# The HAVi Specification

## Specification of the Home Audio/Video Interoperability (HAVi) Architecture, Chapter 8 – Level 2 User Interface

HAVi, Inc.

Version 1.1
May 15, 2001

# 8 HAVi Level 2 User Interface

This chapter provides a specification of the Home Audio/Video Interoperability Architecture User-Interface, (also called the HAVi User-Interface). This HAVi User-Interface is designed as a "TV-friendly" user-interface framework and is explicitly designed to be suitable for use and implementation on a variety of consumer electronic (CE) devices. The application programming interfaces (APIs) for the HAVi User-Interface are contained in the org.havi.ui and org.havi.ui.event packages described in Appendix A: HAVi Java APIs. In case of conflicts between the specification and the Java APIs, the Java APIs shall be the normative reference.

## 8.1 HAVi User-Interface Design (informative)

The HAVi User-Interface allows applications, written in Java, to determine the user interface capabilities of its host display device, accept input from the user, draw to the screen and play audio clips. It uses a subset of the AWT as defined in the Java 1.1 Core API (HAVi specification reference [8]) and extends this with packages and classes specific to the HAVi platform. This subset is supported in PersonalJava as defined in PersonalJava 1.1 specification (HAVi specification reference [9]).

### 8.1.1 Remote Control

The user input model from java.awt is extended to support an optional remote control. A large number of events are optional, allowing manufacturers to customize and add value to their products.

### 8.1.1 Television Specific Support

HAVi also adds classes to support graphics and video display functions that are available in typical television-based systems, including: support for non-square pixels, and graphics / video overlays.

## 8.1 java.awt Subset

Only a subset of the Java 1.1 java.awt package is required to be present on a HAVi platform. This subset is described within this section in more detail.

### 8.1.1 Required Elements from AWT

Since most of the widget set in the java.awt package is not "TV friendly", these classes are not required to be present in systems supporting the HAVi UI framework. TV friendly equivalents of these are provided through the HAVi User-Interface framework, which can be extended to support alternative look and feel. Classes of the java.awt package not included in this specification cannot be expected to be present in devices supporting the HAVi User Interface framework. Interoperable HAVi applications shall not make use of these classes. Where an application uses classes which fall outside of the scope of the HAVi specification, the behavior is not determined by this HAVi User Interface specification, rather it shall be determined by the implementation of the underlying platform. This specification does not prevent a manufacturer implementing a particular device using all of AWT, and any applications intended to execute solely in a particular device may exploit any classes or packages known to be in that device, but both the device and application shall not be regarded as interoperable and shall be considered to be proprietary in nature.

The specified set of classes have been chosen such that HAVi applications can implement any missing widget functionality using these classes.

- The main base classes, such as java.awt.Component, are required in order to build the HAVi widgets.

- Other classes, such as java.awt.Color and java.awt.Font, are required for all general drawing and painting.

- The layout classes, such as java.awt.FlowLayout and java.awt.BorderLayout are retained to provide flexible layout of components on various output devices.

The classes from java.awt that are listed in Table 1 are the classes that an HAVi application author can reliably interact with, and use within a HAVi compliant application. Interoperable applications must not use any references from classes in this list to classes not in this list.

**Table 1. java.awt Classes Available to Interoperable HAVi Applications**

| java.awt | java.awt.event | java.awt.image |
|---|---|---|
| Adjustable(intf) | ActionListener(intf) | ImageConsumer(intf) |
| ItemSelectable(intf) | AdjustmentListener(intf) | ImageObserver(intf) |
| LayoutManager(intf) | ComponentListener(intf) | ImageProducer(intf) |
| LayoutManager2(intf) | ContainerListener(intf) | ColorModel |
| MenuContainer (intf) | FocusListener(intf) | DirectColorModel |
| AWTError | ItemListener(intf) | IndexColorModel |
| AWTEvent | KeyListener(intf) | MemoryImageSource |
| AWTEventMulticaster | MouseListener(intf) | PixelGrabber |
| AWTException | MouseMotionListener(intf) | |
| BorderLayout | TextListener(intf) | |
| CardLayout | WindowListener(intf) | |
| Color | ActionEvent | |
| Component | AdjustmentEvent | |
| Container | ComponentAdapter | |
| Cursor | ContainerEvent | |
| Dimension | FocusAdapter | |
| Event | FocusEvent | |
| EventQueue | InputEvent | |
| FlowLayout | ItemEvent | |
| Font | KeyAdapter | |
| FontMetrics | KeyEvent | |
| Graphics | MouseAdapter | |
| GridLayout | MouseEvent | |
| IllegalComponentStateException | MouseMotionAdapter | |
| Image | PaintEvent | |
| Insets | TextEvent | |
| MediaTracker | WindowAdapter | |
| Point | WindowEvent | |
| Polygon | ComponentEvent | |
| Rectangle | ContainerAdapter | |
| Shape | | |
| Toolkit | | |

The classes in Table 1 are not necessarily sufficient to enable a full implementation of a HAVi compliant device, for example a device implementing the HAVi User-Interface could be implemented using Java 1.1, Personal Java , etc., which might require additional requirements on the implementation. The specification is intentionally silent on the mechanisms used to implement the Java environment for a HAVi implementation.

### 8.1.2 User Input Preference Interfaces

Personal Java 1.1 includes some interfaces which are not found in JDK 1.1 but are useful for a TV friendly user-interface API. The HAVi specification includes a number of interfaces intended to allow Java applications to adapt to mouseless environments like systems operated by remote control. These input preference interfaces allow component developers to specify how users can navigate among and interact with their components. They are only available to components which inherit from org.havi.ui.HComponent. The specification is intentionally silent on the mechanisms used to implement the Java environment for a HAVi implementation. An extra HAVi specific interface org.havi.ui.HAdjustmentInputPreferred is also included.

The org.havi.ui.HNoInputPreferred interface disallows user navigation, and hence actioning, etc.

A component that implements org.havi.ui.HNoInputPreferred indicates that the user may not navigate to this component. However, note that if a component which implements this interface is extended, so that the sub-classed component may implement another "XxxInputPreferred" interface, then in all cases, this other interface may take precedence. In contrast, the method isFocusTraversable shall always return true for components implementing the interfaces org.havi.ui.HActionInputPreferred, org.havi.ui.HAdjustmentInputPreferred, org.havi.ui.HKeyboardInputPreferred, org.havi.ui.HNavigationInputPreferred and org.havi.ui.HSelectionInputPreferred.

The org.havi.ui.HKeyboardInputPreferred interface indicates that it is intended to accept component specific keyboard input from the user. Platforms without keyboards may provide another means for generating such input when this component is edited, for example, by offering an on-screen keyboard.

The org.havi.ui.HActionInputPreferred interface indicates that it is intended to be actioned by the user.

The org.havi.ui.HAdjustmentInputPreferred interface indicates that it is intended to offer increment and decrement functionality to the user.

The org.havi.ui.HNavigationInputPreferred interface indicates that it is intended to offer focus traversal between org.havi.ui.HComponents to the user.

The org.havi.ui.HSelectionInputPreferred interface indicates that it is intended to offer selection and deselection to the user.

## 8.2 HAVi Extensions to AWT

### 8.2.1 General API Issues

In this package, passing null to a method or constructor shall generate a java.lang.NullPointerException except in the following circumstances :

- Where null is explicitly documented as being an allowed parameter
- Where the class where the method or constructor is defined inherits from java.util.EventObject or java.lang.Exception

It is an allowable option to override protected, public and package level methods in HAVi using the standard java inheritance conventions.

### 8.2.2 User Input

Java Applications in HAVi can accept input from a keyboard, a mouse or a remote control. The keyboard and mouse inputs are supported by functions in the java.awt and java.awt.event packages.

Remote control input is provided with classes in the org.havi.ui.event package. The org.havi.ui and org.havi.ui.event packages include classes that allow the application to determine the user-input capabilities of the platform on which the application is running.

### 8.2.2.1 Remote Control Support

The HAVi remote control classes are extended from the java.awt.event key event classes. All of the events that are added for the remote control are optional. The remote control keys fall into two categories: colored keys and dedicated keys. The intention of these keys is to provide the user direct access to various functions; however, the platform *may* implement a virtual (on-screen) mechanism to generate these events, but shall take care in this case not to hide the application. Note that it is an implementation option if (remote control) key events are repeated.

#### 8.2.2.1.1 Remote Control Colored Keys

Up to six colored soft keys can be included on a remote control. These are optional, and are to be identified with a color. If implemented, these keys are to be oriented from left to right, or from top to bottom in ascending order. The application can determine how many colored keys are implemented, and what colors are to be used, so that the application can match the controls.

The following identifiers are available for colored key events: VK_COLORED_KEY_0, VK_COLORED_KEY_1, VK_COLORED_KEY_2, VK_COLORED_KEY_3, VK_COLORED_KEY_4, VK_COLORED_KEY_5.

#### 8.2.2.1.2 Remote Control Dedicated Keys

The org.havi.ui.HRcEvent class defines a number of dedicated remote control events that can be used by applications. Although none of the events in the org.havi.ui.event.HRcEvent class are required to be implemented, events for power (VK_POWER), volume up and down (VK_VOLUME_UP and VK_VOLUME_DOWN), and channel up and down (VK_CHANNEL_UP and VK_CHANNEL_DOWN) are highly recommended.

However, whilst the dedicated remote control events are themselves device independent, the precise set of dedicated keys that is implemented is device dependent. The org.havi.ui.event.HRcCapabilities class enables an application to discover which events are implemented and how these are to be labeled to match the platform implementation.

### 8.2.2.2 Keyboard

HAVi supports keyboards via the java.awt.event package. The events supported on a keyboard can be determined by the org.havi.ui.event.HKeyCapabilities class.

Note that systems that do not include a physical keyboard can check each component to see if it implements org.havi.ui.HKeyboardInputPreferred. If this interface is implemented, the system may enable user input of alphanumeric key events, for example, via a “soft” on-screen keyboard.

### 8.2.2.3 Mouse

Mouse support is optional. The presence of a mouse can be detected with the org.havi.ui.event.HMouseCapabilities class.

Mouse functionality is provided by the java.awt.event package. HAVi applications must be written in such a way that a free roaming cursor is not required for correct operation. This does not mean that a HAVi application could not implement, e.g. a drawing program, but rather that the user should not be able to put the application into a state that cannot be exited without a mouse. (A user-friendly drawing package would also notify the user that a mouse is required to use this application properly.)

#### 8.2.2.4 User Input Capabilities

Three classes are available to determine the capabilities of the user input for a given platform: org.havi.ui.event.HKeyCapabilities, org.havi.ui.event.HMouseCapabilities, org.havi.ui.event.HRcCapabilities. Each of these classes includes a method called getInputDeviceSupported, which returns true if the particular device is known to be available.

#### 8.2.2.5 User Input Representation

The org.havi.ui.event.HRcCapabilities class includes a method called getRepresentation, which returns an object of type org.havi.ui.event.HEventRepresentation. This class defines an event as having a known representation as a string, color or symbol, or having no supported representation. The particular text, color, or symbol can be determined by calling getString, getColor or getSymbol respectively. This allows an application to describe a button on an input device correctly for a given platform. All available events should have a text representation from getString.

The six colored key events (VK_COLORED_KEY_0 -- VK_COLORED_KEY_5), if implemented, must also be represented by a color – the getColor method returns a java.awt.Color object.

Key events may also be represented as a symbol – if the platform does not support a symbolic representation for a given event, then the application is responsible for rendering the symbol itself. Application rendering of keys without a symbolic representation, but with a commonly known representation, should follow the guidelines as defined in the Javadoc definition of the class.

### 8.2.3 Graphics Devices and Configurations

#### 8.2.3.1 Background

There are some specialized requirements for running applications within a consumer electronic environment, rather than the simpler situation that occurs when an Applet is displayed within a web-browser. Most notably the screen dimensions and aspect ratios are significantly different between PCs and CE devices. In the current on-screen display (OSD) graphics model of today's set-top box units, video may be output in a number of different configurations, e.g. traditional 4:3 TV display, or 16:9 widescreen TV displays, etc. The graphics resolution and aspect ratio are often locked to the video resolution and aspect ratio. If the video aspect pixel ratio changes then the graphics pixel aspect ratio may also change. Thus, there are requirements to:

- Determine the resolution and physical characteristics of the current display device.
- Detect modifications to the resolution and physical characteristics of the current display device.

#### 8.2.3.2 The HAVi Screen Reference Model

HAVi provides a model for the video output from a consumer electronics device. Instances of the class HScreen represent each independent final video output signal from a device. Each independent final video output signal is made up from the sum of graphics devices, video devices and backgrounds. These are represented by instances of the classes HGraphicsDevice, HVideoDevice and HBackgroundDevice respectively. All of these classes inherit from a common parent class - HScreenDevice.

The HAVi User-Interface specification provides limited support for applications to be displayed so that they are split across multiple concurrent display devices – the HSceneFactory class allows the HGraphicsDevice to be specified in the HSceneTemplate used to generate the HScene's for the application.

#### *8.2.3.3 The HAVi Screen Device Discovery Classes*

HAVi defines a means to allow applications to discover the range of display devices available. The model followed by HAVi is based on the model used in Java2 as described by the following three classes in the java.awt package - GraphicsDevice, GraphicsConfiguration and GraphicsConfigTemplate. In HAVi, this model is generalized to apply to video devices and to background devices.

##### *8.2.3.3.1 Querying the Configuration of a Display Device*

For each display device class (HVideoDevice, HGraphicsDevice and HBackgroundDevice), there are classes whose name ends in “Configuration” which represent distinct possible configurations of a single device. Applications may obtain a list of all possible configurations of a particular device. Applications may also obtain the current configuration using the getCurrentConfiguration method. Subject to security and resource management issues, applications may also set the configuration of a device using methods found on each device class.

Applications that are interested in a particular configuration of a device can request configurations matching a specific set of constraints. The first step in this process is to construct objects whose name ends in “ConfigTemplate”. Instances of these classes can then be populated with the properties by the application and then used to request a configuration supporting those properties. Properties can also have priorities attached to allow applications to express whether support for that property is required by the application, whether support for that property is only preferred by that application, whether support for that property is required to be absent or whether support for that property is preferred to be absent. In some cases, properties such as PIXEL_ASPECT_RATIO require extra information. This extra information can be provided as part of the method used to add the property to the configuration template.

The Configuration for a Device can be acquired, using the getCurrentConfiguration method. A description of this Configuration can be obtained using the getConfigTemplate method that yields a ConfigTemplate that uniquely identifies the given Configuration. Individual properties in this ConfigTemplate can then be examined using the getPreferencePriority and getPreferenceObject methods – features that are implemented will return REQUIRED, features that are not implemented will return REQUIRED_NOT. Values of some properties may also be obtained through a limited set of query methods provided on HScreenConfiguration.

##### *8.2.3.3.2 Compatibility with Existing java.awt Methods*

The java.awt.Toolkit.getScreenSize method shall be equivalent to the pixel resolution of the current configuration of the default screen device returned by HScreen.getDefaultGraphicsDevice.

The value of the java.awt.Toolkit.getScreenResolution method is implementation specific. This method shall not be used by applications.

Where the screen aspect ratio is unknown (such as in the case where a set-top box is connected to an analog display), the default aspect ratio is 4:3. In the case where an analog monitor is used with a HAVi compliant set-top box the resolution returned shall be based on the raster of the set-top box, ignoring any interpolation or other processing that may be present in the monitor.

The java.awt.Toolkit.getNativeContainer method shall return null; interoperable applications should not rely on this method.

Where an input parameter to a method call is specified to be more restrictive than its Java type allows (e.g. only a restricted set of numbers are allowed as inputs), providing values outside the allowed range shall result in a java.lang.IllegalArgumentException being thrown.

#### *8.2.3.4 Detecting Configuration Changes on a Display Device*

It is important for CE devices to be able to detect variations in their settings, since they may be subject to “on-the-fly” modifications of these settings, for example, they may be heavily influenced by the nature of some input video streams. Hence, the HScreenDevice class provides support for detecting when its configuration (settings) have been changed, using the HScreenDevice.addScreenConfigurationListener methods and the HScreenConfigurationListener and HScreenConfigurationEvent classes.

When an HScreenDevice's configuration is modified, then an HScreenConfigurationEvent is generated. Note that after a HScreenConfigurationEvent is obtained any HScreenConfiguration (or HScreenConfigTemplate) associated with that HScreenDevice must be reacquired to obtain the current settings for the device.

In general, a modification to the HScreenDevice might require that the displayed user-interface be modified, e.g. if the resolution has changed, or the pixel aspect ratio has been modified.

#### *8.2.3.5 Emulated Display Devices*

The HAVi User-Interface introduces extra sub-classes that are used to indicate that a device may perform emulations of other device capabilities:

- HEmulatedGraphicsDevice
- HEmulatedGraphicsConfiguration

Instances of these classes can be returned by the same methods that would return the corresponding class without “Emulated” in the class name. Returning the sub-class indicates that the implementation is emulating the requested configuration on one of its actual supported configurations. The class HEmulatedGraphicsConfiguration includes methods to allow applications to compare the configuration being emulated and the actual underlying configuration being really used. The extent of support for emulated configurations is a profile issue. All possible emulated configurations are not required, or guaranteed to be supported. Emulated configurations may have a significant performance penalty with respect to those supported natively on the device.

##### *8.2.3.5.1 Mapping from Authoring to Device Coordinates*

A special case of device emulation is the emulation of various graphics coordinate systems on a single physical device. The HAVi User-Interface provides mechanisms that allow devices to perform such emulations, e.g. by down-sampling a high-resolution system to match the limitations of a standard definition display. Thus, authors can rely on seamless mapping between authoring and device coordinates by the use of the HEmulatedGraphicsDevice class. Authors may determine an appropriate graphics device and request the best configuration that matches their requirements, as in a standard device discovery mechanism – or examine configurations themselves (both emulation and implementation) to determine appropriate settings. The extent to which devices are required to support emulation of other coordinate systems is profile dependent.

#### 8.2.3.6 Integrating HAVi Video Support into Platforms

The HAVi specification includes several classes to represent video in the user interface system. This representation of video devices only includes the display of video. The setup of the video decoder and the video pipeline is not included in this specification.

The class HVideoComponent is intended to be returned by a platform specific controller for video. In platforms based on the Java Media Framework, the Player.getVisualComponent method shall return objects of this class. The class HVideoDevice provides two hooks to platform specific APIs for this setup, the methods getVideoController and getVideoSource. On platforms based on the Java Media Framework (JMF), the getVideoController method shall return a JMF Player. The getVideoSource method shall return a platform specific class encapsulating a reference to the source of the video. Possible examples of the class to be returned here could include java.net.URL or javax.media.MediaLocator.

#### 8.2.3.7 Backgrounds

The HAVi specification includes several classes to represent the background of a screen, i.e. the area that is behind the running graphics and video and not covered by those. Using the same naming convention as video and graphics, these are called HBackgroundDevice, HBackgroundConfiguration and HBackgroundConfigTemplate. The basic HBackgroundConfiguration allows applications to control a single full screen background color.

The HAVi specification includes support for more sophisticated backgrounds - still images. Applications wishing to use these shall request an HBackgroundConfiguration supporting them by using the STILL_IMAGE property in an HBackgroundConfigTemplate. If this feature is supported by the platform concerned, when such a configuration is requested, an instance of the class HStillImageBackgroundConfiguration shall be returned. This class adds two extra methods, over the standard background configuration, which support the loading of background images. This loading is done through the HBackgroundImage class.

Using the HBackgroundImage class rather than the standard java.awt.Image allows for image formats that are decoded using hardware outside of the graphics system. One specific example of this is the decoding of still MPEG I frames using an MPEG video decoder. This is a commonly used feature in some devices since it provides good quality backgrounds without using software decoders or system memory. In systems where the same underlying MPEG video decoder can be used to decode both video and MPEG I frames, this decoder shall be represented both by an HVideoDevice instance and by an HBackgroundDevice instance for each application.  Where an HBackgroundDevice and an HVideoDevice both map onto the same underlying real resource in the device, the ZERO_VIDEO_IMPACT property in HBackgroundConfigTemplate shall be used to discover and limit any impact of one on the other.

- An application requesting a HBackgroundConfiguration using a HBackgroundConfigTemplate containing the ZERO_VIDEO_IMPACT property with priority REQUIRED shall only be returned one which would have absolutely no impact on any HVideoDevice if that HBackgroundConfiguration was set for its HBackgroundDevice. For example, only systems where a separate decoder is used for HBackgroundImages from video shall return a HBackgroundImage for such a template.

- An application requesting a HBackgroundConfiguration using a HBackgroundConfigTemplate containing the ZERO_VIDEO_IMPACT property with priority PREFERRED shall only be returned one which would have no permanent impact on any HVideoDevice if that HBackgroundConfiguration was set for its HBackgroundDevice. For example, systems where the same underlying hardware is used for decoding both video and HBackgroundImages but where once decoded, the HBackgroundImage is copied into a separate decoder memory from video and video decoding resumes, shall return an HBackgroundConfiguration in this case but not the previous case.

- Implementations where decoding of HBackgroundImages interrupts the video for the duration of the still image shall not support any HBackgroundConfiguration where ZERO_VIDEO_IMPACT is either REQUIRED or PREFERRED.

On implementations where the same underlying MPEG video decoder is used for both video and HBackgrounds, the most recent request of an application shall always be granted where the single underlying decoder is already being used by that application.

#### *8.2.3.8 Control of Screen Configurations*

The HAVi specification is silent about whether a single display device is shared between multiple applications or not. For the case where a display device may be shared between applications, it provides a mechanism for applications to assert control over the right to change the configuration of the display device. The HScreenDevice class includes methods to allow applications to reserve and release the right to control this configuration. It also allows an application to register and remove listeners for events that are generated when the applications reserve and release this right.

Applications wishing to be able to control the configuration of an HScreenDevice must define a class implementing the ResourceClient interface and pass an instance of this class to the reserveDevice method of the HScreenDevice that they wish to control. If the reserveDevice method succeeds then the application obtains control over the device configuration. When an application calls the HScreenDevice.getClient method this will return the ResourceClient passed in to the last call to the reserve method on that HScreenDevice instance.

Where there is a conflict between applications, this specification includes a mechanism to allow the platform to arbitrate between conflicting applications. The policy for this arbitration is intentionally not defined in this specification. When it is decided to remove the right to control a screen from an application, this is notified through the ResourceClient interface, the notifyRelease method will always be called. The requestRelease method will only be called when the existing owner of the resource and the application requesting the resource are authenticated to have a secure relationship of some form. This specification is silent about the details of this authentication.

### 8.2.4 Graphics and Video Integration

#### *8.2.4.1 Configurations*

The HAVi specification allows applications to express the relationship between video, graphics and backgrounds. The method HScreen.getCoherentScreenConfigurations allows applications to express a common set of constraints for video, graphics and backgrounds and get back a coherent answer.

In addition to this, there are several means to express constraints between video and graphics. These can be used for applications which already have running video to fit a graphics configuration to that video or which have already running graphics to fit video to that graphics. In HGraphicsConfigTemplate, the constant VIDEO_MIXING allows applications to request configurations where graphics is superimposed above video but without any requirement for pixels to be aligned. In HScreenConfigTemplate, there are constants to allow applications to ask for configurations as follows:

- VIDEO_GRAPHICS_PIXEL_ALIGNED - video & graphics pixels are the same size and aligned
- ZERO_VIDEO_IMPACT - a new graphics configuration must not change the existing video configuration
- ZERO_GRAPHICS_IMPACT - a new video configuration must not change the existing graphics configuration

#### 8.2.4.2 *Coordinate Spaces*

The HAVi specification includes a normalized screen coordinate system that represents the coordinates on the screen as floating point numbers between zero and one. This coordinate system is not pixel based. Such a non-pixel-based coordinate system enables the following:

- meaningful  results, even when the graphics configuration has not been determined
- meaningful results when presented video does not have a java.awt component.
- meaningful results when the video display and the graphics display are not necessarily aligned / share the same origin / share the same resolution, etc.

This screen-based coordinate system is encapsulated in the HScreenRectangle and HScreenPoint classes. This specification is silent about conversion between normalized and video coordinates. This should be addressed as part of the API providing support for control of video.

For graphics, these conversion mechanisms are found on the HGraphicsConfiguration class, since the conversion from screen to graphics coordinates is dependent on the current graphics device settings (Configuration) – especially if e.g. the graphics resolution can be varied independently of the video resolution, etc.

The HScreenRectangle mechanisms can be used to enable the alignment of (portions of) video and (portions of) graphics. The HScreenLocationModifiedListener and HScreenLocationModifiedEvent allow mechanisms to determine if the on-screen location of an HVideoComponent is modified (rather than its relative location within its enclosing container).

#### 8.2.4.3 *Transparency between Graphics and Video*

The HAVi specification includes support for applications to request transparency between graphics and video. This is provided by the getPunchThroughToBackgroundColor method on the HGraphicsConfiguration class. These methods provide a factory that enables applications to provide an opaque java.awt.Color and obtain a java.awt.Color supporting some form of transparency between graphics and video. These Color objects may be used in the drawing methods in the java.awt.Graphics class to cause video to appear in the graphics system.

### 8.2.5 HSceneFactory, HSceneTemplate and HScene

The HAVi User-Interface is deliberately agnostic concerning the implementation of a “coordinating” environment that provides the mechanism for a user to choose and run one or more applications. In

HAVi, this “coordinating” environment is known as a “home navigation shell”. Application writers cannot make any assumptions that their application GUI will always be immediately visible. For example, valid implementations of a coordinating environment might include:

- A simple “full-screen” view on a single application at any one time (with some undefined mechanism to switch between them).
- A multi-window system, where windows may obscure each other.
- A “paned” system where each application occupies an area on-screen – i.e. each application is always visible, but they may be resized if other applications are installed.

Thus, mechanisms are required to initiate an on-screen display and to indicate “user-interest”, or other modifications to the rendering area. These mechanisms should:

- Enable an application to request an area on-screen – however, given the possibility of differing styles of coordinating environment, an application cannot reasonably expect that its request will always be honored perfectly, and thus, a mechanism is required to indicate preferences for the application location “on-screen”.
- Indicate whether the current application is the one which the user is specifically interested in. For example, a “well-behaved” application which the user is not currently using might release, or reduce its consumption of any limited resources.
- Indicate to an application that its extent and position on-screen have been modified somehow by the home navigation shell.
- Allow an application to indicate to the system, that it requires the user’s attention. For example, the system may either indicate to the user that the user should choose the indicating application, or might simply automatically switch to that application.

### 8.2.5.1 Requesting an Area On-screen

#### 8.2.5.1.1 HSceneFactory and HSceneTemplate

The HSceneFactory is a factory class that is used to generate HScene objects. An application can indicate the location and dimensions of the HScene in the associated HSceneTemplate, although it is not guaranteed that the resulting HScene will necessarily match all of these preferences – since this is dependent on the implementation of the controlling shell and its associated policies, etc.

The application should call HSceneFactory.resizeScene if it wishes to re-size the HScene.

#### 8.2.5.1.2 HScene

An HScene is an HContainer representing the displayable area on-screen within which the application can display itself and thus interact with the user. However, HScene does not paint itself on-screen, only its added “child” components and hence there is no requirement to allocate “pixels” to the HScene directly – its only effect is to “clip” its child components. Hence, HScene may be regarded as a simple connection to the window management policy within the device, acting as a “screen resource reservation mechanism” denoting the area within which an application may wish to present a component, at some point in the future. Since an HScene is by definition not painted, i.e. it is effectively transparent, the area behind (all) HScene's in the z-ordering may be exposed by the platform as an HBackgroundDevice, and/or HVideoDevice's. However,  HAVi does not require platforms to provide such device capabilities, this is platform specific. The HScene semantics for transparency need to be specified exactly on a per-platform basis, for example, on some platforms an HScene might be transparent to other HScene's due to other separate applications.

For all interoperable applications, the HScene is considered the main top-level component of the application. No parent component to an HScene should be accessible to applications. Interoperable applications should not use the getParent method in HScene, since results are implementation dependent and valid implementations may generate a run-time error.

In terms of delegation, the HScene shall behave like a java.awt.Window with a native peer implementation, in that it will not appear to delegate any functionality to any parent object. Components which do not specify default characteristics inherit default values transitively from their parent objects. Therefore, the implementation of HScene must have valid defaults defined for all characteristics, e.g. Font, foreground Color, background Color, ColorModel, Cursor and Locale.

The HScene has a null LayoutManager by default – all widgets are placed using an X, Y co-ordinate, specified by the widget.

When created an HScene is not initially visible, and a call to setVisible is required to display the HScene (and also to hide it).

The application should call HSceneFactory.dispose if it wishes to destroy the HScene (and all of its currently added Components) and therefore release their associated resources for future garbage collection by the platform.

### 8.2.5.2 Modifications to the HScene: Focus and Resize events

The HScene object accepts java.awt.event.WindowEvent's, and interprets them as a java.awt.Window, however it is not required for the home navigation shell to generate all types of java.awt.event.WindowEvent.

Applications can use the java.awt.Component.requestFocus method on the HScene to indicate to the home navigation shell that the HScene should be receiving input focus. This request should be treated as a request to make the entire application visible and ready for user input, e.g. by expanding an icon, or changing the stacking order between competing overlapping applications. The decision as to whether or whenever the HScene (application) gains the input focus is entirely platform specific in terms of policy, etc. The java.awt.Component must be visible on the screen for this request to be granted – note that visibility in this context refers to whether the application has called the HScene.setVisible method, rather than any possible non-application-defined-behavior, due to the action of the coordinating shell hiding an application.

The java.awt.event.ComponentEvent's COMPONENT_MOVED and COMPONENT_RESIZED will be received by the HScene when the controlling shell has modified the position of the HScene or changed its dimensions on screen, respectively.

### 8.2.5.3 Application “user-interface” Lifecycle

- Outside the scope of the HAVi User-Interface:
    - The application is acquired by the platform.
    - The application is validated and security checked (possibly including authentication, byte-code verification, etc.).
    - The virtual machine is initialized, ClassLoader created, etc.
    - The application is executed

  - If the application does not require a user-interface, then it may continue as per normal.

- If a user-interface, and hence some screen resource is required, then the application traverses the HScreen, HGraphicsDevice, HGraphicsConfiguration space to determine an appropriate configuration, using HGraphicsConfigTemplate e.g. video-mixable, full-screen graphics, square pixel aspect ratio, resolution 1280 by 1024.

- The application configures the HGraphicsDevice appropriately, using the setGraphicsConfiguration method.

- The application requests that the HSceneFactory effectively grant it access to part of the screen for that device, using HSceneTemplate, e.g. full-screen display.

- The HSceneFactory returns an appropriate HScene container within which the application can display itself.

- The application uses the HScene container to add all of its components to make its user-interface.

- The application may take advantage of java.awt.WindowEvent's, to determine whether it has the user's (input) focus.

- The application may take advantage of the events COMPONENT_RESIZED and COMPONENT_MOVED, to determine when its HScene extent / location has been modified and to tailor its presentation accordingly.

- The application resizes the HScene by using HSceneFactory.resizeScene – if it wishes to resize the HScene, with the caveat that this may not be allowed by the external environment, e.g. due to window-manager policy, etc.

- The application terminates its on-screen presentation by calling the HScene.dispose method

- Outside of the scope of the HAVi User-Interface:
  - The application itself terminates.

### 8.2.6 Effects and Visual Composition using Component Mattes

#### *8.2.6.1 Component Mattes*

With org.havi.ui, the user interface is constructed from a set of components arranged in a hierarchy. The root of the hierarchy is an instance of HScene, leaf nodes are instances of HComponent and intermediate nodes are instances of HContainer. Components within a container are ordered from back to front. An example is shown in Figure 1, where *c3*, a container, is the back most component and *c1* the front most.

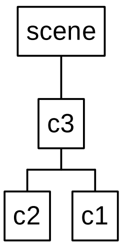


**Figure 1. Scene Hierarchy**

With the HMatte interface, the scene hierarchy can be modified by the inclusion of mattes (additional alpha sources), potentially for each member, i.e.:

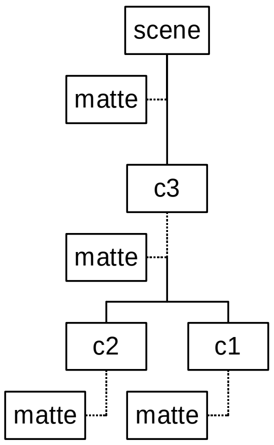


**Figure 2. Scene Hierarchy with Mattes**

The mattes influence the rendering of the scene, their operation can be visualized using a 2½D or layering model. The example below corresponds to the hierarchy in Figure 2 (for simplicity, the scene matte is not shown).

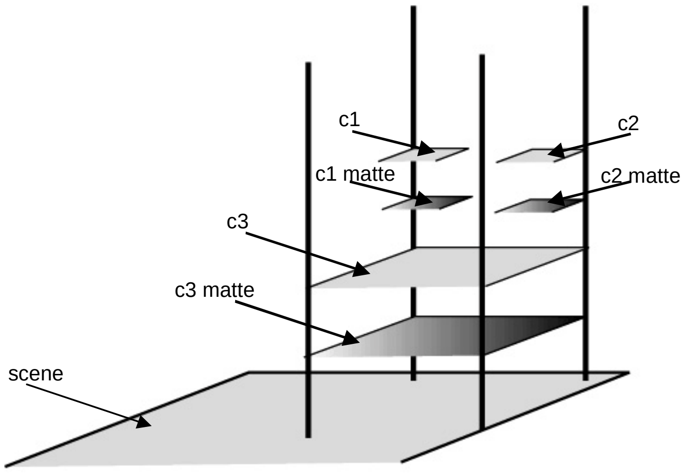


**Figure 3. Component Mattes**

Where pixels in a component already have an alpha value (e.g., from a PNG image), the alpha value from the component and the alpha value from the matte are multiplied together to obtain the actual alpha value to be used for that pixel.

### 8.2.6.2 Component Grouping

A container is either “grouped” or “ungrouped”. When a container is ungrouped, its matte only influences the appearance of those regions of the container not covered by members of the container (i.e., exposed regions of the container’s background). When a container is grouped, its matte influences the appearance of its background and all members of the container. For example, grouping a container and setting its matte to indicate 50% transparency will fade the container’s background and all members of the container. If it is ungrouped only the background will fade.

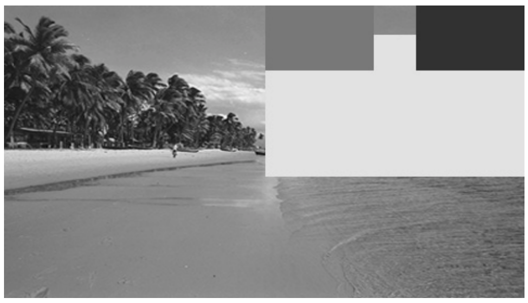
An HContainer may be rendered as follows:

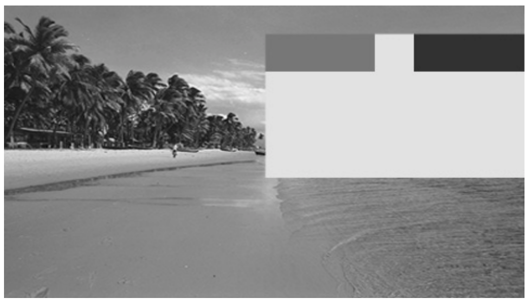
- If the container is ungrouped, the container’s background is first rendered and then composited with the container’s matte (i.e., the RGBA value of the container’s background is combined with the alpha value from the matte). Then, in back to front order, each member of the container is rendered, composited with its matte, and then composited with the container.

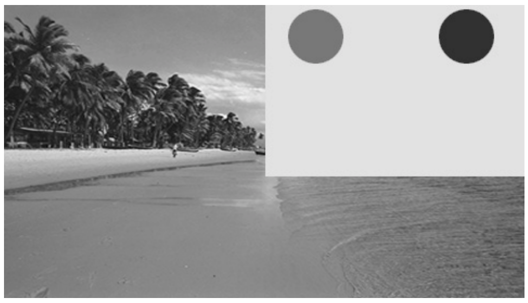
- If the container is grouped, the container’s background is first rendered. Then, in back to front order, each member of the container is rendered, composited with its matte, and then composited with the container. The result is then composited with the container’s matte.

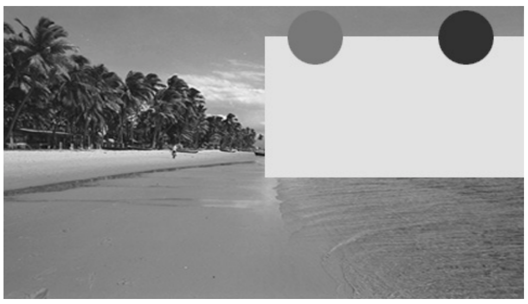
After an HContainer is rendered, it is composited with its parent. Compositing of an HScene is determined by the configuration of display devices.

### 8.2.6.3 Examples of Mattes and Component Composition


a) Bar matte for lower component.


b) As in a), with top components grouped to lower.


c) Circular mattes for top components.


d) As in c), with bar matte for lower component.

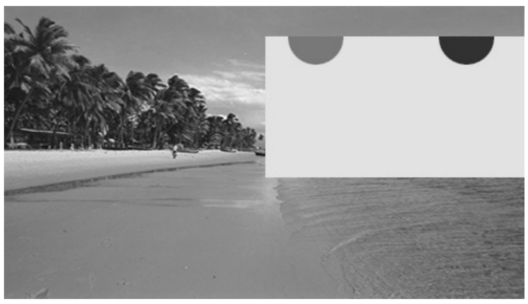

e) As in d), with top components grouped to lower.

**Figure 4. Visual Composition Examples**

### 8.2.6.4 Effects

A great variety of effects (e.g., wipes and fades) can be performed by using *matte animations* – sequences of mattes where the “active” element is changed over time. Matte animations can be combined with other techniques, such as component movement, to produce additional effects. The construction of matte animations is facilitated by the following classification of mattes:

HFlatMatte – the matte is constant over space and time, it can be specified by a float (0.0 is fully transparent and 1.0 fully opaque)

HImageMatte – the matte varies over space but is constant over time, it can be specified by an "image mask" (a single channel image) where the pixels indicate matte transparency

HFlatEffectMatte – the matte is constant over space but varies over time, it can be specified by a sequence of floats

HImageEffectMatte – the matte varies over space and time, it can be specified by a sequence of image masks

#### 8.2.6.5 Matte Sizes and Offsets

When a HImageMatte or HImageEffectMatte is assigned to a component, the associated image (or images) is by default aligned with the component so that their origins – the pixel at (0,0) – coincide. The offset of the matte with respect to the component can be altered using the setOffset method of HImageMatte and HImageEffectMatte. Regions of the component outside the matte (resulting from either a matte being smaller than the component, or from shifting the matte) are not matted.

## 8.3 HAVi Widget Framework

The HAVi widget framework is designed to allow maximum flexibility to implementers of applications. It also provides the necessary extensibility to allow the widget framework to be used as the basis for other application types, such as broadcast applications. By default, the HAVi widget framework only copies object references, and does not clone objects. Cases where objects are clone'd shall be marked explicitly.

### 8.3.1 HAVi Event Mechanism

The HAVi event mechanism is composed of the seven classes listed in Table 2:

**Table 2. HUI Events**

| Event | Use |
|---|---|
| HActionEvent | Interact with a component implementing the HActionInputPreferred interface |
| HFocusEvent | Interact with a component implementing the HNavigationInputPreferred interface |
| HRcEvent | Provide remote control event capabilty |
| HKeyEvent | Interact with a component implementing the HKeyboardInputPreferred interface |
| HAdjustmentEvent | Interact with a component implementing the HAdjustmentValue interface |
| HItemEvent | Interact with a component implementing the HSelectionInputPreferred interface |
| HTextEvent | Interact with a component implementing the HKeyboardInputPreferred interface |

These classes serve as the mechanism by which HAVi components inform each other of event occurrences. They are not intended to be generated from applications.

A HAVi widget must respond to these events in addition to other applicable user-input mechanisms. However, interoperable widgets must not respond to specific key codes received through the Java AWT KeyEvent mechanism.

HXXXEvents are generated and dispatched by the HComponent base class. For example, this class must intercept suitable Java key events and generate HKeyEvents from them. This means that widgets will receive two events - the original KeyEvent and a new HKeyEvent. Although it is possible to "discover" the platform-specific implementation of HXXXEvents via this mechanism, interoperable widgets may not use this information. Widgets may ignore the KeyEvent in favor of only handling the HKeyEvent.

### 8.3.2 Abstraction of “Feel”

In order to provide the necessary flexibility, the HAVi User-Interface widget framework is defined around a core of abstract *Component Behaviors*. These effectively define the functionality (or “feel”) of each widget which is derived from one of the Component Behaviors. Behaviors are defined for widget types having a number of states, which may be used to mimic the behavior of typical widgets.

In summary these Component Behaviors are:

- HVisible – Behavior providing basic display functionality.
- HNavigable– Behavior enabling widgets to receive navigational focus, and to define some kind of display change associated with focus change.
- HActionable– Behavior providing functionality to be invoked in response to an action.
- HSwitchable– Behavior allowing a widget to be actioned and to retain internal state information in addition to simple action behavior.
- HAdjustmentValue, HItemValue, HTextValue - Behaviors permitting the definition of widgets that return values to applications in response to user interaction.

Based upon these fundamental abstract Behaviors, all necessary HAVi functionality can be provided through derived concrete widgets, either for the provision of HAVi specific user-interfaces, or for HAVi specific widgets. In addition, these abstractions form the basis upon which other interactive applications may be built without the requirement for the use of HAVi specific widgets. Thus, the HAVi widget framework is more generally applicable to interactive application execution, rather than exclusively focused upon HAVi. The Javadoc describes these states in more detail, including any valid DISABLED states.

### 8.3.3 Framework Class Hierarchy

The HAVi widget framework consists of a base class (HVisible) and a set of interfaces that model the behaviors different types of widget may exhibit. The behavior is modeled on the number of states a widget may represent. For each such state a widget can present a particular representation (graphical, textual and sound) to the user.

The widget framework allows for simple user interface development by application authors. It also reduces the size of the developed application, since most of the presentation and interaction capability is resident on the device – developers can concentrate on the specific functionality of their application.

#### 8.3.3.1 HContainer

Components in the HAVi User-Interface are explicitly allowed to overlap each other. Hence, the HAVi User-Interface extensions adds additional Z-ordering related methods to org.havi.ui.HContainer:

Additional semantics related to transparency of the HContainer itself and its Components, are also defined via the HMatteLayer interface.

The org.havi.ui.HContainer class also adds the ability to determine whether hardware double buffering is present, using the isDoubleBuffered method.

The org.havi.ui.HContainer class also adds the ability to determine whether it is completely opaque, by applications overriding the isOpaque method.

Additionally, the default LayoutManager for HContainer is defined to be null, i.e. absolute positioning, in contrast to the FlowLayout used in java.awt.Container.

#### 8.3.3.2 HComponent

The base class for all HAVi widgets.

The org.havi.ui.HComponent class extends java.awt.Component to include additional semantics related to transparency of the HComponent, defined via the HMatteLayer interface.

The org.havi.ui.HComponent class also adds the ability to determine whether hardware double buffering is present, using the isDoubleBuffered method.

The org.havi.ui.HComponent class also adds the ability to determine whether it is completely opaque, by applications overriding the isOpaque method.

#### 8.3.3.3 HVisible

Represents a widget that has only  two states, for example HStaticText or HStaticIcon. This widget can be in either a “normal” state or a “disabled” state.

#### 8.3.3.4 HNavigable

An interface that is implemented by classes that are derived from HVisible for adding an additional state that is used to indicate if the widget is currently focused.

The HNavigable interface also provides the functionality necessary to manage the focus navigation between widgets assuming a remote control style UP, DOWN, LEFT, RIGHT form of navigation, using the setFocusTraversal method.

The precise semantics of the HNavigable interface are defined in the supporting Javadoc.

#### 8.3.3.5 HActionable

The HActionable interface extends HNavigable by adding an additional state that is used to indicate when the widget has been actioned.

The HActionable interface provides the functionality necessary to associate HActionListeners with the widget, using the addHActionListener and removeHActionListener methods. These HActionListeners will be called when the widget is actioned.

A widget that implements the HActionable interface is actioned when it receives a havi.ui.event.HActionEvent key event. The widget will move into its Actioned state by presenting its Actioned look. Any associated HActionListeners will be called by the widget calling its HActionInputPreferred.processHActionEvent method. When the HActionListeners have returned the widget will return to its focused state.

The precise semantics of the HActionable interface are defined in the supporting Javadoc.

### 8.3.3.6 HSwitchable

The HSwitchable interface extends HActionable by adding an additional state that is used to maintain an internal (on/off) value.

The state transitions for HSwitchable are as follows:

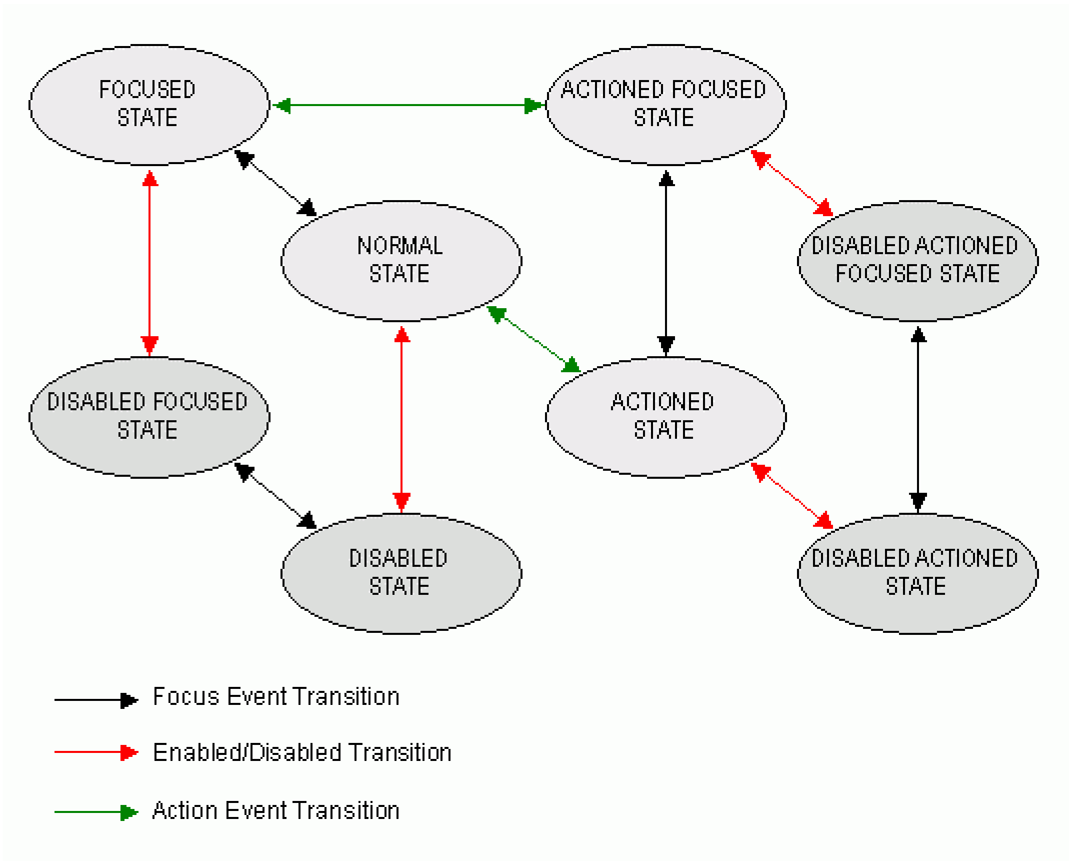


**Figure 5. HSwitchable Transitions**

The precise semantics of the HSwitchable interface are defined in the supporting Javadoc. Note that any state is permitted to change to the “disabled” state.

### 8.3.3.7 HAdjustmentValue, HItemValue, HTextValue

These interfaces extend HNavigable by adding support for managing a widget with an internal value that can be manipulated by user interaction.

All HAVi UI components that require adjustable numerical values, such as range controls, have implemented the HAdjustmentValue interface. Here, the value responds to unit & block increments, and has an optional sound associated with such adjustments.

All HAVi UI components that have selectable content, such as list groups, have implemented the HItemValue interface. An optional sound can be associated with item selection.

All HAVi UI components that have editable text content, such as text entry controls, have implemented the HTextValue interface.

The precise semantics of the HValue interface are defined in the supporting Javadoc.

### 8.3.4 Separation of "Look"

The flexibility of the HAVi widget framework is further enhanced by separating the "look" component from that for "feel". This allows easy construction of many styles of presentation associated with each of the abstract Component Behaviors defined previously.

Content can be associated with each state of a widget. For each widget state, textual, graphical and user defined content can be associated with the widget. The HLook interface defines the mechanism by which the content for the particular state of the widget can be rendered.

The HLook method showLook is used to provide the rendering of the content for the widget. Note that since this method is separated from the widget class, there is no need to subclass the widget to change its look. The showLook method is responsible for repainting the entire component, including its background, subject to the clipRect of the Graphics object passed to it. The showLook method should not modify the clipRect of the Graphics object that is passed to it.

An HLook may also provide some form of border decoration, for example, drawing a rectangle around the widget when it has focus. To allow for predictable layout and presentation the HLook interface provides methods that are used to indicate the size of such a border area.

To support layout managers the HLook interface also defines the following methods, which allow the associated HVisible to query the HLook for its maximum, minimum and preferred sizes: getMaximumSize, getMinimumSize, getPreferredSize.

### 8.3.5 Pluggable Looks

The HAVi Widget framework provides a set of standard classes that implement the HLook interface. These can be regarded as the set of default looks that will be provided by all implementations. The particular rendering of a look is not defined and is manufacturer dependent.

Pluggable Look is defined in such a way as to allow implementers to extend the number of "looks" available to their application. By doing so, new "looks" are automatically available for every Component Behavior and for all widget types derived from those Behaviors. This is described in the Pluggable Look Interface.

To facilitate application development and to limit the size of applications, a set of pre-defined "looks" is provided. These are:

- HAnimateLook – presentation of an animated image sequence.
- HGraphicLook – presentation of graphical content.

- HRangeLook – presentation of a value within a range.

- HListGroupLook - presentation of both the ListGroup itself and the items held on the list.

- HTextLook – a simple presentation mechanism for textual content.

- HSinglelineEntryLook – presentation of a single line of textual content that can be edited by the user.

- HMultilineEntryLook – presentation of multiple lines of textual content that can be edited by the user.

This basic set allows the construction of most typical interactive user interfaces when used in conjunction with the Component Behaviors to define a widget set. It can however be extended in a general fashion to provide new categories of "look".

When a widget is constructed, it is provided with a default look. This default HLook will be one of the standard set of looks listed above. For example, the HGraphicButton is created with the HGraphicLook by default. The default look that is used when the widget is constructed can be changed by calling the static method setDefaultLook that is provided on all widget types. Any widget of that type created after the call will be created with the new HLook that was passed in as the parameter to setDefaultLook. The look of an individual widget can be modified by using the method HVisible.setLook.

The Pluggable Look mechanism is flexible enough so that the application developer can create new HLooks. For example a combined HGraphicLook and HTextLook, where the Text may overlay the Graphic, or be shown in place of the Graphic while the Graphic is being loaded.

### 8.3.6 Content Behavior

Content is associated with the widget through the following methods on HVisible: setTextContent, setGraphicContent, setAnimateContent and setContent. Hence, multiple content (Text, Graphics, Animations and user-defined content) can be associated with a widget. The way multiple content is rendered is dependent on the HLook associated with the widget. The default looks provided by the platform may not render all the content types.

Different content can be associated with the different states of the widget. For example, an HGraphicButton might have six different images to represent its six different states (NORMAL_STATE, FOCUSED_STATE, ACTIONED_STATE, ACTIONED_FOCUSED_STATE, DISABLED_STATE and DISABLED_FOCUSED_STATE) to the user, using the setGraphicContent. The same content can be applied to all states of the widget by using the HState constant ALL_STATES when calling setTextContent, setGraphicContent, setAnimateContent and setContent.

- By default, content associated with a widget is not modified to fit the dimensions of the widget. Refer to the alignment and scaling methods in HVisible to address this issue.

Mechanisms are also available that allow (graphic) content to be resized to match the widget dimensions, etc.

## 8.4 HAVi Resident Widgets

Using the Component Behaviors (HVisible, HNavigable, HActionable and HSwitchable) defined in the previous section a set of resident widgets is provided.

Note that implementations of the HAVi widget set shall be implemented (and behave) as lightweight components. HAVi widgets do not include an associated peer class, irrespective of the exact mechanism for their implementation, i.e., directly implemented in Java, or via some platform specific mechanism.

### 8.4.1 Simple Text/Graphic/Animate Widgets

The HAVi set of resident widgets includes both visible and navigable versions of the HText, HIcon and HAnimation classes:

- Applications providing simple "display-only" non-navigable text, image, or animations may employ the "Static" versions of these classes.
- Applications wishing to provide additional feedback, e.g. "tooltips", or audio feedback – for example, a commentary – may employ the navigable versions of these classes.

| Widget Type | Description | Static | Navigable |
|---|---|---|---|
| Animation | Displays a simple sequence of images | HStaticAnimation | HAnimation |
| Text | Displays a text label | HStaticText | HText |
| Graphic | Displays an Image | HStaticIcon | HIcon |

Refer to the supporting Javadoc for a more detailed description of these widgets.

### 8.4.2 Buttons

The HAVi set of resident widgets includes both textual and graphical version of a push button: HTextButton and HGraphicButton. These buttons implement the HActionable interface that defines their behavior.

The HToggleButton is used to represent a graphical control that has a boolean state that can be toggled on and off by the user (e.g. Checkbox or Radio Button). The HToggleButton implements the HSwitchable interface that defines its behavior. A HToggleButton widget does not have an associated text label as part of the widget. If a text label is required, a separate HStaticText widget should be created.

A set of HToggleButtons can be associated with a HToggleGroup. A HToggleGroup will ensure that a maximum of one HToggleButton is chosen at any time (i.e. a group of radio buttons).

Refer to the supporting Javadoc for a more detailed description of these widgets.

### 8.4.3 Range Widgets

The HAVi set of resident widgets include a group of controls to represent a particular integer value in a range of values i.e. a slider control, or scroll bar. The HStaticRange widget is a non navigable widget, HRange widget is navigable (implements the HNavigable interface) and the HRangeValue is navigable and its value can be modified by user interaction (implements the HAdjustmentValue interface).

Refer to the supporting Javadoc for a more detailed description of these widgets.

### 8.4.4 List Widgets

A HListGroup is a visible that manages a dynamic set of vertically or horizontally scrollable HListElements, allowing either single or multiple HListElements to be chosen by the user. The HListGroup will automatically scroll the HListElements when the user navigates to an element that is currently not visible within the list group.

Refer to the supporting Javadoc for a more detailed description of these widgets.

### 8.4.5 Text Entry Widgets

The HSinglelineEntry component allows a user to enter a single line text string. A typical rendering is as a text entry field, e.g. with an associated on-screen keyboard. The HMultilineEntry widget extends the HSinglelineEntry widget and allows text to be entered over multiple lines. Both these widgets implement the HTextValue interface resulting in the widgets firing HTextEvents when starting to edit, finishing editing, and whenever the content changes.

Refer to the supporting Javadoc for a more detailed description of these widgets.

## 8.5 Profiles

Implementations of the HAVi User-Interface on an FAV should:

- have a minimum screen resolution of 320 by 240 pixels (quarter VGA)
- include support for the image and sound content types, as defined in DDI.
- either provide a physical keyboard or provide a virtual keyboard supporting at least the entry of alphanumeric codes

## 8.6 General Approach to Error Behavior

Where a method call is specified as taking an object as one of its input parameters, if null is passed as a parameter and not explicitly identified as a valid input for that parameter of that method then a java.lang.NullPointerException shall be thrown. Where an input parameter to a method call is specified to be more restrictive than its Java type allows (e.g. only a restricted set of numbers are allowed as inputs), providing values outside the allowed range shall result in a java.lang.IllegalArgumentException being thrown.

## 8.7 Register of Constants

```
org.havi.ui.HAdjustableLook.ADJUST_THUMB = -6;
org.havi.ui.HAdjustableLook.ADJUST_PAGE_MORE = -5;
org.havi.ui.HAdjustableLook.ADJUST_PAGE_LESS = -4;
org.havi.ui.HAdjustableLook.ADJUST_BUTTON_MORE = -3;
org.havi.ui.HAdjustableLook.ADJUST_BUTTON_LESS = -2;
org.havi.ui.HAdjustableLook.ADJUST_NONE = -1;
org.havi.ui.HAnimateEffect.REPEAT_INFINITE = -1;
org.havi.ui.HAnimateEffect.PLAY_REPEATING = 1;
org.havi.ui.HAnimateEffect.PLAY_ALTERNATING = 2;
org.havi.ui.HBackgroundConfigTemplate.CHANGEABLE_SINGLE_COLOR = 10;
org.havi.ui.HBackgroundConfigTemplate.STILL_IMAGE = 11;
org.havi.ui.HFontCapabilities.BASIC_LATIN = 1;
org.havi.ui.HFontCapabilities.LATIN_1_SUPPLEMENT = 2;
org.havi.ui.HFontCapabilities.LATIN_EXTENDED_A = 3;
org.havi.ui.HFontCapabilities.LATIN_EXTENDED_B = 4;
org.havi.ui.HFontCapabilities.IPA_EXTENSIONS = 5;
org.havi.ui.HFontCapabilities.SPACING_MODIFIER_LETTERS = 6;
org.havi.ui.HFontCapabilities.COMBINING_DIACRITICAL_MARKS = 7;
org.havi.ui.HFontCapabilities.BASIC_GREEK = 8;
org.havi.ui.HFontCapabilities.GREEK_SYMBOLS_AND_COPTIC = 9;
org.havi.ui.HFontCapabilities.CYRILLIC = 10;
org.havi.ui.HFontCapabilities.ARMENIAN = 11;
org.havi.ui.HFontCapabilities.BASIC_HEBREW = 12;
org.havi.ui.HFontCapabilities.HEBREW_EXTENDED = 13;
org.havi.ui.HFontCapabilities.BASIC_ARABIC = 14;
org.havi.ui.HFontCapabilities.ARABIC_EXTENDED = 15;
org.havi.ui.HFontCapabilities.DEVANAGARI = 16;
org.havi.ui.HFontCapabilities.BENGALI = 17;
org.havi.ui.HFontCapabilities.GURMUKHI = 18;
org.havi.ui.HFontCapabilities.GUJARATI = 19;
org.havi.ui.HFontCapabilities.ORIYA = 20;
org.havi.ui.HFontCapabilities.TAMIL = 21;
org.havi.ui.HFontCapabilities.TELUGU = 22;
org.havi.ui.HFontCapabilities.KANNADA = 23;
org.havi.ui.HFontCapabilities.MALAYALAM = 24;
org.havi.ui.HFontCapabilities.THAI = 25;
org.havi.ui.HFontCapabilities.LAO = 26;
org.havi.ui.HFontCapabilities.BASIC_GEORGIAN = 27;
org.havi.ui.HFontCapabilities.GEORGIAN_EXTENDED = 28;
org.havi.ui.HFontCapabilities.HANGUL_JAMO = 29;
org.havi.ui.HFontCapabilities.LATIN_EXTENDED_ADDITIONAL = 30;
org.havi.ui.HFontCapabilities.GREEK_EXTENDED = 31;
org.havi.ui.HFontCapabilities.GENERAL_PUNCTUATION = 32;
org.havi.ui.HFontCapabilities.SUPERSCRIPTS_AND_SUBSCRIPTS = 33;
org.havi.ui.HFontCapabilities.CURRENCY_SYMBOLS = 34;
org.havi.ui.HFontCapabilities.COMBINING_DIACTRICAL_MARKS_FOR_SYMBOLS = 35;
org.havi.ui.HFontCapabilities.LETTERLIKE_SYMBOLS = 36;
org.havi.ui.HFontCapabilities.NUMBER_FORMS = 37;
org.havi.ui.HFontCapabilities.ARROWS = 38;
org.havi.ui.HFontCapabilities.MATHEMATICAL_OPERATORS = 39;
org.havi.ui.HFontCapabilities.MISCELLANEOUS_TECHNICAL = 40;
org.havi.ui.HFontCapabilities.CONTROL_PICTURES = 41;
org.havi.ui.HFontCapabilities.OPTICAL_CHARACTER_RECOGNITION = 42;
org.havi.ui.HFontCapabilities.ENCLOSED_ALPHANUMERICS = 43;
org.havi.ui.HFontCapabilities.BOX_DRAWING = 44;
org.havi.ui.HFontCapabilities.BLOCK_ELEMENTS = 45;
org.havi.ui.HFontCapabilities.GEOMETRICAL_SHAPES = 46;
org.havi.ui.HFontCapabilities.MISCELLANEOUS_SYMBOLS = 47;
org.havi.ui.HFontCapabilities.DINGBATS = 48;
org.havi.ui.HFontCapabilities.CJK_SYMBOLS_AND_PUNCTUATION = 49;
org.havi.ui.HFontCapabilities.HIRAGANA = 50;
org.havi.ui.HFontCapabilities.KATAKANA = 51;
org.havi.ui.HFontCapabilities.BOPOMOFO = 52;
org.havi.ui.HFontCapabilities.HANGUL_COMPATIBILITY_JAMO = 53;
org.havi.ui.HFontCapabilities.CJK_MISCELLANEOUS = 54;
org.havi.ui.HFontCapabilities.ENCLOSED_CJK_LETTERS_AND_MONTHS = 55;
org.havi.ui.HFontCapabilities.CJK_COMPATIBILITY = 56;
org.havi.ui.HFontCapabilities.HANGUL = 57;
org.havi.ui.HFontCapabilities.HANGUL_SUPPLEMENTARY_A = 58;
org.havi.ui.HFontCapabilities.HANGUL_SUPPLEMENTARY_B = 59;
org.havi.ui.HFontCapabilities.CJK_UNIFIED_IDEOGRAPHS = 60;
org.havi.ui.HFontCapabilities.PRIVATE_USE_AREA = 61;
org.havi.ui.HFontCapabilities.CJK_COMPATIBILITY_IDEOGRAPHS = 62;
org.havi.ui.HFontCapabilities.ALPHABETIC_PRESENTATION_FORMS_A = 63;
org.havi.ui.HFontCapabilities.ARABIC_PRESENTATION_FORMS_A = 64;
org.havi.ui.HFontCapabilities.COMBINING_HALF_MARKS = 65;
org.havi.ui.HFontCapabilities.CJK_COMPATIBILITY_FORMS = 66;
org.havi.ui.HFontCapabilities.SMALL_FORM_VARIANTS = 67;
org.havi.ui.HFontCapabilities.ARABIC_PRESENTATION_FORMS_B = 68;
org.havi.ui.HFontCapabilities.HALFWIDTH_AND_FULLWIDTH_FORMS = 69;
org.havi.ui.HFontCapabilities.SPECIALS = 70;
org.havi.ui.HGraphicsConfigTemplate.VIDEO_MIXING = 12;
org.havi.ui.HGraphicsConfigTemplate.MATTE_SUPPORT = 13;
org.havi.ui.HGraphicsConfigTemplate.IMAGE_SCALING_SUPPORT = 14;
org.havi.ui.HImageHints.NATURAL_IMAGE = 1;
org.havi.ui.HImageHints.CARTOON = 2;
org.havi.ui.HImageHints.BUSINESS_GRAPHICS = 3;
org.havi.ui.HImageHints.LINE_ART = 4;
org.havi.ui.HKeyboardInputPreferred.INPUT_NUMERIC = 1;
org.havi.ui.HKeyboardInputPreferred.INPUT_ALPHA = 2;
org.havi.ui.HKeyboardInputPreferred.INPUT_ANY = 4;
org.havi.ui.HKeyboardInputPreferred.INPUT_CUSTOMIZED = 8;
org.havi.ui.HListGroup.DEFAULT_ICON_HEIGHT = -4;
org.havi.ui.HListGroup.DEFAULT_ICON_WIDTH = -3;
org.havi.ui.HListGroup.DEFAULT_LABEL_HEIGHT = -2;
org.havi.ui.HListGroup.ITEM_NOT_FOUND = -1;
org.havi.ui.HListGroup.ADD_INDEX_END = -1;
org.havi.ui.HListGroup.DEFAULT_LABEL_WIDTH = -1;
org.havi.ui.HOrientable.ORIENT_LEFT_TO_RIGHT = 0;
org.havi.ui.HOrientable.ORIENT_RIGHT_TO_LEFT = 1;
org.havi.ui.HOrientable.ORIENT_TOP_TO_BOTTOM = 2;
org.havi.ui.HOrientable.ORIENT_BOTTOM_TO_TOP = 3;
org.havi.ui.HScene.IMAGE_NONE = 0;
org.havi.ui.HScene.NO_BACKGROUND_FILL = 0;
org.havi.ui.HScene.IMAGE_STRETCH = 1;
org.havi.ui.HScene.BACKGROUND_FILL = 1;
org.havi.ui.HScene.IMAGE_CENTER = 2;
org.havi.ui.HScene.IMAGE_TILE = 3;
org.havi.ui.HSceneTemplate.GRAPHICS_CONFIGURATION = 0;
org.havi.ui.HSceneTemplate.REQUIRED = 1;
org.havi.ui.HSceneTemplate.SCENE_PIXEL_DIMENSION = 1;
org.havi.ui.HSceneTemplate.PREFERRED = 2;
org.havi.ui.HSceneTemplate.SCENE_PIXEL_LOCATION = 2;
org.havi.ui.HSceneTemplate.UNNECESSARY = 3;
org.havi.ui.HSceneTemplate.SCENE_SCREEN_DIMENSION = 4;
org.havi.ui.HSceneTemplate.SCENE_SCREEN_LOCATION = 8;
org.havi.ui.HScreenConfigTemplate.REQUIRED = 1;
org.havi.ui.HScreenConfigTemplate.ZERO_BACKGROUND_IMPACT = 1;
org.havi.ui.HScreenConfigTemplate.PREFERRED = 2;
org.havi.ui.HScreenConfigTemplate.ZERO_GRAPHICS_IMPACT = 2;
org.havi.ui.HScreenConfigTemplate.DONT_CARE = 3;
org.havi.ui.HScreenConfigTemplate.ZERO_VIDEO_IMPACT = 3;
org.havi.ui.HScreenConfigTemplate.PREFERRED_NOT = 4;
org.havi.ui.HScreenConfigTemplate.INTERLACED_DISPLAY = 4;
org.havi.ui.HScreenConfigTemplate.REQUIRED_NOT = 5;
org.havi.ui.HScreenConfigTemplate.FLICKER_FILTERING = 5;
org.havi.ui.HScreenConfigTemplate.VIDEO_GRAPHICS_PIXEL_ALIGNED = 6;
org.havi.ui.HScreenConfigTemplate.PIXEL_ASPECT_RATIO = 7;
org.havi.ui.HScreenConfigTemplate.PIXEL_RESOLUTION = 8;
org.havi.ui.HScreenConfigTemplate.SCREEN_RECTANGLE = 9;
org.havi.ui.HState.FOCUSED_STATE_BIT = 1;
org.havi.ui.HState.ACTIONED_STATE_BIT = 2;
org.havi.ui.HState.DISABLED_STATE_BIT = 4;
org.havi.ui.HState.ALL_STATES = 7;
org.havi.ui.HState.FIRST_STATE = 128;
org.havi.ui.HState.NORMAL_STATE = 128;
org.havi.ui.HState.FOCUSED_STATE = 129;
org.havi.ui.HState.ACTIONED_STATE = 130;
org.havi.ui.HState.ACTIONED_FOCUSED_STATE = 131;
org.havi.ui.HState.DISABLED_STATE = 132;
org.havi.ui.HState.DISABLED_FOCUSED_STATE = 133;
org.havi.ui.HState.DISABLED_ACTIONED_STATE = 134;
org.havi.ui.HState.DISABLED_ACTIONED_FOCUSED_STATE = 135;
org.havi.ui.HState.LAST_STATE = 135;
org.havi.ui.HStaticRange.SLIDER_BEHAVIOR = 0;
org.havi.ui.HStaticRange.SCROLLBAR_BEHAVIOR = 1;
org.havi.ui.HVideoConfigTemplate.GRAPHICS_MIXING = 15;
org.havi.ui.HVisible.NO_DEFAULT_WIDTH = -1;
org.havi.ui.HVisible.NO_DEFAULT_HEIGHT = -1;
org.havi.ui.HVisible.HALIGN_LEFT = 0;
org.havi.ui.HVisible.VALIGN_TOP = 0;
org.havi.ui.HVisible.RESIZE_NONE = 0;
org.havi.ui.HVisible.NO_BACKGROUND_FILL = 0;
org.havi.ui.HVisible.FIRST_CHANGE = 0;
org.havi.ui.HVisible.TEXT_CONTENT_CHANGE = 0;
org.havi.ui.HVisible.HALIGN_CENTER = 1;
org.havi.ui.HVisible.RESIZE_PRESERVE_ASPECT = 1;
org.havi.ui.HVisible.BACKGROUND_FILL = 1;
org.havi.ui.HVisible.GRAPHIC_CONTENT_CHANGE = 1;
org.havi.ui.HVisible.HALIGN_RIGHT = 2;
org.havi.ui.HVisible.RESIZE_ARBITRARY = 2;
org.havi.ui.HVisible.ANIMATE_CONTENT_CHANGE = 2;
org.havi.ui.HVisible.HALIGN_JUSTIFY = 3;
org.havi.ui.HVisible.CONTENT_CHANGE = 3;
org.havi.ui.HVisible.VALIGN_CENTER = 4;
org.havi.ui.HVisible.STATE_CHANGE = 4;
org.havi.ui.HVisible.CARET_POSITION_CHANGE = 5;
org.havi.ui.HVisible.ECHO_CHAR_CHANGE = 6;
org.havi.ui.HVisible.EDIT_MODE_CHANGE = 7;
org.havi.ui.HVisible.VALIGN_BOTTOM = 8;
org.havi.ui.HVisible.MIN_MAX_CHANGE = 8;
org.havi.ui.HVisible.THUMB_OFFSETS_CHANGE = 9;
org.havi.ui.HVisible.ORIENTATION_CHANGE = 10;
org.havi.ui.HVisible.TEXT_VALUE_CHANGE = 11;
org.havi.ui.HVisible.VALIGN_JUSTIFY = 12;
org.havi.ui.HVisible.ITEM_VALUE_CHANGE = 12;
org.havi.ui.HVisible.ADJUSTMENT_VALUE_CHANGE = 13;
org.havi.ui.HVisible.LIST_CONTENT_CHANGE = 14;
org.havi.ui.HVisible.LIST_ICONSIZE_CHANGE = 15;
org.havi.ui.HVisible.LIST_LABELSIZE_CHANGE = 16;
org.havi.ui.HVisible.LIST_MULTISELECTION_CHANGE = 17;
org.havi.ui.HVisible.LIST_SCROLLPOSITION_CHANGE = 18;
org.havi.ui.HVisible.SIZE_CHANGE = 19;
org.havi.ui.HVisible.BORDER_CHANGE = 20;
org.havi.ui.HVisible.REPEAT_COUNT_CHANGE = 21;
org.havi.ui.HVisible.ANIMATION_POSITION_CHANGE = 22;
org.havi.ui.HVisible.LIST_SELECTION_CHANGE = 23;
org.havi.ui.HVisible.UNKNOWN_CHANGE = 24;
org.havi.ui.HVisible.LAST_CHANGE = 24;
org.havi.ui.event.HAdjustmentEvent.ADJUST_FIRST = 2000;
org.havi.ui.event.HAdjustmentEvent.ADJUST_START_CHANGE = 2000;
org.havi.ui.event.HAdjustmentEvent.ADJUST_LESS = 2001;
org.havi.ui.event.HAdjustmentEvent.ADJUST_MORE = 2002;
org.havi.ui.event.HAdjustmentEvent.ADJUST_PAGE_LESS = 2003;
org.havi.ui.event.HAdjustmentEvent.ADJUST_PAGE_MORE = 2004;
org.havi.ui.event.HAdjustmentEvent.ADJUST_LAST = 2005;
org.havi.ui.event.HAdjustmentEvent.ADJUST_END_CHANGE = 2005;
org.havi.ui.event.HBackgroundImageEvent.BACKGROUNDIMAGE_FIRST = 1;
org.havi.ui.event.HBackgroundImageEvent.BACKGROUNDIMAGE_LOADED = 1;
org.havi.ui.event.HBackgroundImageEvent.BACKGROUNDIMAGE_FILE_NOT_FOUND = 2;
org.havi.ui.event.HBackgroundImageEvent.BACKGROUNDIMAGE_IOERROR = 3;
org.havi.ui.event.HBackgroundImageEvent.BACKGROUNDIMAGE_INVALID = 4;
org.havi.ui.event.HBackgroundImageEvent.BACKGROUNDIMAGE_LAST = 4;
org.havi.ui.event.HEventRepresentation.ER_TYPE_NOT_SUPPORTED = 0;
org.havi.ui.event.HEventRepresentation.ER_TYPE_STRING = 1;
org.havi.ui.event.HEventRepresentation.ER_TYPE_COLOR = 2;
org.havi.ui.event.HEventRepresentation.ER_TYPE_SYMBOL = 4;
org.havi.ui.event.HFocusEvent.NO_TRANSFER_ID = -1;
org.havi.ui.event.HFocusEvent.HFOCUS_FIRST = 2029;
org.havi.ui.event.HFocusEvent.FOCUS_TRANSFER = 2029;
org.havi.ui.event.HFocusEvent.HFOCUS_LAST = 2029;
org.havi.ui.event.HItemEvent.ITEM_FIRST = 2006;
org.havi.ui.event.HItemEvent.ITEM_START_CHANGE = 2006;
org.havi.ui.event.HItemEvent.ITEM_TOGGLE_SELECTED = 2007;
org.havi.ui.event.HItemEvent.ITEM_SELECTED = 2008;
org.havi.ui.event.HItemEvent.ITEM_CLEARED = 2009;
org.havi.ui.event.HItemEvent.ITEM_SELECTION_CLEARED = 2010;
org.havi.ui.event.HItemEvent.ITEM_SET_CURRENT = 2011;
org.havi.ui.event.HItemEvent.ITEM_SET_PREVIOUS = 2012;
org.havi.ui.event.HItemEvent.ITEM_SET_NEXT = 2013;
org.havi.ui.event.HItemEvent.SCROLL_MORE = 2014;
org.havi.ui.event.HItemEvent.SCROLL_LESS = 2015;
org.havi.ui.event.HItemEvent.SCROLL_PAGE_MORE = 2016;
org.havi.ui.event.HItemEvent.SCROLL_PAGE_LESS = 2017;
org.havi.ui.event.HItemEvent.ITEM_END_CHANGE = 2018;
org.havi.ui.event.HItemEvent.ITEM_LAST = 2018;
org.havi.ui.event.HRcEvent.RC_FIRST = 400;
org.havi.ui.event.HRcEvent.VK_COLORED_KEY_0 = 403;
org.havi.ui.event.HRcEvent.VK_COLORED_KEY_1 = 404;
org.havi.ui.event.HRcEvent.VK_COLORED_KEY_2 = 405;
org.havi.ui.event.HRcEvent.VK_COLORED_KEY_3 = 406;
org.havi.ui.event.HRcEvent.VK_COLORED_KEY_4 = 407;
org.havi.ui.event.HRcEvent.VK_COLORED_KEY_5 = 408;
org.havi.ui.event.HRcEvent.VK_POWER = 409;
org.havi.ui.event.HRcEvent.VK_DIMMER = 410;
org.havi.ui.event.HRcEvent.VK_WINK = 411;
org.havi.ui.event.HRcEvent.VK_REWIND = 412;
org.havi.ui.event.HRcEvent.VK_STOP = 413;
org.havi.ui.event.HRcEvent.VK_EJECT_TOGGLE = 414;
org.havi.ui.event.HRcEvent.VK_PLAY = 415;
org.havi.ui.event.HRcEvent.VK_RECORD = 416;
org.havi.ui.event.HRcEvent.VK_FAST_FWD = 417;
org.havi.ui.event.HRcEvent.VK_PLAY_SPEED_UP = 418;
org.havi.ui.event.HRcEvent.VK_PLAY_SPEED_DOWN = 419;
org.havi.ui.event.HRcEvent.VK_PLAY_SPEED_RESET = 420;
org.havi.ui.event.HRcEvent.VK_RECORD_SPEED_NEXT = 421;
org.havi.ui.event.HRcEvent.VK_GO_TO_START = 422;
org.havi.ui.event.HRcEvent.VK_GO_TO_END = 423;
org.havi.ui.event.HRcEvent.VK_TRACK_PREV = 424;
org.havi.ui.event.HRcEvent.VK_TRACK_NEXT = 425;
org.havi.ui.event.HRcEvent.VK_RANDOM_TOGGLE = 426;
org.havi.ui.event.HRcEvent.VK_CHANNEL_UP = 427;
org.havi.ui.event.HRcEvent.VK_CHANNEL_DOWN = 428;
org.havi.ui.event.HRcEvent.VK_STORE_FAVORITE_0 = 429;
org.havi.ui.event.HRcEvent.VK_STORE_FAVORITE_1 = 430;
org.havi.ui.event.HRcEvent.VK_STORE_FAVORITE_2 = 431;
org.havi.ui.event.HRcEvent.VK_STORE_FAVORITE_3 = 432;
org.havi.ui.event.HRcEvent.VK_RECALL_FAVORITE_0 = 433;
org.havi.ui.event.HRcEvent.VK_RECALL_FAVORITE_1 = 434;
org.havi.ui.event.HRcEvent.VK_RECALL_FAVORITE_2 = 435;
org.havi.ui.event.HRcEvent.VK_RECALL_FAVORITE_3 = 436;
org.havi.ui.event.HRcEvent.VK_CLEAR_FAVORITE_0 = 437;
org.havi.ui.event.HRcEvent.VK_CLEAR_FAVORITE_1 = 438;
org.havi.ui.event.HRcEvent.VK_CLEAR_FAVORITE_2 = 439;
org.havi.ui.event.HRcEvent.VK_CLEAR_FAVORITE_3 = 440;
org.havi.ui.event.HRcEvent.VK_SCAN_CHANNELS_TOGGLE = 441;
org.havi.ui.event.HRcEvent.VK_PINP_TOGGLE = 442;
org.havi.ui.event.HRcEvent.VK_SPLIT_SCREEN_TOGGLE = 443;
org.havi.ui.event.HRcEvent.VK_DISPLAY_SWAP = 444;
org.havi.ui.event.HRcEvent.VK_SCREEN_MODE_NEXT = 445;
org.havi.ui.event.HRcEvent.VK_VIDEO_MODE_NEXT = 446;
org.havi.ui.event.HRcEvent.VK_VOLUME_UP = 447;
org.havi.ui.event.HRcEvent.VK_VOLUME_DOWN = 448;
org.havi.ui.event.HRcEvent.VK_MUTE = 449;
org.havi.ui.event.HRcEvent.VK_SURROUND_MODE_NEXT = 450;
org.havi.ui.event.HRcEvent.VK_BALANCE_RIGHT = 451;
org.havi.ui.event.HRcEvent.VK_BALANCE_LEFT = 452;
org.havi.ui.event.HRcEvent.VK_FADER_FRONT = 453;
org.havi.ui.event.HRcEvent.VK_FADER_REAR = 454;
org.havi.ui.event.HRcEvent.VK_BASS_BOOST_UP = 455;
org.havi.ui.event.HRcEvent.VK_BASS_BOOST_DOWN = 456;
org.havi.ui.event.HRcEvent.VK_INFO = 457;
org.havi.ui.event.HRcEvent.VK_GUIDE = 458;
org.havi.ui.event.HRcEvent.VK_TELETEXT = 459;
org.havi.ui.event.HRcEvent.VK_SUBTITLE = 460;
org.havi.ui.event.HRcEvent.RC_LAST = 460;
org.havi.ui.event.HTextEvent.TEXT_FIRST = 2019;
org.havi.ui.event.HTextEvent.TEXT_START_CHANGE = 2019;
org.havi.ui.event.HTextEvent.TEXT_CHANGE = 2020;
org.havi.ui.event.HTextEvent.TEXT_CARET_CHANGE = 2021;
org.havi.ui.event.HTextEvent.TEXT_END_CHANGE = 2022;
org.havi.ui.event.HTextEvent.CARET_NEXT_CHAR = 2023;
org.havi.ui.event.HTextEvent.CARET_NEXT_LINE = 2024;
org.havi.ui.event.HTextEvent.CARET_PREV_CHAR = 2025;
org.havi.ui.event.HTextEvent.CARET_PREV_LINE = 2026;
org.havi.ui.event.HTextEvent.CARET_NEXT_PAGE = 2027;
org.havi.ui.event.HTextEvent.TEXT_LAST = 2028;
org.havi.ui.event.HTextEvent.CARET_PREV_PAGE = 2028;
```