.*, polygons with vertices stored as world rather than pixel locations. H$^{\underline{V}}$ has a predefined world-based polygon item, so creating a user-defined item based on it has the advantage that H$^{\underline{V}}$ already knows how to handle all the basic drawing and item maintenance. The user only provides a customized drawing procedure that H$^{\underline{V}}$ calls after its basic polygon drawing that converts the item from generic to specific.

3.2 H$^{\underline{V}}$ Coordinate Systems

Each H$^{\underline{V}}$ view has two coordinate systems: an integer-pixel based *local* system and a floating point *world* system (see Fig. 16). Every view's local system has the same origin: the upper left corner of the screen with *y* increasing downward, as is customary for pixel coordinates. Thus, the local system for a view merely serves to locate it on the screen.

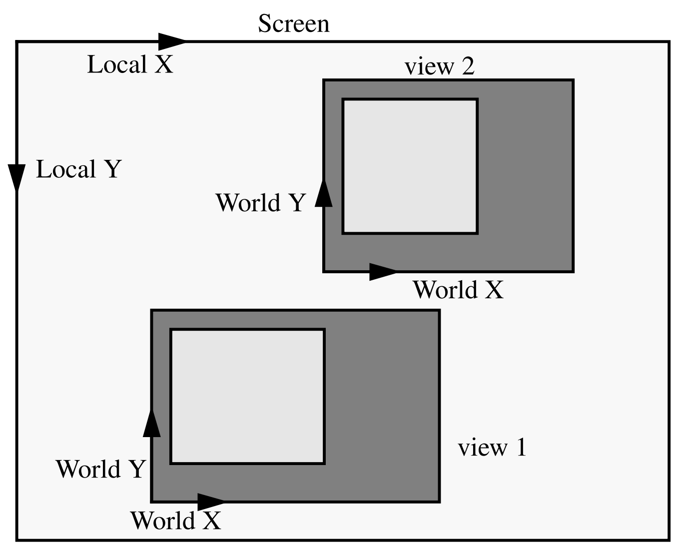


FIGURE 16. The coordinate systems used by H$^{\underline{V}}$ views. The local pixel system positions the view on the screen, relative to the upper-left corner. Each view has its own floating point world system which establishes what items are displayed in the HotRect.

The world system for each view has its origin at the bottom left *of the view*, with *y* increasing upward. The world coordinate system applies to the *entire view* rather than the just the HotRect. However, A moment's consideration will convince you that the most natural way to think, especially in setting up a view, will be in terms of the world coordinates of the HotRect since that is where most of the interesting items will reside. If you were for some reason modeling a (U.S.) football field, you would want to specify the width of the HotRect as being 100 yds. You would probably not even want to know what the dimension of the entire view is in world coordinates.

You are assisted in this by the fact that the View creation routine **`Hv_VaCreateView`** allows you to specify the coordinates you desire *for the HotRect*, it then calculates what the corresponding local and world systems need to be.

Note!

Maintaining two coordinate systems has led to some peculiar problems, especially when it comes to scrolling. Most items that are located in the HotRect are positioned based on *world* coordinates. Imagine for example a representation of equipment in a laboratory layout. Some, however, do not have "natural" world coordinates, such as a text item used for annotation. Scrolling basically is a simple matter of offsetting a view's world system while leaving its local system untouched. To properly zoom and scroll "unnatural" items placed within the HotRect, $H^{\underline{v}}$ will secretly attach world coordinates.

To summarize, you should now have two coordinate systems in mind: the local pixel system and the floating point world system. Each view has its own local and world. $H^{\underline{v}}$ provides routines for transforming from one system to another.

3.3 Regions, Regions, Regions

We have made an effort to shield the developer from the aspects of X and Motif that tend to confuse novices, such as Graphics Contexts, Display and Window variables, etc. One exception is the esoteric *Region* data structure, to which $H^{\underline{v}}$ developers will have to give some attention.

A working definition of a region is: An arbitrarily shaped, possibly disconnected area of your application's main window.

When a window is occluded and then exposed, the X Server will generate an exposure event which ultimately results in a *exposure region*, which is the portion of your window that must be redrawn. $H^{\underline{v}}$ will check what items overlap that region (and thus need to be redrawn) and will call the item's drawing routine passing to it (the drawing routine) the item itself and the exposure region. Thus any drawing routine for an $H^{\underline{v}}$ items is of the form:

```
Hv_DrawButtonItem(Hv_Item    Item,
                  Hv_Region Region)

/* Item:    Item to be drawn
   region;Affected region  */

{
       /* drawing stuff here */
}
```

Drawing routines for "user-items" must follow the same format.

Note!

Most of the time the drawing routine will not even use the region. Occasionally, it might be passed along to a low level drawing routine. The drawing routine should *never* free the region. This could have disastrous consequences.

3.4 Hv Programming Bureaucracy

In this section, we present mundane: naming conventions and argument lists. However, first a word on function prototyping.

Hv was written in ANSI "C", and thus uses function prototyping. Applications are free to use K&R "C" or even C++. The main point here is that on some compilers you may need to set a ansi flag when building the Hv library (See 1.8, page 10).

Note!

If you have read the *Computers in Physics* article about Hv, you may recall that it declared just the opposite: That Hv was built using K&R "C". That was indeed true, but after we returned the galleys we converted to ANSI "C" to facilitate application writing in C++.

3.4.1 Hv Naming Conventions

Hv naming conventions are simple and fairly standard.

- All Hv procedures start with the **Hv_** prefix. The rest of the procedure name is "WordCapped" including the first letter. Some examples (without arguments) are:

```
Hv_Initialize()
Hv_FillPolygon()
```

- All Hv defined types obey the same convention as for procedures. Some examples are:

```
Hv_Rect
Hv_Item
```

- All Hv global variables start with the **Hv_** prefix and are "Word Capped" except for the first letter. Some examples are:

```
Hv_powderBlue
Hv_welcomeView
```

- A conflict arises between the rules for Hv functions and Hv global variables in the case where the global variable in question is actually a pointer to a function. We have chosen to use the rule for function names in such cases:

```
Hv_AfterItemDrag
Hv_AfterResize
```

In brief, the purpose of these function pointers, which will be described in detail in (See 9.9, page 90), is to call (optionally) a user-written procedure *after* Hv has completed an action, such as dragging an item. Many applications will *not* need them.

- All H$^{\underline{v}}$ *#defs* (macros) and attributes start with the Hv_ prefix after which all letters are upper case. Some examples are:

```
Hv_DRAGGABLE
Hv_OFF
```

The **Hv_** prefix was enforced for our benefit rather than your's. Although you are shielded from the underlying X, Xt, and Motif, the code that generates the H$^{\underline{v}}$ library is (obviously) not. The prefix avoided any confusion about the ultimate ancestry of a procedure, type, or variable.

3.4.2 Variable Length Argument Lists

Within H$^{\underline{v}}$, use of variable-length argument list functions occur in several places. These are functions akin to **printf** in "C" that take an arbitrary number of arguments. In H$^{\underline{v}}$, these functions always begin with **Hv_Va**, such as:

```
Hv_VaCreateView
Hv_VaCreateItem
Hv_VaCreateDialogItem
```

The "**Hv_Va**" functions are used to pass, in any order, an arbitrary number of *attribute value pairs*. The *attribute* specifies what quantity you want to set to the accompanying *value*. For example, an attribute value pair of

```
Hv_WIDTH, 80
```

sets the width (of whatever you are creating) to be 80 pixels. Each attribute has a definite type associated with it, either **char, short, int, long, float,** or (**void ***).

Before the attribute-value pairs there will be one or two required arguments. *Most importantly, a NULL must the last argument passed in an* **Hv_Va** *function.* Thus the canonical form is:

```
Hv_VaCreateSomething(required1, required2,
attribute_1, value_1,
attribute_2, value_2, ...
attribute_n, value_n,
NULL);
```

The order of the pairs doesn't matter.

Note!

Sneaky 'ole **printf** and its cousins don't need a NULL terminator because they require a format string. By counting the '%'s in the format string, one knows exactly how many arguments to expect.

3.4.3 Boolean Type

Since H$^{\underline{v}}$ applications necessarily include all the X and Xt headers, variables of type **Boolean** can be declared. Also, the constants **True** and **False** are defined.

Hv
Chapter:

4 The Simplest Hv Applications: *skeleton* and *hvdraw*

It is time that we programmed something. Lets start with *skeleton*: the "hello world" of Hv programming. Then we will look at *hvdraw*, which is (almost) the simplest "complete" Hv application possible with views, items, and feedback.

4.1 *skeleton* source code

The source code for *skeleton* consists entirely of a **main** function. It, like all Hv applications, must include the header file Hv.h. The complete source code for *skeleton*, found in main.c, follows:

```
#include "Hv.h"                   /*only necessary include file*/

/* ----- main program -------*/

void main(int argc,
          char **argv)

{
  Hv_VaInitialize(argc, argv,
                  Hv_WELCOMETEXT, "version 1.00d",
                  Hv_USEWELCOMEVIEW, True,
                  NULL);
  Hv_Go(); /* get and dispatch events */
}
```

The header file `Hv.h` is only one needed for many Hv applications. If the application will use the integrated scientific plotter, then `Hv_xyplot.h` is also required.

There is a *Makefile* for *skeleton* included with the distribution. It relies on the establishment of certain environment variables as discussed earlier (See 1.8, page 10).

4.2 Dissecting the *skeleton* source

To kick-start Hv, one simply passes `argc` and `argv`, along with a variable length argument list, to the function **Hv_VaInitialize**. There is no return value to check from this procedure; if initialization fails **Hv_VaInitialize** will print a message to **stderr** and cause the entire application to exit. In this example, two optional value-attribute pairs (`Hv_WELCOMETEXT, "version 1.0"`) and (`Hv_USEWELCOMEVIEW, True`) were passed to **Hv_VaInitialize**. We will discuss the complete set of value-attribute pairs relevant for **Hv_VaInitialize** in the next section.

Next the function **Hv_Go** is called, which continues to get and process X type events until the user has selected **Quit** from the main or popup menu.

In terms of the three steps of an $H^{\underline{V}}$ application discussed earlier `skeleton` skips step two, the application specific initialization. (See “Top-level view of H^V Programming” on page 26)) The result of running *skeleton* is shown in Fig. 17.

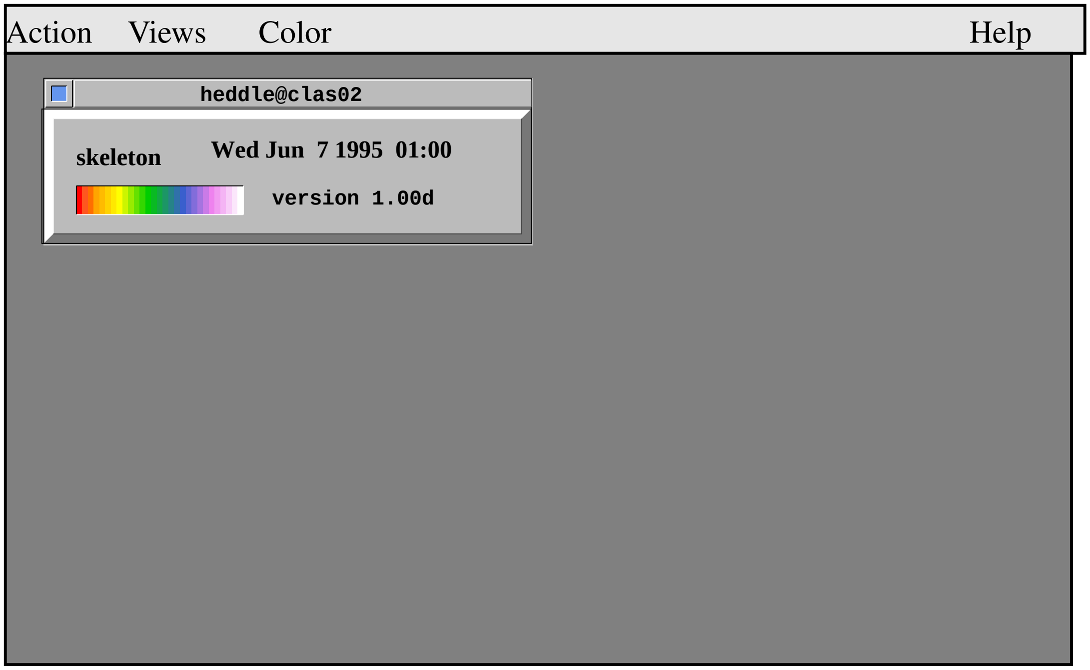


FIGURE 17. `skeleton` main window. On a “real” screen, the Welcome view would take up a much smaller fraction of the available real estate.

Of course, one could pass any string, such as “XYZ.Inc” as the value associated with **`Hv_WELCOMETEXT`** to **`Hv_VaInitialize`**. It would dutifully appear in place of “version 1.00d” on the Welcome view.

Applications are not required to have a Welcome view, as we will see shortly.

4.3 Hv_VaInitialize

The **`Hv_VaInitialize`** variable length argument function performs all the $H^{\underline{V}}$ initialization. It has two required arguments, the command line variables **`argc`** and **`argv`**. Other than initializing $H^{\underline{V}}$, this is the routine the developer uses to include a Welcome View and/or a Virtual Desktop View.

There is a modest set of attribute-value pairs relevant for initialization. Most have to do with the virtual desktop view. They are summarized in the following table:

Attribute	Comment	Type	Default
`Hv_TRAPCOMMANDARGS`	If True, H^{v} will “trap” unknown command line arguments, placing them in a global variable `Hv_leftOverArgs` where they can be processed by the application. Otherwise, H^{v} will give a warning about unknown arguments.	short	0 (False)
`Hv_NORAINBOW`	If True, there will **NOT** be a rainbow strip on the Welcome view	short	0 (False)
`Hv_USEVIRTUALVIEW`	If False, there will **NOT** be a Virtual desktop view	short	0 (False)
`Hv_USEWELCOMEVIEW`	If False, there will **NOT** be a Welcome view	short	0 (False)
`Hv_VERSIONNUMBER`	If this attribute is used, a version number string will be placed on the Welcome view, and the `Hv_WELCOMETEXT` attribute *will be ignored*. The value passed should be 100 times the decimal version, e.g. if the version is 2.13 you should pass 213 as the value. The format of the string will be: “v2.13”. If you don’t like this “automatic” version string, you can use the `Hv_WELCOMETEXT` instead.	short	-1
`Hv_VIRTUALHEIGHT`	The height (pixels) of the virtual desktop.	short	5000
`Hv_VIRTUALWIDTH`	The width (pixels) of the virtual desktop.	short	4000
`Hv_VIRTUALVIEWBMARGIN`	Margin (pixels) below of the Virtual View’s hotrect.	short	4
`Hv_VIRTUALVIEWFEEDBACKDATACOLS`	Characters available for Virtual View’s feedback text.	short	25
`Hv_VIRTUALVIEWFEEDBACKFILL`	Background color of Virtual View’s feedback box.	short	Hv_wheat
`Hv_VIRTUALVIEWFEEDBACKFONT`	Font for Virtual View’s feedback text.	short	Hv_fixed2
`Hv_VIRTUALVIEWHRFILL`	Background color of Virtual View’s hotrect.	short	Hv_rosyBrown
`Hv_VIRTUALVIEWHRW`	Width (pixels) of Virtual View’s hotrect.	short	140
`Hv_VIRTUALVIEWLMARGIN`	Margin (pixels) to the left of the Virtual View’s hotrect.	short	4
`Hv_VIRTUALVIEWRMARGIN`	Margin (pixels) to the right of the Virtual View’s hotrect.	short	4
`Hv_VIRTUALVIEWFBTEXTCOLOR`	Color of text in feedback area.	short	Hv_black
`Hv_VIRTUALVIEWTMARGIN`	Margin (pixels) above the Virtual View’s hotrect. (Its default is bigger than the other margins to make room for the feedback)	short	30
`Hv_WELCOMETEXT`	Arbitrary string (often indicating version) placed on the Welcome view. If the `HV_VERSIONNUMBER` attribute is used, this attribute is ignored.	char *	NULL

TABLE 2. Value-attribute pairs relevant for the Hv_VaInitialize call.

Note!

There is no `Hv_VIRTUALVIEWHRH` attribute (for the height). That is because H^{v} will calculate the virtual view’s hot rect height using its width (via `Hv_VIRTUALVIEWHRW`) and the diemensions of the virtual world (via `Hv_VIRTUALWIDTH` and `Hv_VIRTUALHEIGHT`) in such a way as to give the hotrect the same aspect ratio as the virtual world.

4.4 *hvdraw* source code

The demo *hvdraw* represents the next step in development, since it actually creates a view and attaches some feedback.

hvdraw is a very primitive drawing program that basically creates a "bare" view containing no user items. The view does however, contain the standard H$^{\underline{\mathrm{V}}}$ drawing tools that can be added to any View, and some very simple feedback that merely reports the pointer's screen coordinates. The

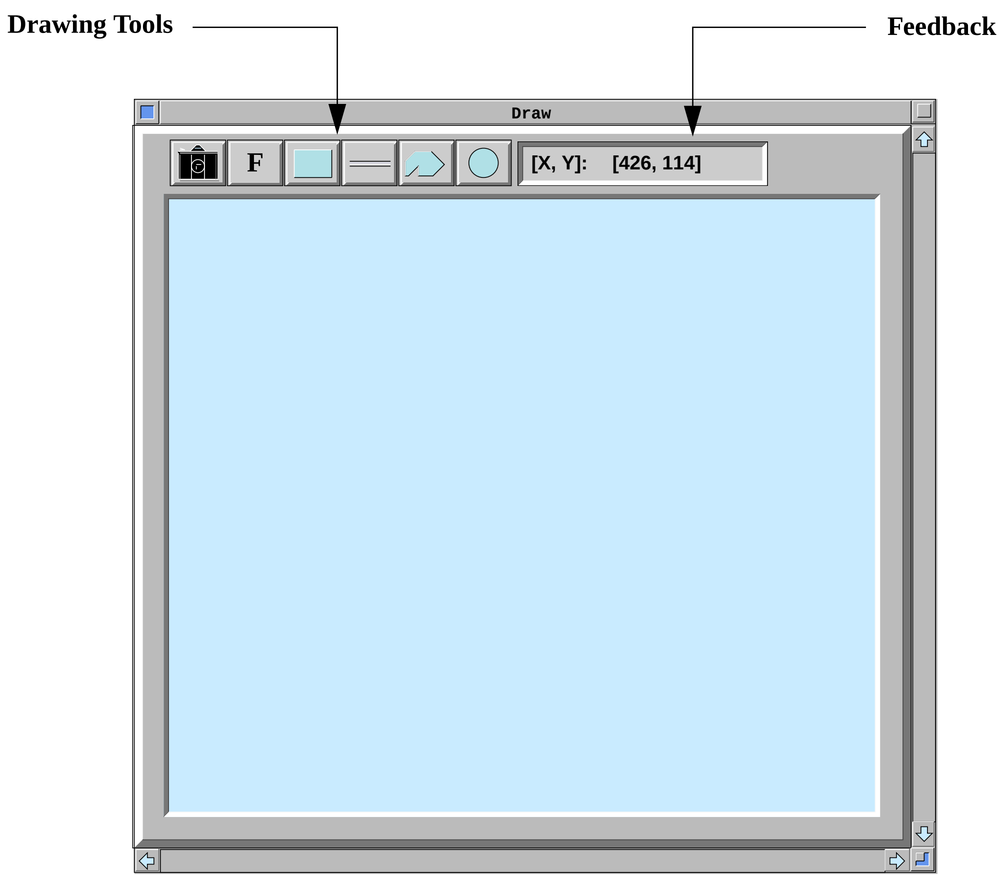


FIGURE 18. Start-up screen for hvdraw. The standard drawing tools are along the top, as is the simple feedback area. Note, as usual, that the main window and menu bar are not shown here.

4.5 Dissecting the *hvdraw* source

The hvdraw subdirectory in the demos directory contains a Makefile, a header file draw.h, four .c files (feedback.c, init.c, main.c and setup.c) and a help file: hvdraw.help) Let us discuss the non- cfiles first.

- hvdraw.help: As discussed earlier (See "The Help Menu" on page 18), any application can use on-line help file via a .help file. This just illustrates the point. When running *hvdraw*, users will have access to the help topics found in hvdraw.help.

- draw.h: This is the header file included by all the .c files in the *hvdraw* directory. It is quite small so we reproduce it here and assume that with the additional comments it is self-explanatory:

```
#ifndef   __DRAWH__
#define   __DRAWH__


#define LOCALXY       0   /* this is a tag for the only feedback used in hvdraw */

/* function prototypes */

extern   void   Init(void);
extern   void   SetupView(Hv_View);
extern   void   Feedback(Hv_View,
                         Hv_Point);

#endif
```

Let us now turn to the source code. First, main.c:

```
#include "Hv.h"
#include "draw.h"

/* ----- main program -------*/

void main(unsigned int argc,
          char **argv)

{
  Hv_VaInitialize(argc, argv, NULL);
  Init();
  Hv_Go();
}
```

This is virtually the same main program as in *skeleton*, so there is no need to repeat that discussion. One difference is that the attribute `Hv_USEWELCOMEVIEW` is not present in the list, so *hvdraw* will not have a welcome view. A more important difference is that, unlike *skeleton*, here we call a private initialization routine **`init()`**, to which we turn our attention.

4.5.1 *hvdraw*'s init.c: an intoduction to View creation

Here is a listing from init.c:

```
#include "Hv.h"
#include "draw.h"

#define  SIMPLE_VIEW     1

/*----- Init -------*/

void Init(void)

{
  Hv_View     view;

  Hv_VaCreateView(&view,
                  Hv_TAG,             SIMPLE_VIEW,
                  Hv_HOTRECTTMARGIN,  38,
                  Hv_TOP,             50,
                  Hv_HOTRECTWIDTH,    650,
                  Hv_HOTRECTHEIGHT,   650,
                  Hv_XMAX,            1.0,
                  Hv_YMAX,            1.0,
```

```
                    Hv_INITIALIZE,      SetupView,
                    Hv_TITLE,           " Draw ",
                    Hv_POPUPCONTROL,    Hv_STANDARDPOPUPCONTROL - Hv_QUICKZOOM,
                    Hv_DRAWCONTROL,     Hv_STANDARDDRAWCONTROL - Hv_ZOOMONRESIZE,
                    Hv_FEEDBACK,        Feedback,
                    Hv_HOTRECTCOLOR,    Hv_aliceBlue,
                    Hv_OPENATCREATION,  True,
                    NULL);

}
```

Hopefully it is clear that the only purpose of the private initialization routine is to create a single view using the view creation routine **Hv_VaCreateView**. We will discuss view creation at length in the next section, but it is probably useful to give a quick synopsis of the arguments passed to **Hv_VaCreateView**.

The first *and only required* argument is a pointer to the variable that will hold the view. No pre-allocation is necessary.

The remaining arguments are the attribute-value pairs (which can appear in any order.) The first attribute Hv_TAG is used to, in effect, give an ID to this view or, more commonly, to this type of view. The next attribute, Hv_HOTRECTTMARGIN, increases the default margin of 15 pixels to 38 in order to make room for the drawing tools and the feedback. The attribute Hv_TOP positions the top of the view. Since no Hv_LEFT attribute was used, H$^{\underline{V}}$ will choose the left placement.

The attributes Hv_HOTRECTWIDTH and Hv_HOTRECTHEIGHT fix the size in pixels of the hotrect. The attributes Hv_XMAX and Hv_YMAX set the extent of the world system (Since their MIN counterparts are not present, the defaults values of 0.0 are used. Note that *hvdraw* doesn't event really need a world system since there are no user defined items representing physical objects.)

The attribute Hv_INITIALIZE is an important one: It identifies the user provided function (in this case **SetupView**) that H$^{\underline{V}}$ will call as soon as it creates the view. That is where we will add items to the view. We'll examine that process presently.

The attribute Hv_TITLE is presumably self-explanatory.

The attributes Hv_POPUPCONTROL (See 5.6, page 52) and Hv_POPUPCONTROL (See 5.7, page 53) are masks that set various properties of the view. They will be discussed in detail in the noted sections.

The attribute Hv_HOTRECTCOLOR sets the hotrect's background color.

The attribute Hv_OPENATCREATION causes the view to be immediately visible. The default is that it is "closed", and the **Views** menus must be used to open it.

The Hv_FEEDBACK attribute tells H$^{\underline{V}}$ what function to notify when the pointer is moving in the View's hotrect.

Finally, the argument list concludes, as it must in all **Hv_Va** functions, with a NULL terminator.

4.5.2 *hvdraw*'s **setup.c**: and intoduction to "setting up" a View

As stated in the previous subsection, the `Hv_INITIALIZE` attribute in the view creation causes $H^{\underline{v}}$ to call the user routine **`SetupView`** immediately after it creates the view. Lets examine the file setup.c, which contains that routine:

```
#include "Hv.h"
#include "draw.h"

/* --------- SetupView -------- */

void SetupView(Hv_View View)
{

  Hv_Item   first, last, Feedback;
  short     l, t, r, b;
  Hv_Rect   *area;

  Hv_GetRectLTRB(Hv_GetViewHotRect(View), &l, &t, &r, &b);

/* create a camera button for printing
   and the standard drawing tools */

  Hv_StandardTools(View,
                   l,
                   t - 37,
                   Hv_HORIZONTAL,
                   Hv_DONOTHINGONRESIZE,
                   Hv_ALLTOOLS,
                   &first,
                   &last);

/* create a feedback item */

  area = Hv_GetItemArea(last);
  Hv_GetRectLTRB(area, &l, &t, &r, &b);
  Feedback = Hv_VaCreateItem(View,
                             Hv_TYPE,              Hv_FEEDBACKITEM,
                             Hv_LEFT,              r+4,
                             Hv_TOP,               t+1,
                             Hv_RESIZEPOLICY,      Hv_DONOTHINGONRESIZE,
                             Hv_FEEDBACKDATACOLS,  11,
                             Hv_FEEDBACKENTRY,     LOCALXY,
                                                   Hv_helveticaBold14,
                                                   Hv_black,
                                                 " [X, Y]: ",
                             NULL);
}
```

Note!

Any routine that is used for setting up a view, and is passed via the `Hv_INITIALIZE` attribute to **`Hv_VaCreateView`**, should be declared as expecting one argument: the actual view being initialized.

The first step in this function is a call to create the "camera" print button and the drawing tools, which are actually Hv_Items (**`Hv_BUTTONITEM`**s, to be precise.) This is one of the ugliest $H^{\underline{v}}$ calls. The prototype for the function is:

```
void Hv_StandardTools(Hv_View_View,
                       short left,
                       short top,
                       short orientation,
                       short resizepolicy,
                       int   whattools,
                       Hv_Item *first,
```

```
                                Hv_Item *last);
```

The first argument is the view that will hold the items. In this case it was the one that H$\underline{\text{v}}$ passed into the **`SetupView`** function.

The next two arguments are the left and top of the first item created, which in this case will be the camera button used for printing. We hope that it is clear how we are positioning the item just above the upper left of the hotrect.

The orientation argument specifies whether you want a horizontal row of tools, by passing `Hv_HORIZONTAL`, or a vertical column of tools by passing `Hv_VERTICAL`.

The **`resizepolicy`** argument specifies how we want the tools to behave if the View is resized. In this case, we don't want them to do anything. The resizepolicy is relevant for control items (as these are) and its value depends on the position of the item relative to the hotrect. This will be discussed later (See 7.7, page 67).

The **`whattools`** argument is a mask specifying which of the set of possible tools we want added. In this case we want all of them. If we didn't want the camera, for example, we would pass `Hv_ALLTOOLS - Hv_CAMERATOOL`.

The last two arguments, **`first`** and **`last`**, are pointers to Hv_Items. The first and last tool added will be returned in those variables. This is often useful for aligning other items in more complicated views.

After we create the tools, we get the **`area`** of the last tool, which is a rectangle that surrounds the item. We use this to position the next and last item we are going to add, the feedback. The feedback is created with the variable length argument list function **`Hv_VaCreateItem`**.

The first and only required argument is the view that will hold the item. The next attribute, `Hv_TYPE`, specifies what type of item we are creating, in this case an `Hv_FEEDBACKITEM`. The next three attributes `Hv_LEFT`, `Hv_TOP`, and `Hv_RESIZEPOLICY` should now be familiar.

The `Hv_FEEDBACKDATACOLS` attribute tells H$\underline{\text{v}}$ how to size itself: anticipate no more than this many characters in the feedback string. Nothing will "bomb" if you ask it to display more text, it just might get clipped.

The most interesting attribute in all of H$\underline{\text{v}}$ just might be the `Hv_FEEDBACKENTRY` attribute. That is because it is the only one that uses more than one value: it uses four. The four values must appear in order after the attribute. They are: tag (short), font (short), color (short) and label (char *). In our example we have just one feedback entry. Its tag is the user provided USERXY. Its font is Hv_HelveticaBold14. Its color is Hv_black, and its label is `" [X, Y]: "`.

Usually there are many feedback entries, not just one. Generally they are added by placing additional `Hv_FFEDBACKENTRY, tag, font, color, label` "quintets" in the same **`Hv_VaCreateItem`** call.

The last item of business is to see how *hvdraw* makes use of its sole feedback entry.

4.5.3 *hvdraw*'s feedback.c: and introduction to pointer tracking

Back when we created the view with **Hv_VaCreateView**, the Hv_FEEDBACK attribute specified the user provided function **Feedback** to be called to track the pointer. That is found in the file feedback.c:

```
/* ------ Feedback ---------- */

void Feedback(Hv_View View,
              Hv_Point pp)

{
  char            text[40];
  short           l, t, r, b;

  Hv_GetRectLTRB(Hv_GetViewHotRect(View), &l, &t, &r, &b);

/* make the feedback relative to the hotrect corner */

  pp.x -= l;
  pp.y -= t;

  sprintf(text, "[%-d, %-d]", pp.x, pp.y);
  Hv_ChangeFeedback(View, LOCALXY, text);
}
```

This routine will be called *anytime* there is a pointer motion event when the pointer is over a non-occluded portion of this particular view's hotrect. All user feedback routines should be declared with the same prototype. This is a particularly naive routine. It is hopefully clear that all it does is take the pointer location, passed in local coordinates via the Hv_Point pp, and, after offsetting so that (0, 0) is the upper left of the hotrect rather than the main window, displays them.

Most feedback routines will be much more complicated, and will search over items in the hotrect, find out which one the pointer is over, and display information.

4.6 Summary

In this section, especially the part dealing with *hvdraw*, we have provided an introduction into all the essential components of an $H^{\underline{V}}$ application: Initialization, view creation, view "setup", item creation, and feedback creation. We will now proceed to higher levels of detail.

Hv Chapter:

5 Creating Hv Views

5.1 View Attributes

As we have already seen, views are created by calling **Hv_VaCreateView**. As an additional example, here is how a view might be created in an application that will display maps:

```
Hv_VaCreateView(&view
                Hv_TAG               MERCATORVIEW,
                Hv_INITIALIZE,       SetupMercatorView,
                Hv_CUSTOMIZE,        DoMercatorCustomize,
                Hv_FEEDBACK,         MercatorFBFill,
                Hv_XMIN,             -Hv_PI,
                Hv_XMAX,             Hv_PI,
                Hv_YMIN,             -1.51,
                Hv_YMAX,             2.43625,
                Hv_HOTRECTWIDTH,     621,
                Hv_HOTRECTHEIGHT,    390,
                Hv_HOTRECTRMARGIN    100,
                Hv_TITLE,           " Mercator View ",
                NULL);
```

The value associated with the `Hv_TAG` attribute is a **short** (in this case via a #def `MERCATORVIEW`) that serves as a tag for that specific view, or more commonly as a tag for a *type* of view, so that the application might create any number of `MERCATORVIEW` views that then could be readily distinguished from, say, `ORTHOGRAPHICVIEW` views. The value provided to the `Hv_TAG` attribute in the example above is not an Hv constant (hence no **Hv_** prefix.) It is an application specific tag. Developers use the tags so that later they can perform a switch:

```
tag = Hv_GetViewTag(view)
switch (tag) {  /*or the politically incorrect switch(view->tag) */
case MERCATORVIEW:
       {whatever}
       break;
case ORTHOGRAPHICVIEW
       {whatever}
       break;
}
```

Note!

Developers should use positive numbers for tags, leaving the negative numbers for Hv's internal use. Presently Hv only predefines three views: the Welcome view, the XY plot view, and the Virtual Desktop view to which it assigns the tags **Hv_WELCOMEVIEW** (-1001), **Hv_XYPLOTVIEW** (-1002) and **Hv_VIRTUALVIEW** (-1003).

Continuing with the above view creation example, The value associated with the **Hv_INITIALIZE** attribute is a function that will be called *immediately* after Hv creates the view. It should be declared similarly to:

```
void SetupMercatorView(View)

Hv_View   View;

{
}
```

It is in this setup function where a view will typically create its initial suite of items.

The next two attribute-value pairs refer to user-provided functions (**`DoMercatorCustomize`** and **`MercatorFBFill`**) that will be called, respectively, when the user selects **`Customize`** from the view popup and when the pointer is in the HotRect. These will be discussed more in following sections.

The next four, corresponding to the attributes `Hv_XMIN` through `Hv_YMAX`, specify the world system that you want to correspond *to the HotRect*.

The remaining attributes `Hv_HOTRECTWIDTH` to `Hv_TITLE` were discussed in the *hvdraw* section and we assume that they are now familiar. *As always, the list must be NULL terminated!*

Note!

All the possible attributes have default values. For example, we have chosen to let $H^{\underline{V}}$ choose values for the placement of the view. (It will stagger them from the upper left of the main window). We could have used Hv_LEFT and Hv_TOP attributes to place the view ourselves. The attributes in this example represent a fairly typical view creation but by no means are they exhaustive.

This would seem to be the time for tabulating *all* attributes that are relevant for view creation:

Attribute	Comment	Type	Default
`Hv_ADDHOTRECT`	Determines whether the view has a HotRect	short	True
`Hv_CUSTOMIZE`	Procedure called when the user selects `customize` fro the view's popup menu. Should be declared as: `void ViewCustomize(Hv_View View) { }`	void * ()	NULL
`Hv_DATA`	A place for the developer to hang any data structure.	void *	NULL
`Hv_DESTROY`	Procedure called when the user selects delete from the view's popup menu. It is only needed if the user has allocated memory using the view's data field. The user code should use this routine to free the memory, especially any making use of the the data hook provided by the `Hv_DATA` attribute. The basic memory used by a view will be freed by $H^{\underline{V}}$. The routine should be declared as: `void ViewDestroy(Hv_View View) { }`	void * ()	NULL
`Hv_DRAWCONTROL`	A mask used to specify various drawing properties.	int	0

TABLE 3. Attributes relevant for $H^{\underline{V}}$ views. The more important attributes for new users are shown in boldface.

Attribute	Comment	Type	Default

`Hv_FEEDBACK`	The function that will be called to update the feedback area. It should be declared as:
	`void Feedback(Hv_View View,`
	`               Hv_Point pp)`
	`{`
	`}`
		void * ()	NULL

`Hv_HEIGHT`	Used to force a height (pixels) on the view. Usually, it is better to use the “margin” attributes and let H^{V} calculate this.	short	calculated
`Hv_HOTRECTBMARGIN`	The margin (pixels) placed below the bottom of the hotrect.	short	15
`Hv_HOTRECTCOLOR`	The background color of the hotrect.	short	Hv_gray11
`Hv_HOTRECTHEIGHT`	The height (pixels) of the hotrect.	short	400
`Hv_HOTRECTLMARGIN`	The margin (pixels) placed to the left of the hotrect.	short	15
`Hv_HOTRECTRMARGIN`	The margin (pixels) placed to the right of the hotrect.	short	15
`Hv_HOTRECTTMARGIN`	The margin (pixels) placed above the top of the hotrect	short	15
`Hv_HOTRECTWIDTH`	The width (pixels) of the hotrect.	short	400

`Hv_INITIALIZE`	The function that will be called immediately after H^{V} creates the view. It is where each view should create its initial suite of items. It should be declared as:
	`void Initialize(Hv_View View)`
	`{`
	`}`
		void * ()	NULL

`Hv_LEFT`	The left (pixels) of the view relative to the main window.	short	calculated
`Hv_MAXZOOMHEIGHT`	The maximum height in world coordinates to which the view can be zoomed to.	float	calculated
`Hv_MAXZOOMWIDTH`	The maximum width in world coordinates to which the view can be zoomed to.	float	calculated
`Hv_MINHEIGHT`	The minimum height (pixels) to which the view can be resized.	short	300
`Hv_MINWIDTH`	The minimum width (pixels) to which the view can be resized.	short	400
`Hv_MINZOOMHEIGHT`	The minimum height in world coordinates to which the view can be zoomed to.	float	calculated
`Hv_MINZOOMWIDTH`	The minimum width in world coordinates to which the view can be zoomed to.	float	calculated

`Hv_OFFSCREENUSERDRAW`	An advance attribute. It specifies a function called at the end of H^{V}’s offscreen drawing for a view that has been set up for off-screen drawing. It should be declared as:
	`void UserOffscreen(Hv_View  View,`
	`                   Hv_Region region)`
	`{`
	`}`
	For example, the demo hvmap uses offscreen drawing. Its maps are “background” items and are drawn offscrren. It then declares an offscreen user-draw to draw any overlays on the map and also a lat-long grid.
		void * ()	NULL

`Hv_OPENATCREATION`	If true, the view is automatically opened when created	short	False

TABLE 3. Attributes relevant for H^{V} views. The more important attributes for new users are shown in boldface.

Attribute	Comment	Type	Default
Hv_POPUPCONTROL	A mask used to determine what popup menu items are available for this view.	int	Hv_STANDARDPOPUPCONTROL
Hv_SIMINTERVAL	Interval (in milliseconds) between calls to the view's simulation process, if there is one.	int	1000
Hv_SIMPROC	The view's simulation process. This function will be called at definite intervals determined by the Hv_SIMINTERVAL attribute. It should be declared as: `void SimProc(Hv_View View) { }`	void * ()	NULL
Hv_TAG	An identifier, usually for a class of views. Note: developers should use positive tags leaving negative numbers reserved for Hv internal use.	short	-32767
Hv_TITLE	The title to appear on the top of the view	char *	NULL
Hv_TOP	The top (pixels) of the view relative to the main window.	short	calculated
Hv_USERDRAW	Drawing routine called after Hv's routine drawing. It should be declared as: `void ViewUserDraw(Hv_View View, Hv_Region region) { }`	void * ()	NULL
Hv_USERDRAWCONTROL	An extra int that can be used by the developer for any purpose.	int	0
Hv_WIDTH	Used to force a width (pixels) on the view. Usually, it is better to use the "margin" attributes and let Hv calculate this.	short	calculated
Hv_XMAX	left of hotrect in world coordinates.	float	1.0
Hv_XMIN	right of hotrect in world coordinates.	float	0.0
Hv_YMIN	bottom of hotrect in world coordinates.	float	1.0
Hv_YMAX	top of hotrect in world coordinates.	float	0.0

TABLE 3. Attributes relevant for Hv views. The more important attributes for new users are shown in boldface.

5.2 Setting up a View

Setting up a view refers to the user routine specified via the `Hv_INITIALIZE` attribute when the view was created. It is in this routine where the view's initial suite of items is created. Consider a view-creation example from the *simple* demo included in the Hv distribution:

```
Hv_VaCreateView(&view,
          Hv_TAG,             SIMPLE_VIEW,
          Hv_LEFT,            left,
          Hv_TOP,             top,
          Hv_HOTRECTHMARGIN,  53,
          Hv_HOTRECTWIDTH,    400,
```

```
Hv_HOTRECTHEIGHT,   680,
Hv_XMAX,            0.4,
Hv_YMAX,            0.66,
Hv_INITIALIZE,      SetupSimpleView,
Hv_TITLE,           text,
Hv_POPUPCONTROL,    Hv_STANDARDPOPUPCONTROL - Hv_QUICKZOOM,
Hv_DRAWCONTROL,     Hv_STANDARDDRAWCONTROL - Hv_ZOOMONRESIZE,
Hv_FEEDBACK,        simpleFeedback,
Hv_OPENATCREATION,  True,
NULL);
```

After H^{v} has created the view, it will call the user routine **`SetupSimpleView`**, with the newly created view as the only argument. The canonical structure for a setup routine is:

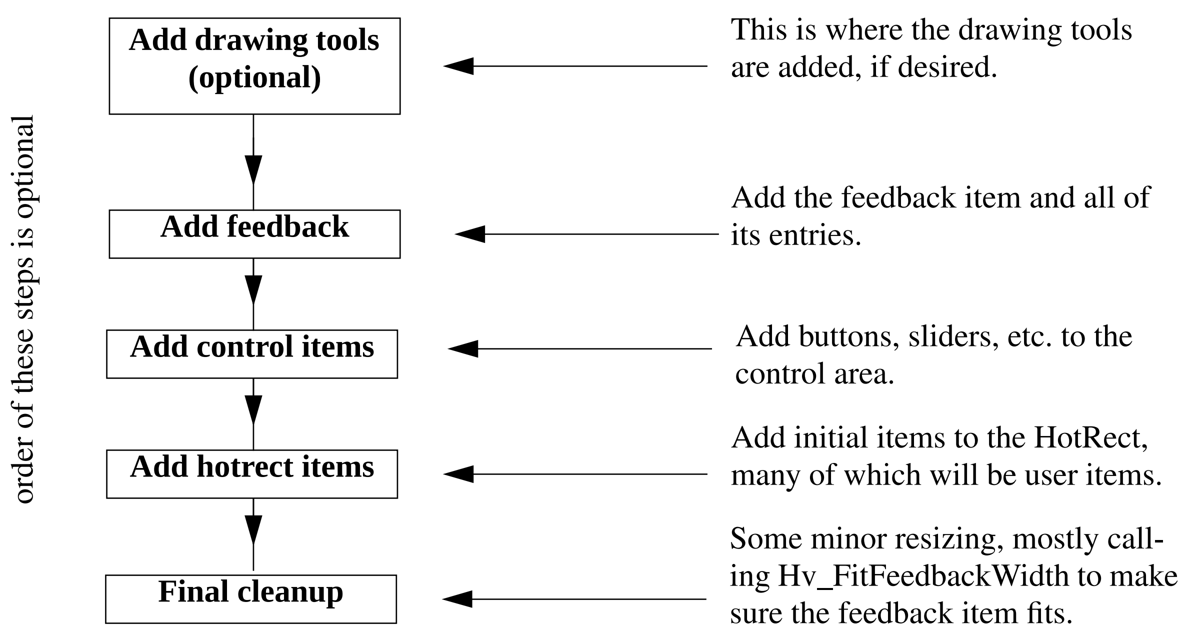


FIGURE 19. Normal sequence inside a View's setup routine.

We demonstrate this by example: the code that sets up the view in the *simple* demo. Note that *simple* is a do-nothing application, its only purpose is to demonstrate many of the control items available in H^{v}. It is somewhat equivalent to the *Motif* "periodic table of widgets" demo. See the actual source for more details regarding the **`simpleControlItems`** and **`simpleHotRectItems.`** Adding feedback will be discussed in more detail in the next section.

```
/* --------- SetupSimpleView -------- */

void SetupSimpleView(Hv_View View)

/* setup the View */

{
  Hv_Item   Item, Feedback;
```

```
  Hv_Rect    *hotrect = View->hotrect;

/* create a camera and drawing tools */

  Hv_StandardTools(View, hotrect->left - 45, hotrect->top, Hv_VERTICAL,
          Hv_DONOTHINGONRESIZE, Hv_ALLTOOLS, &Item, &Item);

/* create a feedback item */

  Feedback = Hv_VaCreateItem(View,
                 Hv_TYPE,                Hv_FEEDBACKITEM,
                 Hv_LEFT,                hotrect->right + 15,
                 Hv_TOP,                 hotrect->top-15,
                 Hv_FEEDBACKDATACOLS,    24,
                 Hv_FEEDBACKENTRY,       LOCALX, Hv_fixed2, Hv_black, "Local X: ",
                 Hv_FEEDBACKENTRY,       LOCALY, Hv_fixed2, Hv_black, "Local Y: ",
                 Hv_FEEDBACKENTRY,       Hv_SEPARATORTAG,
                 Hv_FEEDBACKENTRY,       WORLDX, Hv_fixed2, Hv_blue, "World X: ",
                 Hv_FEEDBACKENTRY,       WORLDY, Hv_fixed2, Hv_blue, "World Y: ",
                 Hv_FEEDBACKMARGIN,      15,
                 NULL);

  simpleControlItems(View, Feedback);
  simpleHotRectItems(View);
```

5.3 Adding Feedback

Adding feedback is very easy. The example above shows how it is done in the *simple* demo. Feedback consists of:

- A Feedback *Item*, and
- one or more feedback *entries*.

Like all H^{V} items, the feedback entry is added via a call to **`Hv_VaCreateItem`**. The entries are included in the variable length argument list. The feedback entries are the most unusual of arguments in a variable length argument list in that instead of an attribute-value pair, they consist of an attribute (`Hv_FEEDBACKENTRY`) and four values: an integer *tag*, a *font*, a *color*, and a *prompt*. The tag should be a positive number unique to that view, it is how you will identify which entry to modify, as discussed below. The feedback entries will be displayed in the feedback item as the prompt followed by the data. The feedback created in this example is shown in Fig. 20.

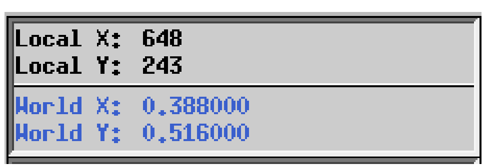


FIGURE 20. The feedback created for the *simple* demo.

To see how the feedback is updated, refer back to the previous example of view creation (See 5.2, page 45). There the function **`simpleFeedback`** was passed with the `Hv_FEEDBACK` attribute. This function will be called anytime the pointer is moving in the HotRect of the corresponding view. The developer has to take the pointer position (which will be passed to the feedback function) and update feedback entries as appropriate. This is perhaps best shown by referring to the code for **`simpleFeedback`**:

```
/* ------ simpleFeedback ---------- */

void simpleFeedback(Hv_View View,
                    Hv_Point pp)

/* NOTE the View and mouse location will be passed to any feedback
   function, thus they should always be declared as in this example */

{
  float             worldx, worldy;
  char              text[20];

/* obtain the world coordinates from the mouse location */

  Hv_LocalToWorld(View, &worldx, &worldy, pp.x, pp.y);

/* use text as a buffer to update all the entries */

  sprintf(text, “%-d”, pp.x);
  Hv_ChangeFeedback(View, LOCALX, text);

  sprintf(text, “%-d”, pp.y);
  Hv_ChangeFeedback(View, LOCALY, text);

  sprintf(text, “%-f”, worldx);
  Hv_ChangeFeedback(View, WORLDX, text);

  sprintf(text, “%-f”, worldy);
  Hv_ChangeFeedback(View, WORLDY, text);
}
```

Note!

Many things to note here! The first is that only one feedback item is permitted for each view. This restriction may be lifted in the future. One can use **`Hv_BoxItem`**s to effectively get around this restriction, if you find it necessary to do so. Also, entries can be added (or deleted) after the feedback item was created via the **`Hv_AddFeedbackEntry`** and **`Hv_AddFeedbackSeparator`** functions. Finally, in more serious examples of feedback, the developer will take the mouse location passed to the feedback function and see what HotRect items it is inside and then display information accordingly.

5.4 Adding Customization

Customization refers to the routine that is called when the user chooses **`Customize`**... from the view’s popup menu. H^{V} will call the routine that was passed along with the `Hv_CUSTOMIZE` attribute when the view was created (See the example in Sec. 5.1, page 42). It should bring up a dialog box that permits the user to customize the view in some manner. There is no standard H^{V} customize routine, since they are very view-specific.

Below is an example of a customization routine taken from the *hvmap* demo. It will be a bit mysterious if you haven’t read the sections on dialog boxes yet, but may be useful later as a reference. Note that all customization routines must be declared like the example below, accepting one argument: the corresponding view from which **`Customize`**... was selected.

```
/* -------- ViewCustomize --------*/

void ViewCustomize(View)

Hv_View     View;
```

```
{
  static Hv_Widget       dialog = NULL;
  Hv_Widget              dummy, rowcol, rc, rca, rcb, rc1, rc2;
  static Hv_Widget       dlat, dlong;
  Hv_ViewMapData         mdata;
  int                    answer;

/* mdata is a private data structure holding info about how maps are displayed*/

  mdata = Hv_GetViewMapData(View);

/* if the dialog wasn't created yet, do it now */

  if (!dialog) {
    Hv_VaCreateDialog(&dialog, Hv_TITLE, " Customize ", NULL);

    rowcol = Hv_DialogColumn(dialog, 3);

    rc  = Hv_DialogRow(rowcol, 8);
    rca = Hv_DialogColumn(rc, 3);
    rcb = Hv_DialogColumn(rc, 3);
    rc1 = Hv_DialogTable(rca, 4, 2);
    rc2 = Hv_DialogTable(rcb, 3, 2);

    glabel     = Hv_SimpleColorLabel(rc1, "        Grid ", EditColors);
    gtlabel    = Hv_SimpleColorLabel(rc1, " Grid Labels ", EditColors);
    wflabel    = Hv_SimpleColorLabel(rc1, "  Water fill ", EditColors);
    wblabel    = Hv_SimpleColorLabel(rc1, "Water border ", EditColors);
    lflabel    = Hv_SimpleColorLabel(rc2, "   Land fill ", EditColors);
    lblabel    = Hv_SimpleColorLabel(rc2, " Land border ", EditColors);
    spacelabel = Hv_SimpleColorLabel(rc2, " Outer Space ", EditColors);

    dummy = Hv_SimpleDialogSeparator(rowcol);
    rc = Hv_DialogTable(rowcol, 2, 4);
    dlat   = Hv_SimpleTextEdit(rc, "Latitide  grid size", NULL, 4);
    dlong  = Hv_SimpleTextEdit(rc, "Longitude grid size", NULL, 4);

    rc = Hv_StandardActionButtons(rowcol, 80, Hv_OKBUTTON + Hv_APPLYBUTTON);
 }


  outerspace    = mdata->outerspace;
  waterfill     = mdata->waterfill;
  waterborder   = mdata->waterborder;
  landfill      = mdata->landfill;
  landborder    = mdata->landborder;
  gridcolor     = mdata->gridcolor;
  gridtextcolor = mdata->gridlabelcolor;

/* set the colors to their present values */

  Hv_ChangeLabelColors(wflabel, -1, waterfill);
  Hv_ChangeLabelColors(wblabel, -1, waterborder);
  Hv_ChangeLabelColors(lflabel, -1, landfill);
  Hv_ChangeLabelColors(lblabel, -1, landborder);
  Hv_ChangeLabelColors(glabel,  -1, gridcolor);
  Hv_ChangeLabelColors(gtlabel, -1, gridtextcolor);
  Hv_ChangeLabelColors(spacelabel,  -1, outerspace);

/* set the text edit fields to their present values */

  Hv_SetFloatText(dlat,  mdata->latstep,  2);
  Hv_SetFloatText(dlong, mdata->longstep, 2);

/* ready to go! */

  while ((answer = Hv_DoDialog(dialog, NULL)) > 0) {

    mdata->waterfill      = waterfill;
    mdata->waterborder    = waterborder;
    mdata->landfill       = landfill;
    mdata->landborder     = landborder;
```

```
    mdata->gridcolor      = gridcolor;
    mdata->gridlabelcolor = gridtextcolor;
    mdata->outerspace     = outerspace;

    if (mdata->projection == Hv_MERCATOR)
      View->hotrectborderitem->scheme->fillcolor = mdata->waterfill;
    else
      View->hotrectborderitem->scheme->fillcolor = mdata->outerspace;

    mdata->latstep  = Hv_fMax(0.1, Hv_GetFloatText(dlat));
    mdata->longstep = Hv_fMax(0.1, Hv_GetFloatText(dlong));

    Hv_SetViewDirty(View);
    Hv_DrawViewHotRect(View);

    if (answer != Hv_APPLY)
      break;
  }
}
```

As mentioned, if you haven't studied dialog boxes yet, do not sweat over the details of this example.

If you don't want to provide any view customization, then disable the choice via the `Hv_POPUPCONTROL` attribute (See "View Attributes" on page 42).

5.5 Adding Quick Zooms

Quick zooms are accessed through the view's popup menu. Their purpose is so that the user can zoom in (or out) to interesting predefined worlds. Using customized quick zoom requires the following steps:

- In your application's private initialization phase, set the global function pointer **`Hv_CustomizeQuickZoom`** to the function that you will use to make sure all the labels are appropriate for the given view. This is called just before the popup is popped up, and allows you to have different labels for different views. (This is necessary because in actuality there is but one popup which is shared by all the views, so the labels have to be reset every time.) This function should be declared as:

```
void CustomizeQuickZoom(Hv_View View)
```

- Also in the private initialization phase, set the global function pointer **`Hv_QuickZoom`** to the function that you want H$^{\underline{V}}$ to call when the quick zoom menu is used. The function should be declared as:

```
void DoQuickZoom(Hv_Widget   w,
                 Hv_Pointer  clientdata)
```

- In the function declared above, you must manually zoom based on which quickzoom was selected.

All this is probably best illustrated by some sample code from *ced*. As usual, we advise you to ignore details and just concentrate on how everything fits together. (Remember that any routine that doesn't start with "Hv_" are private.)

In the initialization it sets up the global variables:

```
Hv_CustomizeQuickZoom = CustomizeQuickZoom;   /* to modify labels */
Hv_QuickZoom = DoQuickZoom;                   /* to handle selection */
```

The customization routine is:

```
/*------------- CustomizeQuickZoom ------------*/

void CustomizeQuickZoom(Hv_View View)

{
  ViewData     viewdata;
  int          sect;
  int          i;

  viewdata = GetViewData(View);

/* called EVERYTIME view popup is initiated */

  if (View->tag == SECTOR) {
    sect = abs(viewdata->visiblesectors);
    Hv_SetQuickZoomLabel(0, SectorViewQZLabel(sect, 1));
    Hv_SetQuickZoomLabel(1, SectorViewQZLabel(sect, 2));
    Hv_SetQuickZoomLabel(2, SectorViewQZLabel(sect, 3));
    Hv_SetQuickZoomLabel(3, SectorViewQZLabel(sect, 4));
    sect += 3;
    Hv_SetQuickZoomLabel(4, SectorViewQZLabel(sect, 1));
    Hv_SetQuickZoomLabel(5, SectorViewQZLabel(sect, 2));
    Hv_SetQuickZoomLabel(6, SectorViewQZLabel(sect, 3));
    Hv_SetQuickZoomLabel(7, SectorViewQZLabel(sect, 4));

    for (i = 8; i < Hv_NUMQUICKZOOM; i++)
      Hv_SetQuickZoomLabel(i, "unused");

  }
  else {
    Hv_SetQuickZoomLabel(0, "Sector 1");
    Hv_SetQuickZoomLabel(1, "Sector 2");
    Hv_SetQuickZoomLabel(2, "Sector 3");
    Hv_SetQuickZoomLabel(3, "Sector 4");
    Hv_SetQuickZoomLabel(4, "Sector 5");
    Hv_SetQuickZoomLabel(5, "Sector 6");
    for (i = 6; i < Hv_NUMQUICKZOOM; i++)
      Hv_SetQuickZoomLabel(i, "unused");


  }

}
```

Notice how it will change the labels depending on the view's tag. That's the whole idea: $H^{\underline{v}}$ will call your customizer just before the popup is display and it is up to you to change the labels appropriately.

The routine that actually handles the selection is:

```
/*----------- DoQuickZoom --------*/

void DoQuickZoom(Hv_Widget   w,
        Hv_Pointer  clientdata)

{
  int            menuitem;
```

```
  menuitem = (int)clientdata;

  if (Hv_hotView->tag == SECTOR) {
    SectorViewQuickZoom(Hv_hotView, menuitem+1);
    return;
  }
  else if (Hv_hotView->tag == FORWARDEC) {
    if (menuitem == 0)
      Hv_QuickZoomView(Hv_hotView, -580.0, 0.0, -290.0, 290.0);
    else if (menuitem == 1)
      Hv_QuickZoomView(Hv_hotView, -628.0, 0.0, 0.0, 628.0);
    else if (menuitem == 2)
      Hv_QuickZoomView(Hv_hotView, 0.0, 628.0, 0.0, 628.0);
    else if (menuitem == 3)
      Hv_QuickZoomView(Hv_hotView, 0.0, 580.0, -290.0, 290.0);
    else if (menuitem == 4)
      Hv_QuickZoomView(Hv_hotView, 0.0, 628.0, -628.0, 0.0);
    else if (menuitem == 5)
      Hv_QuickZoomView(Hv_hotView, -628.0, 0.0, -628.0, 0.0);
  }
  else
    return;

}
```

Notice how it turns the **`clientdata`** into an integer (starting at zero) that tells you which quickzoom was selected, and then manually zooms the view based on the view tag.

Note!

The example above uses the global variable **`Hv_hotView`** (which is of type **`Hv_View`**). This variable refers to the view that has most recently received a mouse button event. That is how we know which view the popup is referring to. There is also a global variable **`Hv_hotItem`** (of type **`Hv_Item`**) which, if not NULL, refers to the most recent item that received a mouse button event.

5.6 View User-Draw

The standard algorithm for drawing a view is to go through the view's list of items drawing each item in turn. However, the developer can add a custom drawing routine that $H^{\underline{V}}$ will call *after* drawing the view's item list. An example would be a program that draws maps and then overlays a latitude-longitude grid. You *could* make the grid a user-item. However, if you are sure that you always want the grid drawn last, then you could use the user-draw feature, activated at view creation through the `Hv_USERDRAW` attribute. Modifying the previous view creation example

```
Hv_VaCreateView(&view
             Hv_TAG              MERCATORVIEW,
             Hv_USERDRAW,        DrawLatLongGrid
             Hv_INITIALIZE,      SetupMercatorView,
             Hv_CUSTOMIZE,       DoMercatorCustomize,
             Hv_FEEDBACK,        MercatorFBFill,
             Hv_XMIN,            -Hv_PI,
             Hv_XMAX,            Hv_PI,
             Hv_YMIN,            -1.51,
             Hv_YMAX,            2.43625,
             Hv_HOTRECTWIDTH,    621,
             Hv_HOTRECTHEIGHT,   390,
             Hv_HOTRECTRMARGIN  100,
             Hv_TITLE,          " Mercator View ",
             NULL);
```

You would then declare the custom drawing routine as:

```
void DrawLatLongGrid(Hv_View   View,
                     Hv_Region  region)


{
}
```

Note!

The view drawing algorithm is, as you might expect, actually a bit more sophisticated. First, before any part of a view is drawn, other views that fall on top of the given view are “clipped”, so that only the visible part of the view is redrawn. Furthermore, this reduced drawing area is further clipped against only the portion of the main window that needs redrawing, which in many instances is only a small area. Also, the formal parameter **`region`** in the custom drawing routine is this “reduced” region that needs to be drawn. *It will have already been clipped to insure that no drawing occurs outside the view’s visible region. Thus novice programmers can safely ignore it.* Experts can use it to optimize the drawing.

5.7 Using View DrawControl Bits

Both views and items uses bits in a drawcontrol field to control behavior. Each bit is given a mnemonic. The table below lists the bits available for views. Note that some of the view draw control bits should be set only during creation. When the view is created, the `Hv_DRAWCONTROL` attribute is used to specify the initial drawcontrol.

Bit Mnemonic	Comment	When it can be set
Hv_SQUARE	If set, HotRect aspect ratio will be maintained during a zoom.	anytime
Hv_ACTIVE	If set, view is initially visible (afterwards controlled by H$\underline{\text{V}}$).	creation
Hv_ZOOMABLE	If set, view can be zoomed.	creation
Hv_ENLARGEABLE	If set, view can be enlarged.	creation
Hv_SCROLLABLE	If set, view can be scrolled (when appropriate).	creation
Hv_ZOOMONRESIZE	Controls resize policy. Toggled via view popup.	anytime
Hv_NOHOTRECTFILL	Hotrect is not “painted”. Can be used to prevent “flashing” updates.	anytime
Hv_SAVEBACKGROUND	Causes the background to be save for faster (memory expensive) updates	anytime
Hv_DIRTY	Used (mostly internally) in conjunction with Hv_SAVEBACKGROUND. When set, it indicates that the saved background needs updating.	anytime
Hv_STANDARDDRAWCONTROL	Defined as the combination of bits: Hv_STANDARDDRAWCONTROL = Hv_ZOOMABLE+Hv_ENLARGEABLE+Hv_SCROLLABLE+Hv_SQUARE+Hv_ZOOMONRESIZE	

TABLE 4. drawcontrol bits for views.

5.8 Using the Save-Background Feature

As discussed, $H^{\underline{v}}$ restores application views after it has been notified about an exposure event. There are two common ways that GUIs accomplish this:

- Simply redraw the affected region by redrawing all items that have any overlap.
- "Instantly" restore by copying an offscreen buffer containing a copy of what was visible back onto the affected view.

The choice between the two methods is the classic trade-off between memory and speed. Many windowing systems allow you to choice between the two methods.

$H^{\underline{v}}$ takes a novel, more flexible approach by allowing you to do a combination of both. By default, it will use the first (i.e. redraw) method. However, you can also do "background" saves by

- Setting the `Hv_SAVEBACKGROUND` bit in a view's drawcontrol.
- Designating which HotRect items are to be in this saved background via the `Hv_INBACKGROUND` bit of the items draw control.

Only HotRect items can be put into the saved background. $H^{\underline{v}}$ will then save an offscreen buffer containing all background items and restore it instantly when it has to redraw, and then draw the remaining items normally.

For example, if one is creating an application where items are drawn on maps, it is a good idea to have the map be a background item, Then you can place items on the map and drag them around without worrying the potentially slow map redraw.

Hv
Chapter:

6 Modifying and Creating Menus

If you followed the view creation of the previous chapter, then modifying and creating menus will be a snap.

Note!

Terminology: *menu* refers to the entire pulldown and is referred to by the title, e.g. the **`Action`** menu. The menu has *menu items* also known as *selections* which are referred to by the label of the individual choice, e.g. "**`Quit...`**".

6.1 Modifying a standard Hv menu

First we discuss how one adds to one of the predefined Hv menus: **`Action`**, **`Views`**, and **`Help`**. Actually, you probably never want to any selections to the **`Views`** menu, since that will automatically grow and shrink as views are added and destroyed.

The three predefined menus are available as global variables of type Hv_Widget:

```
extern Hv_Widget     Hv_actionMenu, Hv_viewMenu, Hv_helpMenu;
```

Suppose we want to add a "New View" selection to the **`Action`** menu. This code can be found in the *hvmap* demo:

```
  item = Hv_VaCreateMenuItem(Hv_actionMenu,
              Hv_LABEL,             "New View",
              Hv_CALLBACK,          GetNewView,
              NULL);
```

GetNewView is an *hvmap* private routine:

```
/*--------- GetNewView -------*/

void GetNewView()

{
  Hv_View      view;

  view = NewView(MAPVIEW);
  Hv_OpenView(view);
}
```

Details are of course unimportant (it calls another private routine to create the new view and then **`Hv_OpenView`** to display the view; see the *hvmap* source.) Generally, new menu items are added to the end of the menu. The exception is the **`Action`** menu, where Hv will always keep the "Quit" selection at the bottom.

6.2 Creating an entirely new menu

To create an entirely new menu we must:

- First create the *menu*
- Then add the *items* (the selections)

Each item will have a callback. The callbacks can all be distinct, all (or some) of the items can share callbacks. It depends on how related the menu items are. If the item has a unique callback then there is no question of which selection invoked the callback. For shared callbacks there are two ways to decide which was the causal selection: The actual item (an **`Hv_Widget`**) will be passed to the callback, this could be compared to a set of global Hv_Widgets. Or, the second argument of the callback (an **`Hv_Pointer`**) can be cast to an integer. It will be (0, 1, 2, ...) depending on whether it is ordinally the (1st, 2nd, 3rd, ...) item on its menu.

Here is an example from the *ced* application, where an “Events” menu is added. Notice that the menu and items are created with variable length argument calls. It should be evident how the accelerators (keystroke equivalents) are being added. This is an example where all the items have unique callbacks. Notice at the end how the “Next Event” and “Previous Event” are “ghosted” (made unselectable) at initialization.

```
/* create an Event menu  */

Hv_Widget EventMenu, evopenmenu, evgotomenu, evnextmenu, evprevmenu, item.

  Hv_VaCreateMenu(&EventMenu, Hv_TITLE, " Events", NULL);

  item = Hv_VaCreateMenuItem(EventMenu,
               Hv_LABEL,             "Event Control...",
               Hv_CALLBACK,          EventControl,
               Hv_ACCELERATORCHAR,   'E',
               NULL);

  evopenmenu = Hv_VaCreateMenuItem(EventMenu,
                 Hv_LABEL,             "Open Event File...",
                 Hv_CALLBACK,          OpenEvent,
                 Hv_ACCELERATORCHAR,   'O',
                 NULL);

  item = Hv_VaCreateMenuItem(EventMenu, Hv_TYPE, Hv_SEPARATOR, NULL);

  evgotomenu = Hv_VaCreateMenuItem(EventMenu,
                 Hv_LABEL,             "Goto Event...",
                 Hv_CALLBACK,          GotoEvent,
                 Hv_ACCELERATORTYPE,   Hv_CTRL,
                 Hv_ACCELERATORCHAR,   'G',
                 NULL);

  evnextmenu = Hv_VaCreateMenuItem(EventMenu,
                 Hv_LABEL,             "Next Event",
                 Hv_CALLBACK,          NextEvent,
                 Hv_ACCELERATORTYPE,   Hv_CTRL,
                 Hv_ACCELERATORCHAR,   'N',
                 NULL);

  evprevmenu = Hv_VaCreateMenuItem(EventMenu,
                 Hv_LABEL,             "Previous Event",
                 Hv_CALLBACK,          PrevEvent,
                 Hv_ACCELERATORTYPE,   Hv_CTRL,
                 Hv_ACCELERATORCHAR,   'P',
```

```
                 NULL);

  item = Hv_VaCreateMenuItem(EventMenu, Hv_TYPE, Hv_SEPARATOR, NULL);

 Hv_SetSensitivity(evprevmenu,  False);   /* not available until events are read*/
 Hv_SetSensitivity(evgotomenu,  False);   /* not available until events are read*/
```

The callback for a menu item should follow this template:

```
/* generic menu item callback */

void DoQuickZoom(Hv_Widget   menuitem
                 Hv_Pointer  clientdata)

{
  int             menuitem;

  menuitem = (int)clientdata;

}
```

Hv
Chapter:

7 Creating Hv Items

In this chapter we discuss the "built-in" items available in the Hv library. Examples will be provided throughout the chapter. To start, we examine attributes and other aspects such as "Draw Control" that are common to all items.

7.1 Common Item Attributes

As with views, attributes used for Hv Items are sets of attribute-value pairs. The first element in the set is the *attribute specifier*, which identifies the attribute whose value is to be set. The second member of the set is the value. For example, to specify the width of an Hv Item to be 80 pixels, one would pass the set:

```
Hv_WIDTH,     80
```

in the argument list. The order of the sets is irrelevant. For example

```
newItem = Hv_VaCreateItem(View,
       Hv_TYPE,                 Hv_BUTTONITEM,
       Hv_WIDTH,                100,
       Hv_HEIGHT,               80,
       NULL);
```

and

```
newItem = Hv_VaCreateItem(View,
       Hv_HEIGHT,               80,
       Hv_WIDTH,                100,
       Hv_TYPE,                 Hv_BUTTONITEM,
       NULL);
```

are equivalent. All attributes have default values that will be used if an overwriting value is not passed as an argument.

Note!

In the code example above, the attribute sets can be in any order, but the first and last arguments (View and NULL) must appear as shown.

Not all attributes are relevant for all items. For example the `Hv_MAXVALUE` attribute is meaningful for wheels and sliders, but meaningless for text items.

Note!

Using “meaningless” attributes yields unpredictable results. Often they will be safely ignored. However, some of the attributes “piggy-back” on others, and thus even though it may appear to be an innocuous attribute, it may get interpreted as one that is meaningful for the item at hand. As a result, one might inadvertently set an attribute to an undesired value. You are safe if you stay within the set of “universal” attributes described below or the item specific attributes provided in each item’s description.

Attribute	Comment	type	Default
`Hv_DRAWCONTROL`	Sets the item’s draw control bits (See “Draw Control” on page 61)	int	0
`Hv_TAG`	Sets the item’s “tag”, used to distinguish among different types of user items.	short	
`Hv_USERDRAWCONTROL`	Another drawcontrol field provided for use by application.	int	
`Hv_USER1`	For use by application.	int	0
`Hv_USER2`	For use by application.	int	0
`Hv_CHILDPLACEMENT`	Specifies how an item should be placed relative to its parent. Should be one of: `Hv_POSITIONLEFT`, `Hv_POSITIONRIGHT`, `Hv_POSITIONABOVE`, `Hv_POSITIONBELOW`, or `Hv_POSITIONCENTER`.	int	none
`Hv_GRANDCHILDPLACEMENT`	Same as Hv_CHILDPLACMENT above, except placement is relative to the grandparent.	int	none
`Hv_LEFT`	The left pixel coordinate of the item’s area rectangle.	short	
`Hv_TOP`	The top pixel coordinate of the item’s area rectangle.	short	
`Hv_WIDTH`	The pixel width of the item’s area rectangle.	short	
`Hv_HEIGHT`	The pixel height of the item’s area rectangle.	short	

`Hv_SINGLECLICK`	Process that will be called if a single click occurs inside the item. It should be declared as:
	`void SingleClick(Hv_Event hvevent)`
	`{`
	`}`
		void *()	type dependent

`Hv_DOUBLECLICK`	Process that will be called if a single click occurs inside the item. It should be declared as:
	`void DoubleClick(Hv_Event hvevent)`
	`{`
	`}`
		void *()	type dependent

`Hv_USERDRAW`	Process that is called AFTER the standard drawing of an item. For user items, it is the only drawing routine called. It should be declared as:
	`void UserDraw(Hv_Item Item,`
	`Hv_Region region)`
	`{`
	`}`
		void *()	NULL

TABLE 5. Universal attributes applicable to <u>ALL</u> H^V Items.

Attribute	Comment	type	Default

`Hv_INITIALIZE`	The optional function that will be called immediately after H^{V} creates the item. It should be declared as:
	`void Initialize(Hv_Item Item,`
	`Hv_AttributeArray attributes)`
	`{`
	`}`
		void * ()	type dependent

`Hv_DESTROY`	Process that is called when an item is deleted. It should free any special data allocated and attached to the data pointer. It should be declared as
	`void DestroyItem(Hv_Item Item)`
	`{`
	`}`
		void * ()	NULL

`Hv_BALLOON`	This will be the string displayed in a balloon if the user turns balloon help on.	char *	NULL

`Hv_AFTERDRAG`	This option user process is called after an item is dragged, in case the item is complicated enough that additional maintenance is required. It should be declared as:
	`void AfterDrag(Hv_Item Item,`
	`short dh,`
	`short dv)`
	`{`
	`}`
	where dh and dv are the horizontal and vertical displacements (pixels) associated with the drag.
		void *()	NULL

`Hv_AFTEROFFSET`	This option user process is called after an item is offset, in case the item is complicated enough that additional maintenance is required. It should be declared as:
	`void AfterOffset(Hv_Item Item,`
	`short dh,`
	`short dv)`
	`{`
	`}`
	where dh and dv are the horizontal and vertical displacements (pixels) associated with the offset.
		void *()	NULL

`Hv_FIXREGION`	Process that is called when H^{V} suspects that the region may be corrupted. This routine should be able to recalculate an item's region and in general is needed only for comlicated user items based on a private coordinate system such as latitude-longitude.
	`void FixRegion(Hv_Item Item)`
	`{`
	`}`
		void *()	NULL

`Hv_CHECKENLARGE`	If the item has the **`Hv_ENLARGEABLE`** draw control bit set, this process will be called prior to a drag to see if the item should actually be enlarged rather than dragged.	void *()	NULL
`Hv_COLOR`	A basic color that may be used by the item.	short	Hv_black
`Hv_DRAWATCREATION`	If true, item is drawn as soon as created.	Boolean	False
`Hv_TEXT`	The text, for any item that displays text.	char *	NULL
`Hv_FONT`	The font, for any item that displays text.	short	Hv_fixed2

TABLE 5. Universal attributes applicable to <u>ALL</u> H^{V} Items.

Attribute	Comment	type	Default
Hv_TEXTCOLOR	Text color, for any item that displays text.	short	Hv_black
Hv_DATA	Generic pointer to user data. (For user items, it can be used to point at anything the developer wants. For standard items, it should not be used since they may use it to point at private data structures.)	void *	NULL
Hv_DOMAIN	One of two possibilities: Hv_INSIDEHOTRECT or Hv_OUTSIDEHOTRECT depending on where the item is to live. By default: all control items (buttons, sliders, etc.) are given the domain Hv_OUTSIDEHOTRECT while all user items are Hv_INSIDEHOTRECT.	short	type dependent
Hv_RESIZEPOLICY	This is relevant for control items only. It specifies how an item should respond to a resizing of the main window. It almost always should be Hv_SHIFTXONRESIZE, Hv_SHIFTYONRESIZE, or Hv_DONOTHINGONRESIZE. There are also the rarely used: Hv_OFFSETONRESIZE, Hv_RESIZEONRESIZE, Hv_RECENTERONRESIZE, and Hv_SHIFTXOFFYONRESIZE,	short	type dependent
Hv_RELATIVEPLACEMENT	When you want to position one item relative to another, use this (and the next two) attributes. It is usually used for control items, for example to create a row of buttons or to place the feedback just below a button. It should be one of five self-explanatory values: Hv_POSITIONLEFT, Hv_POSITIONRIGHT, Hv_POSITIONABOVE, Hv_POSITIONBELOW, or Hv_POSITIONCENTER. (See 7.7, page 67).	int	none
Hv_PLACEMENTITEM	This is the item to which a relative placement is measured against. For example, if you specified a relative placement of Hv_POSITIONABOVE, the current item will be placed above this item. The default placement item is the most recently created item (for the same view) , i.e., the previous item.	Hv_ITEM	previously created item
Hv_PLACEMENTGAP	In relative placements, sets the gap (pixels) between the current item and the placement item.	short	1 pixel

TABLE 5. Universal attributes applicable to ALL H^{V} Items.

7.2 Draw Control

Each Item has a 32 bit draw-control word wherein one sets and clears various draw-control "bits". The behavior of the item is affected by which bits are set. For example, one bit controls whether or not the item can be dragged, another whether it can be rotated, etc.

Draw control bits can be set at creation, via the Hv_DRAWCONTROL attribute, or after creation through the **Hv_SetItemDrawControlBit** and **Hv_ClearItemDrawControl** functions, and they can bet tested through the **Hv_CheckItemDrawControlBit** function. A list of draw-control bits is provided in Table 6. ***The default draw-control has all bits cleared***.

Mask (Bit)	Comment
Hv_HASBALLOON	This should be set for Items that have a balloon help string. It will automatically be set at creation if a string is passed via the Hv_BALLOON attribute.
Hv_ZOOMABLE	Typically set for HotRect Items and cleared otherwise, this bit contols whether or not an Item changes size what the view it belongs to is zoomed.
Hv_ENLARGEABLE	Item has one or more small rectangle “handles” used to enlarge/shrink
Hv_DRAGABLE	Item can be dragged about the HotRect.
Hv_DRAGGABLE	Same as Hv_DRAGABLE. Included because no agreement could be reached on how to spell “draggable”.
Hv_FEEDBACKWHENDRAGGED	If this bit is set, the Feedback will be updated as the Item is dragged.
Hv_FANCYDRAG	Item will be drawed more completely as it is dragged
Hv_ROTATABLE	Item can be rotated.
Hv_NOPOINTERCHECK	Item is not checked for pointer clicks.
Hv_CHECKAREAONLY	Only the item’s area (not its region) is used for checking pointer clicks.
Hv_NOUPDATEAFTERDRAG	No post-drag update function is called.
Hv_REDRAWHOTRECTWHENCHANGED	If this item is changed in any way, redraw the entire HotRect.
Hv_ALWAYSREDRAW	Always redraw this item (in response to an exposure) without checking for overlap.
Hv_HIGHLIGHTITEM	Highlight this item in some manner when editing.
Hv_CONFINETOPARENT	Keep this item within its parent’s boundary when dragging.
Hv_DONTDRAW	Do not draw this item (used for temporarily hiding an item.)
Hv_CONFINETOGRANDPARENT	Keep this item within its grandparent’s boundary when dragging.
Hv_DONTAUTOSIZE	(For text items) do not resize when text is changed
Hv_DIRTYWHENCHANGED	This is relevant for views that use the background save feature. It marks the view as “dirty” so that its offscreen background is redrawn.
Hv_INBACKGROUND	This item should be included in the offscreen drawing if the view is using that feature. Relevant only for HotRect items.
Hv_ROUNDED	Relevant only for buttons, it specifies that this is a “round” button.
Hv_WPOLYBASED	This item is a user item “based” on a world polygon
Hv_STRICTHRCONFINE	This item is STRICTLY confined to the HotRect, meaning that when it is dragged, no portion of it can slide off the edge.

TABLE 6. Draw Contol masks (bits) for Hv_Items. The default setting at Item creation is for all bits to be off.

7.3 A First Introduction to Creating Items

Items, like views are created by a single call to a NULL-terminated variable-length argument list H$\underline{\text{v}}$ function, **`Hv_VaCreateItem`**:

```
newitem = Hv_VaCreateItem(view, ..., NULL)
```

The first argument is required: **`view`** is the **`Hv_View`** wherein the item is to reside. It will have been previously created via **`Hv_VaCreateView`**.

The type of item is specified via the `Hv_TYPE` attribute. The value should be one of the following:

```
Hv_BOXITEM
Hv_BUTTONITEM
Hv_CHOICEITEM
Hv_CHOICESETITEM
Hv_CLOCKITEM
Hv_COLORBUTTONITEM
Hv_CONNECTIONITEM
Hv_CONTAINERITEM
Hv_FEEDBACKITEM
Hv_FRACTIONDONEITEM
Hv_GLOBEITEM
Hv_HOTRECTBORDERITEM
Hv_LEDBARITEM
Hv_MAPITEM
Hv_OPTIONBUTTONITEM
Hv_PLOTITEM
Hv_RAINBOWITEM
Hv_SEPARATORITEM
Hv_SLIDERITEM
Hv_SLIDERTHUMBITEM
Hv_SMALLICONITEM
Hv_TEXTENTRYITEM
Hv_TEXTITEM
Hv_USERITEM
Hv_WHEELITEM
Hv_WORLDLINEITEM
Hv_WORLDOVALITEM
Hv_WORLDPOLYGONITEM
Hv_WORLDRECTITEM
```

An example of item creation is:

```
Camera = Hv_VaCreateItem(View,
Hv_TYPE,         BUTTONITEM,
Hv_RESIZEPOLICY, Hv_SHIFTXONRESIZE,
Hv_SINGLECLICK,  Hv_CameraCallback,
Hv_USERDRAW,     Hv_DrawCamera,
Hv_LEFT,         left,
Hv_TOP,          top,
Hv_BALLOON,      "Use this \"camera\" to print this view.",
NULL);
```

which creates the “camera” button used to print views. You see that it is actually a button, has something called a resize policy, a routine that is called if we click on the button, some specialty drawing (in this case a representation of a camera) above and beyond the normal button drawing, it is being positioned at pixel coordinates (left, top), and has a balloon that might be displayed.

Note!

When items are created, they are given one of two *domains* via the `Hv_DOMAIN` attribute (or by default). The two possible domains are `Hv_INSIDEHOTRECT` and, not surprisingly, `HV_OUTSIDEHOTRECT`. You can put any item in either domain, but once created it cannot change domains. This allows $H^{\underline{v}}$ to provide for sensible behavior with regards to clipping, zooming and especially scrolling.

7.4 Anatomy of an Hv_Item

$H^{\underline{v}}$ decides whether an item needs to be drawn by checking for an overlap between the portion of the window being updated and either the item’s *region* or its *area*. The item’s area is an **`Hv_Rect`** that (usually) completely encloses it. This is all that is needed for most items, *even if they are not rectangular in shape*.

Complicated items can choose to use their region (an **`Hv_Region`**) instead. An **`Hv_Region`** can have a arbitrarily complicated shape (Fig. 21).

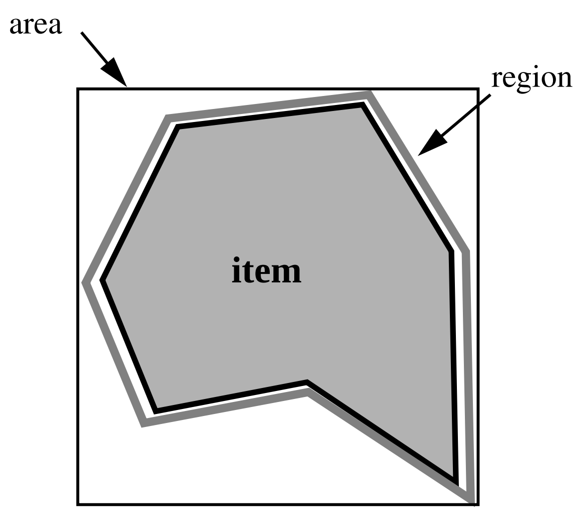


FIGURE 21. An item's region (if not NULL) is a form fitting shape about the item. Its area is an enclosing rectangle that may include a lot of "air".

Note!

For an item that uses its region, it is actually safe (and often convenient) to have the item' s area <u>smaller</u> than the region, perhaps as a small rectangle at the center of the item for use as a dragging "handle". For items not using a region, the area must completely enclose the item or redraws after exposure events may not work properly. That is because Hv first checks whether the region (if present) overlaps the update region, and if it does, sends a redraw message to the item. If the region isn't present Hv will check whether the item's area overlaps the update region.

When Hv is checking a particular item it uses the following algorithm. If the item's region is not NULL, it will use it to check for overlap. Otherwise it will use the item's area.

There are two reasons that you might want an item to use its region rather than its area. If it is expensive to draw, then checking the tightly fitting region instead of the loose fitting area will eliminate unnecessary drawing. Also, if your feedback function needs to know when the pointer is *absolutely, positively* inside (and not just nearby) an item it might have to use the region.

Regions are complicated and potentially slow. Use them only when necessary. Most items draw so quickly that an occasional unnecessary redraw due to its imprecise area will go unnoticed.

7.5 Hv Callback Functions

Users can designate callback functions for single and double clicks for useritems, or override callback functions for standard Hv items. The callback functions always take a single **`Hv_Event`** argument. As an example, we present the $H^{\underline{V}}$ provided callback used by the Camera button, present on many views, and used for printing the view.

```
void Hv_CameraCallback(Hv_Event hvevent)

{
  Hv_Item   Item = hvevent->item;
  Hv_View   View = hvevent->view;

  Hv_ToggleButton(Item);          /* push the button in*/
  Hv_SendViewToFront(View);       /* bring the view to the front */

  Hv_psWholeScreen = False;       /* only print this view */
  Hv_PrinterSetup(View);          /* get the print dialog box */
  Hv_ToggleButton(Item);          /* reset the button */
}
```

The callbacks can be added during item creation via the `Hv_SINGLECLICK` and/or `Hv_DOUBLECLICK` attributes.

The defaults for HotRect items, which are identified as those for which the `Hv_DOMAIN` attribute has the value `Hv_INSIDEHOTRECT`, are **`Hv_DefaultSingleClickCallback`** for single clicks and nothing **`(NULL)`** for double clicks. The effect of the default callback **`Hv_DefaultSingleClickCallback`** is to bring the item to the front, so that it is “on top” of all other items. (Or, if SHIFT is pressed simultaneously, to send the item to the back).

There is an issue of uniformity in style. We almost never change the default single click callback for HotRect items. We use the doubleclick callback if the item has an editor, meaning a dialog box that is popped up allowing the user to change attributes about the item. For control items, the single click callback is used for the action, such as clicking the camera button causing a view to be printed, while the double click callback is used less frequently since control items are not as likely to have editors.

Note!

Some control items should have editors, especially if they somehow imply a “range” that the user might want to change. `Hv_WHEELITEM`s, `Hv_RAINBOWITEMs`, and `Hv_SLIDERITEMs` all have editors.

Finally, we discuss how the **`modifiers`** field in an **`Hv_Event`** is used to check for either a shift, control, or an alt (mod1) key was pressed while the click that initiated the callback occurred. This is done via the bits `Hv_SHIFT`, `Hv_CONTROL` and `Hv_MOD1`, as in the following example:

```
void SampleCallback(Hv_Event hvevent)

{
Boolean   shiftwaspressed;
Boolean   controlwaspressed;
Boolean   mod1waspressed;

shiftwaspressed = Hv_CheckBit(hevent->modifiers, Hv_SHIFT);
controlwaspressed = Hv_CheckBit(hevent->modifiers, Hv_CONTROL);
mod1waspressed = Hv_CheckBit(hevent->modifiers, Hv_MOD1);

}
```

7.6 Item placement

Items can be placed

- Arbitrarily, using the `Hv_LEFT` and `Hv_TOP` attributes
- Relative to some other item
- Relative to their parent or grandparent

The first case, placing an item at a specific pixel location is presumably clear. We have already seen examples of this method.

Placing an item relative to another item is done at creation through three attributes summarized in the following table.

Attribute	Comment	Default
Hv_RELATIVEPLACEMENT	Should be one of `Hv_POSITIONBELOW`, `Hv_POSITIONABOVE`, `Hv_POSITIONLEFT`, `Hv_POSITIONRIGHT`, or `Hv_POSITIONCENTER`	none
Hv_PLACEMENTITEM	The item relative to which you want to position the new item	most recently created item
Hv_GAP	Gap in pixels between the item and the placement item.	1 pixel

TABLE 7. Attributes used in placing an item relative to another item.

A row of buttons could the be created by placing the first one absolutely and then all the others to the right of the previous one. Here is some sample code that sets up a row of buttons:

```
/*----- AddButtonRow -------*/

void AddButtonRow(Hv_View View, int numbutton, char **labels)

/* put a row of buttons to the right of the hotrect */

{

   short  left, top;
   Hv_Item button;

   left = View->hotrect->right + 4;
   top  = View->hotrect->top;

   for (i = 0; i < numbutton, i++)
      Button = AddButton(View, i == 0, left, top, labels[i]);
}


/*------ AddButton ------*/

/* add a single button */

void AddButton(Hv_View View, Boolean first, short left, short top, char *label)

{
   Hv_Item Item;
```

```
/* place the first button absoultely and the others relative-right */

    if (first) {
          Item = Hv_VaCreateItem(View,
                   Hv_RESIZEPOLICY,      Hv_DONOTHINGONRESIZE,
                   Hv_TYPE,              Hv_BUTTONITEM,
                   Hv_LEFT,              left,
                   Hv_TOP,               top,
                   Hv_SINGLECLICK,       MyCallbackk,
                   Hv_LABEL,             label,
                   Hv_FONT,              Hv_fixed2,
                   Hv_WIDTH,             BIGBUTTONWIDTH,
                   Hv_HEIGHT,            BIGBUTTONHEIGHT,
                   NULL);
    }
    else
          Item = Hv_VaCreateItem(View,
                   Hv_RESIZEPOLICY,      Hv_DONOTHINGONRESIZE,
                   Hv_TYPE,              Hv_BUTTONITEM,
                   Hv_RELATIVEPLACEMENT, Hv_POSITIONRIGHT,
                   Hv_SINGLECLICK,       MyCallbackk,
                   Hv_LABEL,             label,
                   Hv_FONT,              Hv_fixed2,
                   Hv_WIDTH,             BIGBUTTONWIDTH,
                   Hv_HEIGHT,            BIGBUTTONHEIGHT,
                   NULL);

    return Item;
}
```

We have assumed that the macros `BIGBUTTONWIDTH` and `BIGBUTTONHEIGHT` have been defined elsewhere as was the callback function **`MyCallback`**.

Placing a item relative to its parent/grandparent is very similar. Here are the relevant attributes:

Attribute	Comment	Default
Hv_CHILDPLACEMENT	Should be one of `Hv_POSITIONBELOW`, `Hv_POSITIONABOVE`, `Hv_POSITIONLEFT`, `Hv_POSITIONRIGHT`, or `Hv_POSITIONCENTER`	none
Hv_GRANDCHILDPLACEMENT	Should be one of `Hv_POSITIONBELOW`, `Hv_POSITIONABOVE`, `Hv_POSITIONLEFT`, `Hv_POSITIONRIGHT`, or `Hv_POSITIONCENTER`	none
Hv_GAP	Gap in pixels between the item and the parent or grandparent.	1 pixel

TABLE 8. Attributes used in placing an item relative its parent or grandparent.

If you do something silly like give the item a relative placement to another item AND a child placement I don’t actually know what would happen.

7.7 Resize Policy

An Item must be told what to do if its view is resized. The is done via the `Hv_RESIZEPOLICY` attribute at the item’s creation. If an item does not have the right resize policy, then when the view is resized the item might be lost or end up in some unexpected and unattractive location.

Note!

Resize policy is only relevant for control items. HotRect items will place themselves properly based on their world coordinates.

The notion of how the item should behave is simplified somewhat by the fact that H$^{\underline{V}}$ views can only be resized from the bottom right. A moment's thought should convince you that the following table covers most cases:

If the item is located:	Its resize policy should be
Above the HotRect	**`Hv_DONOTHINGONRESIZE`**
Right of the HotRect	**`Hv_SHIFTXONRESIZE`**
Below The HotRect	**`Hv_SHIFTYONRESIZE`**
Left of The HotRect	**`Hv_DONOTHINGONRESIZE`**

TABLE 9. Suggested resize policies for items based on their location relative to the HotRect.

7.8 Creating H$^{\underline{V}}$ User-Items

User-defined items are created much like standard items. The user must provide a drawing function via the `Hv_USERDRAW` attribute. For example:

```
Item = Hv_VaCreateItem(View,
                    Hv_Type,             Hv_USERITEM,
                    Hv_DRAWCONTROL,      Hv_DRAGGABLE,
                    Hv_USERDRAW,         userDrawMyItem,
                    Hv_BALLOON,          ("A simple user-item.",
                    Hv_LEFT,             View->hotrect->left + 80,
                    Hv_TOP,              View->hotrect->top  + 250,
                    Hv_WIDTH,             80,
                    Hv_HEIGHT,            80,
                    NULL);
```

The function **`userDrawMyItem`** must be declared as:

```
/*------ userDrawMyItem------*/

static void userDrawMyItem(Hv_Item Item, Hv_Region region)

{
}
```

Note!

The `Hv_USERDRAW` attribute can be set for standard items such as `Hv_BUTTONITEMs`. H$^{\underline{V}}$ will call the provided drawing function *after* it has called its own. This is, for example, how the camera is drawn on top of the button used for printing. The generic routine `Hv_DrawButtonItem` draws the basic 3D button; the specialized `Hv_DrawCamera` draws the camera on top. For `Hv_USERITEM`s, there is no "internal" drawing routine and thus the only one called is the one provided by the user via the `Hv_USERDRAW` attribute.

7.8.1 Piggy-Backing a User-Item onto a Standard H$^{\underline{V}}$ Item

Piggy-backing on an existing item, when applicable, is very fruitful. For example many items are drawn as world based polygons and can take advantage of the fact that H$^{\underline{V}}$ knows how to deal with them. The basic steps are:

- Create the item as one of the predefined item (e.g. **`Hv_WORLDPOLYGONITEM`**)
- Add your own private drawing routine via the `Hv_USERDRAW` attribute, which will be called after the standard drawing routine.
- Alternatively, if you don't want the standard drawing routine to be called prior to the drawing routine that you provide (i.e. you want to do all the drawing), then you can accomplish this by the politically incorrect assignment:

```
Item->standarddraw = MyUserDraw;
```

- Add your own editor via the `Hv_DOUBLECLICK` attribute if the standard editor is not sufficient.
- Manually change the type from the standard type the item was created with to `Hv_USERITEM`.

Here is an example from the ced application, where a "cerenkov" user item piggy-backs on an **`Hv_WORLDPOLYGONITEM`**:

```
  Item = Hv_VaCreateItem(View,
           Hv_TYPE,         Hv_WORLDPOLYGONITEM,
           Hv_TAG,          CERENKOVITEM,
           Hv_NUMPOINTS,    4,
           Hv_DATA,         fpts,
           Hv_NUMROWS,      1,
           Hv_NUMCOLUMNS,   1,
           Hv_DOMAIN,       Hv_INSIDEHOTRECT,
           Hv_DOUBLECLICK,  EditCerenkovItem,
           Hv_DRAWCONTROL,  Hv_ZOOMABLE + Hv_INBACKGROUND,
           Hv_AFTEROFFSET,  Hv_PolygonAfterOffset,
           Hv_FIXREGION,    FixWPolygonBasedRegion,
           Hv_USER1,        ID,
           Hv_USER2,        (short)sect,
           Hv_BALLOON,      (void *)"Cerenkov Counter",
           NULL);

/* final fixes to make it do what I want */

  Item->type = Hv_USERITEM;
  Item->standarddraw = DrawCerenkovItem;
  return  Item;
```

7.8.2 Item Region Maintenance

There is a slight complication for user items that use regions as opposed to just areas. Any number of things such as dragging cause the region to be invalid. But H$^{\underline{V}}$ does not know how to reconstruct regions for arbitrarily complicated user items. Thus, when the region of an item is potentially corrupted, H<u>v</u> will call a user provided routine that should calculate the Item's region given its present location. The routine is passed, as in the example above, using the `Hv_FIXREGION` attribute. The routine should be declared as in:

```
/*---------- FixWPolygonBasedRegion --------*/

void     FixWPolygonBasedRegion(Item)

Hv_Item     Item;

/***************************************
  Should be a good fixregion algorithm for
  any ced world polygon based items.
***************************************/

{
  Hv_View               View = Item->view;
  Hv_WorldPolygonData  *wpoly;
  Hv_FPoint            *fpts;
  int                   i;
  short                 np;

  Hv_DestroyRegion(Item->region);
  Item->region = NULL;

  wpoly = (Hv_WorldPolygonData *)(Item->data);

  np = wpoly->numpnts;
  fpts = (Hv_FPoint *)Hv_Malloc(np*sizeof(Hv_FPoint));

  if (wpoly->poly != NULL)
    Hv_Free(wpoly->poly);

  if (wpoly->fpts != NULL) {

    for (i = 0; i < np; i++) {
      fpts[i].h = wpoly->fpts[i].h;
      fpts[i].v = MidplaneXtoProjectedX(View, wpoly->fpts[i].v);
    }

    wpoly->poly = Hv_NewPoints((int)np);
    Hv_WorldPolygonToLocalPolygon(View, np, wpoly->poly, fpts);
    Item->region = Hv_CreateRegionFromPolygon(wpoly->poly, np);
    Hv_ClipBox(Item->region, Item->area);
    Hv_LocalRectToWorldRect(View, Item->area, Item->worldarea);
  }
}
```

Note!

If you piggy-back on an Item and all you are really doing is enhancing the drawing of the item, then you don't have to worry about region-fixing. H<u>v</u> will do it properly for all of its internal items. In the example above, while the item was based on an `Hv_WORLDPOLYGON` its representation also depended on a user-selected view plane.

7.8.3 Submitting items for inclusion in the $H^{\underline{v}}$ library

If you create an item that may be of use to others, please submit it via email to *heddle@cebaf.gov* for inclusion into the next release of $H^{\underline{v}}$. Your submission should include a description of the item, all relevant source code (drawing, initializing, destroying, etc.) and some sample code wherein the item was used.

Hv
Chapter:

8 Dialog Boxes

Before getting started, we make a preemptive strike: Dialog Boxes (or just "dialogs", for short) are the most difficult upset of programming in Hv. They never look like what we expect the first time they are created, and a fair amount of trial and error is required. Now it may be just a coincidence, but dialog boxes are also the part of Hv that make the most extensive use of *Motif*. In fact, all Hv provides are convenience functions for demystifying the use of *Motif* dialogs.

Unfortunately, we have adopted the habit of calling the widgets that get placed on the dialog boxes "dialog items" adding to the confusion over what is from Hv and what is from *Motif*. Just remember that if it is on a View it is an Hv item; if it is on a dialog it is from *Motif*.

8.1 Standard Dialog Use

Dialogs should be confined to a specific routine that is often a callback for an item (for example, a doubleclick will bring up an item's editor in a dialog.) The canonical form for one of these routines is:

```
/*-------- GenericDoDialog -------*/

static void   GenericDoDialog(Hv_Event hvevent)


{
  Hv_Item                  Item = hvevent->item;
  Hv_View                  View = hvevent->view;
  static Hv_Widget         dialog = NULL;
  Hv_Widget                dummy rowcol;
  int                      answer;

/* has the dialog been created? */

  if (!dialog) {
    Hv_VaCreateDialog(&dialog, Hv_TITLE, " Dialog Title ", NULL);
    rowcol = Hv_DialogColumn(dialog, 6);

/*------- MORE DLOG ITEM CREATION SHOULD GO HERE ---------*/

    dummy = Hv_StandardActionButtons(rowcol,
                                     20,
                                     Hv_OKBUTTON + Hv_CANCELBUTTON + Hv_APPLYBUTTON);
  }

/* the dialog has been created */

/*-------- STUFF/SET DLOG ITEMS WITH INITIAL VALUES -----------*/


/* process the dialog */

  while ((answer = Hv_DoDialog(dialog, NULL)) > 0) {

    if (answer != Hv_APPLY)
      break;
```

```
    }
}
```

The steps are hopefully clear from the code and the following explanation:

- Upon the first entry, the dialog is created. The dialog is saved as a static Hv_Widget.
- An Hv_Widget (dialog item) “rowcol” is created which will be a frame for subsequent dialog items.
- A hole is left were additional dialog items will be created.
- A “close-out” consisting of OK, Cancel, and Apply buttons is added at the end.

If memory is not an issue, use a static variable to create the dialog the first time the routine is called. On subsequent calls it will pop up very quickly. Create the dialog with a call to **`Hv_VaCreateDialog`**.

8.2 H^{V} Dialog Items

Here is the set of dialog items understood by Some are mere synonyms for Motif widgets, some are collections of Motif widgets.

Dialog Item	Comment
Hv_BUTTONDIALOGITEM	Motif pushbutton widget
Hv_COLORSELECTORDIALOGITEM	A Color-selector
Hv_LABELDIALOGITEM	Motif label widget (these are basically inert strings)
Hv_LISTDIALOGITEM	Motif list widget
Hv_RADIODIALOGITEM	Motif “radio” button
Hv_SCALEDIALOGITEM	Motif scale widget
Hv_SCROLLEDTEXTDIALOGITEM	Motif scrolled text widget
Hv_SEPARATORDIALOGITEM	Motif separator widget
Hv_TEXTDIALOGITEM	Motif text widget (these are often editable)
Hv_TOGGLEDIALOGITEM	Motif toggle button

TABLE 10. The set of H^{V} dialog items.

The utility H^{V} provides is not so much creating new widgets, but for (hopefully) making it a little easier to lay them out and use them.Hv_RadioListPtr Hv_GetActiveRadioButton(Hv_RadioHeadPtr radiohead)

8.3 Creating Dialogs

Dialogs are created with the variable length argument list function **`Hv_VaCreateDialog`**. Its only required argument is a pointer to the **`Hv_Widget`** that will hold the dialog. There are presently only two relative attribute value pairs: `Hv_TYPE` which is either `Hv_MODAL` (the default) or `Hv_MODELESS`. A modal dialog is one that has to be "answered" before continuing to interact with the main application. The other recognized attribute is `Hv_TITLE` which is self-explanatory.

Attribute	Type	Comment	Default
Hv_TYPE	short	Either `Hv_MODAL` or `Hv_MODELESS`.	Hv_MODAL
Hv_TITLE	char *	Title of the dialog.	NULL

TABLE 11. Attributes used in creating a dialog.

As an example:

```
Hv_VaCreateDialog(&dialog, Hv_TITLE, " World Rectangle Editor ", NULL);
```

creates a modal dialog (using the variable named "dialog") with the given title.

8.4 Creating Dialog Items

Dialogs items are created with the variable length argument list function **`Hv_VaCreateDialogItem`**. The only required argument is an **`Hv_Widget`** parent which is either the dialog itself or another dialog item. The type of item to be created is governed by the `Hv_TYPE` attribute whose value is one of the types from Table 10. Below we show a table of all attributes used in creating dialogs, including a specification of which items the attribute can be applied to.

Attribute	Type	Comment	Default	Relevant Items
Hv_TYPE	short	Specifies type of the dialog item. See Table 10.	none	all
Hv_LABEL	char *	A string label	NULL	Hv_LABELDIALOGITEM Hv_TOGGLEBUTTONITEM Hv_RADIODIALOGITEM Hv_BUTTONDIALOGITEM
Hv_NUMCOLUMS	short	Number of columns of text to display	0	Hv_TEXTDIALOGITEM Hv_SCROLLEDTEXTDIALOGITEM
Hv_DEFAULTTEXT	char *	Default text in an editable field	NULL	Hv_TEXTDIALOGITEM
Hv_DATA	void *	User supplied data	NULL	Hv_RADIODIALOGITEM
Hv_OPTION	char *	This attribute can appear multiple times and is used to pass an array of labels, For example an array of names of radio buttons.	NULL	Hv_RADIODIALOGITEM
Hv_NOWON	short	Which radio buttonwil be on (starting the count at 1)	1	Hv_RADIODIALOGITEM

TABLE 12. Attributes relevant in creating dialog items.

Attribute	Type	Comment	Default	Relevant Items
Hv_RADIOLIST	*Hv_Radio HeadPtr	When creating radio button, use this to pas the a pointer to the"radiohead" whichwill mange the set..	NULL	Hv_RADIODIALOGITEM
Hv_BACKGROUND	short	Background color	calculated	Hv_LABELDIALOGITEM Hv_SCROLLEDTEXTDIALOGITEM Hv_TEXTDIALOGITEM
Hv_TEXTCOLOR	short	Color of text used by the ite,	Hv_black	Hv_LABELDIALOGITEM Hv_SCROLLEDTEXTDIALOGITEM Hv_TEXTDIALOGITEM
Hv_BORDERWIDTH	short	Pixel width of border	0	Hv_LABELDIALOGITEM Hv_TEXTDIALOGITEM
Hv_BORDERCOLOR	short	Color of border	Hv_black	Hv_LABELDIALOGITEM Hv_TEXTDIALOGITEM
Hv_FONT	short	Font used in item's text.	Hv_fixed2	all
Hv_NUMITEMS	short	Number of items in a list	0	Hv_LISTDIALOGITEM
Hv_NUMVISIBLEITEMS	short	Number of items to accomodate without scrolling.	8	Hv_LISTDIALOGITEM
Hv_LISTSELECTION	int	Either Hv_SINGLELISTSELECT (choose one only) or Hv_MULTIPLELISTSELECT (choose many)	Hv_SINGLELISTSE-LECT	Hv_LISTDIALOGITEM
Hv_LISTRESIZEPOLICY	int	One of Hv_VARIABLE, Hv_CONSTANT, or Hv_RESIZE_IF_POSSIBL	Hv_VARIABLE	Hv_LISTDIALOGITEM
Hv_LIST	char**	Array of strings to be added to a list	NULL	Hv_LISTDIALOGITEM
Hv_CALLBACK	void *	Callback function (should be declared with an Hv_Widget argument.)	NULL	Hv_LISTDIALOGITEM Hv_SCALEDIALOGITEM Hv_RADIODIALOGITEM Hv_TOGGLEDIALOGITEM Hv_BUTTONDIALOGITEM
Hv_VALUECHANGED	void *	Called as text is being modified (should be declared with an Hv_Widget argument.)	NULL	Hv_TEXTDIALOGITEM
Hv_ORIENTATION	short	Hv_HORIZONTAL or Hv_VERTICAL	Hv_HORIZONTAL	Hv_SCALEDIALOGITEM
Hv_MINVALUE	int	Minimum value of scale	0	Hv_SCALEDIALOGITEM
Hv_MAXVALUE	int	Maximum value of scale	100	Hv_SCALEDIALOGITEM
Hv_CURRENTVALUE	int	Current value of scale	50	Hv_SCALEDIALOGITEM
Hv_WIDTH	short	Width in pixels (usually not needed unless trying to "force" it).	calculated	all
Hv_HEIGHT	short	Height in pixels (usually not needed unless trying to "force" it).	calculated	all
Hv_LEFT	short	Left in pixels (usually not needed unless trying to "force" it).	calculated	all
Hv_TOP	short	Top in pixels (usually not needed unless trying to "force" it).	calculated	all

TABLE 12. Attributes relevant in creating dialog items.

8.5 Dialog Geometry

Dialog items can always be positioned using the `Hv_LEFT`, and `Hv_TOP` attributes. However, this is not good practice because whenever you want to extend the dialog (which happens often) you'll find yourself manually rearranging items. It is better to use the **`Hv_DialogRow`**, **`Hv_DialogColumn`**, and **`Hv_DialogTable`** routines to create *Motif* row-column widgets to serve as the parents of our items. These will manage the geometry for us. However, this takes some practice. We start by describing the differences of these three functions:

- **`Hv_DialogColumn`** returns a "vertical" row-column widget; its children will appear arranged in a nice column.
- **`Hv_DialogRow`** returns a "horizontal" row-column widget; its children will appear arranged in a nice row.
- **`Hv_DialogTable`** returns a "table" row-column widget; its children will appear arranged in a table filled row-wise.

The first dialog item created after the dialog is almost always a vertical row-column, i.e. through the use of **`Hv_DialogColumn`**. We refer to this item as the "main" rowcol.

The dialog should be designed on paper first. For example, consider the design of the editor for rectangles created from the $H^{\underline{V}}$ drawing tool:

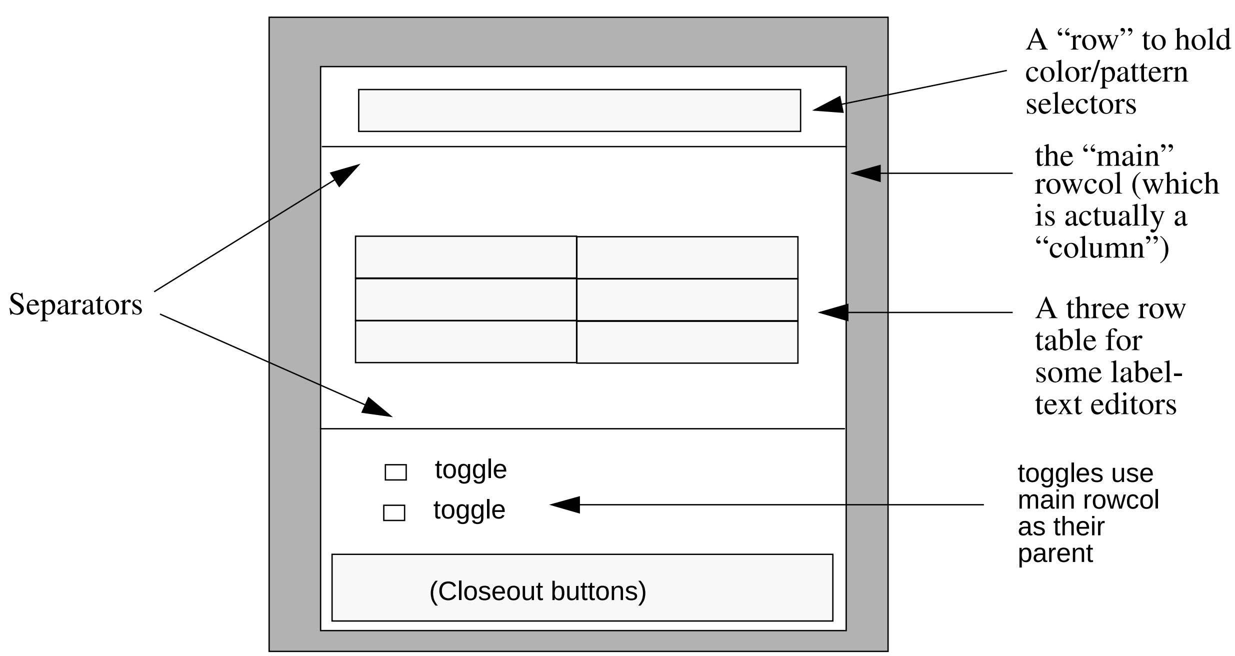


FIGURE 22. Paper design of a rectangle editor.

We examine the code for constructing this dialog in the next section.

8.6 An Example Of Dialog Creation

Putting this all together we create the dialog we designed in the previous section. Here we concentrate only on creating the dialog, although the "processing of the dialog" code is also provided so that the example is complete.

```
/*-------- Hv_EditWorldRectItem -------*/

void   Hv_EditWorldRectItem(Hv_Event hvevent)

/* item edit for Hv_WORLDRECTITEMs  */

{
  Hv_Item              Item = hvevent->item;
  static Widget        dialog = NULL;
  static Widget        thickness;
  static Widget        frame, grid, rows, cols;
  Widget               dummy, rowcol, rc;
  Hv_WorldRectData     *wrect;
  int                  answer;

  if (Item == NULL)
    return;

  wrect = (Hv_WorldRectData *)(Item->data);

  if (!dialog) {
    Hv_VaCreateDialog(&dialog, Hv_TITLE, " World Rectangle Editor ", NULL);
    rowcol = Hv_DialogColumn(dialog, 8);/* the main rowcol*/
    rc = Hv_DialogRow(rowcol, 5);      /* create a row for color/pattern edit*/

    linecolorlabel = Hv_SimpleColorLabel(rc, " line ", Hv_EditWorldRectColor);
    rectcolorlabel = Hv_SimpleColorLabel(rc, " fill ", Hv_EditWorldRectColor);
    patternlabel   = Hv_SimplePatternLabel(rc, "pattern",
                                   Hv_EditWorldRectPattern);

    dummy = Hv_SimpleDialogSeparator(rowcol);

    rc = Hv_DialogTable(rowcol, 3, 8);

    thickness = Hv_SimpleTextEdit(rc, "     Line Width ", NULL, 4);
    rows      = Hv_SimpleTextEdit(rc, " Number of Rows ", NULL, 4);
    cols      = Hv_SimpleTextEdit(rc, " Number of Cols ", NULL, 4);
    dummy = Hv_SimpleDialogSeparator(rowcol);


/* 3D frame? */

    frame = Hv_SimpleToggleButton(rowcol, (char *)"3D frame");

    grid = Hv_SimpleToggleButton(rowcol,  (char *)"Grid lines");
    dummy = Hv_StandardActionButtons(rowcol, 26, Hv_ALLACTIONS);
  }

/* the dialog has been created */

  Hv_SetDeleteItem(dialog, Item);

  newpattern = Item->pattern;
  newlinecolor = Item->color;
  newrectcolor = wrect->fillcolor;

  Hv_ChangeLabelPattern(patternlabel, Item->pattern);
  Hv_ChangeLabelColors(linecolorlabel, -1, Item->color);
  Hv_ChangeLabelColors(rectcolorlabel, -1, wrect->fillcolor);


  Hv_SetToggleButton(frame, wrect->frame);
  Hv_SetToggleButton(grid,  wrect->grid);
```

```
  Hv_SetIntText(thickness, wrect->thickness);
  Hv_SetIntText(rows, wrect->nrows);
  Hv_SetIntText(cols, wrect->ncols);

  while ((answer = Hv_DoDialog(dialog, NULL)) > 0) {
    Item->pattern = newpattern;
    Item->color  = newlinecolor;
    wrect->fillcolor = newrectcolor;
    wrect->frame = Hv_GetToggleButton(frame);
    wrect->grid  = Hv_GetToggleButton(grid);
    wrect->thickness = Hv_sMax(1, Hv_sMin(6, (short)Hv_GetIntText(thickness)));

    wrect->nrows = Hv_sMax(1, Hv_sMin(100, (short)Hv_GetIntText(rows)));
    wrect->ncols = Hv_sMax(1, Hv_sMin(100, (short)Hv_GetIntText(cols)));

    Hv_StandardRedoItem(Item);

    if (answer != Hv_APPLY)
      break;
  }
}
```

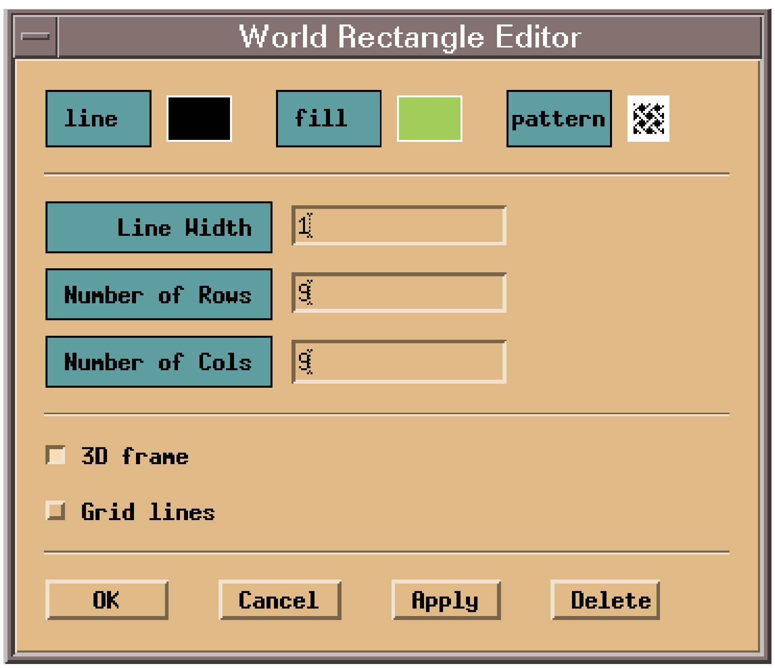


FIGURE 23. Sample dialog box (it is the actual editor resulting from a double click on a rectangle created from the drawing tool.)

8.7 Dialog Closeout

Every dialog must have a closeout, which is a method of terminating the dialog. Standard closeouts include combinations of an OK, CANCEL, and APPLY button. These are usually the last items added in an HV dialog, through the poorly named **`Hv_StandardActionButtons`** function:

```
Hv_Widget Hv_StandardActionButtons(Hv_Widget parent,
                                   short spacing
                                   int whatbuttons);
```

The parent is usually the main “rowcol” widget. The pixel spacing depends on the width of the dialog and usually requires a few iterations. The argument whatbuttons should be combinations of the following bits:

```
Hv_OKBUTTON
Hv_CANCELBUTTON
Hv_APPLYBUTTON
Hv_DELETEBUTTON
Hv_DONEBUTTON
```

Note!

A separator is automatically added above the close-out buttons.

8.8 Stuffing And Unstuffing Dialogs

These refer to the process of taking the local variables that you want to permit the user to edit (or at least seeing it.) The routines used to stuff dialogs are

Name	Comment
Hv_ChangeLabel(Hv_Widget w, char *label, short font)	Changes the string and font for a label item. Note: will NOT work for text items.
Hv_ChangeLabelColors(Hv_Widget w, short fg, short bg)	Changes the foreground and background colors of labels only.
Hv_ChangeLabelPattern(Hv_Widget, w, short pattern)	Changes the label background pattern.
void Hv_NewPosSelectListItem(Hv_Widget w, int pos, Boolean CB)	For lists: w is the list widget, pos is the position you want selected (starting at 1), CB cotrols whether or not you want the callback to be called.
void Hv_PosReplaceListItem(Hv_Widget w, int pos, char *str)	For the given list it replaces the text a the designated position (starting at 1) with the new string.
void Hv_PosSelectListItem(Hv_Widget w, int pos)	Like Hv_NewPosSelectListItem, except the callback is automatically called.
void Hv_PosSetActiveRadioButton(short pos, Hv_RadioHeadPtr radiohead)	For a given set of radio buttons specified by “radiohead” this sets the one specified by “pos” (starting at 1) to be on.
void Hv_SetActiveRadioButton(Hv_Widget nowon, Hv_RadioHeadPtr radiohead)	For a given set of radio buttons specified by “radiohead” this sets the button “nowon” to be on.
void Hv_SetFloatText(Hv_Widget w, float val, int numdec)	Displays the value in the widget w with “numdec” numbers to the right of the decimal point.
void Hv_SetIntText(Hv_Widget w, int val)	Displays the value in the widget w.
void Hv_SetScaleMinMax(Hv_Widget w, int min, int max)	Sets the limits on a sliding scale.
void Hv_SetScaleValue(Hv_Widget w, int val)	Sets the current value of a scale.
void Hv_SetString(Hv_Widget w, char *str)	Sets the text displayed in a text dialog item.
void Hv_SetToggleButton(Hv_Widget w, Boolean state)	Sets the state of a toggle button.

TABLE 13. Routines used to “stuff” dialogs with their default values and/or settings.

Unstuffing dialogs is just the opposite. That’s when we pull numvers out of text items, check which toggle buttons are set etc. and set our program variables accordingly.

Name	Comment
Hv_RadioListPtr Hv_GetActiveRadioButton(Hv_RadioHeadPtr radiohead)	Returns the list node for the active radio button in the set controlled by "radiohead". Not used as often as Hv_PosGetActiveRadioButton.
void *Hv_GetActiveRadioButtonData(Hv_RadioHeadPtr radiohead)	Little used routine that reutrns user data optionally stored with the set of radio buttons controlled by "radiohead".
int *Hv_GetAllListSelectedPos(Hv_Widget list)	Reurns array of positions (starting at 1) of all selected items in a list. The terminatineg entry in the array is -1. The caller should free the array with Hv_Free.
int Hv_GetListSelectedPos(Hv_Widget list)	For a select-one-only list, returns the position (starting at 1) of the selected item.
char *Hv_GetString(Hv_Widget w)	Obtains the text from a text dialog item.
float Hv_GetFloatText(Hv_Widget w)	Obtains a floating point value from the text dialog item.
int Hv_GetIntText(Hv_Widget w)	Obtains an integer value from the text dialog item.
int Hv_GetScaleValue(Hv_Widget w)	Obtains the current value of a scale dialog item.
Boolean Hv_GetToggleButton(Hv_Widget w)	Obtains the current state os a toggle button dialog item.
short Hv_PosGetActiveRadioButton(Hv_RadioHeadPtr radiohead)	Returns the position (starting at 1) for the currently active radio button in the set controlled by "radiohead".

TABLE 14. Routines to "Unstuff" dialogs

8.9 Processing Dialogs

Typical (i.e., modal dialogs, that require an answer before the let you go back to the main program) are processed with the **`Hv_DoDialog`** function. For modeless dialogs, which can coexist in parallel with the main program, one has to use the **`Hv_OpenDialog`** and **`Hv_CloseDialog`** functions explicitly.

```
int  Hv_DoDialog(Hv_Widget dialog,
        Hv_Widget def_btn)
```

The first argument is the dialog that is to be processed. The second (which can be **`NULL`**) is the "default" button. It is supposed to be the button that is "preselected" in the sense that if you hit return it's as if you clicked the button. However, try as we might we cannot issue the correct *Motif* incantation to make this work reliably. Maybe someone can look at the code in `Hv_dlogs.c` and tell us what is wrong.

`Hv_DoDialog` returns one of the following values:

```
Hv_DELETE      -1
Hv_CANCEL       0
Hv_OK           1
Hv_DONE         2
Hv_APPLY        3
```

Note that `Hv_CANCEL` is zero, a fact that we sometimes use explicitly even though we shouldn't.

How the dialog is processed depends somewhat on what method of closeout is used. The three most common situations that occur in modal dialogs are

- There are OK and CANCEL buttons provided
- There are OK, CANCEL, and APPLY buttons provided

- There is only an OK or some other type of "DONE" button

We describe these in turn, with the assumptions that the following procedures have

- **`BuildDialog`** is a procedure that creates the dialog
- **`StuffDialog`** stuffs the dialog based on currents values of the variables being displayed
- **`UnstuffDialog`** resets the variables based on state of the dialog

For other possibilities, such as `Hv_DELETE`, we refer you to the $H^{\underline{V}}$ source code. Any of the files that support one of the drawing tools, such as Hv_worldrect.c, will provide a good example of both a dialog (the respective editor) and the use of `Hv_DELETE`.

8.9.1 OK and CANCEL

This is the most common: you want to provide the user with a way to change some values but also to change his mind:

The canonical form is:

```
void MyDoDIalog(Hv_Event hvevent)

{
       int answer;
       static Hv_Widget dialog = NULL:

       if (dialog == NULL)
             dialog = BuildDialog();

/* dialog has been created */

       StuffDialog(dialog);

       answer = Hv_DoDialog(dialog, NULL);

       if (answer == Hv_OK)
             UnstuffDialog(dialog);

}
```

8.9.2 OK, CANCEL, and APPLY

In this case, the user is also offered and APPLY button which we take to mean: The dialog is unstuffed just as if OK was selected but, unlike the OK case, it is not closed and editing can continue.

```
void MyDoDIalog(Hv_Event hvevent)

{
       int answer;
       static Hv_Widget dialog = NULL:

       if (dialog == NULL)
             dialog = BuildDialog();

/* dialog has been created */

       StuffDialog(dialog);

       while ((answer = Hv_DoDialog(dialog, NULL)) > 0) {

             UnstuffDialog(dialog);
```

```
                if (answer != Hv_APPLT)
                        break;
        }

        if (answer == Hv_CANCEL) {
                /* restore vals */
        }

}
```

Note!

The “CANCEL” option is much less trivial when APPLY is an option, since pressing APPLY in the above code will cause the dialog to be unstuffed. A copy of the original values of all variables must be kept in case the user changes values, selects “APPLY” followed by “CANCEL”,

8.9.3 Only OK or some other “DONE” button

This is the easiest. There is only one button that causes the dialog to closeout:

```
void MyDoDIalog(Hv_Event hvevent)

{
        static Hv_Widget dialog = NULL:

        if (dialog == NULL)
                dialog = BuildDialog();

/* dialog has been created */

        StuffDialog(dialog);

        Hv_DoDialog(dialog, NULL);
        UnstuffDialog(dialog)

}
```

8.10 Final Word on Dialogs

We have tried to make dialogs easier to use than they are in standard *Motif*. Whether or not we succeeded, well, you’ll have to let us know. Finally, we would be remiss if we didn’t point out that the mother of all dialogs as far as this release is concerned is the plot editor found in Hv_xyplotedit.c. It uses most of the dialog items available.

Hv
Chapter:

9 Hv Miscellany

9.1 Tiling the background window

Tiling the main window refers usually to filling it with some sort of propaganda, such as a logo. It is accomplished through the following steps (usually performed during your applications private initialization):

- Set the global (and badly named) variables **`Hv_tileX`** and **`Hv_tileY`** to thw width and height of the tiling pattern.
- Set the global function pointer **`Hv_tileDraw`** to your function for drawing the tile.
- Write your tile-drawing routine, declaring it as a function with a void argument list.

For example, the demo *hvmap* application has as its private initialization (see init.c):

```
/*------ Init() ----*/

void Init()

{

/* set the background color */

  Hv_canvasColor = Hv_lightSeaGreen;

  WindowTile();  /* establishes tile drawing */
  MakeMenus();
  InitControls();
  AddLogo();
  InitQuickZoom();
  FinalInit();
}
```

The **`WindowTile`** function is (in *hvmap*):

```
/*-------- WindowTile -----------*/

static void WindowTile()

/* this sets the size of the rectangle that is tiled
   on the background */

{
  Hv_tileX = 200;
  Hv_tileY = 80;
  Hv_TileDraw = DrawTile;
}
```

And the DrawTile function is

```
/*------- DrawTile --------------*/

static void DrawTile()

{
  Hv_Rect    rect;

  if ((Hv_mainWindow == 0) || (Hv_display == NULL))
    return;

  Hv_SetRect(&rect, 0, 0, (short)Hv_tileX, (short)Hv_tileY);
  Hv_FillRect(&rect, Hv_canvasColor);

  DrawLogo(Hv_tileX, Hv_tileY);
}
```

The **DrawLogo** function can be found in logo.c in the hvmap directory. You should run that demo to see the effect of window tiling.

9.2 Making a Tool-like Item

Suppose that we want to add a tool that plops something down onto the HotRect, much like the drawing tools. The typical process is:

- Select the button corresponding to the tool
- Interact with the HotRect either by clicking or rubberbanding as appropriate

In doing this, $H^{\underline{V}}$ has to be told, in effect, that the tool has temporarily taken over the HotRect, so that mouse events are used to create whatever the tool is creating rather than being interpreted as routine actions. This is best demonstrated using the code from one of the drawing tools (the rectangle tool):

```
/* ------- Hv_WorldRectButtonCallback -------*/

void Hv_WorldRectButtonCallback(Hv_Event hvevent)

/* call back for when the user selects the rectangle drawing tool */

{
  Hv_View                View = hvevent->view;
  static Hv_View         activeView = NULL;
  static Hv_Item         activeItem = NULL;
  Hv_Item                newItem;
  static Cursor          tempcursor;
  Hv_Point               pp;
  Hv_Rect                therect, limitrect;
  float                  xmin, xmax, ymin, ymax;
  Hv_Item                parent = NULL;

/* activeView holds the view this button is on. We allow only one active
   tool/view at a time. If it is NULL, we have just clicked the
   button and not started drawing the rect. The Hv_StandardFirstEntry will:

       1) set the static variables
       2) redirect the next mous click BACK TO THIS ROUTINE
       3) change the cursor

*/

  if (activeView == NULL) {
    Hv_StandardFirstEntry(hvevent,
            Hv_WorldRectButtonCallback,
```

```
            &activeView,
            &activeItem,
            &tempcursor);
    return;
  }

/* are we on the same view that the tool was selected from? */

  if (View != activeView) {
    Hv_Warning("World Rect selector active on another view!");
    return;
  }

/* Ready to go. Reset static vars, reset mouse clicks to regular handler */

  Hv_ResetActiveView(hvevent, &activeView, &activeItem, tempcursor);

/* see if click occured inside hotrect */

  Hv_SetPoint(&pp, hvevent->where.x, hvevent->where.y);
  if (Hv_PointInRect(pp, View->hotrect)) {

/* for certain items, we assume the desire is to contain the
   new item */

    parent = Hv_PointInPlotItem(View, pp);

    if (parent == NULL)
      Hv_CopyRect(&limitrect, View->hotrect);
    else
      Hv_CopyRect(&limitrect, parent->area);

    Hv_SelectRect(&therect, limitrect, pp, 1, Hv_NOSELECTPOLICY, 0.0);
    if ((therect.width > 1) && (therect.height > 1)) {
      Hv_LocalToWorld(View, &xmin, &ymax, therect.left,  therect.top);
      Hv_LocalToWorld(View, &xmax, &ymin, therect.right, therect.bottom);
      newItem = Hv_VaCreateItem(View,
             Hv_TYPE,          Hv_WORLDRECTITEM,
             Hv_XMIN,          xmin,
             Hv_XMAX,          xmax,
             Hv_PARENT,        parent,
             Hv_YMIN,          ymin,
             Hv_YMAX,          ymax,
             Hv_NUMROWS,       9,
             Hv_NUMCOLUMNS,    9,
             Hv_DOMAIN,        Hv_INSIDEHOTRECT,
             Hv_DRAWCONTROL,   Hv_ENLARGEABLE+ Hv_ZOOMABLE + Hv_FANCYDRAG,
             Hv_CHECKENLARGE, Hv_CheckWorldRectEnlarge,
             NULL);

      hvevent->item = newItem;
      Hv_DrawItem(newItem, NULL);
    }
  }

}
```

9.3 Using the Function Keys

The function keys F1 through F10 can be registered by a call to **`Hv_AddFunctionKeyCallback`**. For example, to register the F6 key to call the function BosGeometryDump:

```
Hv_AddFunctionKeyCallback(6, BosGeometryDump);
```

The callback function should be declared with a void argument list:

```
 void BosGeometryDump(void)
{
```

```
}
```

9.4 Animating Items

Items can be animated in three ways:

- The item can be given a "redraw" time. Its drawing routine will be called at intervals of this time.This works nicely for control items. For example, the standard text drawing tool uses this method to scroll its label through the alphabet.
- The application can call **`Hv_UpdateItemOffScreen`**, which draws the item off screen and then copies it on screen. This works well for items than change appearance but don't move or grow, like a plot being updated.
- If the item is in a view that saves its background, and it is a HotRect item with its `Hv_INBACKGROUND` bit set, then it can 1) update (but not redraw) itself and 2) set the view to be dirty using **`Hv_SetViewDirty`**.

We provide only an example of the first method, from the $H^{\underline{v}}$ code that animates the text tool button:

```
/*------- Hv_DrawTextButton -------*/

void Hv_DrawTextButton(Hv_Item    Item,
                       Hv_Region   region)

/* draws the icon for text annotation buttons */

{
  Hv_Rect           area;
  short             xmw, xmh;        /* String width  in pixels */
  short             xc, yoff;
  Hv_String         *str;

  Hv_ButtonDrawingArea(Item, &area);
  Hv_FillRect(&area, Item->scheme->fillcolor);

  str = Hv_CreateString("A");
  str->font = Hv_timesBold20;

/* see if this is actually a timed redraw */

  if (Item->drawinterval != NULL)
    str->text[0] = 'A' + (Item->drawinterval->timedredrawcount % 26);
  else
    Hv_SetItemTimedRedraw(Item, 60000);

/* set up the motif string and get some characteristics */

  Hv_CompoundStringDimension(str, &xmw, &xmh);

  xc = (area.left + area.right - (short)xmw)/2 - 1;
  yoff = 1 + ((area.height - (short)xmh)/2);

  Hv_DrawCompoundString(xc + 2, area.top + yoff, str);
  Hv_DestroyString(str);
}
```

9.5 Item Connections

We are developing ways of “connecting” Hv items. This would include convenient ways to connect items with a connection tool (for example to create process flow.) The connection package is not ready yet, but will be available in a near-term future release.

9.6 Simulations

Hv allows you to register simulations; specifically it permits you to register one per view. This is done at view creation time through the `Hv_SIMPROC` and `Hv_SIMINTERVAL` attributes. The former indicates what process Hv will call at regular intervals (in milliseconds) specified by the latter.

As an example, consider the *hvmap* demo included with the Hv distribution. The view is created via:

```
Hv_VaCreateView(&View,
                .
                .
                .
                Hv_SIMPROC,         (void *)SimulationCB,
                Hv_SIMINTERVAL,     1000,
                .
                .
                .
                NULL);
```

Where we have omitted mention of those attributes that are not important. The user provided function **`SimulationCB`** will be called once per second (1000 ms) starting as soon as the view is created. Its prototype is:

```
void SimulationCB(Hv_View View);
```

There are also three routines available for manipulating simulations:

```
void           Hv_StopSimulation(Hv_View  View);

void           Hv_ChangeSimulationInterval(Hv_View View,
                                    unsigned long interval);

void           Hv_StartSimulation(Hv_View     View,
                                  Hv_FunctionPtr  simproc,
                                  unsigned long  interval);
```

`Hv_StopSimulation` stops any simulation registered to the given View.
`Hv_ChangeSimulationInterval` modifies the frequency with which a simulation is called, and **`Hv_StartSimulation`** can restart a simulation after it was stopped or start a new simulation (it is an alternative to using the at-creation attributes.)

In principle, this feature allows the developer to create a simulation that doesn’t interfere with the GUI. It is effectively (but not actually) the case that as the simulation runs, the GUI can still be used. Since we are not actually “threading” the simulation as a different process (maybe in a future java release?) you have to take some precautions that your simulation will not bog down the user interface. In this model, the simulation is being called synchronously at regular intervals. You need to have your simulation return quickly from the call to the SimProc since the interface will be frozen during that time. The *hvmap* demo results in a slightly sluggish GUI on some slower platforms, indicating that a rotation of all the map points is a little too much for a single step. The developer will have to optimize the trade-off between the interval between calls and the amount of processing performed on each call.

9.7 The Hv ASCII IO Package

Many of the Hv applications we have written require reading configuration and other ASCII files. We have developed a primitive collection of database type tools for doing this. This is not an integral part of Hv. You are completely free to ignore this feature. We include it here for completeness and on the chance that it may be useful in a limited number of cases.

The features of the Hv ASCII IO package are:

- It expects files to be in the form of records delimited by a header and a trailer. These are usually names in square brackets such as “[MYRECNAME]” and “[ENDREC]”
- Searches for records by name are case insensitive
- Blank lines or lines starting with a ‘!’ are comments

Thus an Hv database file might look like:

```
!This is an Hv database

[RECNAME 1]
23 45
hey man

!comment

1.23
[ENDREC]

[RECNAME 2]
.
.
.
[ENDREC]

!another record of the first variety

[RECNAME 1]
211 456
hey dude
4.56
[ENDREC]
```

etc. There can be any number of records of any given type. Order is irrelevant. Hv provides a number of tools for accessing this records. In a nutshell you find the record based on its name in the header and then ask Hv to give you non-comment lines one at-a-time from within the record. You then parse the lines as you see fit. For “tables” you can ask Hv how many non-comment lines are in the record before stepping through it.

Here is a typical example (from the *caps* application):

```
/*-------- ReadTargets ------*/
```

```
static void ReadTargets(View, fp)

Hv_View   View;
FILE      *fp;

{
  char              *line;
  int               numlines; /*number of lines in the record*/
  int               numnclines; /*number of NON-COMMENT lines*/
  char             *targetrecname = "[TARGETREC]";
  char             *endrecname = "[ENDREC]";


  rewind(fp);
  line = (char *)Hv_Malloc(256);

/* there may be mutilple target records */

  while (Hv_FindRecord(fp, targetrecname, False, line)) {
    Hv_RecordLineCount(fp, targetrecname, endrecname, &numlines, &numnclines);
    if (numnclines > 0) {
      while(Hv_ReadNextLine(fp, line, targetrecname, endrecname) == 0) {
        if(NotAComment(line))
          ParseLine(line);
    }
  }

  rewind(fp);
  Hv_Free(line);
}
```

Here is a summary of IO related routines:

Function	**Comment**
Boolean Hv_IsAComment(char *line);	True if the line is interpreted as a comment.
Boolean Hv_NotAComment(char *line);	True if the line is interpreted as not being a comment.
Boolean Hv_ReportDatabaseSearch(FILE *fp, char *rec-name, Boolean rewind);	True if the specified record is found, leaving the file pointer positioned for reading at the start of the record. The "rewind" variable controls whether the file is rewound before searching.
char *Hv_FindRecord(FILE *fp, char *recname, Boolean rewind, char *line);	Returns the line containing the recname, and a copy of it in the "line" variable, if the specified record is found, leaving the file pointer positioned for reading at the start of the record. The "rewind" variable controls whether the file is rewound before searching. Returns NULL on failure.
char *Hv_GetNextNonComment(FILE *fp, char *line);	Returns the next line that is NOT a comment line starting at the present file position.
int Hv_CountSegments(char *line);	Counts the number of segments in a line, where a segment is characters surrounding by whitespace. For example, '23.4 hello gimme a break 22' contains six segments. If the global variable Hv_quoteDelimitsSegment is set to True, then the line '23.4 hello "gimme a break" 22' has four segments.

TABLE 15. Routines in H^{V}'s primative ASCII database package.

Function	Comment
char *Hv_NextTextSegment(char **line);	Returns the next segment in a line and moves the line "past" the returned segment. The segment should be freed by the caller. If line is "abc def ghi" then this returns "abc" and afterward line is "def ghi".
int Hv_ReadNextLine(FILE *fp, char *line, char *startrec, char *endrec);	Reads the next line, NOT excluding comments, placing a copy into the "line" variable. Returns zero on sucess.Failure codes are `Hv_FOUNDEOF` if and end-of-file is encountered, `Hv_FOUNDSTARTREC` if we unexpectedly come upon a new record, and `Hv_FOUNDENDREC` if we hit the end.
void Hv_NextRecordSize(FILE *fp, char *startrec, char *endrec, int *numlines, int *numchars);	If it finds a record delimited by startrec and endrec, it returns the number of lines (comments included) and the number of characters.
void Hv_RecordLineCount(FILE *fp, char *startrec, char *endrec, int *numlines, int *numnclines);	If it finds a record delimited by startrec and endrec, it returns the total number of lines and the number of von-comment lines.

TABLE 15. Routines in Hv's primative ASCII database package.

9.8 Requests from Hv to the application

Hv has only two requests it makes to your application, which you can choose to ignore. One has to do with the "feedback when dragged" feature, the other has to do "fancy dragging" where the item is drawn repeatedly (in XOR mode) as it is being dragged. This is in contrast with an ordinary drag, where only the item's area is dragged.

Normally when an item is dragged, the automatic updating of the feedback area is suspended. Feedback *during* a drag can be requested by setting the `Hv_FEEDBACKWHENDRAGGED` drawcontrol bit for the item in question. This might be useful, for example, if you want to drag an item to a particular world coordinate.

Because feedback can be CPU intensive if many complicated items are present, when an item is dragged with its `Hv_FEEDBACKWHENDRAGGED` bit set, Hv sets the global variable **`Hv_completeFeedback`** to False (it is normally True). Thus, if you choose to be a good citizen you could have your feedback routine check this variable and skip any really difficult or irrelevant feedback when it is set. If not, the dragging might be "jerky" as it competes for cycles with the feedback. If your feedback is simple, this will be unnecessary.

Similarly, when an item with its `Hv_FANCYDRAG` bit set is dragged, Hv will set the global variable **`Hv_extraFastPlease`** to True (it is normally False). Your drawing routine, which at that time is competing with the drag, could check for this variable and, when it is True, could skip any CPU intensive drawing.

9.9 Redirection using the Hv Global Function Pointers

You may need to do some housekeeping on your items following various operations or to replace what the normal Hv would do. The more complicated the application, the more likely that this will occur. This is done through the following global function pointers, *which all have NULL as their default value*.

• **Hv_UserZoom**, if not NULL, will be called after $H^{\underline{v}}$ redefines the world system due to a zoom, but before it redraws the HotRect. If needed, it should point to a function declared as

```
void myAfterZoom(Hv_View view)
{
}
```

• **Hv_UserDefaultWorld,** if not NULL, will be used to “restore the default world” of a view. This is needed only in the rare case where the View has a non-rectangular world system, for example if it is displaying an orthographic map projection. It should be declared as

```
void myDefaultWorld(Hv_View view)
{
}
```

• **Hv_UserMove**, if not NULL, will be called after $H^{\underline{v}}$ redefines the local system due to dragging a view, but before it redraws itself. If needed, it should point to a function declared as

```
void myAfterMove(Hv_View view, short dh, short dv)
{
}
```

• **Hv_UserResize**, if not NULL, will be called after $H^{\underline{v}}$ resizes a view, but before it redraws itself. If needed, it should point to a function declared as

```
void myAfterResize(Hv_View view, short dh, short dv)
{
}
```

• **Hv_AfterItemRotate**, if not NULL, will be called after $H^{\underline{v}}$ rotates an item but before it does any redrawing. If needed, it should point to a function declared as

```
void myAfterItemRotate(Hv_Item item, float ang)

/* ang is the angle of rotation in radians */

{
}
```

SECTION 3 Reference

Hv
Chapter:

10 Library of Standard Hv Items

Below we tabulate the presently recognized Hv items, including attributes specific to that item. The attributes listed are in addition to the common set of attributes described earlier(See “Common Item Attributes” on page 58.) The “HotRect or Control” column refers to the default domain (**`Hv_INSIDEHOTRECT`** or **`Hv_OUTSIDEHOTRECT`**) given to the item, The default domain can be overwritten (although why you would want to is not clear) via the **`Hv_DOMAIN`** attribute. For example, in the simple demo we place a button in the HotRect, purely for amusement purpose.

i

Item Name	HotRect or Control?	Comment	(Attribute, type, default, comment)
Hv_BOXITEM	Control	A 3D box that is for holding other control items, such as option buttons. The items it holds are usually created as “children” of the box.	(Hv_FILLCOLOR, short, Hv_gray12, the background color of the box)
Hv_BUTTONITEM	Control	A rectangular or rounded button that is used as a control item.	(Hv_RADIUS, short, -1, radius in pixels for round buttons) (Hv_FRAMEWIDTH, short, 2, size in pixels of frame giving 3D appearance (Hv_FILLCOLOR, short, Hv_gray12, the normal background color of the button) (Hv_HOTCOLOR, short, Hv_gray10, the background color of the button when pressed) (Hv_LABEL, char *, NULL, the button’s label) (Hv_TEXTCOLOR, short, Hv_gray12, the color of the button’s label)
Hv_CHOICEITEM	Control	A button similar to the Motif radio button but in this case an Hv item.	(Hv_STATE, short, False, sets the initial state of the button)

TABLE 16. Presently recognized Hv Items. Note the attributes listed here are in addition to the set of common attributes (see Table 5 on page 59) The “world” drawing items are not included as they are created from the drawing tools. You cans examine the relevant source code for the attributes they use.

Item Name	HotRect or Control ?	Comment	(Attribute, type, default, comment)
Hv_CHOICESETITEM	Control	A kind of single-purpose box-like item that holds Hv_CHOICEITEMs.	(Hv_FONT, short, Hv_fixed2, font used to display choices) (Hv_NOWON, short, 1, position (starting at 1) of the button that is on by default) (Hv_FILLCOLOR, short, Hv_gray12, background fill) (Hv_ARMCOLOR, short Hv_red, color of “on” button) (Hv_TITLE, char *, NULL, optional title)
Hv_CLOCKITEM	Control	Displays a clock (which will actually work!). It appears on the “Welcome” view. Probably applications will not use this item.	none
Hv_COLORBUTTONITEM	Control	A small button whose color is significant and is usually attached to some internal variable. The color can be changed by clicking the button.	(Hv_FILLCOLOR, short, Hv_gray12, the color of the button) (Hv_ALLOWNOCOLOR, short, False, if true “No Color” will be an allowed choice from editor)
Hv_CONNECTIONITEM	HotRect	EXPERIMENTAL connector between (among) hotrect items.	n/a
Hv_FEEDBACKITEM	Control	The item displaying the feedback text.	(Hv_FEEDBACKDATACOLS, short, 16, number of columns to size for in data text) (Hv_FILLCOLOR, short, Hv_gray12, background fill) (the entries, as described on page 47
Hv_FRACTIONDONEITEM	Control	An item that shows a rectangle getting filled as some loop is being executed.	none
Hv_LEDBARITEM	Control	A row or column od LED’s that can be an effective indicator of the state of a View.	(Hv_ORIENTATION, short, Hv_HORIZONTAL, only other possibility is Hv_VERTICAL) (Hv_ONCOLOR, short, Hv_red, color of “on” LED) (Hv_OFFCOLOR, short, Hv_green, color of “off” LED) (This item is a little irregular. See the example in the *simple* demo)
Hv_OPTIONBUTTONITEM	Control	A Usual toggle type button.	(Hv_ARMCOLOR, short, no color) (Hv_CONTROLVAT, void *, NULL, should be a pointer to a boolean, any time the option button is hit the variables value will be changed) (Hv_STATE, shoort, Hv_OFF, other possibility is Hv_ON)

TABLE 16. Presently recognized $H^{\underline{V}}$ Items. Note the attributes listed here are in addition to the set of common attributes (see Table 5 on page 59) The “world” drawing items are not included as they are created from the drawing tools. You cans examine the relevant source code for the attributes they use.

Item Name	HotRect or Control?	Comment	(Attribute, type, default, comment)
Hv_RAINBOWITEM	Control	Used to create rainbow scales.	(Hv_ORIENTATION, short, Hv_HORIZONTAL, only other possibility is Hv_VERTICAL) (Hv_MINVALUE, int, 0, minimum value of scale) (Hv_MAXVALUE, int, 100, maximum value of scale) (Hv_MINCOLOR, short, Hv_red, minimum color on scale) (Hv_MAXCOLOR, short, Hv_white, maximum color on scale) (Hv_NUMLABELS, short, 2, number of numeric labelse) (Hv_PIXELDX, short, 1, pixel spacing) (Hv_PIXELDY, short, 1, pixel spacing) (Hv_PRECISION, short, 0, precision of numerical labels (#digits right of decimal) (Hv_SCALEFACTOR, float, 1.0, overall scale factor) (Hv_EDITABLE, short, False, can rainbow be edited?)
Hv_SEPARATORITEM	Control	Usually the child of an Hv_BOXITEM, it is used to separate (usually the box's) children.	(Hv_ORIENTATION, short, Hv_HORIZONTAL, only other possibility is Hv_VERTICAL)
Hv_SLIDERITEM	Control	A standard slider control.	(Hv_ORIENTATION, short, Hv_HORIZONTAL, only other possibility is Hv_VERTICAL) (Hv_MINVALUE, int, 0, minimum value of scale) (Hv_MAXVALUE, int, 100, maximum value of scale) (Hv_CURRENTVALUE, int, 50, current value of scale) (Hv_TITLE, char *, NULL, optional title)
Hv_TEXTENTRYITEM	Control	An in-place text entry item	no additional attributes
Hv_TEXTITEM	HotRect	A compound text item.	no additional attribute
Hv_USERITEM	HotRect	User defined items (See 7.8, page 68)	No predefined attributes, but you can use any of the specialized attributes from this table.
Hv_WHEELITEM	Control	A Thumb wheel.	(Hv_ORIENTATION, short, Hv_HORIZONTAL, only other possibility is Hv_VERTICAL) (Hv_MINVALUE, int, 0, minimum value of scale) (Hv_MAXVALUE, int, 100, maximum value of scale) (Hv_CURRENTVALUE, int, 50, current value of scale) (Hv_TITLE, char *, NULL, optional title)

TABLE 16. Presently recognized H^{V} Items. Note the attributes listed here are in addition to the set of common attributes (see Table 5 on page 59) The "world" drawing items are not included as they are created from the drawing tools. You cans examine the relevant source code for the attributes they use.

In addition, there are some other items (Hv_CONTAINERITEM, Hv_MAPITEM, Hv_GLOBEITEM, Hv_HOTRECTBORDERITEM, and Hv_SLIDERTHUMBITEM) that are used internally by H^{V} and should never be created by an application.

Note!

The demo application is a showcase for most of these items. We strongly suggest you build it and refer to it as you learn about these items. Also, almost all of the items have built in editors -- try double-clicking them.

Hv
Chapter:

11 Function Reference

In this chapter, we present tables of the Hv functions. The first set of simple tables lists functions that are useful for specific items or tasks, such as sliders. The final table (Table 17 on page 117) is a comprehensive, alphabetical listing of all Hv public functions.

Functions dealing with Hv_BOXITEMs

void Hv_DrawBoxItem(Hv_Item, Hv_Region)

void Hv_UpdateBoxItem(Hv_Item, char *)

Functions dealing with Hv_BUTTONs

void Hv_DrawButtonItem(Hv_Item, Hv_Region)

void Hv_ButtonDrawingArea(Hv_Item, Hv_Rect *)

void Hv_ToggleButton(Hv_Item)

void Hv_ButtonSingleClick(Hv_Event)

short Hv_GetButtonState(Hv_Item)

void Hv_SetButtonState(Hv_Item, short)

Functions related to CALLBACKs

void Hv_ResetActiveView(Hv_Event, Hv_View *, Hv_Item *, Hv_Cursor)

void Hv_StandardFirstEntry(Hv_Event, Hv_FunctionPtr, Hv_View *, Hv_Item *, Hv_Cursor *)

Functions dealing with Hv_CHOICEBUTTONs

short Hv_HandleChoice(Hv_Item)

void Hv_PosSetChoice(short pos, Hv_Item)

Functions dealing with COLOR

void Hv_DarkenColor(short, unsigned short) void Hv_BrightenColor(short, unsigned short)

void Hv_AdjustColor(short, short, short, short)

void Hv_ChangeColor(short, unsigned short, unsigned short, unsigned short)

Functions dealing with Hv_COLORBUTTONs

void Hv_DrawColorButtonItem(Hv_Item, Hv_Region)

void Hv_ChangeColorButtonText(Hv_Item, char *, Boolean)

short Hv_GetColorButtonColor(Hv_Item)

void Hv_EditColorButtonItem(Hv_Event)

void Hv_ToggleColorButtonItem(Hv_Event)

Functions dealing with CONNECTIONs

void Hv_ToggleConnection(Hv_Item)

void Hv_DisableConnection(Hv_Item)

void Hv_EnableConnection(Hv_Item)

Hv_Item Hv_AddConnection(Hv_Item, Hv_Item, short, short, short, short, short, short, Boolean, Hv_CheckProc, Boolean)

Hv_Item Hv_MultiplexConnection(Hv_Item, Hv_Item, short, short, short, short, short, short, short, Boolean, Hv_CheckProc)

void Hv_UpdateConnections(Hv_View, Hv_Item, Boolean)

void Hv_RemoveConnections(Hv_Item, short, Boolean, Boolean, Boolean)

void Hv_RemoveConnection(Hv_Item, Hv_Item, short, Boolean)

void Hv_RemoveViewConnections(Hv_View, Boolean, short)

Hv_Region Hv_ConnectionRegion(Hv_Item)

Hv_Item Hv_DuplicateConnection(Hv_Item, Hv_Item, short)

void Hv_ChangeConnectionStyles(Hv_View, Hv_Item, short, short, short, short, short, short)

Functions dealing with DIALOGS

void Hv_CloseOutCallback(Hv_Widget, int)

void Hv_VaCreateDialog(Hv_Widget *, ...)

Hv_Widget Hv_VaCreateDialogItem(Hv_Widget, ...)

int Hv_Question(char *)

void Hv_Warning(char *)

void Hv_Information(char *)

void Hv_TimedInformation(char *, int duration)

int Hv_ModifyColor(Hv_Widget, short *, char *, Boolean)

int Hv_ModifyPattern(Hv_Widget, short *, char *)

int Hv_DoDialog(Hv_Widget, Hv_Widget)

void Hv_OpenDialog(Hv_Widget, Hv_Widget)

void Hv_CloseDialog(Hv_Widget)

int Hv_Wait(int)

void Hv_VariableZoom(Hv_View)

Hv_Widget Hv_GetDialogActionButton(Hv_Widget, int)

int Hv_GetDialogAnswer(Hv_Widget)

void Hv_NotYet(void)

void Hv_SetDeleteItem(Hv_Widget, Hv_Item)

Hv_Widget Hv_SimpleTextEdit(Hv_Widget, char *, char *, short)

Hv_Widget Hv_SimpleDialogSeparator(Hv_Widget)

Hv_Widget Hv_SimpleColorLabel(Hv_Widget, char *, Hv_FunctionPtr)

Hv_Widget Hv_SimplePatternLabel(Hv_Widget, char *, Hv_FunctionPtr)

Hv_Widget Hv_SimpleToggleButton(Hv_Widget, char *)

void Hv_ChangeLabelColors(Hv_Widget, short, short)

void Hv_ChangeLabelPattern(Hv_Widget, short)

void Hv_ChangeLabel(Hv_Widget, char *, short)

void Hv_SetToggleButton(Hv_Widget, Boolean)

void Hv_SetFloatText(Hv_Widget, float, int)

void Hv_SetIntText(Hv_Widget, int)

void Hv_SetString(Hv_Widget, char *)

Boolean Hv_GetToggleButton(Hv_Widget)

float Hv_GetFloatText(Hv_Widget)

int Hv_GetIntText(Hv_Widget)

char *Hv_GetString(Hv_Widget)

Functions dealing with DIALOG LISTS

Boolean Hv_CheckListItem(Hv_Widget, char *, int **, int *)

void Hv_PosReplaceListItem(Hv_Widget, int, char *)

void Hv_DeleteAllListItems(Hv_Widget)

void Hv_DeselectAllListItems(Hv_Widget)

void Hv_AddListItem(Hv_Widget, char *, Boolean)

void Hv_PosSelectListItem(Hv_Widget, int)

void Hv_NewPosSelectListItem(Hv_Widget, int, Boolean)

void Hv_SelectListItem(Hv_Widget, char *)

int Hv_GetListSelectedPos(Hv_Widget)

int *Hv_GetAllListSelectedPos(Hv_Widget)

Functions dealing with DIALOG SCALES

int Hv_GetScaleValue(Hv_Widget)

void Hv_SetScaleValue(Hv_Widget, int val)

void Hv_SetScaleMinMax(Hv_Widget, int, int)

Functions dealing with DIALOG RADIO BUTTONS

void Hv_RadioListSensitivity(Hv_RadioHeadPtr, short, Boolean)

void Hv_SetActiveRadioButton(Hv_Widget nowon, Hv_RadioHeadPtr)

void Hv_PosSetActiveRadioButton(short pos, Hv_RadioHeadPtr)

Hv_RadioListPtr Hv_GetActiveRadioButton(Hv_RadioHeadPtr)

void *Hv_GetActiveRadioButtonData(Hv_RadioHeadPtr)

short Hv_PosGetActiveRadioButton(Hv_RadioHeadPtr)

Functions dealing with DIALOG TEXT

void Hv_AppendLineToScrolledText(Hv_Widget, char *)

void Hv_DeleteLinesFromTop(Hv_Widget, int)

void Hv_JumpToBottomOfScrolledText(Hv_Widget, Hv_TextPosition *)

void Hv_ClearScrolledText(Hv_Widget, Hv_TextPosition *)

void Hv_AddLineToScrolledText(Hv_Widget, Hv_TextPosition *, char *)

void Hv_DeleteScrolledTextLines(Hv_Widget, int, Hv_TextPosition *)

Functions dealing with DRAGGING/RUBBERBANDING

void Hv_DragRect(Hv_Rect *, Hv_Rect *, Hv_Point, Hv_Point *, short, Boolean, Boolean)

void Hv_ConnectLine(Hv_Rect, Hv_Point, Hv_Point *)

void Hv_CreateLine(Hv_Rect, Hv_Point, Hv_Point *StopPP)

void Hv_EnlargeLine(Hv_Rect, Hv_Point, Hv_Point *, short, Hv_Point *)

void Hv_EnlargeRect(Hv_Rect *, Hv_Rect, Hv_Point, Boolean)

void Hv_EnlargeOval(Hv_Rect *, Hv_Rect, Hv_Point, Boolean)

void Hv_SelectPolygon()

void Hv_SelectOval(Hv_Rect *, Hv_Rect, Hv_Point, short)

void Hv_SelectRect(Hv_Rect *, Hv_Rect, Hv_Point, short, short, float)

Functions dealing with DRAWING

void Hv_StringDraw(short x, short y, Hv_String *, Hv_Region)

void Hv_DrawPoints(Hv_Point *, int, short)

void Hv_DrawSymbolsOnPoints(Hv_Point *, int, short, short, short)

void Hv_DrawLines(Hv_Point *, short, short)

void Hv_DrawSegments(Hv_Segment *, short, short)

void Hv_DrawWorldCircle(Hv_View, float, float, float, Boolean, short, short)

void Hv_FrameRect(Hv_Rect *, short)

void Hv_FillRect(Hv_Rect *, short)

void Hv_FillPatternRect(Hv_Rect *, short, short)

void Hv_Simple3DRect(Hv_Rect *, Boolean, short)

void Hv_Simple3DDiamond(short, short, short, short, Boolean)

void Hv_BlockISymbol(short, short, short, short, short)

void Hv_BlockHSymbol(short, short, short, short, short)

void Hv_FillCircle(short, short, short, Boolean, short, short)

void Hv_FrameCircle(short, short, short, short)

void Hv_FrameWedge(short, short, short, int, int, short)

void Hv_FillPolygon(Hv_Point *, short, Boolean, short, short)

void Hv_FramePolygon(Hv_Point *, short, short)

void Hv_EraseRect(Hv_Rect *)

void Hv_FrameArc(short, short, short, short, int, int, short) void Hv_Draw3DRect(Hv_Rect *, Hv_ColorScheme *)

void Hv_Draw3DCircle(short, short, int, Hv_ColorScheme *)

void Hv_DrawLine(short, short, short, short, short)

void Hv_DrawPoint(short, short, short)

void Hv_DrawItem(Hv_Item, Hv_Region)

void Hv_SetPattern(int)

void Hv_SetLineStyle(int, int)

void Hv_SetBackground(unsigned long)

void Hv_SetForeground(unsigned long)

void Hv_WorldPointToLocalPoint(Hv_View, Hv_FPoint *, Hv_Point *)

void Hv_WorldPoint2ToLocalRect(Hv_View, Hv_FPoint *, Hv_FPoint *, Hv_Rect *, short)

void Hv_TileMainWindow(Hv_Pixmap)

void Hv_ReTileMainWindow(void)

void Hv_ClipBox(Hv_Region, Hv_Rect *)

void Hv_ClipRect(Hv_Rect *)

void Hv_SetFillStyle(int)

void Hv_SetClippingRegion(Hv_Region)

void Hv_RestoreClipRegion(Hv_Region)

void Hv_FullClip(void)

void Hv_SetSegment(Hv_Segment *, short, short, short, short)

void Hv_RotateFPoints(float, Hv_FPoint *, short)

void Hv_RotateFPoint(float, Hv_FPoint *)

Boolean Hv_PolygonIntersectsPolygon(Hv_Point *, Hv_Point *, short, short)

Boolean Hv_PolygonIntersectsRect(Hv_Point *, short, Hv_Rect *)

Boolean Hv_RectIntersectsRect(Hv_Rect *, Hv_Rect *)

void Hv_PolygonEnclosingRect(Hv_Rect *, Hv_Point *, int)

void Hv_OffsetPolygon(Hv_Point *, int, short, short)

void Hv_CreatePolygonFromRect(Hv_Point *, Hv_Rect *)

Boolean Hv_PointInPolygon(Hv_Point, Hv_Point *, int)

Boolean Hv_PointInRegion(Hv_Point, Hv_Region)

Boolean Hv_RectInRegion(Hv_Rect *, Hv_Region)

void Hv_SetPoint(Hv_Point *, short, short)

void Hv_InvertColorScheme(Hv_ColorScheme *) void Hv_ClipItem(Hv_Item, short, short)

Hv_Region Hv_ItemDrawingRegion(Hv_Item, short, short)

void Hv_SetLineWidth(int)

Functions dealing with FEEDBACK

void Hv_UpdateFeedbackText(Hv_Item)

void Hv_ChangeFeedbackLabel(Hv_View, short, char *)

void Hv_RefreshFeedbackRegion(Hv_Item, Hv_Rect *)

void Hv_UpdateFeedbackArea(Hv_Item)

void Hv_DeleteFeedbackEntry(Hv_Item, short)

void Hv_DestroyFeedbackItem(Hv_Item)

void Hv_ChangeFeedback(Hv_View, short, char *)

void Hv_AddFeedbackEntry(Hv_Item, short, char *, short, short)

void Hv_AddFeedbackSeparator(Hv_Item)

Functions dealing with FILE SELECTION

char *Hv_FileSelect(char *, char *, char *)

char *Hv_DirectorySelect(char *, char *)

Functions dealing with Hv_FRACTIONDONEITEMs

void Hv_FractionDoneUpdate(Hv_Item, float)

Functions dealing with INITIALIZATION

void Hv_VaInitialize(unsigned int, char **, ...)

void Hv_Go(void)

void Hv_SetFKeysActive(Boolean)

void Hv_AddFunctionKeyCallback(int, Hv_FunctionPtr)

Functions dealing with I/O

char *Hv_NextTextSegment(char **)

int Hv_CountSegments(char *)

Boolean Hv_GetNextFile(DIR *, char *, char *, FILE **, char *)

int Hv_AlphaSortDirectory(char *, char *, char ***, Boolean, Boolean)

DIR *Hv_OpenDirectory(char *)

void Hv_AddPath(char *)

void Hv_NextRecordSize(FILE *, char *, char *, int *, int *)

char *Hv_PrependHostToFilename(char *)

FILE *Hv_LookForNamedFile(char *, char **, char *)

Boolean Hv_NotAComment(char *)

Boolean Hv_IsAComment(char *)

char *Hv_GetNextNonComment(FILE *, char *)

char *Hv_FindRecord(FILE *, char *, Boolean, char *)

int Hv_ReadNextLine(FILE *, char *, char *, char *)

Boolean Hv_ReportDatabaseSearch(FILE *, char *, Boolean)

void Hv_RecordLineCount(FILE *, char *, char *, int *, int *)

Functions dealing with GENERIC ITEMS

Hv_Item Hv_VaCreateItem(Hv_View,...)

void Hv_SetItemToDraw(Hv_Item)

void Hv_SetItemNotToDraw(Hv_Item)

Boolean Hv_CheckItemToDraw(Hv_Item)

void Hv_SetItemState(Hv_Item, short)

short Hv_GetItemState(Hv_Item)

void Hv_SetItemDrawControlBit(Hv_Item, bit)

Boolean Hv_CheckItemDrawControlBit(Hv_Item, bit)

void Hv_ClearItemDrawControlBit(Hv_Item, bit)

void Hv_HotRectItemsBoundary(Hv_ItemList, Hv_Rect *)

Hv_Item Hv_ItemMatch(Hv_ItemList, short, short, short)

Hv_Item Hv_PointInItemMatch(Hv_ItemList, short, short, short, Hv_Point)

Boolean Hv_PointInItem(Hv_Point, Hv_Item)

void Hv_RedrawItem(Hv_Item)

char *Hv_GetItemText(Hv_Item)

void Hv_SetItemText(Hv_Item, char *)

void Hv_SetItemTimedRedraw(Hv_Item, unsigned long)

void Hv_RemoveItemTimedRedraw(Hv_Item)

Hv_Region Hv_GetItemDrawingRegion(Hv_Item, short, short)

Hv_Region Hv_GetMinimumItemRegion(Hv_Item)

void Hv_SetItemArea(Hv_Item, short, short, short, short)

Hv_Rect *Hv_GetItemArea(Hv_Item)

void Hv_GetTotalItemArea(Hv_Item, Hv_Rect *)

void Hv_SetItemRegion(Hv_Item, Hv_Region)

Hv_Region Hv_GetItemRegion(Hv_Item)

void Hv_SetItemString(Hv_Item, Hv_String *)

Hv_String *Hv_GetItemString(Hv_Item)

void Hv_SetItemData(Hv_Item, void *)

void *Hv_GetItemData(Hv_Item)

void Hv_SetItemColor(Hv_Item, short)

short Hv_GetItemColor(Hv_Item)

void Hv_SetItemTag(Hv_Item, short)

short Hv_GetItemTag(Hv_Item)

Hv_View Hv_GetItemView(Hv_Item)

void Hv_MoveItemBehindItem(Hv_Item, Hv_Item)

void Hv_SendItemToBack(Hv_Item)

void Hv_SendItemToFront(Hv_Item)

void Hv_DumpItemList(Hv_ItemList, FILE, int nspace)

void Hv_DeleteItem(Hv_Item)

void Hv_DestroyItem(Hv_Item)

void Hv_EnableItem(Hv_Item)

void Hv_DisableItem(Hv_Item)

void Hv_EnableAllItems(Hv_View)

void Hv_DisableAllItems(Hv_View)

void Hv_SetItemSensitivity(Hv_Item, Boolean, Boolean)

Boolean Hv_ItemIsEnabled(Hv_Item)

void Hv_KillItemList(Hv_ItemList)

void Hv_OffsetItem(Hv_Item, short, short, Boolean)

void Hv_OffsetItemList(Hv_ItemList, short, short) void Hv_OffsetHotRectItems(Hv_ItemList, short, short)

void Hv_ResizeItem (Hv_Item, short, short, Boolean)

void Hv_ResizeItemList(Hv_ItemList, short, short)

void Hv_GetItemCenter(Hv_Item, short *, short *)

void Hv_DrawItemList(Hv_ItemList, Hv_Region)

void Hv_UpdateItem(Hv_Item)

Hv_Item Hv_GetItemFromTag(Hv_ItemList, short)

Hv_Region Hv_CopyItemRegion(Hv_Item, Boolean)

void Hv_AddItemToList(Hv_ItemList, Hv_Item)

void Hv_MaybeDrawItem(Hv_Item, Hv_Region)

void Hv_FixItemWorldArea(Hv_Item, Boolean)

Functions dealing with Hv_LEDITEMs

void Hv_UpdateLED(Hv_Item, int, int)

Functions dealing with MAPS

void Hv_GetGoodMapLimits(Hv_View, float, float, float, float, float *, float *, float *, float *, Boolean)

void Hv_LatLongToLocal(Hv_View, short *, short *, float, float)

void Hv_LocalToLatLong(Hv_View, short, short, float *, float *)

void Hv_XYToLatLong(Hv_View, float, float, float *, float *)

void Hv_LatLongToXY(Hv_View, float *, float *, float, float)

void Hv_ChangeMapCenter(Hv_View, float, float)

void Hv_FixMapParameters(Hv_View)

float Hv_MercatorYToLat(float)

float Hv_MercatorLatToY(float)

float Hv_MercatorXToLong(float, float)

float Hv_MercatorLongToX(float, float)

float Hv_GCDistance(float, float, float, float)

Hv_ViewMapDataHv_GetViewMapData(Hv_View)

void Hv_MallocViewMapData(Hv_View)

void Hv_DestroyViewMapData(Hv_View)

Boolean Hv_PointIsOnLand(Hv_View, Hv_Point)

Hv_Region Hv_LandMapRegion(Hv_View)

void Hv_DrawMapItem(Hv_Item, Hv_Region)

void Hv_DrawOneSpecialMap(Hv_View, Hv_Map, Hv_Region)

void Hv_InitializeMap(Hv_Item, Hv_AttributeArray)

void Hv_ChangeMapProjection(Hv_View, int)

void Hv_DestroyLocalMaps(Hv_View)

Hv_Point *Hv_MapPointsToPoly(Hv_View, Hv_Map, int *, Boolean)

void Hv_DrawSpecialMapsOnly(Hv_View, Hv_Region)

void Hv_DrawMapBordersOnly(Hv_View, Hv_Region, Boolean)

void Hv_DrawOverlayRect(Hv_View, float, float, float, float, short)

void Hv_DrawMapGrid(Hv_View, Hv_Region)

Hv_SpecialMapData Hv_GetSpecialMapData(Hv_Map)

Hv_SpecialMapData Hv_MallocSpecialMapData(void)

Hv_Map Hv_MallocMap(unsigned int)

void Hv_InitMaps(char *f)

void Hv_DestroyMap(Hv_Map *)

Functions dealing with MATH

short Hv_InRange(float)

void Hv_WorldToLocal(Hv_View, float, float, short *, short *)

void Hv_LocalToWorld(Hv_View, float *, float *, short, short)

int Hv_RandomInt(int *, int)

void Hv_IndexSort(int, float *, int *)

Functions dealing with MENUS

void Hv_VaCreateMenu(Hv_Widget *, ...)

Hv_Widget Hv_VaCreateMenuItem(Hv_Widget, ...)

void Hv_SetMenuItemString(Hv_Widget, char *, short)

void Hv_SetQuickZoomLabel(short, char *)

Hv_Widget Hv_CreateFontPopup(Hv_Widget, short)

Hv_Widget Hv_CreateSymbolPopup(Hv_Widget, short) void Hv_ViewPopup(Hv_View, XEvent *)

Functions dealing with OFFSCREEN DRAWING

Hv_Pixmap Hv_DrawItemOnPixmap(Hv_Item, Hv_Region)

void Hv_OffScreenViewUpdate(Hv_View, Hv_Region)

void Hv_UpdateItemOffScreen(Hv_Item, Hv_Region)

void Hv_SaveViewBackground(Hv_View, Hv_Region)

void Hv_RestoreViewBackground(Hv_View)

Boolean Hv_ViewSavesBackground(Hv_View)

Boolean Hv_IsViewDirty(Hv_View)

void Hv_SetViewDirty(Hv_View)

void Hv_FreePixmap(Hv_Pixmap)

Hv_Pixmap Hv_CreatePixmap(unsigned int, unsigned int)

Functions dealing with Hv_OPTIONBUTTONs

Hv_Item Hv_StandardOptionButton(Hv_View, Hv_Item, Hv_Item, Boolean *, char *, int)

void Hv_OptionButtonCallback(Hv_Event)

Boolean Hv_GetOptionButton(Hv_Item)

void Hv_SetOptionButton(Hv_Item, Boolean)

Functions dealing with PLOTS

void Hv_ReasonableBarPlotLimits(Hv_PlotData)

void Hv_DrawBarPlotTicks(Hv_Item, Boolean, Boolean)

void Hv_DrawBarCurves(Hv_Item)

void Hv_PlotItemInitialize(Hv_Item, Hv_AttributeArray)

Hv_View Hv_NewXYPlotView(char *, Boolean)

Hv_View Hv_BaseXYPlotView(char *, Hv_Item *, Hv_Item *, short)

void Hv_UpdatePlotView(Hv_View)

void Hv_UpdateXYPlot(Hv_Item)

void Hv_AddDataSetToXYPlot(Hv_Item)

Hv_Item Hv_MallocPlotItem(Hv_View)

Hv_Item Hv_PointInPlotItem(Hv_View, Hv_Point) void Hv_ReasonablePlotLimits(Hv_PlotData)

void Hv_ReasonableYPlotLimits(Hv_PlotData)

void Hv_ZoomPlot(Hv_View, Hv_Point, XEvent *)

void Hv_TieXAxis(Hv_Item splot)

void Hv_ZoomPlotToRect(Hv_Item, Hv_Rect *)

void Hv_DestroyPlotItem(Hv_Item)

Hv_PlotData Hv_GetPlotData(Hv_Item)

Hv_PlotViewData Hv_GetPlotViewData(Hv_View)

void *Hv_GetPlotViewUserData(Hv_View)

void Hv_SetPlotViewUserData(Hv_View, void *)

void Hv_InitPlotViewData(Hv_View)

Hv_Item Hv_StandardPlotLabel(Hv_Item, short, short, char *, short)

void Hv_LocalToPlot(Hv_Item, short, short, float *, float *)

void Hv_PlotToLocal(Hv_Item, short *, short *, float, float)

void Hv_PlotDataMinMax(float *, int, float, float *, float *)

short Hv_FontFromRestrictedFont(int)

void Hv_BigThreeItems(Hv_Item, Hv_Item *, Hv_Item *, Hv_Item *)

void Hv_EditPlot(Hv_Event)

int Hv_GetCurvePos(void)

FILE *Hv_PrepareFile(char *)

void Hv_DemoButtonCallback(Hv_Event)

void Hv_CheckPlotViewSensitivity(Hv_View)

void Hv_WritePlotFile(Hv_View, FILE *)

void Hv_LoadPlotData(Hv_Event)

void Hv_ReadPlotFile(Hv_View, FILE *, char *)

int Hv_CountPlots(Hv_View)

void Hv_DeletePlots(Hv_View)

float Hv_ConvertToANiceValue(float)

int Hv_Decade(float)

int Hv_GetPlotType(Hv_Item)

void Hv_AttachLegend(Hv_Item)

int Hv_SizeLegendItem(Hv_Item)

Hv_Region Hv_TotalPlotRegion(Hv_Item)

Boolean Hv_PlotIsHorizontal(Hv_Item)

void Hv_DrawXTick(Hv_Item, float, short, Hv_Rect *, char *)

void Hv_DrawYTick(Hv_Item, float, short, Hv_Rect *, char *)

Hv_PlotData Hv_MallocPlotData(void)

void Hv_DestroyPlotData(Hv_PlotData)

Hv_CurveData Hv_MallocCurveData(char *)

void Hv_DestroyCurveData(Hv_CurveData)

Functions dealing with RECTANGLES

short Hv_RectRight(Hv_Rect *)

short Hv_RectBottom(Hv_Rect *)

void Hv_GetRectCenter(Hv_Rect *, short *, short *)

void Hv_GetFRectCenter(Hv_FRect *, float *, float *)

void Hv_FixRectWH(Hv_Rect *)

void Hv_FixRectRB(Hv_Rect *)

void Hv_FixFRectWH(Hv_FRect *)

void Hv_InitRect(Hv_Rect *)

void Hv_CopyRect (Hv_Rect *, Hv_Rect *)

void Hv_CopyFRect (Hv_FRect *, Hv_FRect *)

void Hv_GetRectLTRB(Hv_Rect *, short *, short *, short *, short *)

void Hv_SetRect(Hv_Rect *, short, short, short, short)

void Hv_SetFRect(Hv_FRect *, float, float, float, float)

void Hv_ResizeRect (Hv_Rect *, short, short)

void Hv_OffsetRect (Hv_Rect *, short, short)

Boolean Hv_PointInRect(Hv_Point, Hv_Rect *)

Boolean Hv_FPointInFRect(Hv_FPoint, Hv_FRect *)

void Hv_ShrinkRect(Hv_Rect *, short, short)

Functions dealing with REGIONS

void Hv_IntersectRegionWithRegion(Hv_Region *, Hv_Region)

void Hv_EraseRegion (Hv_Region)

void Hv_FillRegion(Hv_Region, short, Hv_Region)

Hv_Region Hv_CreateRegion(void)

Hv_Region Hv_CreateRegionFromPolygon(Hv_Point *, short)

Hv_Region Hv_CreateRegionFromLine(short, short, short, short, short)

Hv_Region Hv_CreateRegionFromLines(Hv_Point *, short, short)

void Hv_DestroyRegion(Hv_Region)

void Hv_KillRegion(Hv_Region *)

Hv_Region Hv_UnionRegion(Hv_Region, Hv_Region)

void Hv_ShrinkRegion(Hv_Region, short, short)

Hv_Region Hv_IntersectRegion(Hv_Region, Hv_Region)

Hv_Region Hv_SubtractRegion(Hv_Region, Hv_Region)

void Hv_RemoveRectFromRegion(Hv_Region *, Hv_Rect *)

Hv_Region Hv_SubtractRectFromRegion(Hv_Region, Hv_Rect *)

void Hv_CopyRegion(Hv_Region *, Hv_Region)

void Hv_AddRegionToRegion(Hv_Region *, Hv_Region)

void Hv_SubtractRegionFromRegion(Hv_Region *, Hv_Region)

void Hv_UnionPolygonWithRegion(Hv_Point *, int, Hv_Region *)

void Hv_UnionRectWithRegion(Hv_Rect *, Hv_Region)

Hv_Region Hv_RectRegion(Hv_Rect *)

Functions dealing with Hv_SLIDERITEMs

void Hv_SetSliderCurrentValue(Hv_Item, short, Boolean)

short Hv_GetSliderCurrentValue(Hv_Item)

Functions dealing with Hv_SEPARATORITEMs

void Hv_CenterSeparator(Hv_Item)

void Hv_SeparatorResizeToParent(Hv_Item)

Convenience functions mostly designated as STANDARD

void Hv_StandardRedoItem(Hv_Item)

Hv_Widget Hv_StandardLabel(Hv_Widget rc, char *, int)

Hv_Widget Hv_StandardClearListButton(Hv_Widget, Hv_Widget, char *, short)

Hv_Widget Hv_StandardDeleteButton(Hv_Widget)

Hv_Widget Hv_StandardButton(Hv_Widget, char *, Hv_FunctionPtr)

Hv_Widget Hv_StandardScale(Hv_Widget, int, int, int, Hv_FunctionPtr)

Hv_Widget Hv_StandardTextField(Hv_Widget)

Hv_Widget Hv_StandardToggleButton(Hv_Widget, char *, Hv_FunctionPtr)

Hv_Widget Hv_StandardDoneButton(Hv_Widget, char *)

Hv_Widget Hv_StandardActionButtons(Hv_Widget, short, int)

Hv_Widget Hv_DialogTable(Hv_Widget, short, short)

Hv_Widget Hv_DialogRow(Hv_Widget, short)

Hv_Widget Hv_DialogColumn(Hv_Widget, short)

void Hv_StandardTools(Hv_View, short, short, short, short, int, Hv_Item *, Hv_Item *)

Functions dealing with Hv_TEXTITEMs

void Hv_SizeTextItem(Hv_Item, short, short)

void Hv_ConcatTextToTextItem(Hv_Item, char *)

void Hv_ChangeTextItemText(Hv_Item, char *, Boolean)

General UTILITY functions

short Hv_ColorFromText(char *)

short Hv_PatternFromText(char *)

short Hv_SymbolFromText(char *)

void Hv_Flush(void)

void Hv_SetCursor(Cursor)

void Hv_DestroyWidget(Hv_Widget)

Hv_Point *Hv_NewPoints(int)

Hv_Rect *Hv_NewRect(void)

Hv_FRect *Hv_NewFRect(void)

void Hv_InitCharStr(char **, char *)

void Hv_DisableWidget(Hv_Widget)

void Hv_EnableWidget(Hv_Widget)

void Hv_SetSensitivity(Hv_Widget, Boolean)

void Hv_SetWidgetBackgroundColor(Hv_Widget, short)

void Hv_SetWidgetForegroundColor(Hv_Widget, short)

char *Hv_GetHvTime(void)

void Hv_Pause(int, Hv_Region)

void *Hv_Malloc(size_t)

void Hv_Free(void *)

void Hv_CopyString(Hv_String **, Hv_String *)

Hv_String *Hv_CreateString(char *)

void Hv_DestroyString(Hv_String *)

Boolean Hv_Contained(char *, char *)

void Hv_SysBeep(void)

void Hv_CompoundStringDimension(Hv_String *, short *, short *)

void Hv_StringDimension(Hv_String *, short *, short *)

void Hv_CharDimension(char *, short, short *, short *)

void Hv_NumberToString(char *, float, short, float)

Functions dealing with VIEWS

void Hv_DrawHotRectItems(Hv_View, Hv_Region)

void *Hv_GetViewData(Hv_View)

Hv_Rect *Hv_GetViewHotRect(Hv_View)

void Hv_VaCreateView(Hv_View *View, ...)

void Hv_SetViewPopupControl(Hv_View, int)

void Hv_SetViewMinimumSize(Hv_View, short, short)

void Hv_StopSimulation(Hv_View)

void Hv_ChangeSimulationInterval(Hv_View, unsigned long)

void Hv_StartSimulation(Hv_View, Hv_FunctionPtr, unsigned long)

short Hv_GetViewTag(Hv_View)

Hv_ItemList Hv_GetViewItemList(Hv_View)

Boolean Hv_ViewIsActive(Hv_View)

Boolean Hv_ViewIsVisible(Hv_View)

void Hv_SetViewBrickWallItem(Hv_Item)

Hv_Region Hv_GetViewRegion(Hv_View)

void Hv_DrawView(Hv_View, Hv_Region)

void Hv_DrawControlItems(Hv_View, Hv_Region)

void Hv_DrawViewHotRect(Hv_View)

void Hv_RedrawViewHotRect(Hv_View, short, Hv_Region)

void Hv_DrawNonBackgroundItems(Hv_View, Hv_Region)

Hv_Region Hv_HotRectRegion(Hv_View)

Hv_Region Hv_ClipHotRect(Hv_View, short)

Hv_Region Hv_ClipView(Hv_View)

void Hv_FreeView(Hv_View)

void Hv_EnableView(Hv_View)

void Hv_DisableView(Hv_View)

char *Hv_GetViewTitle(Hv_View)

void Hv_SetViewTitle(Hv_View, char *)

void Hv_FitFeedback(Hv_View, short)

void Hv_ChangeViewLocal(Hv_View, Hv_Rect *)

Functions dealing with Hv_WHEELITEMs

void Hv_GetWheelValues(Hv_Item, short *, short *, short *)

short Hv_GetWheelCurrentValue(Hv_Item)

void Hv_SetWheelValues(Hv_Item, short, short, short, Boolean)

void Hv_SetWheelCurrentValue(Hv_Item, short, Boolean)

Functions dealing with Hv_WORLDLINEs

void Hv_EditWorldLineItem(Hv_Event)

void Hv_DrawWorldLineItem(Hv_Item, Hv_Region)

void Hv_FixLineRegion(Hv_Item)

Functions dealing with Hv_WORLDOVALs

void Hv_EditWorldOvalItem(Hv_Event)

void Hv_OvalAfterDrag(Hv_Item, short, short)

void Hv_OvalAfterOffset(Hv_Item, short, short)

void Hv_DrawWorldOvalItem(Hv_Item, Hv_Region)

void Hv_FixOvalRegion(Hv_Item)

Functions dealing with Hv_WORLDPOLYGONs

void Hv_EditWorldPolygonItem(Hv_Event)

void Hv_PolygonAfterDrag(Hv_Item, short, short)

void Hv_PolygonAfterOffset(Hv_Item, short, short)

void Hv_OffsetWorldPolygon(Hv_Item, float, float)

void Hv_WorldPolygonToLocalPolygon(Hv_View, short, Hv_Point *, Hv_FPoint *)

void Hv_LocalPolygonToWorldPolygon(Hv_View, short, Hv_Point *, Hv_FPoint *)

void Hv_DrawWorldPolygonItem(Hv_Item, Hv_Region)

void Hv_FixPolygonRegion(Hv_Item)

void Hv_CheckWorldPolygonEnlarge(Hv_Item, Hv_Point, Boolean, Boolean *)

Functions dealing with Hv_WORLDRECTs

void Hv_EditWorldRectItem(Hv_Event)

void Hv_DrawWorldRectItem(Hv_Item, Hv_Region)

void Hv_WorldRectToLocalRect(Hv_View, Hv_Rect *, Hv_FRect *)

void Hv_LocalRectToWorldRect(Hv_View, Hv_Rect *, Hv_FRect *)

void Hv_CheckWorldRectEnlarge(Hv_Item, Hv_Point, Boolean, Boolean *)

Finally, the comprehensive table:

Function	**Comment**
void Hv_AddFeedbackEntry(Hv_Item fbitem, short tag, char *prompt, short font, short color);	Adds a feedback entry to the feedbackitem with the specified tag and prompt and displays the entry in the specified font and color.
void Hv_AddFeedbackSeparator(Hv_Item);	Adds a horizontal separator accross the feedback item.
void Hv_AddFunctionKeyCallback(int whichkey, Hv_FunctionPtr fkeyfunc);	Registers a a function to be called anytime the user hits the “F” key specified by “whickkey”. The function shoid be declared with a void argument list.
void Hv_AddItemToList(Hv_ItemList items,Hv_Item Item);	Adds the item to the end of the list of items in “items”.
void Hv_AddLineToScrolledText(Hv_Widget w, Hv_TextPosition *pos, char *line);	Adds one line to a scrolled text widget in a dialog box. The cursor position, passed in pos, will be updated.
void Hv_AddListItem(Hv_Widget w, char *text, Boolean selected);	Adds a new entry specified by “text” to a list item in a dialog box.
void Hv_AddPath(char *path);	Adds “path” to the search path used by the appication when looking for a file.
void Hv_AddRegionToRegion(Hv_Region *mainreg, Hv_Region addreg);	Adds the region “addreg” to “mainreg”.
void Hv_CenterSeparator(Hv_Item Item);	Center a separator relative to its parent.

TABLE 17. Alphabetical listing of public functions available in H^{V}.

Function	Comment
void Hv_ChangeColorButtonText(Hv_Item,char *,Boolean);	Change a color button’s text and optionally redraw.
void Hv_ChangeFeedback(Hv_View View, short tag, char *label);	Change the data portion of a feedback entry
void Hv_ChangeFeedbackLabel(Hv_View View, short tag, char *label);	Change the label portion of a feedback entry.
void Hv_ChangeLabel(Hv_Widget w, char *label, short font);	Changes the string and font for a label dialog item. Note: will NOT work for text items
void Hv_ChangeLabelColors(Hv_Widget w, short fg, short bg);	Changes the foreground and background colors of label dialog items only
void Hv_ChangeLabelPattern(Hv_Widget w, short pattern);	Changes the label (dialog item) background pattern
void Hv_ChangeSimulationInterval(Hv_View View, unsigned long interval);	Change how often the simulation is called.
void Hv_ChangeTextItemText(Hv_Item Item, char *str, Boolean redraw);	Change the text in a text item and optionally redraw it.
void Hv_ChangeViewLocal(Hv_View View, Hv_Rect *newlocal);	Change a view’s local coordinate system.
Boolean Hv_CheckItemDrawControlBit(Hv_Item Item, int bit);	Returns True if the bit is set.
void Hv_ClearItemDrawControlBit(Hv_Item Item, int bit);	Clear the specified bit.
Boolean Hv_CheckListItem(Hv_Widget list, char *liststr, int **poslist, int *num)	Retruns True if the string “liststr” is one of the entries in the “lisT” dialog item. “num” is the number of times the entry was found. “poslist” is an array (malloced internally but freed by the caller) contianining the positions (starting at 1) of all the entries found.
void Hv_ClearScrolledText(Hv_Widget w, Hv_TextPosition *pos);	Clear text from the specified dialog item.
void Hv_ClipBox(Hv_Region region, Hv_Rect *rect);	Returns the smalles rect that fits around the given region.
Hv_Region Hv_ClipHotRect(Hv_View View, short shrink);	Sets and returns the clipping region to be the view’s HotRect (allowing for shrinking in both directions)
void Hv_ClipRect(Hv_Rect *);	Set the clipping region based on the
Hv_Region Hv_ClipView(Hv_View View);	Set the clipping region to the visible portion of the specified view.
void Hv_CloseDialog(Hv_Widget);	Closes “unmanages” the dialog, but does not free its memory.
short Hv_ColorFromText(char *line);	Used for IO: tryes to convert a line into a color as in if “red” appears in the line it returns the constant Hv_red.
void Hv_CompoundStringDimension(Hv_String *str, short *w, short *h);	Returns the string size in pixels taking into account special formatting.
void Hv_ConcatTextToTextItem(Hv_Item Item, char *concattext);	Concatenates the specified texdt to what already is in the text item.
Boolean Hv_Contained(char *str, char *substr);	CASE INSENSITIVE substring check.
void Hv_CopyFRect (Hv_FRect *dest, Hv_FRect *src);	Copy a world based rectangle.
Hv_Region Hv_CopyItemRegion(Hv_Item Item, Boolean IncludeChildren);	Returns a copy of an item’s region, optionally including children.
void Hv_CopyRect (Hv_Rect *dest, Hv_Rect *src);	Copy a pixel based rectangle.
void Hv_CopyRegion(Hv_Region *dreg, Hv_Region sreg);	Copy a region.
void Hv_CopyString(Hv_String **dstr, Hv_String *sstr);	Copy an Hv_String.

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
int Hv_CountSegments(char *line);	Counts the number of segments in a line, where a segment is characters surrounding by whitespace. For example, ‘23.4 hello gimme a break 22’ contains six segments. If the global variable Hv_quoteDelimitsSegment is set to True, then the line ‘23.4 hello “gimme a break” 22’ has four segments.
Hv_Pixmap Hv_CreatePixmap(unsigned int w, unsigned int h);	Allocate a pixmap of specified dimensions.
Hv_Region Hv_CreateRegion(void);	Allocate an empty region.
Hv_Region Hv_CreateRegionFromLine(short x1, short y1, short x2, short y2, short slop);	Convert a line to a region allowing for some “slop”.
Hv_Region Hv_CreateRegionFromLines(Hv_Point *pp,short nlines,short slop);	Convert lines to a region allowing for some “slop”
Hv_Region Hv_CreateRegionFromPolygon(Hv_Point *poly, short n);	Create a region that corresponds to the given polygon.
Hv_String *Hv_CreateString(char *text);	Creates an Hv_String from the given text. The default font is Hv_fixed2 abd the default color is Hv_black.
Hv_Widget Hv_CreateSymbolPopup(Widget rc, short font);	Places a symbol popup on a dialog.
void Hv_DeleteAllListItems(Hv_Widget);	Removes all items in a list dialog widget.
void Hv_DeleteFeedbackEntry(Hv_Item, short);	Remove a feedback entry.
void Hv_DeleteItem(Hv_Item Item);	Deletes the specified item.
void Hv_DeletePlots(Hv_View View);	Deletes all the plots on the given view.
void Hv_DeselectAllListItems(Hv_Widget w);	Deselect all items in the list (a dialog item)
void Hv_DestroyItem(Hv_Item Item);	Free an items memory and remove it from the View’s linked list.
void Hv_DestroyRegion(Hv_Region region);	Frees the memory associated with the given region.
void Hv_DestroyString(Hv_String *str);	Free an Hv_String.
void Hv_DestroyWidget(Hv_Widget w);	Frees the memory associated with the give widget.
Hv_Widget Hv_DialogColumn(Hv_Widget parent, short spacing);	Create a “column” in a dialog with given spacing for subsequently added children.
Hv_Widget Hv_DialogRow(Hv_Widget parent, short spacing);	Create a “row” in a dialog with given spacing for subsequently added children.
Hv_Widget Hv_DialogTable(Hv_Widget parent, short numrow, short spacing);	Create a “table” in a dialog with given spacing (and number of rows) for subsequently added children.
void Hv_DisableAllItems(Hv_View View);	Make all items on a view “unselecatble” (temporarily)
void Hv_DisableItem(Hv_Item Item);	Make a specified item unselecatble.
void Hv_DisableWidget(Hv_Widget w);	Make a dialog item unselectable.
int Hv_DoDialog(Hv_Widget, Hv_Widget);	The dialog event-loop.
void Hv_Draw3DCircle(short xc short yc , int rad, Hv_ColorScheme *);	Draws a “3D” circle at specified location and radius using the provided color scheme.
void Hv_Draw3DRect(Hv_Rect *rect, Hv_ColorScheme *scheme);	Draws a “3D” rect using the provided color scheme
void Hv_DrawBoxItem(Hv_Item, Hv_Region);	The standard drawing routine for box items.
void Hv_DrawButtonItem(Hv_Item, Hv_Region);	The standard drawing routine for button items.
void Hv_DrawColorButtonItem(Hv_Item, Hv_Region);	The standard drawing routine for color button items.

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
void Hv_DrawItem(Hv_Item, Hv_Region);	The toplevel drawing routine for all items. This routine will then dispatch the standard drawing routine (if there is one) and then any user provided drawing routine.
void Hv_DrawLine(short x1, short y1, short x2, short y2, short color);	Draws a line from (x1, y1) and (x2, y2) (local coordinates) in the specified color.
void Hv_DrawLines(Hv_Point *pts, short np, short color);	Draw lines as specified.
void Hv_DrawPoint(short x, short y, short color);	Draw point as specified.
void Hv_DrawPoints(Hv_Point *pts, int np, short color);	Draw points as specified.
void Hv_DrawSegments(Hv_Segment *segs, short nseg, short color);	Draw line segments as specified.
void Hv_DrawSymbolsOnPoints(Hv_Point pnts*, int np, short size, short color, short symbol);	Draw symbols centered aroun each point in “pnts” with specified attributes.
void Hv_DrawView(Hv_View View, Hv_Region region);	The basic view drawing routine. It will call Hv_DrawItem for all the items in its item list that overlap the given region and then call any user supplied specialty view drawing.
void Hv_DrawViewHotRect(Hv_View View);	This will cause all the view’s hotrect items to be redrawn. Control items will not be redrawn.
void Hv_DrawWorldCircle(Hv_View View, float x, float y, float radius, Boolean fill, short framecolor, short fillcolor);	Draws a circle based on world coordinates (x, y) for the center and with the given radius. Filled determines whether the circle is filled or hollow.
void Hv_DrawWorldLineItem(Hv_Item Item, Hv_Region region);	Standard drawing routine for world based line items.
void Hv_DrawWorldOvalItem(Hv_Item Item, Hv_Region region);	Standard drawing routine for world based oval items
void Hv_DrawWorldPolygonItem(Hv_Item Item, Hv_Region region);	Standard drawing routine for world based line items
void Hv_DrawWorldRectItem(Hv_Item Item, Hv_Region region);	Standard drawing routine for world based line items
void Hv_EditColorButtonItem(Hv_Event);	The standard editor for color buttons.
void Hv_EditWorldLineItem(Hv_Event hvevent);	The standard editor for world based lines.
void Hv_EditWorldOvalItem(Hv_Event hvevent);	The standard editor for world based ovals.
void Hv_EditWorldPolygonItem(Hv_Event hvevent);	The standard editor for world based polygons.
void Hv_EditWorldRectItem(Hv_Event hvevent);	The standard editor for world based rects.
void Hv_EnableAllItems(Hv_View View);	Enable all items on a view.
void Hv_EnableItem(Hv_Item Item);	Enables an Item so that it will respond to the pointer.
void Hv_EnableWidget(Hv_Widget w);	Make a dialog item selectable.
void Hv_EnlargeLine(Hv_Rect, Hv_Point, Hv_Point *, short, Hv_Point *);	Mouse-enlarging routine for lines.
void Hv_EnlargeOval(Hv_Rect *, Hv_Rect, Hv_Point, Boolean);	Mouse-enlarging routine for ovals.
void Hv_EnlargeRect(Hv_Rect *, Hv_Rect, Hv_Point, Boolean);	Mouse-enlarging routine for rects.
void Hv_EraseRect(Hv_Rect *);	This will erase the specified rect by filling it with the background color.
void Hv_EraseRegion (Hv_Region region);	This will erase the specified region by filling it with the background color

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
Boolean Hv_FPointInFRect(Hv_FPoint pp, Hv_FRect *rect);	Returns True if the given world point “pp” is contained in the world rect “rect”.
char *Hv_FileSelect(char *prompt, char *mask, char *deftext);	Call a file selection dialog to select a file. “prompt” will appear in the dialog, and only files matching “mask” (e.g. “*.txt”) will be display. “deftxt” is the default text in the selector.
void Hv_FillCircle(short xc, short yc , short radius, Boolean frame, short fillcolor, short framecolor);	Fill the given circle (and optionally frame it in a different color).
void Hv_FillPatternRect(Hv_Rect *rect, short color, short pattern);	Fill a rect in the given color and pattern.
void Hv_FillPolygon(Hv_Point *poly, short np, Boolean fill, short fillcolor, short framecolor);	Fill the given polygon (and optionally frame it in a different color).
void Hv_FillRect(Hv_Rect *rect, short color);	Fill rect in given color.
void Hv_FillRegion(Hv_Region region, short color, Hv_Region clipregion);	Fill a region in the given color (clipped agains “clipreg”, if not NULL)
char *Hv_FindRecord(FILE * fp, char *recname, Boolean rewind, char *line);	Returns the line containing the recname, and a copy of it in the “line” variable, if the specified record is found, leaving the file pointer positioned for reading at the start of the record. The “rewind” variable controls whether the file is rewound before searching. Returns NULL on failure
void Hv_FitFeedback(Hv_View View, short margin);	Make sure the view accomodates the feedback, with the given margin (pixels) to spare.
void Hv_FixFRectWH(Hv_FRect *thefSR);	Make sure a world rect’s width and height are internally consistent.
void Hv_FixItemWorldArea(Hv_Item Item, Boolean children);	Make sure an items world area is internally consistent.
void Hv_FixPolygonRegion(Hv_Item Item);	Update an Hv_WORLDPOLYGONITEM’s region
void Hv_FixRectRB(Hv_Rect *theSR);	Makes sure a rect’s right and bottom are consistent with its left, top, width, and height.
void Hv_FixRectWH(Hv_Rect *theSR);	Makes sure a rect’s width and height are consistent with its left, top, right, and bottom.
void Hv_Flush(void);	Flushes all pending X events.
void Hv_FractionDoneUpdate(Hv_Item, float);	Update a fraction-done item accordingly.
void Hv_FrameArc(short x, short y, short width, short height, int ang1, int ang2 , short color);	Frame arc in specified color. (x, y, width, and height determin the bounding rect) (ang1, ang2) are the limiting angles in degrees.
void Hv_FrameCircle(short xc, short yc , short radius, short color);	Frame the circle with the specified attributes.
void Hv_FramePolygon(Hv_Point *poly, short np, short color);	Frame a polygon of “np” points in the given color.
void Hv_FrameRect(Hv_Rect *rect, short color);	Frames a pixel rect in the given color.
void Hv_Free(void *ptr);	$H^{\underline{V}}$ wrapper for usual “free” function.
void Hv_FreePixmap(Hv_Pixmap pmap);	Frees the memory associated with a pixmap.
void Hv_FreeView(Hv_View View);	Deletes a view and frees all its memory, except memory allocated by the user and placed in data->user. The user should provide a “dstroy” routine via the Hv_DESTROY attribute to handle this.
void Hv_FullClip(void);	Sets the cliiping region to the entire main window.
Hv_RadioListPtr Hv_GetActiveRadioButton(Hv_RadioHeadPtr radiohead);	Returns the list node for the active radio button in the set cotrolled by “radiohead”. Not used as often as Hv_PosGetActiveRadioButton.

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
void *Hv_GetActiveRadioButtonData(Hv_RadioHeadPtr radiohead);	Little used routine that reutrns user data optionally stored with the set of radio buttons controlled by “radiohead”.
int *Hv_GetAllListSelectedPos(Hv_Widget list);	Reurns array of positions (starting at 1) of all selected items in a list. The terminatineg entry in the array is -1. The caller should free the array with Hv_Free.
short Hv_GetButtonState(Hv_Item);	Returns the button state, either Hv_ON or Hv_OFF.
short Hv_GetColorButtonColor(Hv_Item);	Returns the current color of an H$^{\underline{V}}$ color button item.
Hv_Widget Hv_GetDialogActionButton(Hv_Widget dialog, int which);	A way to obtain the individual buttons from a dialog closeout. For example, Hv_GetDialogActionButton(dialog, Hv_OKBUTON) returns that dialog’s OK button.
int Hv_GetDialogAnswer(Hv_Widget);	Returns the reason (answer) that caused the dialog to close. One of Hv_DONE, Hv_CANCEL, Hv_OK, Hv_APPLY, or Hv_DELETE.
void Hv_GetFRectCenter(Hv_FRect *r, float *xc, float *yc);	Returns the centroid of a world rect.
float Hv_GetFloatText(Hv_Widget);	Obtains a floating point value from the text dialog item.
int Hv_GetIntText(Hv_Widget);	Obtains an integer value from the text dialog item.
Hv_Rect *Hv_GetItemArea(Hv_Item Item);	Returns the rectangular area surrounding an item.
void Hv_GetItemCenter(Hv_Item Item,short *xc,short *yc);	Returns the pixel center of an item.
short Hv_GetItemColor(Hv_Item Item);	Returns the “main” color of an item.
void *Hv_GetItemData(Hv_Item Item);	Returns the “user” data of an item.
Hv_Region Hv_GetItemDrawingRegion(Hv_Item Item, short dh, short dv);	Returns the item’s drawing region.
Hv_Item Hv_GetItemFromTag(Hv_ItemList items, short tag);	Returns the FIRST item in the given list that it finds with a tag matching the input “tag”.
Hv_Region Hv_GetItemRegion(Hv_Item Item);	Returns an item’s region (which may be NULL if the item has elected to just use an “area” rect to mark its boundary.
short Hv_GetItemState(Hv_Item Item);	Returns the state of an item.
Hv_String *Hv_GetItemString(Hv_Item Item);	Returns the item’s string, if any.
short Hv_GetItemTag(Hv_Item Item);	Returns the item’s tag.
char *Hv_GetItemText(Hv_Item Item);	Returns an item’s text (from within its Hv_String).
Hv_View Hv_GetItemView(Hv_Item Item);	Simple information hider that returns the view that the item lives on (i.e., Item->view).
int Hv_GetListSelectedPos(Hv_Widget);	For a select-one-only list, returns the position (starting at 1) of the selected item.
char *Hv_GetNextNonComment(FILE *fp, char *line);	Returns the next line that is NOT a comment line starting at the present file position.
Boolean Hv_GetOptionButton(Hv_Item Item);	Treurns the stat of an H$^{\underline{V}}$ option button item.
void Hv_GetRectCenter(Hv_Rect *r,short *xc,short *yc);	Return the center of a rect.
void Hv_GetRectLTRB(Hv_Rect *rect, short *l, short *t, short *r, short *b);	For a given rect, this returns the left, top, right and bottom in local coordinates.
int Hv_GetScaleValue(Hv_Widget scale);	For a scale widget on a dialog, this returns its current value. Do NOT use this call for Hv_Items.
char *Hv_GetString(Hv_Widget);	Returns the string associated with an Hv_Widget, presumably from a dialog box. Do NOT use this call for Hv_Items.
Boolean Hv_GetToggleButton(Hv_Widget);	Returns state of a toggle button dialog item.

TABLE 17. Alphabetical listing of public functions available in H$^{\underline{V}}$.

Function	Comment
void *Hv_GetViewData(Hv_View View);	Returns the view's user data.
Hv_Rect *Hv_GetViewHotRect(Hv_View View);	Returns the HotRect (politically incorrect equivalent: View->hotrect)
Hv_ItemList Hv_GetViewItemList(Hv_View View);	Returns a view's item list.
Hv_Region Hv_GetViewRegion(Hv_View View);	Returns the visible region of a view.
short Hv_GetViewTag(Hv_View View);	Returns the views tag. Most applications written so far use the politically incorrect equivalent: View->tag
char *Hv_GetViewTitle(Hv_View View);	Returns the title of a view.
short Hv_GetWheelCurrentValue(Hv_Item wheel);	Returns the current value of a wheel item.
void Hv_GetWheelValues(Hv_Item wheel,short *minval,short *maxval,short *val);	Returns all parameters of a wheel item
void Hv_Go(void);	Launches the $H^{\underline{V}}$ main loop.
short Hv_HandleChoice(Hv_Item);	Handles the choice of a choice button, Returns the "position" (starting at 1) of the choice in its choice set.
Hv_Region Hv_HotRectRegion(Hv_View View);	Returns (but does not set) the region corresponding to the visible part of a view's HotRect.
void Hv_Information(char *message);	Displays a simple information dialog box with the specified message.
void Hv_InitCharStr(char **dst, char *sst);	Mallocs a string and initializes it.
void Hv_InitRect(Hv_Rect *theSR);	Initialize the given rect to (0, 0, 0, 0)
Hv_Region Hv_IntersectRegion(Hv_Region reg1, Hv_Region reg2);	Computes the intersection of reg1 and reg2 and returns the result.
void Hv_InvertColorScheme(Hv_ColorScheme *);	Changes the data within a color scheme so what previously had a 3D appearance suggesting an "outsie" will now look like an "insie", and vice versa.
Boolean Hv_IsAComment(char *line);	True if the line is interpreted as a comment.
Boolean Hv_IsViewDirty(Hv_View View);	Applicable for views that save their background. If True, the view's off-screen buffer is no longer valid and needs to be redrawn.
Hv_Region Hv_ItemDrawingRegion(Hv_Item, short, short);	Returns an item's visible region.
Boolean Hv_ItemIsEnabled(Hv_Item Item);	True if the item is selecatable.
void Hv_KillRegion(Hv_Region *region);	Frees the memory associated with the given region and sets the region variable to NULL.
void Hv_LocalPolygonToWorldPolygon(Hv_View View, short npts, Hv_Point *poly, Hv_FPoint *wpoly);	Convert a polygon from local (pixel) to world coordinates.
void Hv_LocalRectToWorldRect(Hv_View View, Hv_Rect *area, Hv_FRect *wr);	Convert a rect from local (pixel) to world coordinates
void Hv_LocalToWorld(Hv_View View, float *fx, float *fy, short x, short y);	Convert coordinates from local (pixel) to world coordinates
FILE *Hv_LookForNamedFile(char *filenamae, char **paths, char *mode);	Looks for and opens (with mode = "mode") the file of the given name looking first in the current working directory and then in every path in the NULL terminated array "paths". Usually you add paths to the global variable Hv_ioPaths via the function Hv_AddPath and then use Hv_ioPaths as the second argument.
void *Hv_Malloc(size_t size);	An $H^{\underline{V}}$ wrapper around the standard malloc.
int Hv_ModifyColor(Hv_Widget w, short *color, char *prompt, Boolean allownocolor);	Call a standard color picker to choose a color. If nocolor is True, then "no color" will be one of the choices.

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
int Hv_ModifyPattern(Hv_Widget w,short *pattern,char *prompt);	Call a standard pattern picker to choose a pattern.
void Hv_MoveItemBehindItem(Hv_Item moveItem, Hv_Item stayItem);	Move one item behind another (so that it appears on top).
Hv_FRect *Hv_NewFRect(void);	Mallocs and returns a pointer to a world-based rectangle
Hv_Point *Hv_NewPoints(int n);	Mallocs "n" Hv_Points.
void Hv_NewPosSelectListItem(Hv_Widget w, int pos, Boolean CB);	For list dialog items: w is the list widget, pos is the position you want selected (starting at 1), CB cotrols whether or not you want the callback to be called
Hv_Rect *Hv_NewRect(void);	Mallocs and returns a pointer to a rect.
Hv_View Hv_NewXYPlotView(char *title, Boolean autoopen);	Creates a new plot view with the given title and optionally opens it at creation.
void Hv_NextRecordSize(FILE *fp, char *startrec, char *endrec, int *numlines, int *numchars);	If it finds a record delimited by startrec and endrec, it returns the number of lines (comments included) and the number of charac-ters.
char *Hv_NextTextSegment(char **line);	Returns the next segment in a line and moves the line "past" the returned segment. The segment should be freed by the caller. If line is "abc def ghi" then this returns "abc" and afterward line is "def ghi".
Boolean Hv_NotAComment(char * line);	True if the line is interpreted as not being a comment
void Hv_NumberToString(char*text,floatv,shortp,float eps);	
void Hv_OffsetHotRectItems(Hv_ItemList items, short dh, short dv);	Shift all the HotRect items in the given list by the specified amount (pixels).
void Hv_OffsetItem(Hv_Item Item, short dh, short dv, Bool-ean children);	Shifts an item by the specified number of pixels in the horizontal (dh) and vertical (dv) directions. If children is True, then all the item's children are also shifted.
void Hv_OffsetItemList(Hv_ItemList items,short dh,short dv);	Shifts all items on an item (linked) list by the specified number of pixels in the horizontal (dh) and vertical (dv) directions.
void Hv_OffsetPolygon(Hv_Point *,int,short,short);	Shifts a polygon by the specified number of pixels in the horizontal (dh) and vertical (dv) directions.
void Hv_OffsetRect (Hv_Rect *rect, short dh, short dv);	Shifts a rect by the specified number of pixels in the horizontal (dh) and vertical (dv) directions.
void Hv_OpenDialog(Hv_Widget dlog, Hv_Widget default);	Opens a dialog box (dlog) setting the default widget to "default" (i.e. the button that will be effectively pressed if the user hits return.) However, the latter feature (the default button) doesn't always work because *Motif* doesn't always work as advertised.
void Hv_OptionButtonCallback(Hv_Event hvevent);	Default callback for option buttons.
short Hv_PatternFromText(char *line);	Used for IO: tryes to convert a line into a pattern as in if "star" appears in the line it returns the constant Hv_STARPAT
Boolean Hv_PointInItem(Hv_Point pp, Hv_Item Item);	Returns True if the given mouse point falls inside the item's region or, if it has no region, its area.
Hv_Item Hv_PointInPlotItem(Hv_View View, Hv_Point pp);	It the specified point is inside a plot on the specified view, this returns the plot item, else it returns NULL.
Boolean Hv_PointInPolygon(Hv_Point pp, Hv_Point *poly, int np);	Ruturns True if the local point "pp" is inside the local polygon "poly" which contains "np" vertices.
Boolean Hv_PointInRect(Hv_Point pp, Hv_Rect *rect);	Returns True if the given point is inside the rect.
Boolean Hv_PointInRegion(Hv_Point,Hv_Region);	Returns True if the point is in the specified region.

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
void Hv_PolygonEnclosingRect(Hv_Rect *rect, Hv_Point *poly, int np);	Sets |"rect" to just fit around the polygon specified by "poly" with "np" points.
Boolean Hv_PolygonIntersectsPolygon(Hv_Point *poly1, Hv_Point *poly2, short np1, short np2);	Returns True if the two polygons intersect.
Boolean Hv_PolygonIntersectsRect(Hv_Point *poly , short np, Hv_Rect *rect);	Checks whether poly of np points intersects the given rect.
short Hv_PosGetActiveRadioButton(Hv_RadioHeadPtr radiohead);	Returns the position (starting at 1) of the acitve radio button from a set on a dialog. This is for dialog widgets only.
void Hv_PosReplaceListItem(Hv_Widget w, int pos, char * str);	For the given list it replaces the text a the designated position (starting at 1) with the new string
void Hv_PosSelectListItem(Hv_Widget w, int pos);	Like Hv_NewPosSelectListItem, except the callback is automatically called
void Hv_PosSetActiveRadioButton(short pos, Hv_RadioHeadPtr radiohead);	For a given set of radio buttons specified by "radiohead" this sets the one specified by "pos" (starting at 1) to be on
void Hv_PosSetChoice(short pos, Hv_Item choiceset);	For the given choice set (which are items on views, NOT dialog widgets), this sets the one at position "pos" (counting starts at 1) to be selected.
int Hv_Question(char * questio);	A simple dialog that asks the provided question and returns either Hv_YES or Hv_NO.
void Hv_RadioListSensitivity(Hv_RadioHeadPtr radiohead, short pos, Boolean val);	Controls the sensitivity for the radio button at the specified position (starting at 1) in the set of radio buttons managed by "radiohead". "val" determines if it is selectable (True) or unselectable and ghosted (False). If pos <=0 "val" is applied to ALL the buttons.
int Hv_ReadNextLine(FILE *fp, char *line, char *startrec, char *endrec);	Reads the next line, NOT excluding comments, placing a copy into the "line" variable. Returns zero on sucess.Failure codes are Hv_FOUNDEOF if and end-of-file is encountered, Hv_FOUNDSTARTREC if we unexpectedly come upon a new record, and Hv_FOUNDENDREC if we hit the end.
void Hv_RecordLineCount(FILE *fp, char *startrec, char *endrec, int *numlines, int *numnclines);	If it finds a record delimited by startrec and endrec, it returns the total number of lines and the number of von-comment lines
short Hv_RectBottom(Hv_Rect *r);	returns the bottom of the specified rect
Boolean Hv_RectInRegion(Hv_Rect *, Hv_Region);	Returns True if the rect intersects the region.
Boolean Hv_RectIntersectsRect(Hv_Rect *, Hv_Rect *);	Returns True if the two rects intersect.
Hv_Region Hv_RectRegion(Hv_Rect *rect);	Converts a rect to an equivalent region.
short Hv_RectRight(Hv_Rect *r);	Returns the right side of the specified rect.
void Hv_RedrawItem(Hv_Item Item);	Redraws the item, if it is visible.
void Hv_RedrawViewHotRect(Hv_View View, short type, Hv_Region cregion);	More complicated cousin to Hv_DrawViewHotRect. If type is not zero, this will redraw only matching types/
void Hv_RemoveItemTimedRedraw(Hv_Item Item);	Remove a timed redraw (used in animation) from an item.
void Hv_RemoveRectFromRegion(Hv_Region *reg, Hv_Rect *rect);	Subtract the area of a rect from an existing region.
Boolean Hv_ReportDatabaseSearch(FILE * fp, char * recname, Boolean rewind);	True if the specified record is found, leaving the file pointer positioned for reading at the start of the record. The "rewind" variable controls whether the file is rewound before searching.

TABLE 17. Alphabetical listing of public functions available in H^{V}.

Function	Comment
void Hv_ResetActiveView(Hv_Event, Hv_View *, Hv_Item *, Hv_Cursor);	Used when placing an item on the hotrect from a “tool”.
void Hv_ResizeItem (Hv_Item Item, short dh, short dv, Boolean children);	Resize an item by the specified amounts (both dh and dv can be negative) and optionally all children as well.
void Hv_ResizeItemList(Hv_ItemList items,short dh,short dv);	Resize all items by the specified amount.
void Hv_ResizeRect (Hv_Rect *rect, short dh, short dv);	Resize a rect by the specified amount.
void Hv_RestoreClipRegion(Hv_Region);	Restore a clipping region.
void Hv_RestoreViewBackground(Hv_View View);	Copy a view’s offscreen backgron onscreen.
void Hv_SaveViewBackground(Hv_View View, Hv_Region region);	Save the present background in an offscreen buffer.
void Hv_SelectListItem(Hv_Widget w, char *name);	Select the item of the given name from a list dialog item.
void Hv_SendItemToBack(Hv_Item Item);	Send the item behind all other items.
void Hv_SendItemToFront(Hv_Item Item);	Put item on top of all other items.
void Hv_SeparatorResizeToParent(Hv_Item Item);	Resize a separator so that it spans its parent.
void Hv_SetActiveRadioButton(Hv_Widget nowon, Hv_RadioHeadPtr radiohead);	For a given set of radio buttons specified by “radiohead” this sets the button “nowon” to be on.
void Hv_SetButtonState(Hv_Item button, short state);	Set the state of a button to Hv_ON or Hv_OFF.
void Hv_SetClippingRegion(Hv_Region);	Set the current clipping region to “region”.
void Hv_SetCursor(Cursor curs);	Set the cursor to the one passed in “curs”.
void Hv_SetDeleteItem(Hv_Widget,Hv_Item);	Set which item will be deleted if the user selects “Delete” from a dialog box -- this will almost always be the item whose callback caused the dialog to popup.
void Hv_SetFRect(Hv_FRect *frect, float xmin, float xmax, float ymin, float ymax);	Set a world based rect according to the limits provided.
void Hv_SetFloatText(Hv_Widget w, float val, int numdec);	Displays the value in the widget w with “numdec” numbers to the right of the decimal point
void Hv_SetIntText(Hv_Widget, int);	Displays the value in the widget w
void Hv_SetItemArea(Hv_Item Item, short left, short top, short width, short height);	Set the area of an item.
void Hv_SetItemColor(Hv_Item Item, short color);	Set the Item’s main color field.
void Hv_SetItemData(Hv_Item Item, void *data);	Set the Item’s user-data field.
void Hv_SetItemDrawControlBit(Hv_Item Item, int bit);	Set the specified bit.
void Hv_SetItemRegion(Hv_Item Item, Hv_Region region);	Set the region of an item.
void Hv_SetItemSensitivity(Hv_Item Item, Boolean val, Boolean Redraw);	Set whether an item
void Hv_SetItemString(Hv_Item Item, Hv_String *str);	Set the string field of an item.
void Hv_SetItemText(Hv_Item Item, char *text);	Set the text portion of an items’ Hv_String
void Hv_SetItemTimedRedraw(Hv_Item Item, unsigned long interval);	Set the interval until the item redraws itself (for animation)
void Hv_SetLineStyle(int width, int style);	Sets the line width and style for subsequent drawing.
void Hv_SetLineWidth(int);	Sets the line widht for subsequent drawing (pixels). You should always reset it to one or zero (which mean the same thing.)

TABLE 17. Alphabetical listing of public functions available in H$^{\underline{V}}$.

Function	Comment
void Hv_SetMenuItemString(Widget menuitem, char *str, short font);	Change a menu item's label.
void Hv_SetOptionButton(Hv_Item Item, Boolean control);	Set the state of an option button.
void Hv_SetPattern(int pattern);	Set the pattern for subsequent fills.
void Hv_SetPoint(Hv_Point *pp, short x, short y);	Stuff a pixel point.
void Hv_SetQuickZoomLabel(short item, char *str);	Set one of the quick zoom labels to "str".
void Hv_SetRect(Hv_Rect *rect, short left, short top, short width, short height);	Sets the given rect to have the specified left, top, width and height.
void Hv_SetScaleMinMax(Hv_Widget w, int min, int max);	Sets the limits on a sliding scale dialog item
void Hv_SetScaleValue(Hv_Widget w,int val);	Sets the current value of a scale dialog item.
void Hv_SetSegment(Hv_Segment *seg, short x1, short y1, short x2, short y2);	Stuff the cordinates into a (line) segment.
void Hv_SetSensitivity(Hv_Widget w, Boolean val);	Set a dialog item's sensitivity (selectability) according to "val".
void Hv_SetSliderCurrentValue(Hv_Item slider,short val, Boolean redraw);	Set the current value of a slider and optionally redraw.
void Hv_SetString(Hv_Widget w, char *str);	Sets the text displayed in a text dialog item
void Hv_SetToggleButton(Hv_Widget w, Boolean state);	Sets the state of a toggle button dialog item
void Hv_SetViewBrickWallItem(Hv_Item Item);	Set the item that no other item can be "underneath".
void Hv_SetViewDirty(Hv_View View);	Flag the view for a refresh of its offscreen buffer, if it uses one.
void Hv_SetViewMinimumSize(Hv_View View, short minw, short minh);	Set the size beyond which the view cannot be reduced.
void Hv_SetViewPopupControl(Hv_View View, int bits);	Set which items are selectable from a view's popup menu.
void Hv_SetViewTitle(Hv_View View, char *title);	Change a view's title.
void Hv_SetWheelCurrentValue(Hv_Item wheel, short val, Boolean redraw);	Set the current value of a wheel item.
void Hv_SetWheelValues(Hv_Item wheel,short vmin,short vmax,short val,Boolean redraw);	Set all parameters of a wheel item.
void Hv_ShrinkRect(Hv_Rect *r, short dh, short dv);	Reduce (or expand, if dh/dv are negative) the rect by the specified amount.
void Hv_ShrinkRegion(Hv_Region region, short h, short v);	Reduce (or expand, if dh/dv are negative) the region by the specified amount.
void Hv_Simple3DDiamond(short x, short y, short size, short fillcolor, Boolean outsie);	Draws a simple 3D diamond (like in a choice button) in the given color. "outsie" determines whether the diamond appears "out" or "in".
void Hv_Simple3DRect(Hv_Rect *rect, Boolean outsie, short fillcolor);	Draws a simple 3D rect (like in a. option button) in the given color. "outsie" determines whether the diamond appears "out" or "in".
Hv_Widget Hv_SimpleColorLabel(Hv_Widget parent, char *label, Hv_FunctionPtr callback);	Convenience routine for creating a dialog item used for selecting a color
Hv_Widget Hv_SimpleDialogSeparator(Hv_Widget parent);	Convenience routine for creating a separator in a dialog.
Hv_Widget Hv_SimplePatternLabel(Hv_Widget pattern, char *label, Hv_FunctionPtr callback);	Convenience routine for creating a dialog item used for selecting a pattern.
Hv_Widget Hv_SimpleTextEdit(Hv_Widget parent, char *label, char *deftext, short numcol);	Convenience routine for creating a dialog item pair: a label and a text entry field.

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
Hv_Widget Hv_SimpleToggleButton(Hv_Widget parent, char *label);	Convenience routine for creating a toggle button dialog item.
void Hv_SizeTextItem(Hv_Item Item, short x, short y);	Size a text item to accomodate its string.
Hv_Widget Hv_StandardActionButtons(Hv_Widget parent,short spacing,int whichones);	Convenience routine for creating close-out buttons (dialogs)
Hv_Widget Hv_StandardButton(Hv_Widget parent, char *label, Hv_FunctionPtr CB);	Convenience routine for creating buttons (dialogs).
Hv_Widget Hv_StandardClearListButton(Hv_Widget parent, Hv_Widget list, char *label, short font);	Convenience routine for creating a clear button for a list dialog item.
Hv_Widget Hv_StandardDeleteButton(Hv_Widget parent);	Convenience routine for creating delete buttons (dialogs).
Hv_Widget Hv_StandardDoneButton(Hv_Widget parent, char *donestr);	Convenience routine for creating done buttons (dialogs).
void Hv_StandardFirstEntry(Hv_Event, Hv_FunctionPtr, Hv_View *, Hv_Item *, Hv_Cursor *);	Used for “tools” that lay things down on the HotRect.
Hv_Widget Hv_StandardLabel(Hv_Widget rc,char *label,int type);	Convenience routine for creating labels.
Hv_Item Hv_StandardOptionButton(Hv_View View, Hv_Item parent, Hv_Item pitem, Boolean *control, char *label, int drawcontrol);	Convenience routine for creating option buttons.
void Hv_StandardRedoItem(Hv_Item Item);	Good for redrawing items after the have been edited and may have changed size.
Hv_Widget Hv_StandardScale(Hv_Widget parent,int min,int max,int current,Hv_FunctionPtr CB);	A convenience function for scales.
Hv_Widget Hv_StandardTextField(Hv_Widget parent);	A convenience function for creating text fields.
Hv_Widget Hv_StandardToggleButton(Hv_Widget parent, char *label, Hv_FunctionPtr CB);	A convenience function for creating toggle buttons.
void Hv_StandardTools(Hv_View View, short left, short top, short orientation, short resizepolicy, int whichtools, Hv_Item *first, Hv_Item *last);	Create the standard suite of drawing tools, positioned and oriented as specified. The resize policy depends on where the tools are placed. “whichtools” is a combination of bits, usually Hv_ALLTOOLS.
void Hv_StartSimulation(Hv_View View, Hv_FunctionPtr proc, unsigned long interval);	Start (or restart) a simulation on a view.
void Hv_StopSimulation(Hv_View View);	Pauses the simulation running on the given View.
void Hv_StringDimension(Hv_String *str, short *w, short *h);	Returns the size of a string in pixels.
void Hv_StringDraw(short x, short y, Hv_String *str, Hv_Region region);	Draws an Hv_String at the specified pixel location.
Hv_Region Hv_SubtractRectFromRegion(Hv_Region reg, Hv_Rect *rect);	Removes a region from an existing region.
Hv_Region Hv_SubtractRegion(Hv_Region reg1, Hv_Region reg2);	Returns the difference of two regions.
void Hv_SubtractRegionFromRegion(Hv_Region *mainreg, Hv_Region subreg);	Removes the region “subreg” from “mainreg”.
short Hv_SymbolFromText(char *line);	Used for IO: tryes to convert a line into a symbol as in if “circle” appears in the line it returns Hv_CIRCLESYMBOL
void Hv_SysBeep(void);	Causes your terminal to beep.
void Hv_ToggleButton(Hv_Item);	Toggles a button from on to off (or vice versa).

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Function	Comment
void Hv_ToggleColorButtonItem(Hv_Event);	Toggles a color button from on to off (or vice versa).
void Hv_UnionPolygonWithRegion(Hv_Point *poly, int np, Hv_Region *region);	Adds the region inside the polygon (with “np” vertices)
void Hv_UnionRectWithRegion(Hv_Rect *sr, Region region);	Adds the rect to the given region.
Hv_Region Hv_UnionRegion(Hv_Region reg1, Hv_Region reg2);	Returns the sum of two regions.
void Hv_UpdateBoxItem(Hv_Item box,char *text);	Redraws the box item with new text.
void Hv_UpdateFeedbackArea(Hv_Item Item);	Redraws the feedback item
void Hv_UpdateFeedbackText(Hv_Item Item);	Redraws changed feedback text.
void Hv_UpdateItem(Hv_Item Item);	Basically redraws the item by updating the view with the item’s visible area.
void Hv_UpdateItemOffScreen(Hv_Item Item, Hv_Region region);	Like Hv_UpdateItem, but for off-screen drawing.
void Hv_UpdateLED(Hv_Item Item, int pos, int state);	Updates the LED at position pos (starting at 1) to the state (Hv_ON of Hv_OFF).
void Hv_VaCreateDialog(Hv_Widget *, ...);	The variable length argument function used to create Motif Dialog shells.
Hv_Widget Hv_VaCreateDialogItem(Hv_Widget, ...);	The variable length argument function used to create Dialog items, which are really Motif widgets. This is ONLY used in conjuction with dialogs, NOT with views.
Hv_Item Hv_VaCreateItem(Hv_View,...);	The standard Item creation routine.
void Hv_VaCreateMenu(Hv_Widget *menu, ...);	The standard menu creation routine.
Hv_Widget Hv_VaCreateMenuItem(Hv_Widget menu, ...);	The standard menu item creation routine.
void Hv_VaCreateView(Hv_View *View, ...);	The standard View creation routine.
void Hv_VaInitialize(unsigned int, char **, ...);	$H^{\underline{V}}$’s main initialization routine.
Boolean Hv_ViewIsActive(Hv_View View);	If True the View is somewhere on the Virtual desktop, although it may not be visible. (i.e., the View is opened.)
Boolean Hv_ViewIsVisible(Hv_View View);	Returns True if any par of the view is presently visible -- this means that the view is open AND it has not been moved completely off screen. This does NOT take into account the possibility that the view is occluded by other views.
Boolean Hv_ViewSavesBackground(Hv_View View);	Returns True if this view was created with the **Hv_SAVEBACKGROUND** draw control bit set.
int Hv_Wait(int);	Pauses for specified number of seconds and displays a “count-down” dialog box
void Hv_Warning(char *);	Displays a simple warning dialog with the given message.
void Hv_WorldPoint2ToLocalRect(Hv_View, Hv_FPoint *, Hv_FPoint *, Hv_Rect *, short);	Converts two world points to a local rectangle.
void Hv_WorldPointToLocalPoint(Hv_View, Hv_FPoint *, Hv_Point *);	Converts a world point to a local point.
void Hv_WorldPolygonToLocalPolygon(Hv_View View, short npts, Hv_Point *poly, Hv_FPoint *wpoly);	Converts a world point to a local point.
void Hv_WorldRectToLocalRect(Hv_View View, Hv_Rect *area, Hv_FRect *wr);	Converts a world rect to a local rect.
void Hv_WorldToLocal(Hv_View View, float fx, float fy, short *x, short *y);	Converts a set of world coordinates to local coordinates..

TABLE 17. Alphabetical listing of public functions available in $H^{\underline{V}}$.

Appendix 1 Data Structures

Before launching into a discussion of Hv data structures, a short philosophical interlude is in order. These days it is politically incorrect for the developer to know details of data structures, "information-hiding" demands instead that all access occur through interface functions. We weakly support this notion and provide some "get" and "put" routines to preclude the need for direct access. However, since Hv is in its infancy and we are hoping for substantive user contributions, and furthermore since it will illuminate some later discussion, we will present here *some* explicit data structures. (Of course one can simply look at the Hv header file **`Hv.h`** to examine any and all data structure definitions).

A1.1 Synonyms to X data types

In a few cases, we simply give some of the types from X and Motif an Hv name for purely aesthetical uniformity purposes:

X Name	Hv Name
Region	Hv_Region
XPoint	Hv_Point
XSegment	Hv_Segment
XtPointer	Hv_Pointer
Pixmap	Hv_Pixmap
XmTextPosition	Hv_TextPosition
Widget	Hv_Widget
XtIntervalId	Hv_IntervalID

TABLE 18. Providing a few X and Motif types with Hv synonyms.

A1.2 Simple Geometric Types

In addition to the pixel-based **`Hv_Point`**, we need a floating point analog **`Hv_FPoint`** for use in the world system:

```
typedef struct
{
  float  h;   /* horizontal coordinate (in world coordinates) */
  float  v;   /* vertical coordinate (in world coordinates) */
}  Hv_FPoint;
```

In terms of rectangles, we have made a radical departure from **`XRectangles`**, in that in addition to the left, top, width and height of the rectangle, we also store the right and bottom. This is, of course, redundant. However we found that for efficient coding of rectangle routines it is sometimes convenient to have one set of parameters (left, top, width, height) and sometimes convenient to have the other (left, top, right, bottom). Thus we traded the (small) additional space required for the CPU savings of not needing repeated calculations of the form `right = left + width`.

To be fair, we also introduced the danger that a given rectangle is internally inconsistent. Use of H$\underline{^V}$ routines for creating and manipulating rectangles ensures that this will not happen. At any rate, here are the definitions for pixel-based **`Hv_Rect`** and its world-based cousin **`Hv_FRect`**:

```
typedef struct
{
  short   left;                    /* Left pixel coordinate */
  short   top;                     /* Top pixel coordinate */
  short   right;                   /* Right pixel coordinate */
  short   bottom;                  /* Bottom pixel coordinate */
  short   width;                   /* right - left */
  short   height;                  /* bottom - top */
}  Hv_Rect;

typedef struct
{
  float   xmin;                    /* xmin (left) */
  float   ymin;                    /* ymax (top) */
  float   xmax;                    /* xmax (right) */
  float   ymax;                    /* ymin (bottom) */
  float   width;                   /* xmax - xmin */
  float   height;                  /* ymax - ymin */
}  Hv_FRect;
```

A1.3 Strings

We have defined a type **`Hv_String`**:

```
typedef  struct
{
short         font;                /* font */
char          *text;               /* for items using text */
short         strcol;              /* string color */
short         fillcol;             /* background fill color */
Boolean       horizontal;          /* orientation flag */
}  Hv_String;
```

`font` is one of the predefined fonts. **`text`** is the actual array of characters. **`strcol`** (the color of the text itself) is one of the predefined H$\underline{^V}$ colors, such as **`Hv_black`**, as is **`fillcol,`** the color of the background of the string. The **`horizontal`** field determines whether the text is horizontal or vertical, such as on a y axis of a plot.

A1.4 Mouse Button Events

As in X, but in a simpler manner, mouse button events in H$\underline{^V}$ are passed to callback routines. The structure of an H$\underline{^V}$ event is:

```
typedef  struct  hvevent *Hv_Event;

typedef  struct  hvevent

{
  Hv_View    view;                 /* view where click occurred */
```

```
  Hv_Item    item;              /* item clicked(or null) */
  Hv_Point   where;             /* location of click */
  int        modifiers;         /* modifiers */
} Hv_EventData;
```

Note that in this case **`Hv_Event`** is a pointer type. The first three fields are self-explanatory. The last field, **`modifiers`**, can be used to see if, for example, the shift button was held down when the click occurred. This will be discussed in detail later.

A1.5 Function Pointers

H$\underline{^{\mathrm{V}}}$ defines a single function pointer type, **`Hv_FunctionPtr`**, used ubiquitously:

```
typedef void (*Hv_FunctionPtr)();      /*pointer to function returning a void*/
```

A1.6 Color Schemes

`Hv_ColorScheme` is a structure used by some items to create a 3D “sculptured” appearance. It is defined as:

```
typedef struct
{
short       fillcolor;          /* Interior area fill color*/
short       topcolor;           /* Color for left top sides */
short       bottomcolor;        /* Color for right bottom sides */
short       lighttrimcolor;     /* Trim color for topcolor */
short       darktrimcolor;      /* Trim color for bottommolor*/
short       hotcolor;           /* for fill when selected, etc */
short       borderwidth;        /* Pixel Width of border*/
short       framewidth;         /* Pixel Width of Frame (3D effect part)*/
} Hv_ColorScheme;
```

A1.7 Hv_Interval Structure

These are used when items are animated by being animated at intervals.

```
typedef struct hvinterval
{
  unsigned long    interval;           /* used in timed redraws */
  Hv_IntervalID    intervalid;         /* used in timed redraws */
  unsigned long    timedredrawcount;   /* used in timed redraws */
} Hv_Interval;
```

A1.8 Hv_View Data Structure

Here is the structure for a view, which is basically a Macintosh likewindow on our main Motif window.

```
typedef struct view
{
  Boolean          enabled;            /* if TRUE, then can “click” items on view*/
  short            tag;                /* for IDing  this view*/
  short            reserved;           /* not presently used */
  short            minwidth;           /* minimum pixel width  of this view */
  short            minheight;          /* minimum pixel height of this view */

  int              modifiers;          /* to remember modifier clicks */
  int              drawcontrol;        /* holds drawing options (bitwise) */
  int              userdrawcontrol;    /* another one for user defined options */
```

```
  int              popupcontrol;       /* holds which popup items are active */

  Hv_FRect         *world;             /* world coordinate system*/
  Hv_FRect         *defaultworld;      /* default world coordinate system */
  Hv_Rect          *local;             /* local pixel coordinate system*/
  Hv_Rect          *previouslocal;     /*  prev loc system, to toggle with full screen */
  Hv_Rect          *hotrect;           /* holds the “hot” for zooming and scrolling*/
  Hv_Item          containeritem;      /* holds a pointer to the container data */
  Hv_Item          feedbackitem;       /* holds a pointer to the feedback data */
  Hv_Item          hotrectborderitem;  /* holds a pointer to the feedback data */
  Hv_Item          brickwallitem;      /* unmovable, no item may move in front of it */

/* auxillary functions */

  Hv_FunctionPtr   customize;          /* user defined customization via popup menu */
  Hv_FunctionPtr   initialize;         /* function for init. and creating item lists */
  Hv_FunctionPtr   feedback;           /* function called to update feedback */
  Hv_FunctionPtr   userdraw;           /* specialized drawing AFTER item list is drawn */
  Hv_FunctionPtr   destroy;            /* to destroy extra data */

  float            minzoomw, minzoomh; /* cannot zoom to a world w or h smaller than this */
  float            maxzoomw, maxzoomh; /* cannot zoom to a world w or h bigger  than this */
  Hv_ItemList      items;              /* llinked list of item’s controlled by this view */

  Widget           menu;               /* corresponding menu item in Views menu */
  void             *data;              /* for any purpose*/

  Hv_ScrollInfoPtr  hscroll;           /* for horizontal scrolling*/
  Hv_ScrollInfoPtr  vscroll;           /* for vertical scrolling*/

  Hv_View          next;               /* next view in linked list*/
  Hv_View          prev;               /* previous view in linked list*/

/* connection stuff */

  Boolean          editconnections;    /* permit individual editing*/
  Boolean          drawconnections;
  Boolean          drawarrowheads;
  Hv_ConnectionNodePtr  connections;   /* list of items connected to each other*/

/* simulation stuff */

  Hv_FunctionPtr   simproc;            /* function called when sim timer times out*/
  unsigned long    siminterval;        /* interval until simproc is called again*/
  Hv_IntervalID    simintervalid;      /* id of the timer event*/
  void             *mapdata;           /* points to Hv_ViewMapDataRec if view has maps */

  Hv_Pixmap        background;         /* for saving the background if requested */

  Hv_FunctionPtr   animationproc;      /* used in conjunction with savebg for anim*/

/* related to undo */

  Hv_Item          lastdragitem;
  Hv_Item          lastrotateitem;

  Hv_Point         lastdragdelta;
  float            lastazimuth;

/* off screen user draw */

  Hv_FunctionPtr   offscreenuserdraw;
}  ViewRecord;
```

A1.9 List-Head Data Structures

List-Heads are used for bookkeeping purposes: to maintain a pointer to the first and last node in a linked list.

```
/* here is the structure of the listhead “bookkeeeper” for lists of Hv_Items */

typedef struct hvitemlistdata        *Hv_ItemList;

typedef struct hvitemlistdata
{
```

```
  Hv_Item    first;       /* first item added to linked list */
  Hv_Item    last;        /* last item added to linked list */
} Hv_ItemListData;

/* here is the structure of the listhead “bookkeeeper” for lists of Views */

typedef struct view_listhead
{
  Hv_View    first;       /* first view added to linked list */
  Hv_View    last;        /* last  view added to linked list */
} View_ListHead;
```

A1.10 Radio Dialog Items

Radio dialog items have two associated data structures: An **Hv_RadioHead** that is basically the whole set of radio buttons, and **Hv_RadioList**s which are the data associated with individual nodes.

```
typedef struct  hvradiohead
{
  Hv_Widget           activewidget;  /* active (selected) button
  short               activepos;     /* corresponding position (starting at 1) */
  Hv_RadioListPtr     head;          /*first button (start of list) */
} Hv_RadioHead;
```

Here is the individual node structure, one for each button in the set (in the form of a linked list)

```
typedef struct hvradiolist  *Hv_RadioListPtr;

typedef struct hvradiolist
{
  Hv_Widget           radio;      /* the actual button */
  Hv_RadioListPtr     next;       /* next one in the list */
  void                *data;      /* user data associated with button */
  short               pos;        /* position (starting with 1) within the list */
} Hv_RadioList;
```

A1.11 Hv_Item Data Structure

Here is the structure for an item. The items drawn will be stored as a doubly linked list.New items are added at the end.

```
typedef struct item
{
  Hv_View          view;               /* view owner of this item */
  short            type;               /* i.e., rectangle, coil, line, etc */
  short            tag;                /* user sets this tag to quickly find the item */
  short            color;              /* use this for simple one color items */
  short            state;              /* for buttons, especially*/
  short            resizepolicy;       /* how a resize affects this item */
  short            domain;             /* where this items lives (hotrect or not hotrect */

  int              drawcontrol;        /* holds drawing options (bitwise) */
  int              user1;              /* for any purpose*/
  int              user2;              /* for any purpose*/

  float            azimuth;            /* relevant for rotatable items */

  Hv_Interval      *drawinterval;      /*used in timed redraw */
  Hv_Rect          *area;              /* pixel coordinates of item’s bounding rect */
  Hv_FRect         *worldarea;         /* wrect corresponding to area for HOTRECT items */
  Hv_Rect          *rotaterect;        /* used for complicated items -- often NULL */
  Hv_String        *str;               /* pointer to string stuff, if needed */
  void             *data;              /* generic pointer for private item specific data*/
```

```
  char            *balloon;           /* points to balloon help text */
  Hv_Item         next;               /* for making linked list */
  Hv_Item         prev;               /* for making linked list */
  Hv_ItemList     children;           /* an internal list of items contained in this one!
*/

  Hv_FunctionPtr  standarddraw;       /* standard drawing routine */
  Hv_FunctionPtr  singleclick;        /* single click callback */
  Hv_FunctionPtr  doubleclick;        /* double click callback */
  Hv_FunctionPtr  userdraw;           /* specialized drawing */
  Hv_FunctionPtr  initialize;         /* specialized initialization */
  Hv_FunctionPtr  customize;          /* specialized customization */
  Hv_FunctionPtr  destroy;            /* specialized destruction */
  Hv_FunctionPtr  afteroffset;        /* called after an offset */
  Hv_FunctionPtr  fixregion;          /* called to fix region */
  Hv_FunctionPtr  checkenlarge;       /*check for item enlargement*/

  Hv_ColorScheme  *scheme;            /* 3D color scheme for itemss that need it*/
  Hv_Item         parent;             /* if this is part of a sublist, this may be useful*/

  short           pattern;            /* background pattern */

}  Hv_ItemData;
```

Appendix 2 Global Variables and Constants

A2.1 Colors

Here is a table of predefined H^{V} colors, which are global variables of type **`short`**:

Hv_aliceBlue	Hv_aquaMarine	Hv_black	Hv_blue	Hv_brown
Hv_cadetBlue	Hv_chocolate	Hv_coral	Hv_cornFlowerBlue	Hv_darkBlue
Hv_darkGreen	Hv_darkSeaGreen	Hv_dodgerBlue	Hv_forestGreen	Hv_gray0
Hv_gray1	Hv_gray10	Hv_gray11	Hv_gray12	Hv_gray13
Hv_gray14	Hv_gray15	Hv_gray16	Hv_gray2	Hv_gray3
Hv_gray4	Hv_gray5	Hv_gray6	Hv_gray7	Hv_gray8
Hv_gray9	Hv_green	Hv_honeydew	Hv_lavender	Hv_lawnGreen
Hv_lightSeaGreen	Hv_maroon	Hv_mediumForestGreen	Hv_mistyRose	Hv_navyBlue
Hv_olive	Hv_orange	Hv_peachPuff	Hv_peru	Hv_powderBlue
Hv_purple	Hv_red	Hv_rosyBrown	Hv_royalBlue	Hv_seaGreen
Hv_skyBlue	Hv_slateBlue	Hv_steelBlue	Hv_tan	Hv_tan3
Hv_violet	Hv_wheat	Hv_white	Hv_yellow	

TABLE 19. Predefined colors.

A2.2 Fonts

Here is a table of predefined H^{V} fonts, which are global variables of type **`short`**:

Hv_courier11	Hv_courier14	Hv_courier17	Hv_courier20	Hv_courier25
Hv_helvetica11	Hv_helvetica14	Hv_helvetica17	Hv_helvetica20	Hv_helvetica25
Hv_times11	Hv_times14	Hv_times17	Hv_times20	Hv_times25
Hv_symbol11	Hv_symbol14	Hv_symbol17	Hv_symbol20	Hv_symbol25
Hv_courierBold11	Hv_courierBold14	Hv_courierBold17	Hv_courierBold20	Hv_courierBold25
Hv_helveticaBold11	Hv_helveticaBold14	Hv_helveticaBold17	Hv_helveticaBold20	Hv_helveticaBold25
Hv_timesBold11	Hv_timesBold14	Hv_timesBold17	Hv_timesBold20	Hv_timesBold25
Hv_fixed	Hv_fixed2			

TABLE 20. Predefined fonts.

A2.3 Patterns

Here is a table of predefined $H^{\underline{V}}$ patterns. Note that patterns (and symbols) are defined as macros rather than global variables (as are colors and fonts):

Hv_FILLSOLIDPAT	Hv_80PERCENTPAT	Hv_50PERCENTPAT	Hv_SCROLLPAT	Hv_20PERCENTPAT
Hv_HATCHPAT	Hv_DIAGONAL1PAT	Hv_DIAGONAL2PAT	Hv_HSTRIPEPAT	Hv_VSTRIPEPAT
Hv_WEAVEPAT	Hv_SCALEPAT	Hv_STARPAT		

TABLE 21. Predefined patterns.

A2.4 Symbols

Here is a table of predefined $H^{\underline{V}}$ symbols Note that symbols (and patterns) are defined as macros rather than global variables (as are colors and fonts):

Hv_BIGPOINTSYMBOL	Hv_BLOCKHSYMBOL	Hv_BLOCKISYMBOL	Hv_CIRCLESYMBOL
Hv_CROSSSYMBOL	Hv_DEFOCUSSYMBOL	Hv_DIAMONDSYMBOL	Hv_DOWNARROWSYMBOL
Hv_FOCUSSYMBOL	Hv_HEXAGONSYMBOL	Hv_LEFTARROWSYMBOL	Hv_NOSYMBOL
Hv_POINTSYMBOL	Hv_RECTSYMBOL	Hv_RIGHTARROWSYMBOL	Hv_TRIANGLEDOWNSYMBOL
Hv_TRIANGLEUPSYMBOL	Hv_UPARROWSYMBOL		

TABLE 22. Predefined Symbols.

A2.5 Global Variables

We conclude with a list some of the global varaibles that may be of interest to developers.

Global Variable	Type	Comment
Hv_AfterItemRotate	Hv_FunctionPtr	Optional routine called after an item is rotated.
Hv_CustomizeQuickZoom	Hv_FunctionPtr	Routine used to set quickzoom labels (See “Adding Quick Zooms” on page 50.)
Hv_QuickZoom	Hv_FunctionPtr	Routine called when a quickzoom is selected.
Hv_TileDraw	Hv_FunctionPtr	Routine called to tile the main window.
Hv_UserDefaultWorld	Hv_FunctionPtr	Optional routine called after a View’s default world is restored.
Hv_UserMove	Hv_FunctionPtr	Optional routine called aftera View is moved.
Hv_UserResize	Hv_FunctionPtr	Optional routine called after a View is resized.
Hv_UserZoom	Hv_FunctionPtr	Optional routine called after a view is zoomed.

TABLE 23. Partial list of global variables.

Global Variable	Type	Comment
Hv_actionMenu	Hv_Widget	The standard Action menu.
Hv_bottomRightCursor	Cursor	The cursor used when resizing the view from the bottom right corner.
Hv_completeFeedback	Boolean	When False, H^V is asking your app to do minimal feedback
Hv_crossCursor	Cursor	The cursor used when the pointer is inside the HotRect.
Hv_defaultCursor	Cursor	The standard arrow cursor.
Hv_dialogAnswer	int	Dialog result. Hv_OK, Hv_CANCEL, Hv_DONE or Hv_APPLY
Hv_dotboxCursor	Cursor	The cursor used when a drawing tool is active.
Hv_drawingOffScreen	Boolean	If True, H^V is presently drawing onto an off screen buffer.
Hv_drawingOn	Boolean	If set to False ALL DRAWING IS DISABLED.
Hv_extraFastPlease	Boolean	H^V sometimes sets this variable to encourage you to draw faster. For example, if you have a complicated item with its Hv_FANCYDRAG draw control bit set, H^V will try to draw the item as it is being dragged (as opposed to just its area). You may want to simplify the item's drawing in such a case.
Hv_helpMenu	Hv_Widget	The standard **`Help`** menu.
Hv_hotItem	Hv_Item	If not NULL, it's the item that has be "clicked" most recently.
Hv_hotView	Hv_View	If not NULL, it's the item that has be "clicked" most recently.
Hv_inPostscriptMode	Boolean	If True, H^V is creating a postscript document.
Hv_lastLabel	Hv_Widget	Remebers the last dialog label created.
Hv_offsetDueToDrag	Boolean	True if the reason that an item is being offset is that it was dragged.
Hv_offsetDueToScroll	Boolean	True if the reason that an item is being offset is that a scroll occurred.
Hv_simpleTextBg	short	Background used in "Simple" dialog item convenience creation.
Hv_simpleTextBorder	short	Border used in "Simple" dialog item convenience creation.
Hv_simpleTextFg	short	Foreground used in "Simple" dialog item convenience creation.
unsigned int	Hv_tileX	Badly named width of the nain window tile.
unsigned int	Hv_tileY	Badly named height of the nain window tile.
Hv_timedRedraw	Boolean	If True, the item is being redrawn NOT because of an exposure but rather becaus its timer expired.
Hv_trapCommandArgs	Boolean	If True, unrecognized command line arguments are saved for the application to process (default: False).
Hv_version	int	100 x H^V version (e.g. 123 -> version 1.23)
Hv_viewMenu	Hv_Widget	The standard **`Views`** menu.
Hv_views	View_ListHead	A list of all the views in the application.
Hv_virtualView	Hv_View	The standard Virtual Desktop View.
Hv_waitCursor	Cursor	A clock cursor.
Hv_welcomeView	Hv_View	The standard Welcome View.
*Hv_hostName	char	The host name where the application is reunning.
*Hv_leftOverArgs	char	Unprocessed command line arguments (assuming Hv_trapCommandArgs was set)
Hv_quoteDelimitsSegment	Boolean	Used in the IO package. If True, text withing quotes is assumed to be one segment. Default is False.
*Hv_userName	char	The name of the user that launched the application.

TABLE 23. Partial list of global variables.