Orca Documentation Series

Sun Microsystems, Inc.

Orca Documentation Series
by Sun Microsystems, Inc.

Orca User Experience Design

Orca User Experience Design

Table of Contents

Foreword

Orca is a flexible, extensible, and powerful assistive technology that provides end-user access to applications and toolkits that support the AT-SPI (e.g., the GNOME desktop). With early input and continued engagement from its end users, Orca has been designed and implemented by the Sun Microsystems, Inc., Accessibility Program Office.

NOTE: Orca is currently a work in progress. As a result, this and other books in the Orca Documentation Series are under continuous modification and are also in various states of completeness.

This book covers the design of the user experience for Orca.

Chapter 1. Introduction

The development of Orca focused primarily on the requirements of its end users. Orca's users may require non-visual access methods such as speech and/or braille and they may also require alternative visual access methods such as magnification.

When designing Orca, its authors at Sun Microsystems engaged the user community from the start, soliciting feedback on questions such as the following:

- What tasks do they want to accomplish?
- How do they accomplish these tasks?
- What should the screen reader present to them?
- What is the user experience like?

This book covers the user design of Orca, ranging from a poetic use of Cooper's "Persona" concept [cooper99>] to the development of the end user requirements and default interaction styles for speech synthesis, braille, magnification, and input devices.

Chapter 2. Personas

The Orca user personas are intended to be a tool that helps elicit use cases and user requirements. The approach is to think of "real" users with specific disabilities performing specific actions. These personas are meant to cover a range of typical users, both in the tasks they want to accomplish as well as their disabilities. The Orca user design explored four personas, each of which is discussed in the following sections:

- Lee: a programmer who uses braille
- Pat: a manager who uses magnification and speech
- Kim: an administrative assistant who uses speech
- Sam: an IT staff member who uses speech and braille

Lee the Programmer

Lee is a programmer who is primarily a braille user and typically performs the tasks outlined in the following sections.

Write code in NetBeans

Lee edits a program module in NetBeans. Over the course of this editing, Lee must be able to do the following:

- Determine proper indenting. On the braille display, Orca shows this by using blank spaces. Orca also has an option to compress horizontal spacing so as to better make use of braille real-estate.

  Lee sometimes uses speech as well, and uses separate commands to have Orca read the indentation level. When in focus tracking mode, Lee also has the option to turn on the automatic speaking of this information.
- See highlighting, attributes and coloring. On the braille display, Orca shows this in status cells, where the status cells reflect the attributes of the character under the current cursor position. For braille displays without status cells, attribute information is shown using dots 7 and 8, where a "key on the braille display" is used to cycle through the various attribute types (underline, bold, etc.).
- Navigate to and activate buttons and other controls with the braille display, especially those that cannot gain focus using native keystrokes. For example, some code completion and information windows may not have this ability. Lee uses functions to quickly move the braille display to various parts of the screen such as "top of screen" or "line 6." This is typically done under flat review mode, and Lee uses the panning and navigation keys on the braille display to do this.

  When Lee presses touch cursors on the braille display, they behave as if a Lee were performing a single left mouse click on the given object associated with the touch cursor.
- Monitor an object on the screen. When Lee tells Orca to monitor an object by pressing a key in combination with cursor routing buttons on the braille display Orca will reserve an area on the braille display and will continually display all current information for this object.
- Set a part of the screen to jump to, such as a status bar. When Lee lands on an object of interest, Lee will instruct Orca to reserve a keystroke by pressing a key and a touch cursor on the braille display to jump back to the object without moving the active focus. Since Lee is a braille user, Orca also allows Lee to jump to these objects by pressing a combination of a braille display key and touch cursor to jump to a bookmark (e.g., cursor 1 goes to bookmark 1).

Compile code in NetBeans

Lee compiles a program module and fixes errors in NetBeans. Over the course of this task, Lee must be able to do the following:

- See all errors and warnings. For example, when Lee compiles a program module, focus will be moved to a window containing errors and warnings. Focus tracking should behave properly here, and the braille display will show the first line in the errors and warnings window.

  When Lee has speech enabled, Orca will speak the first line of the message window.
- Click on an error with a touch cursor to go to the error. This acts just like performing a left mouse click on the error to take Lee to the editor window and source line for the error. Focus tracking behaves properly here, and the braille display will show the line containing the error.
- When reviewing the screen, Orca compresses white space on the braille display to allow Lee to review more quickly. For example, when there are only blanks to the right and Lee presses the pan-right button on the braille display, Orca will take Lee to beginning of the next non-blank line. Orca also indicates positional information by displaying optional "beginning of line" and "end of line" characters which are standard well known characters on the braille display. Orca also optionally shows vertical screen position in status cells. Orca also optionally plays sound effects to indicate line changes.
- Run the application being created and access it. General screen reading functionality applies here.

Compose document in StarOffice

Lee writes rough draft of an architecture document in StarOffice. Over the course of this editing, Lee must be able to do the following:

- Detect text selection. When Lee selects text in the document, Orca uses dots 7 and 8 on the braille display to indicate the text selection. Note that this will hide any capitalization held in dot 7, but that's OK.
- Use spellcheck to determine misspelled word and read list of possible replacements. NOTE: Lee wants to be able to see both the spellcheck dialog and the misspelled word in context. When spellcheck dialog comes up, Orca provides a well known command to say "read the word in its context". Speech will speak it. Braille will display it, and any other action snaps Lee back to the spellcheck dialog.
- Read documents in grade two braille. Lee uses a well known command to toggle between grade one and grade two braille.

  TODO: Perhaps not a 1.0 requirement.
- Detect attributes of text. Lee does this in the same way as described in "highlighting, attributes and coloring" above.
- Easily comprehend tabs and spacing. Less does this in the same way as described in "determine proper indenting" above.
- Bring up a list of toolbar items to quickly access with the keyboard. For example, some applications are well-behaved and let you move to the toolbar. Others do not. Orca provides a well known command that takes Lee to toolbar navigation mode. Another well known command (e.g., escape) gets you out of this mode.

  NOTE: this may also be workable via review mode, but the screen reader always attempts to provide Lee with the most efficient means for accomplishing any task.
- Review the currently visible window with the braille display without moving the active cursor (e.g., the caret or the object with focus). Lee uses Orca's flat review

mode to do this, and Orca's flat review mode also allows Lee to select an object (and caret position in that object, if caret position applies).

- Navigate and read all prompts and controls (including static text) in dialogs such as "Save as..." both in flat review mode and in logical order.
- View labels on the same line as the control they refer to followed by the control type. NOTE: Orca automatically does this in braille when in focus tracking mode, but will not do this when in flat review mode.
- Obtain "where am I" info. In focus tracking mode, Orca's braille display always tells Lee this (e.g., it displays "file open dialog, filename text area, role" on the braille display - see the braille specification document for more information). Updates to text areas cause automatic panning of braille so as to keep as much of text area on display as possible, with weight given to the caret.

Read and write e-mail with Evolution

Lee reads and writes e-mail with Evolution to communicate with team. Over the course of this task, Lee must be able to do the following:

- Read all e-mail formats (e.g., plain text and html). Orca relies on Evolution to give it the proper AT-SPI object information for this.
- Track the message list and easily be able to determine such info as time, date and message priority. Orca places all this information on one line of the braille display if at all possible.
- Spellcheck an outgoing message as described above.
- See autocompletion information as it is appearing. Orca displays this information to Lee on the braille display and keeps the cursor at the current caret position.

  When Lee has speech enabled, the speech output also speaks the autocomplete values.

Use terminal windows

Lee also interacts with terminal windows on a daily basis. While BrlTTY is most likely the better solution for Lee (who is primarily a braille user), Lee may also want to access a GUI terminal on occasion. Lee needs to be able to do the following:

- Navigate the terminal window. Orca updates the braille display to track the cursor in apps such as emacs and vi. Orca also updates the braille display to track the navigation of fields in terminal apps (such as curses applications).

  TODO: Handling curses applications may be a lower priority task at this point.

Use the calendar tool

Lee sets up appointment for team design discussion using a calendar tool. Over the course of this task, Lee needs to be able to do the following:

- See relevant information about each time slot. Orca uses braille to show this relevant information (e.g., is that slot available and if not, what is scheduled?).
- See the prompts for each field when filling out an appointment. Orca displays all information for the current field on the braille display if it will fit. If it will not, Orca supports panning of the braille display. Orca also makes it possible to restrict

panning to the braille concept of the current element which means that the user can not pan away from the information about the current control.

Browses the web using Mozilla

Lee uses Mozilla to read JDK JavaDoc, the latest Java Tips and Tricks article, and the latest slashdot content. Lee needs to be able to do the following:

- View all links in a list view and also be able to activate them. Orca allows Lee to arrow through the links and hit first letter (or perhaps enter key) to activate them.

  TODO: might need to be concerned about conflict with web forms with buttons such as "Submit All Changes" that will react to presses of the enter key.
- Jump past links on the page to get directly to large blocks of text. Orca provides a well known command to do this. This is useful for reading documentation and articles where Lee is reading one page after another.
- Select, cut, copy and paste blocks of text from a web page. Orca relies on the semantics of the application to do this.
- View the URL of imagemap links with no useful information. Orca provides this as part of one of its navigation modes. Ideally, the browser will allow any user to navigate to any link, thus allowing Orca's focus tracking mode to handle this. If not, however, Orca's flat review mode will provide this functionality.
- See labels properly associated with input fields on the braille display; relevant table information should be shown on the display when moving between columns and rows.
- Move back and forth between headers (e.g., H1, H2, etc), tables, and frames. Orca relies on the built-in keyboard navigation of Mozilla to do this.
- Distinguish links from regular text. Lee does this in the same manner as described in the "highlighting, attributes and coloring" section above.

Pat the Manager

Pat is a manager who is primarily a magnifier and speech user and typically performs the tasks outlined in the following sections:

Browses the web using Mozilla

Each morning before going to work, Pat uses Mozilla to read the daily news from CNN. To do so, Pat needs to do the following:

- Jump past links on the page to get directly to large blocks of text. Orca provides a well known command to do this. This is useful for reading documentation and articles where Pat is reading one page after another.
- Select, cut, copy and paste blocks of text from a web page. Orca relies on the semantics of Mozilla to do this.
- View the URL of imagemap links with no useful information. Orca provides this as part of one of its navigation modes. Ideally, Mozilla will allow any user to navigate to any link, thus allowing Orca's focus tracking mode to handle this. If not, however, Orca's flat review mode will provide this functionality.
- Hear and see links, menu items, and buttons as Pat navigates to them using Mozilla's built-in keyboard navigation methods.

- Move back and forth between headers (e.g., H1, H2, etc), tables, and frames. Orca relies on the built-in keyboard navigation of Mozilla to do this.
- Distinguish links from regular text. Pat does this in the same manner as described in the "highlighting, attributes and coloring" section above.
- Have the document spoken starting from the current position. Orca provides a well known command to to start reading at the current position and stop when desired. Orca also automatically tracks and magnifies the text while it is being read. When reading is stopped, Orca places the caret (if possible) at the end of the last word spoken.
- Pat uses Orca's option of using alternative voice styles to indicate various attributes of the text. Orca also provides a well known command to speak all the attributes of the text at the cursor (e.g., "bold underlined 12-point helvetica").

Read and write e-mail with Evolution

Pat is also a heavy e-mail user, especially to communicate with Pat's staff and boss. As such, Pat needs the following:

- See and hear the highlighted item in the message list. Orca also gives Pat the ability to hear descriptive information about the currently highlighted item. For example, Pat can determine information such as if the item is unread and/or has an attachment.
- Hear and see links, menu items, and buttons as Pat navigates to them using Evolution's built-in keyboard navigation methods.
- Have the document spoken starting from the current position. Orca provides a well known command to start reading at the current position and stop when desired. Orca also automatically tracks and magnifies the text while it is being read. When reading is stopped, Orca places the caret (if possible) at the end of the last word spoken.
- Pat uses Orca's option of using alternative voice styles to indicate various attributes of the text. Orca also provides a well known command to speak all the attributes of the text at the cursor (e.g., "bold underlined 12-point helvetica").
- Change the color of the magnified area. If the full screen is magnified, the entire screen will have the same scheme. If the lens view is on, then only the lens has that color scheme. This is important because Pat sometimes loses the location of the lens and the contrast between the magnified area and rest of the screen helps Pat find it.
- See and hear prompts and controls in dialogs. For example, when a search dialog comes up, Pat sees the focused component in the magnified display and hears "subject: edit."
- Use one or more well known commands to quickly change magnification level and style. Magnification styles include things such as: magnification lens that moves around display and fixed magnification window that follows mouse and focused object. For following the mouse, Orca also provides additional options such as "mouse centered", "optimal least movement", etc.
- Commands to turn magnification tracking on and off.
- Use multiple zoomers to track both the "locus of focus" as well as other areas of the screen (e.g., status bar).

Compose presentation in StarOffice

Pat periodically needs to create an "All Hands" presentation in StarOffice. To do, Pat needs the following:

- Select from the slide templates. This might require review mode.
- Determine the position on the slide that typed text will appear.

  TODO: is this perhaps more for someone who doesn't use magnification?
- Determine if the slide is full or if part of text or an object won't fit.

  TODO: is this perhaps more for someone who doesn't use magnification?
- Cut and paste objects and text from and to a slide.

  TODO: the application should provide this, but Pat will still need to be able to determine what is selected. How will Pat do this?
- Move and size objects on a slide.

  TODO: is this perhaps more for someone who doesn't use magnification?
- Determine bullets and indentation levels for text.

  TODO: is this perhaps more for someone who doesn't use magnification?
- Pat sometimes invokes a spreadsheet while writing documents. When doing so, Pat brings up multiple zoomers that place cell location, formulas, etc., in a small area on one of the corners of the screen. Orca allows the location of the zoomers and what the zoomers track to be completely customized.

Read help documentation

Pat needs to be able to read the help documentation, and often needs to do the following:

- Navigate the contents magnified and with speech. If a topic has sub-topics Orca makes it clear by speaking so when the topic is highlighted. Speech also tells Pat what level in the help tree Pat is in.
- See and hear each content pane. Orca automatically reads each content pain when it gains focus and magnifies each word/line as it is read.

Compose text document in StarOffice

Pat also writes performance reviews in StarOffice. To do so, Pat often performs the following tasks:

- Use spellcheck to determine misspelled word and read list of possible replacements. By default, Orca speaks the misspelled word followed by the default choice. NOTE: Pat often wants to be able to see the misspelled word in context. When spellcheck dialog comes up, Orca provides a well known command to say "read the word in its context". Speech will speak it and the magnifier will show it. Any other action snaps Pat back to the spellcheck dialog.
- Pat uses Orca's option of using alternative voice styles to indicate various attributes of the text. Orca also provides a well known command to speak all the attributes of the text at the cursor (e.g., "bold underlined 12-point helvetica").
- Hear current character, word and line. Orca provides one or more well known commands to do this. When the well known command is pressed a second time, Orca spells the word. When the well known command is pressed a third time, Orca spells the word phonetically.

- Hear capitalization. For example, when arrowing across the name "Mike" Orca says 'cap m' 'i' 'k' 'e'. Optionally, Orca changes the pitch for the capital letter.
- Bring up a list of toolbar items to quickly access with the keyboard. For example, some applications are well-behaved and let you move to the toolbar. Others do not. Orca provides a well known command that takes Lee to toolbar navigation mode. Another well known command (e.g., escape) gets you out of this mode. NOTE: this may also be workable via review mode, but the screen reader always attempts to provide Lee with the most efficient means for accomplishing any task.
- Have dialog static text and focus information should automatically be read if a dialog such as "save" or "print" appears. Orca also provides commands to read this information as well as the entire dialog.
- Detect text selection by both seeing the text in the magnifier and also hearing "selected" when navigating character-by-character or word-by-word. Orca also provides a well known command (e.g., "say selection") to read the currently selected text.

  TODO: what happens when there are multiple discontiguous regions selected?

Use address book

Pat uses his address book often to locate contact information. To do so, Pat needs Orca to do the following:

- Track the address list, speaking and magnifying the currently selected person.
- Speak and magnify all controls in the "find" utility as Pat navigates them.
- Hear details about an individual contact. When navigating the found contact, Pat needs to hear and see the label and content of each field. Orca also provides a well known command to re-read the current label and roll for the focused item.

Use the calendar tool

Pat sets up a staff meeting using a calendar tool. See Lee's stuff above, but apply to speech and mag.

TODO: need to flesh this out some more.

Kim the Admin

Kim is an administrative assistant who is primarily a speech user. Kim regularly performs the tasks outlined in the following sections.

Use the address book

Kim often uses the address book to add and find contact information for Pat. As such, Kim needs to be able to do the following tasks:

- Track the address list, speaking the currently selected person.
- Speak all controls in the "find" utility as Kim navigates them using the system's built in keyboard navigation commands.
- Hear details about an individual contact. When navigating the found contact, Kim needs to hear the label, content, and roll of each field. Orca also provides a well known command to re-read the current label and roll for the focused item.

- Detect text selection by hearing "selected" when navigating character-by-character or word-by-word. Orca also provides a well known command (e.g., "say selection") to read the currently selected text.
- Read the title and status bar of the current application. Orca provides this via a well known command to invoke the "Where am I" functionality of Orca.

Use the calendar tool

Kim often sets up calendar appointment for Pat and others. To do so, Pat needs Orca to provide the following abilities:

- Track the date and time views of the calendar. When navigating, Kim hears, for example, the current time and if an appointment is currently set.

  TODO: Mike - write down your ideas for this!
- Hear the label for each field and review the information that has been input.
- Read the entire appointment. Orca provides this via a well known command to read an entire dialog.

  TODO: this assumes the appointment information is input via a dialog.

Browses the web using Mozilla

Kim often orders office supplies and books travel for others on line. Kim does this in a manner similar to Pat, but using only speech instead of speech and magnification. [[[*TODO:* - probably need to flesh this out more.]]] In addition, since Kim often fills out forms, Orca needs to allow for the following:

- Keystrokes to move between form fields only. Orca provides functionality to move from one form field to the next or prior skipping all other links.
- Automatically read field labels when moving to form fields.

Read and write e-mail with Evolution

Kim often coordinates and organizes group meetings using e-mail. As such, Kim needs to be able to do the following:

- Hear contacts in the address book and know if they are selected.
- Use spellcheck to determine misspelled word and read list of possible replacements. By default, Orca speaks the misspelled word followed by the default choice. NOTE: Kim often wants to be able to hear the misspelled word in context. When spellcheck dialog comes up, Orca provides a well known command to say "read the word in its context". Speech will speak it.
- Kim uses Orca's option of using alternative voice styles to indicate various attributes of the text. Orca also provides a well known command to speak all the attributes of the text at the cursor (e.g., "bold underlined 12-point helvetica").
- Hear current character, word and line. Orca provides a well known command to do this. When the well known command is pressed a second time, Orca spells the word. When the well known command is pressed a third time, Orca spells the word phonetically.
- Hear capitalization. For example, when arrowing across the name "Mike" Orca says 'cap m' 'i' 'k' 'e'. Optionally, Orca changes the pitch for the capital letter.

Creates spreadsheet in StarOffice

Kim sometimes views and updates the group budget, which is maintained as a spreadsheet in StarOffice. To do so, Kim needs to be able to do the following:

- Automatically hear cell location and content when navigating between cells. Orca speaks, for example: "B4" followed by the cell content. Orca also provides a well known command to speak this information. Orca also provides an option to automatically speak this information when Kim moves from one cell to the next.
- Related to the above, hear row and column headers. Orca supports this via separate well known commands.
- Automatically hear formulas if they appear in a cell. Orca speaks these after it speaks the location and content information.
- Determine and modify size of cell.

  TODO: this might require some emulation of mouse behavior if StarOffice doesn't provide dialogs to do this.
- Select a range of cells and know that they are selected. Orca speaks "selected" followed by the begin and end coordinates.
- Produce and read simple charts and graphs.
- Interact with a print dialog to print finished report.
- Automatically hear static text and focus information when a dialog appears such as "file" or "reformat." Orca also provides commands to re-read this information as well as the entire dialog.

Sam the IT Guru

Sam is a member IT staff who is primarily a speech and braille user. Sam regularly performs the following tasks.

Use terminal windows

Sam regularly interacts with terminal windows. While BrlTTY is most likely the better solution, Sam may also want to access a GUI terminal to do the following:

- Navigate the terminal window. Orca updates the braille display to track the cursor in apps such as emacs and vi. Orca also updates the braille display to track the navigation of fields in terminal apps (such as curses apps), and provides a "speak the current line" to display this information.

Use GUI-based administration tools

Sam interacts with GUI-based tools such as update and network configuration on a daily basis. This includes the following:

- Automatically hear static text and focus information when a dialog appears such as "configure network." Orca also provides commands to re-read this information as well as the entire dialog.
- In logical order, navigate and read all prompts and controls (including static text).
- See all information for the current field on the braille display if it will fit. If it will not, Orca supports panning of the braille display. Orca also makes it possible to

restrict panning to the braille concept of the current element. This will restrict panning from moving away from the braille information presented for the current control.

- Re-read information as requested.
- Read an entire dialog with speech.
- Watch a particular field in a dialog box (e.g., the "% CPU utilization" field and be notified when it changes.

  TODO: how does Sam specify what to watch and how does Orca notify Sam when the value changes?

Chapter 3. User Requirements

Based upon discussions with end users as well as the tasks and ideas from the personas, Orca must supply at least the following end-user features.

Availability at All Times

Orca will often be the only vehicle by which many users will be able to access the system. As such, it must be available at all times, including at system login and screensaver screens.

Orca must also be available after the system returns from "standby" or "sleep" mode.

Finally, in the event the user's choice of speech synthesis engine becomes unavailable (e.g., expired license), Orca must attempt to find and use an alternative synthesis engine.

Failure Resiliency

In the event that Orca fails or a component of the system that Orca depends upon fails, the system should be able to heal (and perhaps restart) itself appropriately.

Seamless Interaction with Traditional Keyboard Navigation Methods

Orca must allow users to navigate through the desktop and applications on the desktop using the system keyboard navigation gestures (e.g., Alt+Tab to select the next window). In other words, Orca must not interfere with traditional keyboard navigation.

Seamless Interaction with Other Assistive Technology

It is not uncommon for users to simultaneously use other assistive technologies, such as the AccessX features of XKB [*XKB*>], to access their displays. As such, Orca must be able to co-exist (i.e., not interfere) with other assistive technologies in use by the user.

Consistent Style

It is expected that access to applications will be driven primarily via customized "scripts," with a fallback ("default") script to be used in the absence of a customized script.

While each script can provide dramatically different access to an application, it is expected that scripts will provide a users with a consistent style to access applications. To help with this consistent style, Orca will provide a style guide, a well-documented "default" script, and several application scripts that demonstrate and promote this style.

Customizable Behavior Per Application ("Scripting")

It is expected that the default behavior will provide reasonable access to all applications that use the AT-SPI. However, to provide dramatically improved access, Orca must be able to provide customized behavior for individual applications.

For example, one can envision a script for an e-mail application that can provide prioritized access to one's inbox. Another example may be that the script provides keyboard access to select and copy displayed text to the system clipboard in the event the application doesn't support this (e.g., the only way to select text in a terminal window is to use a mouse - a script might create new keybindings to allow a user to do this from the keyboard).

Configurable Presentation and Interaction

Different users have different capabilities (e.g., some may be able to hear some synthesis voices better than others; some may use braille while others do not) and desires (e.g., some may prefer faster speaking rates). As such, the general manner and means by which Orca presents information to the user must be configurable by the end user.

Users must also be able to change configuration settings while Orca is running, including the ability to turn presentation modes (i.e., speech, braille, magnification) on or off without the need to restart Orca.

Focus Tracking as the Default Mode

Known as "focus tracking," Orca must provide a mode to track the current region of interest, which is usually the object that has keyboard focus.

When first starting up and when the region of interest has changed, Orca must provide a short summary of the region of interest, and must also provide a mechanism for the user to obtain more detailed information. The presentation will follow the style guide.

When navigating text areas in a character-by-character or word-by-word fashion (or any linear fashion for that matter), Orca must provide an option to play an audio cue when the caret crosses line boundaries.

Flat Review Mode

Orca must provide the ability for a user to review the contents of the desktop or a given application. This is typically done by the user making a well known command indicating "review," with Orca reacting by presenting the area to be reviewed. At any time, the user can interrupt the review mode, causing presentations such as speech output to stop immediately.

Another option for review mode includes the ability to use a set of well known commands to quickly skim the contents of the display.

The review of the desktop will follow the style guide, and will at least provide a short summary of the available applications. While the review of an application will also follow the style guide, the overall presentation depends largely upon the application being reviewed.

TODO: Add notions of "hierarchical review," and "hybrid review (e.g., in/out of lists)". Need to flesh out the primary purposes of each.

TODO: Add ability to read a dialog/window.

TODO: As with navigating text, should Orca have an option to play an audio cue when the "next" or "prev" object command takes user to a new line?

Presentation of Text

For any given piece of text, Orca must provide the ability to speak the current character, word, or line. When speaking the current word, Orca must provide the ability to spell it out in either letter-by-letter or phonetic (e.g., military spelling) mode. When speaking a line or set of lines, Orca must offer the ability to verbalize punctuation.

Orca must provide the ability to present the following to the user:

- *Attributes* - font size, face (bold, italic, etc.), underline, color, etc.
- *Capitalization* - is it capital or lower case?
- *Selection* - is the text selected or not?
- *Indentation* - what is the indentation level?
- *Bounds* - what is the bounding box of the text (where and size)?

Learning Mode (Command Echo)

Orca must provide a feature that optionally presents the command to be invoked when the user presses a key on the keyboard or braille display. All key events in this mode will be consumed by Orca, thus allowing the user to freely type anything to learn what the effect of the key will be. A command being spoken must be able to be interrupted at any time.

Customizable Gestures

While Orca will provide a default set of well known keyboard and braille input gestures, users must be able to override these gestures and extend them with gestures of their own choosing.

The user interface for defining these gestures must be easy to use.

"List All Applications" Command

Orca will provide a means for a user to determine all applications running on the desktop as well as all open windows on the desktop.

TODO: the Alt+Tab functionality of the desktop may be sufficient to meet this requirement.

"Where Am I?" Command

Orca will provide a means for a user to determine information about the current object of interest, including the object itself, which window it is in, which application, which workspace, etc.

The object of interest can vary depending upon the navigation mode the user is using at the time. For example, in focus tracking mode, the object of interest is the object with keyboard focus. In review mode, the object of interest is the object currently being visited, whether it has keyboard focus or not.

"Jump to Well Known Object" Commands

There are common locations a user wants to jump to at any given time. These include the title bar, the status bar, the toolbar, the beginning of a window, the end of a window, the system applet tray, etc. Orca will provide well known commands for Orca to enter review mode and set the current object of interest to one of these objects. From this point, the user can use a well known command to go back to the object with focus, give the jumped-to object focus if possible, or perform the equivalent of a mouse click on the jumped-to object.

"Find" Command

Orca will allow users to enter a string to search for a visible text string in the current window or entire desktop. If the string is found (e.g., the string matches the accessible name of an object or the string matches a string in a text area), Orca will switch to "review mode" and make the associated object the object of interest. From this point, the user can use a well known command to go back to the object with focus, give the found object focus if possible, or perform the equivalent of a mouse click on the found object.

In the event the user is a magnification user, the magnifier will moved to the found object.

The search will be performed in the text flow of the locale (e.g., left-to-right, top-to-bottom) from the current point of interest, wrapping back to the beginning of the window if necessary.

The search will provide options for requiring a full word match (e.g., "ok" will not match "token") or substring match (e.g., "ok" will match "token").

Finally, the "find" command will also allow the user to search for text based on attribute (e.g., "bold"), and the search will allow the user to search based solely on attribute type, text string, or a combination of both.

NOTE: this feature is for searching for visible text only. Traditional searching for text in documents will be supported by application functionality (e.g., the "search" menu item in the application.

Watched Objects

Orca will allow users to indicate interest in an object. Once interest has been given, Orca will notify the user of changes to that object, whether it or its window has focus or not. Orca will also provide an option to automatically enter review mode and make the watched object the object of interest. From this point, the user can use a well known command to go back to the object with focus, give the watched object focus if possible, or perform the equivalent of a mouse click on the watched object.

In the event the user is a magnification user, the magnifier will moved to the found object. Alternatively, the user may have set up a separate "zoomer" to watch the object. In this case, the zoomer will always reflect the current state of the watched object.

Bookmarked Objects

Orca will allow users to define a gesture (keyboard or braille) that will tell Orca to enter review mode and make a particular object the object of interest. From this point, the user can use a well known command to go back to the object with focus, give the bookmarked object focus if possible, or perform the equivalent of a mouse click on the bookmarked object.

In the event the user is a magnification user, the magnifier will moved to the bookmarked object.

Document Reading

Orca must provide the ability for a user to read the contents of a document, such as e-mail or a word processing document. As the document is read, Orca will instruct the associated application to scroll so as to keep the portion being read visible on the screen (and magnifier). The invocation of the document reading will be triggered via a well known command and may be stopped at any time by the user. When the review stops, the caret will be positioned at the end of the last word spoken (if possible - this depends upon the capabilities of the text area as well as the speech engine).

Orca will also provide options to highlight the text being spoken.

NOTE: while the reading of a document will follow the style guide, the overall presentation depends largely upon the document being read (e.g., is it a text document, spreadsheet, web page, etc.?).

Ability of Non-focused Windows to Present of Information

Orca must allow for the presentation of information of objects that do not have keyboard focus. While the presentation will typically be information on objects that the user has requested interest in, it may also come from custom scripts that have decided it is important for them to present something. Examples of such information include announcing the status of a progress bar, announcing the subject/sender of incoming e-mail, etc.

Speech Synthesis

One of the primary non-visual ways to present a graphical display to a user is via speech synthesis. Note that Orca need not be a speech synthesizer, but it must be able to drive one. The most important functionality Orca needs for speech synthesis are as follows:

- *General Synthesis* - Orca must be able to speak an utterance (e.g., a word, a label, a sentence, etc.) or set of utterances (e.g., a paragraph or entire document). The utterance being spoken must be able to interrupted at any time.

  IMPLEMENTATION DETAIL: if at all possible, the synthesis engine should be able to tell Orca what has been spoken, allowing Orca to synchronize its internal data structures with what has been presented to the user via speech.

- *Voice Styles* - Orca must be able to identify and allow the user to select between the voices available on the various synthesis engines available on the system. In addition, Orca must allow the user to customize parameter settings for the voices, such as average pitch, speaking rate, and volume. The combination of {voice, pitch, rate, volume} will be known as a "voice style." Orca will provide the user with the ability to select the voice styles to be used for various speaking operations (e.g., default, uppercase, warning, alarm, etc.), and may also provide unique "voice style sets" depending upon the navigation mode being used (e.g., focus tracking, review mode, etc.). At a minimum, Orca will support a "default" style to be used for the majority (if not all) of the speaking operations.

- *Speaking Rate Modification* - While the speaking rate will generally remained fixed once configured, users may sometimes which to speed up or slow down the presentation of speech. While it is ideal that user can do this while the synthesis en-

gine is speaking, such functionality is typically not provided by the majority of speech synthesis engines. Orca must, however, allow the user to change the speaking rate that will be used for the next utterance, should the underlying engine support changes to speaking rate.

- *Multilingual Text* - Orca should provide the ability to speak a single utterance that contains words or phrases from multiple locales. This is an emerging area for speech synthesis engines, however, so Orca will support this feature if the underlying engine(s) support it.
- *Spelling Mode* - Orca must be able to optionally spell out words, either letter by letter or by military (e.g., alpha, bravo, charlie) spelling.
- *Verbalized Punctuation* - Orca must be able to optionally verbalize punctuation.
- *Repeated Character Count* - Orca must be able to optionally compress the repetition of character by saying something such as "25 dashes" instead of "dash dash dash dash dash..."

Key and Text Echo

Orca must provide a set of options to allow the user to enable the automatic speaking of keys as they are typed, alphanumeric characters as they are entered/deleted, and words as they are entered. The speech much be interruptable at any time.

Refreshable Braille

Braille is another primary non-visual presentation mode for screen readers. As with speech synthesis, Orca need not directly support a braille display, but it must be able to drive one. The primary end requirements for a braille display are as follows:

- *Work with BrlTTY* - it is not expected that users will abandon character cell access to their virtual consoles. To preserve this access (which will be done via BrlTTY), Orca will not interrupt or alter the behavior of BrlTTY.
- *Effective use of space* - Orca should try to use the cells on the braille display as effectively as possible. This will be driven primarily by the style guide, and can also include the the option to effectively compress white space.
- *Effective use of input buttons* - Orca should use any input buttons on the display as effectively as possible, with the default behavior being the equivalent effect of the BrlTTY bindings for those buttons. These functions include panning the braille display as well as driving the flat review mode. The functions also include the ability to quickly toggle features on and off, such as: toggle compression of whitespace, toggle attributes to be shown using dots 7-8 or status cells, toggle between grade I and grade II, etc.
- *Effective use of cursor routing keys* - Orca should enable the cursor routing keys to be used to click on objects, set slider values, etc.

  In the event there is a second set of cursor routing keys, Orca will attempt to make effective use of them (e.g., pressing them tells Orca to relay text attribute information via speech).
- *Effective use of input button and cursor routing key combinations* - Orca should allow the user to use combinations of input buttons and/or cursor routing keys to perform operations such as selecting text, jumping to bookmarks, etc.

  NOTE: selection of text using combined keyboard and braille actions (e.g., holding the shift key while pressing a cursor routing key) may also be considered.

  NOTE: input of text using chording of braille keys might be considered, but is not a high priority.

- *Cursor options* - Orca should provide the user with the ability to show the cursor or not. If shown, the user should have the ability to make it blink, and should also have the ability to tell it to either take the entire cell or just dots 7-8.
- *Ability to have multiple pan regions on the display* - watched objects may need to be able to remain on a static area of the display (e.g., the rightmost 10 cells). The braille support should enable this and should also allow the user to pan either the "main" display or the watched area.

Synchronization with Screen Magnifiers

In addition to co-existing with each other, a screen magnifier and Orca must be in sync with each other. For example, when Orca is reviewing an area of the screen that is larger than the screen magnifier can display at once, Orca must inform the screen magnifier that the region of interest has changed.

Acceptable Response Time

Orca must not degrade the perceptible performance of the system. That is, user should be able to detect any decrease is responsiveness of the desktop when the Orca is being used. In addition, a user's interaction with Orca should appear as crisp and as lively as normal interaction with the display via traditional interfaces (e.g., the keyboard).

Acceptable performance of the speech synthesis output is very important. Orca must be able to provide speech synthesis that meets or exceeds the following performance metrics:

- *Time to First Sound* - The time between when a speech synthesizer gets a request to speak and when the synthesizer actually starts speaking must be minimal (e.g., less than 30ms).
- *Time to Cancel* - Orca must be able to cancel speech synthesis in progress, and the time to cancel must be minimal (e.g., less than 30ms). Furthermore, the time between when a cancel is issued and the time the next utterance is to be spoken must be minimal (e.g., less than 30ms).

All updates to the braille display should occur within 50ms of the time the update command was issued.

The magnifier should offer smooth performance and show no visible lag.

> **Performance Scope:** It is understandable that much of the response time may be due to factors outside the control of Orca (e.g., Bonobo and the underlying speech engine). As such, the primary responsibility of Orca and each application script is to process AT-SPI and keyboard events as quickly as possible.

Documentation and Tutorials

Although it is a reasonable goal that Orca should attempt to achieve, users cannot be expected to be able set up and use Orca without documentation. Like other systems, such as JAWS, Orca must provide documentation and tutorials on the installation, configuration and use of Orca. This documentation must come in form(s) that are accessible to people who need to use the screen reader (e.g., accessible text and audio).

Chapter 4. Default Interaction Styles

Associated with the discussion of the personas was the notion of a need for consistent and efficient interaction methods for Orca. This notion resulted in the development of default interaction styles for speech synthesis, braille input and output, magnification, and keyboard control of Orca. The following sections describe these default interaction styles in more detail.

Input Style: Keyboard Mappings

This section describes the default keyboard mappings for the focus tracking and flat review modes of the Orca screen reader. Note that Orca allows these mappings to be overridden and extended, allowing users to define their own keyboard mappings.

As with most other available screen readers, the numeric keypad is the primary location for Orca keystrokes. Where appropriate, the keymap also provides letter-based mnemonics that have been carefully chosen so as to make Orca easier to translate into other languages. Furthermore, the default mappings have been developed with a minimalist approach so as to allow custom scripts to override and extend the default keyboard behavior while reducing conflict with the default keybindings.

Orca will also automatically invoke flat review mode when a command is issued to review any part of the screen that does not have the keyboard focus.

Finally, when in flat review mode, Orca will optionally allow the mouse to follow the object of interest. *TODO:* this probably should be moved to the user requirements instead of the key mappings.

NOTE: Ideally, a screen reader should provide good access on a laptop, which typically means using the numeric keypad is an inconvenient access method. As appropriate, Orca's default keyboard mappings may change to reflect this ideal goal; alternatively, Orca may provide another set of keyboard mappings optimized for laptop use.

In brief summary, the default keyboard mappings break the numeric keypad into the following logical sections:

- The top row is for mouse clicks, managing focus, and switching review mode between focus tracking and flat review.
- Keypad keys 7, 8, and 9 are for navigating lines
- Keypad keys 4, 5, and 6 are for navigating words
- Keypad keys 1, 2, and 3 are for navigating characters

In more detail, the keyboard mappings are as follows, and use the "INSERT" key as a modifier (note that Orca will most likely allow the user to define a different key to use instead of "INSERT"):

- "KEYPAD_/": performs a left mouse click on the current object of interest
- "KEYPAD_*": performs a right mouse click on the current object of interest
- "KEYPAD_-": when in flat review, returns to the object with keyboard focus (note that any user action that causes the screen or focus to change will also do this). If the user is currently in focus tracking mode, pressing this key activates flat review mode.
- "KEYPAD_+": reads from current position to end of document. If any key is pressed afterwards, reading stops and the appropriate object is given appropriate focus (e.g., the text caret will be placed on the last word spoken). If the user is in a dialog box, pressing this key reads the dialog box in a logical order.

- "KEYPAD_ENTER": performs "where am I?" If this key is pressed twice selection information will be spoken ie all the selected items or text.
- "KEYPAD_.": performs a "screen find." Pressing this key brings up a dialog where a user can enter a search string and decide where on the screen the search should begin IE from the top or current position.
- "KEYPAD_7": moves the review position to the prior line, landing on object closest to current object
- "INSERT+KEYPAD_7": moves the review position to the beginning of the current line
- "KEYPAD_8": speaks current line. If flat review has been activated, this acts on the line at the review location. If the user is in focus tracking mode, the line or item with focus is spoken. When this key is pressed twice quickly the line is read with formatting and capitalization details.
- "KEYPAD_9": moves the review position to the next line
- "INSERT+KEYPAD_9": moves the review position to the top of the review area.
- "KEYPAD_4": moves the review position to the prior word or item
- "KEYPAD_5": speaks the current word or item. If this key is pressed twice quickly the word is spelled and any capitalization is be announced; three times results in phonetic spelling
- "KEYPAD_6": moves the review position to the next word or item
- "KEYPAD_1": moves the review position to the prior character
- "INSERT+KEYPAD_1": moves the review position to the end of the line
- "KEYPAD_2": speaks the current character. If this key is pressed twice quickly the character is pronounced phonetically (if it is a letter)
- "KEYPAD_3": moves the review position to the next character
- "INSERT+KEYPAD_3": moves the review position to the bottom of the review area
- INSERT+s: speaks the status bar (if there is one)
- INSERT+t: speaks the title of dialog or app
- INSERT+v: brings up a verbosity preference dialog. Example settings include whether or not to speak indenting when working with text, speaking capitalization, speak the role of items, speak font information if it changes, etc.
- INSERT+RIGHT_ARROW: increases speech rate
- INSERT+LEFT_ARROW: decreases speech rate

Input Style: Braille Mappings

This section describes the default braille mappings for the focus tracking and flat review modes of the Orca screen reader. Note that Orca allows these mappings to be overridden and extended, allowing users to define their own braille mappings.

Where at all possible, the default braille mappings for Orca will map to their equivalent action in BrlTTY. This permits users to have a consistent experience with their braille display when using virtual character-cell consoles on their machine as well as the GUI display.

The braille key functionality includes the following:

TODO: need to map these to BrlTTY commands.

- Pan left: BRL_CMD_FWINLT
- Pan right: BRL_CMD_FWINRT

- Line up: BRL_CMD_LNUP
- Line down: BRL_CMD_LNDN
- Beginning of line: BRL_CMD_LNBEG
- End of line: BRL_CMD_LNEND
- Top of window: BRL_CMD_TOP_LEFT
- Bottom of window: BRL_CMD_BOT
- Return to focus: BRL_CMD_HOME
- Cursor routing/clicking
- Text selection: accomplished by pressing a touch cursor at the beginning and end of the region that the user wishes to select in conjunction with another well known key on the display.
- Set watched area:
- Set Bookmarked area:
- Go to bookmarked area:
- Go to watched area:

In addition, the following commands will be used to cycle between braille features:

- Toggle space compression: BRL_CMD_SLIDEWIN. Toggles whether or not spaces are compressed when navigating with the braille display
- Toggle grade two on and off:
- Toggle what attributes if any are displayed:
- Enter verbosity mode: BRL_CMD_PREFMENU
- Toggle keyboard learn mode: BRL_CMD_LEARN
- Toggle display style: toggles between the default view, which is the logical presentation of the focus and the contents of the current line [[[WDW - ???]]]

Output Style: Speech Synthesis and Braille

This section provides output styles for each relevant Accessible role. *TODO:* this is a work in progress and will be completed via an iterative prototype process that includes feedback from end users.

General Braille Style

For braille, there are two verbosity levels. The first is a more verbose level for use with larger braille displays and for those users who are not expert with the Gnome desktop environment. The second more brief mode is for those who are using a smaller display or who are expert with the Gnome environment. With braille, Orca should always make an effort to show as much of the actual item with focus as possible, and the user should never be left with a completely blank display unless they are on a blank line in a text editor.

A general style for displaying braille is as follows: the line will be built up of several regions and an attempt will be made to start the most meaningful region at a given "homing position." The idea behind the homing position is that it will be the braille cell of the user's choice (e.g., the first cell of the display, the middle of the display, etc.) that the user can navigate to quickly. In the event that the line is too long for the physical display, the homing position will will apply and the line will be clipped to the left and/or right as appropriate; the user can use the panning actions to view the clipped information. The regions of the line follow a general pattern:

- Verbose: context label [value] rolename
- Brief: context label [value]

Where:

- context is information about the container with focus. Examples include "calculator window," "desktop," "save as dialog," "low disk space alert," etc. *TODO:* context can be composed of nested contexts (e.g., a page tab in a dialog); we need to decide what the context is (e.g., logical visual grouping of objects?)
- label is the label of the object, usually obtained by the name of the object or the text of the label that labels the object. Examples include "File," "OK," "First name:," "Volume:," etc. The label is generally viewed as the most meaningful region.
- [value] (optional) is the value of the object, and varies depending upon the object type. For example, "<x>" for a check box, "70%" for a slider, and "Mike Pedersen" for a text field.
- rolename is a localized string representing the name of the role of the object. The user can select between full rolenames (e.g., "check box," "menu item," "push button") or abbreviated rolenames (e.g., "chk," "mit," "pbt"). Note that the rolename may also appear in brief mode if the role of the object is not clear from the label and surrounding context.

The braille cursor (e.g., dots 7 and 8 on the braille display) will be used to indicate meaningful information as well. For example, the cursor will indicate caret position in text areas, and the cursor will also be used to indicate the menu item with focus.

Finally, the touch cursors on the display will "do the right thing." When in focus tracking mode, pressing any touch cursor associated with most objects will perform the default action for that object; the obvious exception is text areas, where the touch cursor will move the text caret to that object. *TODO:* define what happens in flat review - this might move keyboard focus to the object, perform a click, etc.

General Speech Style

For speech, there are two verbosity levels. The first is a more verbose level for use for those users who are not expert with the Gnome desktop environment or those who like to hear more information. The second, more brief mode, is for those who who are expert with the Gnome environment.

TODO: discuss general style.

Output Styles by Role

accelerator label (ROLE_ACCEL_LABEL)

An accelerator label is a short string that appears at the end of the text for a menu item (e.g., "CTRL+Z"), and defines the keystrokes that will invoke the action associated with a menu item.

EXAMPLE: gnome-terminal: the "Paste" menu item in the "Edit" menu has an accelerator.

KNOWN ISSUE: It would be nice to be able to get text bounds information for the accelerator text, but this doesn't appear to be possible with the existing implementations.

Braille

When presented, the accelerator will always be presented after the item it represents and before the marker that separates menu items. The braille text for the accelerator will appear in parentheses with no space before the opening parenthesis. Accelerator labels will only be presented in verbose mode, and will only appear for the item with focus. In the event that there are other things to show (e.g., check buttons or role names), the accelerator will appear to the far right.

- Verbose Example: view menu show mumble <x>(CTRL+M) _ zoom in _ ...
- Brief Example: view menu show mumble <x> _ zoom in _ ...

Speech

- Verbose Example: mumble menu item control m
- Brief Example: mumble

alert (ROLE_ALERT)

EXAMPLE: gtk-demo: "Dialog and Message Boxes" demo; click on the "Message Dialog" button. The "Information" window that appears is an alert.

KNOWN ISSUE: The AT-SPI makes a distinction between alerts and dialogs. But, it appears as though the role "alert" is applied to just about any dialog.

Braille

Any window, regardless if it is an alert, dialog, frame, or window, will appear as the context region on the braille line. The context string will consist of the title of the window followed by its role. The remainder of the braille line will consist of information describing the object with focus in the window. The information will follow the presentation rules for the given role type.

In the case of the "alert", the role of the context will be "alert."

- Verbose Example: low disk space alert
- Brief Example: low disk space alert

Speech

When a dialog pops up, its title will be spoken followed by relevant static text followed by the active object.

- Verbose Example: "low disk space dialog. you only have 10mb of disk space left. please delete some files. OK button.
- Brief Example: "low disk space dialog. you only have 10mb of disk space left. please delete some files. OK button.

animation (ROLE_ANIMATION)

An animation contains a moving or dynamic image.

EXAMPLE: Unknown.

Braille

TODO: should this follow a pattern of "label description role" to be more consistent with the generalized style?

- Verbose Example: animation: Will riding his bike.
- Brief Example: animation: Will riding his bike.

Speech

Orca will say the roll followed by any descriptive information.

- Verbose Example: animation: Will riding his bike.
- Brief Example: animation: Will riding his bike.

arrow (ROLE_ARROW)

An arrow is a 2D directional indicator.

EXAMPLE: Unknown.

KNOWN ISSUE: The AT-SPI does not appear to have a way to determine which way the arrow is pointing.

KNOWN ISSUE: gail/gail/gailarrow.c sets the role to ATK_ROLE_ICON.

Braille

An arrow will be represented by the direction the arrow is pointing followed by "arrow." Any touch cursor associated with the text will click the arrow. *TODO:* should the rolename "arrow" appear in brief mode?

- Verbose Example: left arrow
- Brief Example: up arrow

Speech

Orca will speak the direction followed by "arrow."

- Verbose Example: left arrow
- Brief Example: up arrow

check box (ROLE_CHECK_BOX)

A check box represents a choice that can be checked or unchecked and provides an indicator for the current state.

EXAMPLE: gnome-terminal: "Edit" -> "Current Profile..."; the "General" tab has a number of check boxes.

EXAMPLE: gtk-demo: "Tree Store" demo. Most table cells are check boxes. *KNOWN ISSUE:* the AT-SPI tells us these are tables cells, not check boxes.

Braille

The state of a check box will be represented by "< >" if it is unchecked and "<x>" if it is checked. Any touch cursor associated with the text will click the check box.

- Verbose Example: save password: < > checkbox
- Brief Example: save password: <x>

Speech

Orca will sat the label followed by the word "checkbox" followed its state.

- Verbose Example: save password checkbox checked
- Brief Example: save password checked

check menu item (ROLE_CHECK_MENU_ITEM)

A check menu represents a menu item that can be checked or unchecked and provides an indicator for the current state.

EXAMPLE: gnome-terminal: the "View" menu contains check menu items.

KNOWN ISSUE: It appears as though some toolkits allow menu items to have children (see the 'Menus' demo of the gtk-demo application - the radio menu items have sub menus). This needs to be taken into account.

Braille

All menus items from a single menu will be shown at the same time (see ROLE_MENU). The state of a check menu item will be represented by "< >" if it is unchecked and "<x>" if it is checked. (See also ROLE_ACCEL_LABEL). The touch cursors will result in activating the item.

- Verbose Example: show status bar < >(CTRL+B) checkmenu
- Brief Example: show status bar <x>

Speech

Like a checkbox, Orca will say the item followed by the word "checkmenu" followed by its state.

- Verbose Example: show status bar checkmenu checked
- Brief Example: show status bar checked

column header, table column header (ROLE_COLUMN_HEADER, ROLE_TABLE_COLUMN_HEADER)

A column header is a header for a column of data.

EXAMPLE: gtk-demo: "List Store" demo. The headers are table column headers.

Braille

A column header will be treated like a button (column headers are usually clickable, typically resorting the rows of the associated table), except it will be followed by "column heading" in verbose mode.

- Verbose Example: monthly income column heading
- Brief Example: monthly income

Speech

A column header will be treated like a button (column headers are usually clickable, typically resorting the rows of the associated table), except it will be followed by "column heading" in verbose mode.

- Verbose Example: monthly income column heading
- Brief Example: monthly income

combo box (ROLE_COMBO_BOX)

A combo box is a single line item that contains a list of choices the user can select from, and it can also contain editable text.

EXAMPLE: gedit: "File" -> "Open...". Both combo boxes are simple menu combo boxes.

EXAMPLE: gnome-terminal: "Edit" -> "Current Profile..." -> "Effects". The "Image file" combo box has a list in a scroll pane and a SINGLE_LINE EDITABLE text area as children.

EXAMPLE: gtk-demo: "Size Groups" demo. This creates a "GtkSizeGroup" dialog. Each combo box object in this dialog has a menu as its child.

Braille

The verbose and brief modes will show the label followed by the item followed by the word "combo." If the item is selected, it will be completely underlined with dots 7 & 8 to indicate selection. *TODO:* what do we do if the item is editable text?

- Verbose Example: time zone: GMT-8 pacific combo
- Brief Example: time zone: GMT-8 pacific combo

Speech

The verbose and brief modes will speak the label followed by the item followed by the word "combo." If the item is selected, the word selected will be spoken. *TODO:* what do we do if the item is editable text? When a combo box has focus and its selection changes, just the new selection will be spoken.

- Verbose Example: time zone: GMT-8 pacific combo selected
- Brief Example: time zone: GMT-8 pacific selected

desktop icon (ROLE_DESKTOP_ICON)

A desktop icon is an iconified internal frame within a desktop pane.

EXAMPLE: Unknown.

Braille

- Verbose Example: file.txt icon
- Brief Example: file.txt icon

Speech

- Verbose Example: file.txt icon
- Brief Example: file.txt icon

dial (ROLE_DIAL)

A dial is a rotatable valuator.

EXAMPLE: Unknown.

Braille

- Verbose Example: temperature 50 dial
- Brief Example: temperature 50 dial

Speech

- Verbose Example: temperature 50 dial
- Brief Example: temperature 50 dial

dialog (ROLE_DIALOG)

A dialog is a top level window with a title and border. *TODO:* the AT-SPI makes a distinction between alerts and dialogs. But, it appears as though the role "alert" is applied to just about any dialog.

EXAMPLE: gtk-demo: "Dialog and Message Boxes" demo; click on the "Interactive Dialog" button. The "Dialogs" window is a dialog.

EXAMPLE: gtk-demo: "Expander" demo; the "GtkExpander" window is a dialog.

Braille

Any window, regardless if it is an alert, dialog, frame, or window, will appear as the context region on the braille line. The context string will consist of the title of the window followed by its role. The remainder of the braille line will consist of information describing the object with focus in the window. The information will follow the presentation rules for the given role type.

In the case of the "dialog", the role of the context will be "dialog."

- Verbose Example: Editing profile "foo" dialog
- Brief Example: Editing profile "foo" dialog

Speech

When a dialog gets focus, Orca will say the title of the dialog followed by "dialog" followed by the active object.

- Verbose Example: Editing profile "foo" dialog. OK button.
- Brief Example: Editing profile "foo" dialog. OK.

directory pane (ROLE_DIRECTORY_PANE)

A directory pane is a pane that allows the user to navigate through and select the contents of a directory.

EXAMPLE: Unknown.

Braille

When in a directory pane, it will be treated like a part of a context.

- Verbose Example: Open file dialog select file directory pane filename: text
- Brief Example: Open file dialog select file filename:

Speech

TODO: add speech

icon (ROLE_ICON)

An icon is a small fixed-sized picture typically used to decorate components.

EXAMPLE: desktop: Press Ctrl+Alt+Tab to select the "Desktop." This will give the icons on the desktop focus and you can move between them using the arrow keys.

EXAMPLE: gtk-demo: "Application Main Window" demo. This creates an "Application Window" frame. Click on File->New. This brings up an alert as a child of the gtk-demo application. The name of the alert is "Information." Buried in the widget hierarchy is an icon named "dialog information."

EXAMPLE: gtk-demo: "Dialog and Message Boxes" demo; click on the "Interactive Dialog" button. This brings up a dialog as a child of the gtk-demo app. There is a an icon named "dialog question" in the dialog's hierarchy.

EXAMPLE: gtk-demo: "Stock Item and Icon Browser". This is a big table with four table column header objects and sets of table cell objects. The "Selected Item" panel has the named icon in its hierarchy. NOTE: the first table cell has the icon and text as children table cell objects.

Braille

The label of the icon (or the component for the icon) will be displayed.

- Verbose Example: unread icon
- Brief Example: unread

Speech

- Verbose Example: unread icon
- Brief Example: unread

image (ROLE_IMAGE)

An image is a picture, typically static.

KNOWN ISSUE: Most push buttons objects appear to implement the Image interface whether they show an image or not. So...the consumer needs to be careful to determine if an image is actually showing or not.

Braille

The label of the icon (or the component for the icon) will be displayed. *TODO:* perhaps show accessible name and/or description of the image?

- Verbose Example: new england in the fall image
- Brief Example: new england in the fall

Speech

- Verbose Example: new england in the fall image
- Brief Example: new england in the fall

html container (ROLE_HTML_CONTAINER)

EXAMPLE: mozilla.

KNOWN ISSUE: The HTML container is a hierarchy of test objects. Non-links do not get focus, but links do. This presents some difficulties. For example, the first time a container is displayed, focus might be given to a link that is off the screen.

TODO: considerable thought needs to be done here. An idea for links, however, is that their text should be followed by the type of link: "[mailto,ftp,...] link." In addition, it may actually be more beneficial to use sound effects here so as to not interrupt the natural flow of text. That is, it would be better to speak "press [boop] here to send e-mail to Mike" vs. "press here mailto link to send e-mail to Mike."

Braille

- Verbose Example:
- Brief Example:

Speech

TODO: add speech

label (ROLE_LABEL)

A label is a short string. It typically labels another object (e.g., a text area), but may be standalone. When it is standalone, it will be considered "static text." Special handling of labels will only be done in the case where they are static text. In these cases, the user will discover the text through review mode.

EXAMPLE: gtk-demo: Bring "Color Selector" demo. This creates a "Color Selection" frame as a child of the gtk-demo app. Click on "Change to above color". This creates a "Changing color" color chooser dialog. The numerical entry fields are spin button objects, each which is LABELED_BY a label.

Braille

Labels will be presented as "raw" text on the display.

- Verbose Example: you are low on disk space
- Brief Example: you are low on disk space

Speech

- Verbose Example: you are low on disk space
- Brief Example: you are low on disk space

list (ROLE_LIST)

A list is an object that presents a list of objects to the user and allows them to select one or more of them.

EXAMPLE: mozilla: "Tools" -> "Switch Profile..."

Braille

Orca will display the label for the list followed by the focused item followed by "list" followed by the position in the list if in verbose mode. If the item is selected, Orca will completely underline the item using dots 7 and 8. The touch cursors will toggle the selection of the associated item.

- Verbose Example: state: california list 5 of 50
- Brief Example: state: california list

Speech

- Verbose Example: state: california list 5 of 50
- Brief Example: state: california list

menu (ROLE_MENU)

A menu contains menu items and lives in a menu bar.

Braille

Orca will show the menu name followed by the menu items. The menu item with focus (if there is one) will be the most meaningful object and it will also have the braille cursor. The touch cursors will activate the associated item.

EXAMPLE: Menus abound everywhere.

KNOWN ISSUE: A flurry of events seem to be delivered when going to/from a submenu item in the same menu. One of these events appears to give focus to the containing menu, which causes a dilemna for determining the object that will actually end up with focus (i.e., speech output needs to ignore intermediate objects with focus).

- Verbose Example: file menu new _ save _ quit _ ...
- Brief Example: file menu new _ save _ quit _ ...

Speech

When a menu is entered, its name will be spoken. If the menu selection changes, only the new selection will be spoken.

- Verbose Example: file menu new control n
- Brief Example: file menu new

menu (ROLE_MENU_BAR)

A menu bar contains a set of menus.

EXAMPLE: Menu bars abound.

Braille

Orca will show the "menubar" followed by names of the menu items separated with "_". %todo; for this and maybe all other objects with mnenonics, how does one determine what the mnemonic is?

- Verbose Example: menubar file _ edit _ view _...
- Brief Example: menubar file _ edit _ view _ ...

Speech

Will speak "menubar" followed by name of menu with focus.

- Verbose Example: menubar file
- Brief Example: menubar file

menu item (ROLE_MENU_ITEM)

A menu item lives in a menu.

EXAMPLE: Menu items abound.

KNOWN ISSUE: It appears as though some toolkits allow menu items to have children (see the 'Menus' demo of the gtk-demo application - the radio menu items have sub menus). This needs to be taken into account.

Braille

All menus items from a single menu will be shown at the same time (see ROLE_MENU). The menu item with focus (if there is one) will be the most meaningful object and it will also have the braille cursor. The touch cursors will activate the associated item. If the menu item is a menu (i.e., a submenu), then it will be followed by the word "sub" to indicate it is a submenu.

- Verbose Example: file menu new _ save _ quit _ ...
- Brief Example: file menu new _ save _ quit _ ...

Speech

Speak the item followed by any sub menu information followed by its accelerator.

- Verbose Example: new submenu control n
- Brief Example: new submenu

option pane (ROLE_OPTION_PANE)

An option pane is a specialized pane whose primary used is inside a dialog.

EXAMPLE: Unknown.

Braille

- Verbose Example:
- Brief Example:

Speech

- Verbose Example:
- Brief Example:

page tab (ROLE_PAGE_TAB)

A page tab is a child of a page tab list.

EXAMPLE: gtk-demo: The panel on the right hand side of the main window is a page tab list. Selecting the children causes the associated page tab to appear.

EXAMPLE:: gnome-terminal: Each child of the gnome-terminal app is frame. Each frame has a menu bar and a page tab list. Each page tab list has a page tab.

EXAMPLE: gnome-terminal: "Edit" -> "Current Profile..." is a dialog containing a page tab list.

Braille

A page tab will be viewed as a form of context. If only the page tab is selected (i.e., the focus is on the name of tab), then the page tab will be presented in the context of the page tab list. If an object in the tab has focus, however, then the following will be the context and the object will also be displayed. The object will also be the most meaningful information.

- Verbose Example: preferences dialog effects tab
- Brief Example: preferences dialog effects tab

Speech

- Verbose Example: preferences dialog effects tab. OK button.
- Brief Example: preferences dialog effects tab. OK.

page tab list (ROLE_PAGE_TAB_LIST)

A page tab is an object that presents a series of panels (page tabs), one at a time.

EXAMPLE: See ROLE_PAGE_TAB.

Braille

Orca will display "tab list" followed by the list of tabs separated by "_". The selected tab will get the cursor and will also be the most meaningful information. The touch cursors will select the associated page tab.

- Verbose Example: tab list general _ advanced _ effects _ ...
- Brief Example: tab list general _ advanced _ effects _ ...

Speech

When the tab changes, just speak the new tab name.

- Verbose Example: tab list general tab
- Brief Example: tab list general

password text (ROLE_PASSWORD_TEXT)

Password text objects are used for passwords or other places where the visible text should not be visibly shown.

EXAMPLE: "Launch" -> "Preferences" -> "System Preferences" -> "Network Settings" -> "NFS." Brings up a "Query" alert requesting the root password.

Braille

Orca will treat the password text just like a text object, but will display *'s instead of characters.

- Verbose Example: password: ****** text
- Brief Example: password: ******

Speech

- Verbose Example: password: text 5 characters
- Brief Example: password: 5 characters

progress bar (ROLE_PROGRESS_BAR)

A progress bar indicates how much of a task has been completed.

EXAMPLE: Unknown.

Braille

Orca will show the label followed by the value followed by "progress."

- Verbose Example: uploading: 57% progress
- Brief Example: uploading: 57%

Speech

- Verbose Example: uploading: 57% progress
- Brief Example: uploading: 57%

push button (ROLE_PUSH_BUTTON)

An object the user can manipulate to tell the application to do something.

EXAMPLE: gtk-demo: "Button Boxes" demo. Many pushbuttons here. The buttons have images and accelerators.

EXAMPLE: gnome-terminal: "Edit" -> "Current Profile..." The "General" page tab has a push button labeled "Font." The text of this push button is represented by two child labels.

KNOWN ISSUE: It is important to remember that push buttons can optionally have children, which is the case when they have more than one label.

Braille

Orca will show the name of the button followed by the state of the button.

- Verbose Example: OK pushbutton
- Brief Example: OK

Speech

- Verbose Example: OK pushbutton
- Brief Example: OK

radio button (ROLE_RADIO_BUTTON)

A specialized check box that will cause other radio buttons in the same group to become unchecked when it is checked.

EXAMPLE: gnome-terminal: "Edit" -> "Current Profile..." "Effects" tab. Radio buttons live here.

KNOWN ISSUE: The accessible relation interface was design for things like radio buttons, but it appears to be ignored or sporadically supported. As such, depending upon the relation interface appears to be a poor choice.

KNOWN ISSUE: In Swing, when a radio button gets focus, it is not automatically checked. In GTK+, however, a radio button automatically is checked when it gets focus.

Braille

The state of a radio button will be represented by "& y" if it is unselected and "&=y" if it is selected. All buttons from the same radio group will be displayed at once, with the button with focus being the most meaningful information. Any touch cursor associated with the text will click the radio button. Furthermore, the name of the radio button group will be used as part of the context.

- Verbose Example: Color: Red &=y _ Blue & y _ Green & y
- Verbose Example: Color: Red &=y _ Blue & y _ Green & y

Speech

Will speak radio button group name upon entry into group, and then just each radio button as it is selected.

- Verbose Example: Color button group: red button
- Verbose Example: Color: red

radio menu item (ROLE_RADIO_MENU_ITEM)

A radio menu represents a menu item that behaves like a radio button.

EXAMPLE: gnome-terminal: The "Tabs" menu has radio menu items for each tab.

Braille

All menus items from a single menu will be shown at the same time (see ROLE_MENU). The state of a radio menu item will be represented by "& y" if it is unchecked and "&=y" if it is checked. (See also ROLE_ACCEL_LABEL). The touch cursors will result in activating the item.

- Verbose Example: Red & y(CTRL+R) radiomenu
- Brief Example: Red &=y

Speech

- Verbose Example: Red radiomenu selected
- Brief Example: Red selected

row header, table row header (ROLE_ROW_HEADER, ROLE_TABLE_ROW_HEADER)

A row header is a header for a row of data.

EXAMPLE: Unknown.

Braille

Like a column header, a row header will be treated like a button, except it will be followed by "row heading" in verbose mode.

- Verbose Example: January row heading
- Brief Example: January

Speech

- Verbose Example: January row heading
- Brief Example: January

scroll bar (ROLE_SCROLL_BAR)

A scroll bar allows a user to incrementally view a large amount of information.

EXAMPLE: gtk-demo: "Application Main Window" demo. The text area is in a scroll pane. Type enough and a scroll bar appears. One cannot seem to give it keyboard focus; instead normal keyboard navigation applies to the text area.

Braille

Orca will show the value (in %) followed by the orientation followed by "scrollbar". NOTE: it is rare that users will use scrollbars. They will tend to use the page up/down and other navigation keys to move around large areas of text.

- Verbose Example: 57% vertical scrollbar
- Brief Example: 57% vertical scrollbar

Speech

- Verbose Example: 57% vertical scrollbar
- Brief Example: 57% vertical scrollbar

slider (ROLE_SLIDER)

A slider allows a user to select a value from a bounded range.

EXAMPLE: gnome-terminal: "Edit" -> "Current Profile..." The "Shade transparent or image background" object is a slider.

Braille

Orca will show the label, then the value (in %) followed by "slider".

- Verbose Example: volume: 57% slider
- Brief Example: volume: 57%

Speech

- Verbose Example: volume: 57% slider
- Brief Example: volume: 57%

spin button (ROLE_SPIN_BUTTON)

A spin button is a single line item that contains a list of choices the user can select from.

EXAMPLE: gtk-demo: "Color Selector" demo. This creates a "Color Selection" frame as a child of the gtk-demo app. Click on "Change the above color". This creates a "Changing color" color chooser dialog. The numerical entry fields are spin button objects, each which is LABELED_BY a label and also implements the EditableText interface. These also have the CONTROLLER_FOR relation for the color wheel. These implement the AccessibleValue interface as well. (NOTE: use the arrow keys to change the value.)

Braille

The verbose and brief modes will show the label followed by the item followed by the word "spin."

- Verbose Example: Red: 57 spin
- Brief Example: Red: 57 spin

Speech

- Verbose Example: Red: 57 spin
- Brief Example: Red: 57 spin

split pane (ROLE_SPLIT_PANE)

A split pane is a specialized panel that presents two panels at once. It has a divider than can be dragged.

EXAMPLE: gtk-demo: "Paned Widgets" demo. This brings up a "Panes" frame. The top split pane is a child of the big split pane.

EXAMPLE: gedit: "File" -> "Open" brings up a split pane.

Braille

The words "split pane" will become part of the context and the focused object will be the object of interest. In the event the focus is on the split pane divider, braille will display the percentage of the top or left pane that is shown followed by "split pane divider".

- Verbose Example: open file dialog directory split pane /usr list 5 of 10
- Brief Example: open file dialog directory split pane /usr list 5 of 10

Speech

- Verbose Example: open file dialog directory split pane /usr list 5 of 10
- Brief Example: open file dialog directory split pane /usr list 5 of 10

table, table cell (ROLE_TABLE, ROLE_TABLE_CELL)

Tables are a container of data organized in rows and columns. The interesting aspects of a table include: the row/column headers, the contents of the various cells, and selection.

EXAMPLE: gtk-demo: "Editable Cells" demo brings up a "Shopping list" frame that has a table with cells that have editable text (press the space key to initiate the edit).

Braille

A table will generally be viewed as a context, with focused table component (e.g., header or cell) being the most important information. In verbose mode, Orca will show the title of a table followed by "table" followed by row and column information followed by the cell data. Brief mode will simply show row and column information followed by cell data. The cell data will displayed using the given role styles in this style guide followed by "cell." In brief mode, "cell" will not be shown.

- Verbose Example: home runs table row header Mike c4 0 cell
- Brief Example: c4 0

Speech

- Verbose Example: home runs table row header Mike c4 0 cell
- Brief Example: c4 0

tear off menu item (ROLE_TEAR_OFF_MENU_ITEM)

A tear off menu item allows a menu to be removed from a menu bar and shown in its own window.

EXAMPLE: gtk-demo: "Menus" demo has many tear off menu items. Keyboard navigation skips the tear off menu item at first. To get to it, you need to press the up arrow key after selecting the first "real" menu item.

Braille

All menus items from a single menu will be shown at the same time (see ROLE_MENU). The menu item with focus (if there is one) will be the most meaningful object and it will also have the braille cursor. The touch cursors will activate the associated item. Tear off menu items will be shown with "---" as their label.

- Verbose Example: file menu --- _ new _ save _ quit _ ...
- Brief Example: file menu --- _ new _ save _ quit _ ...

Speech

TODO: maybe say "dash dash dash"?

- Verbose Example: tear off menu item
- Brief Example: tear off

text (ROLE_TEXT)

A text object presents text to the user such that the text typically has a caret position, can be single line or multiline, and can be editable.

EXAMPLE: gtk-demo: (MULTI_LINE) "Application Main Window" demo. This creates an "Application Window" frame. The text is a child of scroll pane which is a child of the panel of this frame. This text object implements the EditableText interface.

EXAMPLE: gtk-demo: (LABELED_BY and SINGLE_LINE) "Color Selector" demo. This creates a "Color Selection" frame as a child of the gtk-demo app. Click on "Change the above color". This creates a "Changing color" color chooser dialog. The "Color Name" text area is a SINGLE_LINE text area that is LABELED_BY the "Color Name" label.

EXAMPLE: gtk-demo: (LABELED_BY and SINGLE_LINE) Bring up the "Dialog and Message Boxes" demo. "Entry 1" and "Entry 2" are SINGLE_LINE text area that are LABELED_BY.

EXAMPLE: gtk-demo: the "Info" and "Source" tabs contain uneditable text.

Braille

Document text will be displayed optionally with the following enhancements. It can optionally be displayed with proper spacing shown, with a certain attribute underlined, style information displayed in status cells. If status cells exist it should be possible to toggle them between the font, the point size and the attribute. Multiline text will be followed by the current line and total lines.

- Verbose Example: first name: willie edit (the cursor is shown and moves with the caret position)

Verbose Example: address: 1024 washington way multiline 1 of 2 (the cursor is shown and moves with the caret position)

- Brief Example: first name: willie

Speech

TODO: add speech

toggle button (ROLE_TOGGLE_BUTTON)

A toggle button is a specialized push button that can be checked or unchecked, but does not provide a separate indicator for the current state.

gtk-demo: "Expander" demo. The "Details" object is a toggle button. *KNOWN ISSUE:* in at-poke, clicking on "Details" doesn't seem to result in the new label appearing in at-poke's display.

Braille

The state of a toggle button will be represented by "& y" if it is unselected and "&=y" if it is selected.

- Verbose Example: Run &=y toggle
- Verbose Example: Run & y

Speech

Only speaks new state if object has focus and state changes.

- Verbose Example: Run toggle pressed
- Verbose Example: Run not pressed

tool bar (ROLE_TOOL_BAR)

A tool bar is a bar or palette usually composed of push buttons or toggle buttons, but may also contain other items such as editable text.

EXAMPLE: gtk-demo: "Application Main Window" demo. This creates an "Application Window" frame. The tool bar is a child of the panel, which, in turn, is a child of this frame. This toolbar is composed of multiple panels, each with its own push button. The vertical separator is just a panel (it's not a separator).

Braille

Verbose mode should show the tool bar followed by "toolbar" Brief mode should simply show the tool bar. Each item will be separated with "_". Same verbosity rules apply to each item. *TODO:* perhaps toolbar should be treated like a context?

- Verbose Example: new button _ save button _ cut button toolbar
- Brief Example: new _ save _ cut

Speech

- Verbose Example: toolbar
- Brief Example: toolbar

tree, tree table (ROLE_TREE, ROLE_TREE_TABLE)

A tree presents hierarchical information to the user, allowing them to expand/collapse nodes to limit what it seen.

EXAMPLE: gtk-demo: "Tree Store" demo brings up a tree table.

Braille

Orca should show the label for the tree followed by the current item followed by its state followed by the level followed by tree.

- Verbose Example: directory myfolder expanded level 4 tree
- Brief Example: directory myfolder leaf level 4 tree

Speech

- Verbose Example: directory myfolder expanded level 4 tree
- Brief Example: directory myfolder leaf level 4 tree

window (ROLE_WINDOW)

A window is a top level window with no title or border.

Braille

On Alt+Tab, the task switcher will appear. As it is on the screen, it will be presented. The text that appears (i.e., the window name) will be shown in braille. When the window gets focus, it will be the context, and the object with focus in the window will be the most meaningful information.

- Verbose Example:
- Brief Example:

Speech

TODO: add speech

Bibliography

Notes

1. http://matrix.netsoc.tcd.ie/hcksplat/work/XKBlib.pdf

Orca Architecture and Functional Specification

Orca Architecture and Functional Specification

Table of Contents

l

Foreword

Orca is a flexible, extensible, and powerful assistive technology that provides end-user access to applications and toolkits that support the AT-SPI (e.g., the GNOME desktop). With early input and continued engagement from its end users, Orca has been designed and implemented by the Sun Microsystems, Inc., Accessibility Program Office.

NOTE: Orca is currently a work in progress. As a result, this and other books in the Orca Documentation Series are under continuous modification and are also in various states of completeness.

This book covers the architecture and functional specification of Orca. The specification was driven primarily by the Orca User Experience Design.

Chapter 1. Introduction

The Orca architecture has been driven primarily by the Orca User Experience Design. There are two primary operating modes of Orca: a focus tracking mode and a flat review mode.

The focus tracking mode generally relies upon applications to provide reasonable keyboard navigation techniques to allow the user to operate the application without requiring the mouse. As the user uses traditional keyboard navigation techniques to move from component to component in the application (e.g., pressing the Tab key to move from pushbutton to text area to toggle button, etc.), Orca will present this to the user via braille, speech, magnification, or a combination thereof. In the cases where more complex navigation is needed, such as structural navigation of complex text documents, Orca also provides a facility to define keyboard and braille input events that it can intercept and handle appropriately.

The flat review mode provides the user with the ability to spatially navigate a window, giving them the ability to explore as well as discover and interact with components in the window. Orca provides a default set of keybindings for flat review, and these keybindings can be easily redefined by the user.

The various modes of Orca are handled by "scripts," which are Python modules that can provide a custom interpretation of an application's interaction model. It is not intended that there will be a unique script for every application. Instead, it is expected that there will be a general purpose script that covers a large number of applications. In the event that more compelling or custom behavior is desired for an application, however, one can use a custom script for the application. Furthermore, scripts can subclass other scripts, allowing them to be quite simple.

Chapter 2. Prerequisites

To help narrow the scope of the Orca development activity, Orca uses existing software where available. For example, as mentioned in the requirements, Orca is a screen reader that needs to be able to interact with speech synthesis, braille, and screen magnification services, but it need not be the provider of such services. Given this, Orca has the following dependencies:

Python v2.4 or better

Orca is written in the Python programming language and depends upon features found in Python versions 2.4 and greater.

pyorbit v2.0.0 or better

PyORBit provides the Python language bindings for ORBit, which is Bonobo's CORBA ORB implementation.

libbonobo v2.0.0 or better

libbonobo provides the Python language bindings for Bonobo, which gives Orca access to the AT-SPI.

gnome-python-2.0 v2.6.0 or better

GNOME-Python provides the Python language bindings for the GNOME libraries.

pygtk-2.0 v2.4.0 or better

PyGTK provides a convenient wrapper for the GTK library for use in Python programs, and takes care of many of the details such as managing memory and type casting. When combined with PyORBit and gnome-python, it can be used to write full featured Gnome applications.

AT-SPI v1.6.6 or better

Orca's means of gathering information about the desktop as well interacting with the desktop will be done through the AT-SPI [*AT-SPI*>]. As such, a functioning AT-SPI environment is mandatory. The AT-SPI provides a CORBA-based approach to detect, examine, and manipulate desktop and application content. It supports the registration of event listeners for changes to desktop and application content. Finally, the AT-SPI supports the registration of listeners for input device events, with an option for these listeners to intercept (and possibly consume) the events before they are processed by the desktop or applications on the desktop.

gnome-speech v0.3.9 or better

GNOME-Speech [*Gnome-Speech*>] provides a CORBA-based approach to access speech synthesizers as network services.

BrlTTY v3.6.1 or better

BrlTTY [*BRLTTY*>] provides access to a variety of Braille displays, and consists of a library and a daemon to provide programmatic interaction with the display.

gnome-mag v0.11.11 or better

GNOME-mag [*Gnome-Mag*>] provides a CORBA-based approach to access and manipulate a screen magnifier as a network service.

Keyboard Navigation

As much as possible, Orca relies upon the keyboard navigation methods built in to the native platform. For example, it is expected that the native platform will provide access via traditional methods such as the "tab" key, keyboard mnemonics, and keyboard accelerators.

Chapter 3. Architecture and Implementation

As illustrated in the high level Orca architecture diagram, the main components of Orca are as follows: desktop applications that support the AT-SPI, the AT-SPI registry and infrastructure, Orca itself, Orca Scripts, and system services (e.g., speech, braille, magnification).

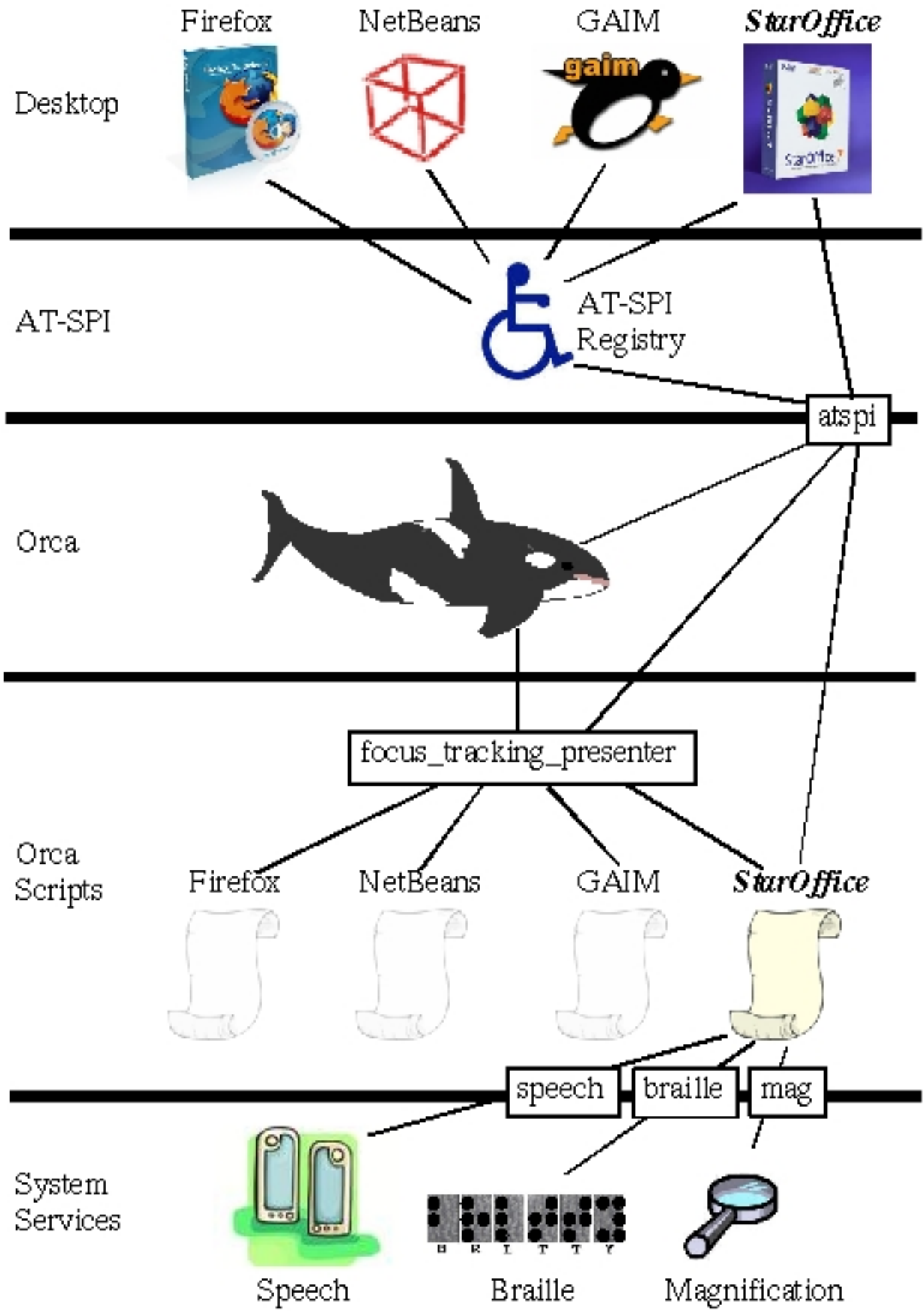

Figure 3-1. High Level Orca Architecture. The main components of Orca are as

follows: desktop applications that support the AT-SPI, the AT-SPI registry and infrastructure, Orca itself, Orca Scripts, and system services. The key communication between the components is depicted.

The following sections describe the architecture in more detail.

Desktop and AT-SPI

Orca's sole view of any application on the desktop is via the AT-SPI [*AT-SPI*>]. The AT-SPI is a CORBA/Bonobo-based technology [*Bonobo*>] that provides a common interface for the desktop and its applications to expose their GUI component hierarchy to assistive technologies such as Orca. AT-SPI support is provided by toolkits such as GNOME's GTK+ toolkit (via gail [*GAIL*>]), the Java platform (via the Java access bridge *TODO:* get link), and the custom toolkits used by applications such as Mozilla and Open Office.

Assistive Technologies interact with the AT-SPI via two primary means: the AT-SPI registry and accessible objects. The AT-SPI registry permits assistive technologies to discover existing applications on the desktop and to register for event notification for AT-SPI events (e.g., window creation, focus changes, object state changes, etc.) and device events (e.g., keyboard input events). Accessible objects provide the assistive technology with information about the application, and tend to mirror the actual GUI component hierarchy. Accessible objects can be obtained in three ways:

1. From the AT-SPI registry via queries on the desktop
2. From an AT-SPI event
3. From another Accessible via parent/child relationships and other relationships

Orca's interaction with the AT-SPI is managed through Orca's `atspi.py` module. The `atspi.py` module communicates directly with the AT-SPI via the AT-SPI IDL interfaces and also provides a number of classes that help with AT-SPI interaction: `Registry`, `Accessible`, and `Event`. The full documentation for each of these classes is available in the pydoc for Orca, and the following paragraphs provide a brief overview.

The `Registry` class provides a singleton (actually a "Borg") class instance to access to the AT-SPI registry. It provides convenience methods for registering AT-SPI event listeners and device event listeners, and also provides the mechanism for starting and stopping event delivery from the AT-SPI registry.

The `Accessible` class provides a wrapper for communicating with CORBA objects that implement the AT-SPI Accessible and Application interfaces. Using Python's ability to add new properties to a class instance at run time, Orca can also annotate Accessible class instances with additional information. The main purpose of an Accessible is to provide a local cache for accessible objects, preventing the need for numerous round trip calls to the AT-SPI registry and application for information.

The `Event` class provides a wrapper for converting AT-SPI events into Python `Event` instances. The main purpose is to convert the AT-SPI accessible source of the event into a Python `Accessible` instance and to also provide an `Event` instance that can be annotated by scripts (the AT-SPI event is read-only).

As illustrated in the high level Orca architecture diagram, the `atspi` module has been used to register event and device listeners with the AT-SPI registry. Each exemplary desktop application (Firefox, NetBeans, GAIM, StarOffice) emits AT-SPI events to the AT-SPI registry which then delivers them to the `atspi` module. The `atspi` module then calls all appropriate listeners for the events it receives from the AT-SPI registry.

In this case, the `orca` module receives keyboard events, which it interprets and also sends on to the `focus_tracking_presenter` module. Of more interest, however, is that the `focus_tracking_presenter` module receives AT-SPI events which it then

passes on the script for the application associated with the event. If there is no script, the `focus_tracking_presenter` will create one. See the Orca Script Writing Guide for more information.

The `atspi` module also registers its own set of event listeners that it uses to keep its local cache of accessible objects up to date.

IMPLEMENTATION DETAIL: Because processing AT-SPI object events can be time consuming, and because the notification of AT-SPI object events is relatively "bursty," the `focus_tracking_presenter` maintains a queue of AT-SPI object events. It adds the events to this queue when it receives them and processes the events on the GLib idle handling thread. This permits Orca to survive a relatively long burst of events and also allows it to handle the events on a thread that is compatible with GLib.

IMPLEMENTATION DETAIL: Unlike AT-SPI object events, Orca must process keyboard events immediately and quickly because it needs to determine if it should consume those events or not. As such, the `atspi` component will dispatch keyboard events directly to the keyboard event listeners registered by Orca.

TODO: The system needs to be able to cope with failure. Right now, if Orca fails while handling a keyboard event, the whole desktop can hang. I'm curious if the AT-SPI-enabled app can heal when the connection to Orca is broken or if there is a timeout? Conversations with Bill Haneman lead me to believe there is no such thing as a timeout for the synchronous form of device event notification.

Orca

The `orca` module is the "main entry point" of Orca. It initializes the components that Orca uses (atspi, speech, braille, mag) and loads the user's settings. It also is the first to receive all keyboard and braille input events and delves them out to other Orca components appropriately.

The `orca` module maintains the current known "locus of focus" in its `locusOfFocus` field that is set via the `setLocusOfFocus` method. This locus of focus represents the current object that effectively has keyboard focus. That is, the object that will change and or react to keyboard input events. Orca scripts call this method to inform Orca when the locus of focus has changed. In addition, in the event that there was a visual appearance change to the object that has the locus of focus, the `orca` module provides a `visualAppearanceChanged` that scripts can use to inform Orca of this event.

The `orca` module also maintains a list of all known applications. This list is available as the `apps` field.

settings

The *settings* module (not depicted in the high level Orca architecture diagram) holds preferences set by the user during configuration. These settings include the following: use of speech and/or braille, voice styles, key echo, text echo, and command echo (see the user requirements for details on these features).

Any Orca module can request the value of a setting by calling the `getSetting` method of the `settings` module. When first called, this module will import the `user-settings` module from the `~/.orca directory`, if it exists. The `user-settings` module is a Python script, allowing it to contain functions, class definitions, etc. Furthermore, the import of the `user-settings` module will cause any executable statements to be run, allowing the module to define/set fields in itself and call functions in other modules. For example, the module will typically define the use of output mode (speech, braille, mag) and may also call functions in other modules, such as the `setDebugLevel` method of the `debug` module.

The `getSetting` method of the `settings` module is intended to discover field attribute values. It will first look to the `user-settings` module for the field, and then

fall back to the settings module. The getSetting method also accepts a default value in the event the field does not exist in either the user-settings or settings module.

IMPLEMENTATION DETAIL: the init method of the orca module obtains settings. As a result, the user-settings module is imported very early in the Orca life cycle.

Orca Scripts

Internally, the orca module keeps track of list of PresentationManager instances (see the pydoc for presentation_manager.py). The FocusTrackingPresenter (see focus_tracking_presenter) is of the most interest, as it is the PresentationManager that manages scripts.

Details on the FocusTrackingPresenter and Orca scripts can be found in the Orca Script Writing Guide.

System Services

Orca relies on existing system services to provide support for speech synthesis, braille, and screen magnification. To interact with these services, Orca provides the modules described in the following sections.

speech

The *speech* module provides Orca's Python interface to system speech services. Each speech service is generated by a "speech server factory." There are currently two such factories: one for [*Gnome-Speech*>] (see gnomespeechfactory.py and one for [*Emacspeak*>] (see espeechfactory.py), though it is expected that support for other factories can be added in the future.

Each speech factory offers up a list of SpeechServers, where each SpeechServer is typically an interface to a particular speech engine. For example, the espeechfactory will offer up a SpeechServer that talks to the Fonix DECtalk engine and a SpeechServer that talks to the IBMTTS engine. Likewise, the gnomespeechfactory will offer up a SpeechServer that uses the gnome-speech interface to talk to the Festival Speech Synthesis System, a separate SpeechServer that also uses the gnome-speech interface to talk to the Fonix DECtalk engine, and so on.

Each SpeechServer instance then provides a set of methods for actually speaking. Each of the methods accepts an ACSS instance, which represents an aural cascading style sheet ([*ACSS*>]) that defines the voice and voice parameter settings to use.

As part of the orca-setup process, the user selects a particular speech factory, SpeechServer, and voice to use as their default voice. When Orca starts, the speech module looks for these settings and connects to the appropriate speech factory and SpeechServer. In the event the a connection cannot be made, the speech module attempts to find a working synthesis engine to use by examining its list of speech factories. The speech module then provides simple methods that delegate to the SpeechServer instance. This model allows scripts to use their own SpeechServer instances if they wish, but to also just rely upon the user's default preferences.

braille

The *braille* module provides Orca's Python interface to the system's BrlTTY [*BRLTTY*>] daemon. The BrlTTY daemon, in turn, provides the interface to braille

devices for both displaying braille and receiving input from the user.

TODO: flesh this section out more.

mag

The *mag* module provides Orca's Python interface to the system's gnome-mag [*Gnome-Mag*>] CORBA service(s). The magnification component provides methods that permit Orca discover screen magnification services and set their desktop region of interest.

TODO: flesh this section out more.

Chapter 4. Internationalization (I18N) Support

All human-consumable text obtained from AT-SPI calls is expected to be in a localized form. As such, Orca does not do any extra localization processing when working with text obtained via the AT-SPI.

For text generated by Orca itself, Orca handles internationalization and localization using the [*gettext*>] support of Python. The gettext support of Python is similar to the GNU gettext module. Each human consumable string of Orca is US English text wrapped in a call to gettext.gettext. The call to gettext.gettext will either return a localized string or default to the US English text. Orca depends upon an active and thriving community of open source translators to provide the localizations.

The synthesis of localized speech is to be provided by the underlying gnome-speech implementation.

The generation of localized braille is to be determined. *TODO:* BrlTTY currently does not support this at the moment, but it is expected that the BrlTTY developers will add this in the future.

Bibliography

Notes

1. http://directory.fsf.org/accessibility/at-spi.html
2. http://www.w3.org/TR/1998/REC-CSS2-19980512/aural.html
3. http://lidn.sourceforge.net/articles/gnomenclatureintrotobonobo/
4. http://directory.fsf.org/accessibility/brltty.html
5. http://emacspeak.sourceforge.net/
6. http://freshmeat.net/projects/gail/
7. http://docs.python.org/lib/module-gettext.html
8. http://directory.fsf.org/accessibility/gnome-mag.html
9. http://directory.fsf.org/accessibility/gnome-speech.html
10. http://directory.fsf.org/accessibility/gnopernicus.html
11. http://www.freedomscientific.com/fsproducts/softwarejaws.asp
12. http://matrix.netsoc.tcd.ie/hcksplat/work/XKBlib.pdf

Orca Script Writing Guide

Orca Script Writing Guide

Table of Contents

Foreword

Orca is a flexible, extensible, and powerful assistive technology that provides end-user access to applications and toolkits that support the AT-SPI (e.g., the GNOME desktop). With early input and continued engagement from its end users, Orca has been designed and implemented by the Sun Microsystems, Inc., Accessibility Program Office.

NOTE: Orca is currently a work in progress. As a result, this and other books in the Orca Documentation Series are under continuous modification and are also in various states of completeness.

This book is intended for programmers intending to extend Orca's functionality by writing custom scripts.

Chapter 1. Introduction

In this document, you will learn how to create your own custom scripts for Orca.

Orca's scripting approach provides a mechanism for providing customized support for the specific user interaction model of an application. The goal is to provide Orca with the capability of providing a natural feeling and compelling user experience for the various user interaction models of different desktop applications.

The Orca scripting approach allows scripts to extend and/or override the behavior of other scripts, thus simplifying the job of a script writer. To further facilitate script writing, Orca provides a "default" script that provides a reasonable default behavior for Orca. This will not only serve as the "fallback script" for Orca, but will also typically serve as the "jumping off" point for writing custom scripts. Furthermore, keep in mind that the "default" script is intended to cover a large variety of applications. As such, you may find that it is not necessary to write a custom script.

Chapter 2. Overview

The primary operating mode of Orca is "focus tracking mode," where Orca keeps track of the most relevant user interface object that has keyboard focus. When Orca detects changes to this object, which Orca refers to as the "locus of focus," Orca will present relevant information to the user.

As such, the primary goal of a script is to assist Orca in tracking of the locus of focus as well as presenting information about the locus of focus. A script does this by registering for one or more AT-SPI events and then reacting appropriately when it receives those events. A script can also intercept and interpret keystrokes and braille input events, allowing it to further extend the behavior of Orca.

Script Life Cycle

BIRTH: Orca's `focus_tracking_presenter` module is the sole maintainer of scripts. Whenever it receives a `window:activated` event from the AT-SPI Registry, the `focus_tracking_presenter` will determine the application associated with that event and create a script for that application if necessary. Only one script instance per application is managed by the `focus_tracking_presenter`.

The script creation process consists of the following steps:

- The `focus_tracking_presenter` will attempt to perform a Python `import` using the application name as the name of an Orca module. For example, for the `gnome-terminal` application, the `focus_tracking_presenter` will look for the `gnome-terminal.py` in the `orca.scripts` package (see the script naming discussion in the debug utilities section to determine what to name your script). If it cannot find such a module in the Python search path, the `focus_tracking_presenter` will create an instance of the `Default` script in default.py.

  NOTE: the `focus_tracking_presenter` also maintains a table to map application names to script names. This is useful in many cases, such as if the application name changes over time or the application contains characters that are awkward in file system names. To extend or override this table, one can call the `setScriptMapping` method of the `settings` module.

  IMPLEMENTATION DETAIL: it is possible to tell Orca to bypass all custom script creation by setting `useCustomScripts=False` in your `~/.orca/user-settings.py` module. This can be useful for debugging purposes.

- Each script module is expected to provide a `Script` class that ultimately extends the `orca.Script` class defined in the `script.py` module. The constructor takes the accessible application object as an argument.

  The constructor for the `Script` instance is expected to define any keystrokes, braille buttons, and AT-SPI event listeners it is interested in (see the Customized Behavior section for how to do this).

- Once it has created a script, the `focus_tracking_presenter` will register event listeners for all AT-SPI events associated with script (i.e., the script should not register these events itself). When the `focus_tracking_presenter` receives the events, it will pass the event directly to the script associated with the event, regardless if the application associated with the script has focus or not.

  IMPLEMENTATION DETAIL: the `focus_tracking_presenter` registers its own `processObjectEvent` method as the AT-SPI event listener. This method finds (and creates if necessary) the script associated with the event and passes the event onto the required `processObjectEvent` method of the script for processing. Each `Event` (see the `atspi` module) has the following fields:

  - `source`: an `Accessible` (see the `atspi` module) instance representing the object associated with the event

  - `type`: a string describing the event (e.g., `window:activated`)
  - `detail1` and `detail2`: integer details for the event (see the AT-SPI documentation)
  - `any_data`: something associated with the event (see the AT-SPI documentation)

- The `focus_tracking_presenter` also keeps track of the active script (as determined by the script associated with the currently active window) and will pass all keyboard and braille input events to the active script.

  IMPLEMENTATION DETAIL: the `focus_tracking_presenter` implements the `processKeyboardEvent` and `processBrailleEvent` methods which are called by the main `orca` module whenever it receives a keystroke or braille input event. The `focus_tracking_presenter` will pass these events onto the `processKeyboardEvent` and `processBrailleEvent` methods of the active script.

  IMPLEMENTATION DETAIL: the `focus_tracking_presenter` provides logic to allow the user's `~/.orca/user-settings.py` module to extend and/or override key and braille bindings. This logic is currently latent, however, and is not documented or supported.

LIFE: Whenever a script receives an event, the script can do whatever it wants. Its primary task, however, is to assist Orca in keeping track of the locus of focus. When a script detects a change in the locus of focus, it should call `orca.setLocusOfFocus` with the `Accessible` object instance that is the new locus of focus.

IMPLEMENTATION DETAIL: The `orca` module has logic to detect if the locus of focus really changed and will propagate the change on as appropriate. The `orca.setLocusOfFocus` method first sends the change to the `locusOfFocusChanged` method of the `focus_tracking_presenter`, which then passes the change onto the required `locusOfFocusChanged` method of the active script. The `locusOfFocusChanged` method is the primary place where a script will present information to the user.

In many cases, the locus of focus doesn't change, but some property of the current locus of focus changes. For example, a checkbox is checked or unchecked, yet remains as the locus of focus. In these cases, a script should also keep Orca informed by calling `orca.visualAppearanceChanged`.

IMPLEMENTATION DETAIL: Like the `locusOfFocusChanged` method, the `visualAppearanceChanged` method of the `orca` module will first call the `visualAppearanceChanged` method of the `focus_tracking_presenter`, which will then call the required `visualAppearanceChanged` of the active script. The `visualAppearanceChanged` is the primary place where a script will present such information to the user.

DEATH: Whenever the `focus_tracking_presenter` detects that an application has gone away (by determining that the application has been removed from the desktop), it will delete the script for that application and unregister any event listeners associated with that script.

IMPLEMENTATION DETAIL: the `focus_tracking_presenter` determines an application has gone away by detecting a `object:children-changed:remove` event on the desktop.

TODO: it needs to be verified that scripts are actually being reclaimed. I think something somewhere is hanging on to a handle of the script, preventing it from being garbage collected.

Script Contract

The contract for a script is documented in detail in the `script.py` module. The `Default` script subclass defined in the `default.py` module provides the default behavior for Orca when it encounters applications and toolkits that behave like the GTK toolkit. It is expected that new scripts will typically extend the `Default` script subclass rather than directly extending the `Script` class defined in the `script.py` module.

Chapter 3. Customized Behavior

The customized behavior of a script is set up in its constructor. In its constructor, each script is expected to extend/override several fields as illustrated in the following diagram and describe below:

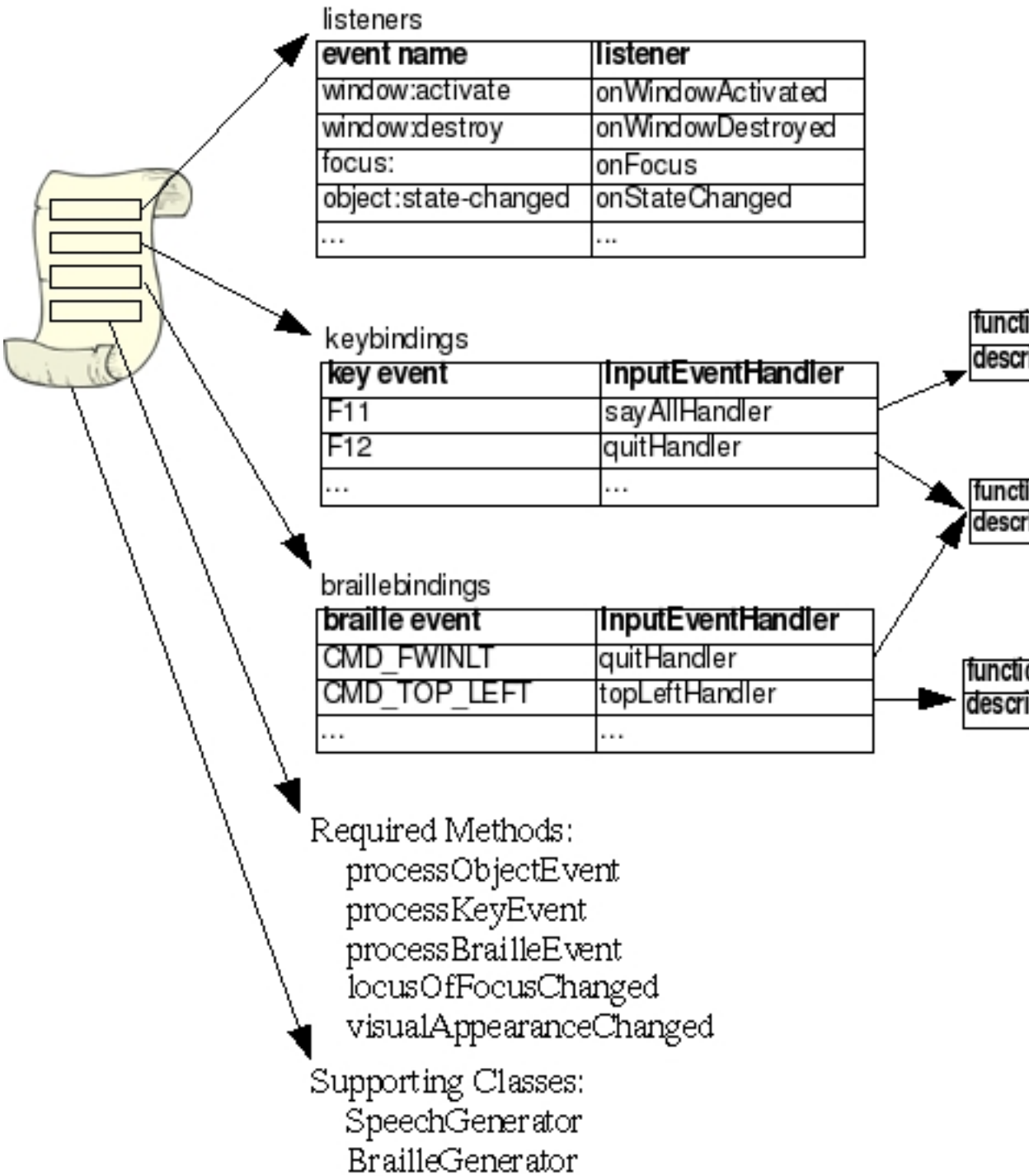

Figure 3-1. Orca Script Diagram

- `listeners`: a dictionary where the keys are strings that match AT-SPI event types (e.g., `focus:`, `object:text-caret-moved`, etc.), and the values are functions to handle the event. Each function is passed an `Event` instance (see the `atspi.py` module) as its sole parameter and no return value is expected.
- `keybindings`: an instance of `keybindings.KeyBindings` (see the `keybindings.py` module) that defines the keystrokes the script is interested in.
- `braillebindings`: a dictionary where the keys are BrlTTY commands (e.g., `CMD_HWINLT`, defined in `braille.py`), and the values are `InputEventHandler` instances.

The constructor for the `Script` class, which all scripts should ultimately extend (most will extend the `Default` class, which in turn extends `Script`), sets up empty values for each of these fields. As such, a subclass merely needs to extend/override these fields. Each of these fields is described in more detail in the following sections.

Defining Event Listeners

As described above, the `listeners` field is a dictionary where the keys are strings that match AT-SPI event types (e.g., `focus:`, `object:text-caret-moved`, etc.), and the values are functions to handle the event. A script's constructor can modify/extend this dictionary by merely defining an entry:

```
self.listeners["focus:"] = self.onFocus
```

In the event there is already an entry in the `listeners` dictionary, it will be overridden by the new value.

As decribed previously, the `focus_tracking_presenter` will register listeners on behalf of a script, and will notify the script of any events via the script's `processObjectEvent` method. The `processObjectEvent` method of the top level `Script` class examines the `type` field of the given `event`, calling any matching functions from the `listeners` dictionary. As such, it is unlikely that a `Script` subclass will ever need to override the `processObjectEvent` method. Instead, it merely needs to populate the `listeners` dictionary as appropriate.

The function for an event listener merely takes an `Event` instance (see the `atspi.py` module) and does whatever it wants; the return value is ignored. For example, the function definition associated with the above `listeners` entry might look like the following, where the `event` is described above:

```
def onFocus(self, event):
    """Called whenever an object gets focus.

    Arguments:
    - event: the Event
    """

    ...
    orca.setLocusOfFocus(event, event.source)
    ...
```

Input Event Handlers

Before describing how to set up keyboard and braille event handlers, it is import to first understand the `InputEventHandler`, which is defined in the `input_event.py` module. `InputEventHandler`s serve a purpose of holding a function to call for a particular input event, and a human consumable string that provides a short description of the function's behavior. The main purpose of the `InputEventHandler` is to provide

support for the "learn mode" of Orca. If learn mode is enabled, the input event handler will consume the input event (i.e., return True) and merely speak and braille the human consumable string. If learn mode is not enabled, the input event handler will pass the active script and the input event on to the function, returning the boolean value of the function as indication of whether the event should be consumed by Orca or passed on to the application.

The best place to look for examples of `InputEventHandler`s is in the `default.py` module. For example, this module defines an input event handler for telling the flat review context to move to the home position of a window:

```
reviewHomeHandler = input_event.InputEventHandler(
            Script.reviewHome,
            _("Moves flat review to the home position."))
```

In this definition, `default.py` is creating an `InputEventHandler` instance whose function is the Script's method, `reviewHome` and whose human consumable text describes what will happen. The Script's `reviewHome` method is defined as follows:

```
def reviewHome(self, inputEvent):
    """Moves the flat review context to the top left of the current
    window."""
    context = self.getFlatReviewContext()
    context.goBegin()
    self.reviewCurrentLine(inputEvent)
    self.targetCursorCell = braille.cursorCell
    return True
```

Note that the method returns `True` to indicate the input event has been consumed.

Defining Keyboard Bindings

The keyboard bindings for a script are held in the `keybindings` field, which is a `KeyBindings` instance. This field maintains a set of `KeyBinding` instances.

Keyboard bindings merely define the keystroke and modifier circumstances needed to invoke an `InputEventHandler` instance. This definition is held in a `KeyBinding` instance (see the `keybindings.py` module):

```
self.keybindings.add(
    keybindings.KeyBinding("KP_7",
                           1 << orca.MODIFIER_ORCA,
                           1 << orca.MODIFIER_ORCA,
                           reviewHomeHandler))
```

The first parameter of a `KeyBinding` is a string that represents an X Window System KeySym string for the key. This is typically a string from `/usr/include/X11/keysymdef.h` with the preceding 'XK_' removed (e.g., XK_KP_Enter becomes the string 'KP_Enter'), and is used as a means to express the physical key associated with the KeySym.

The second parameter is a bit mask that defines which modifiers the keybinding cares about. If it does not care about any modifier state, then this mask can be set to 0. In the example above, the keybinding is being told to pay attention to the `MODIFIER_ORCA` modifier, which is a modifier Orca sets when the "Insert" key is pressed. Other examples of modifier bit positions include those defined in the AT-SPI Accessibility specification: MODIFIER_SHIFT, MODIFIER_SHIFTLOCK, MODIFIER_CONTROL, MODIFIER_ALT, MODIFIER_META, MODIFIER_META2, MODIFIER_META3, and MODIFIER_NUMLOCK. These can be obtained via the `orca.atspi.Accessibility` field. For example, `orca.atspi.Accessibility.MODIFIER_SHIFTLOCK`.

The third parameter is a bit mask that defines what the modifier settings must be. If a bit is set, it means the associated modifier must be set. The only meaningful bits in this mask are those that are defined by the second parameter. In the example above, the keybinding cares about the `MODIFIER_ORCA` modifier, and the third parameter says this modifier must be set.

The last parameter is the `InputEventHandler` to us if the user types a keystroke qualified by the previous parameters. `InputEventHandler`s are described in the previous section.

Defining Braille Bindings

Refreshable braille displays have buttons that the user can press. The BrlTTY system provides a means for standardizing the types of input events one can generate using these buttons, and a script can set up braille bindings to handle these events.

The braille bindings for a script are held in the `braillebindings` field, which is a dictionary. The keys for the dictionary are BrlTTY constants representing braille input events (see `braille.py` for a list), and the values are `InputEventHandler` instances:

```
self.braillebindings[braille.CMD_TOP_LEFT] = reviewHomeHandler
```

In the above example, the BrlTTY `braille.CMD_TOP_LEFT` input event has been set to be handled by the same `reviewHomeHandler` instance described previously.

Chapter 4. Script Utilities

There are many common things a script wants to do: generate speech, update braille, etc. In addition, there are many common things a script writer wants to do, especially getting debug output to determine just what the AT-SPI is sending it. This chapter discusses the debug utilities of Orca as well as a variety of utilities to assist a script in managing speech, braille, and magnification.

Debug Utilities

The debug utilities (defined in the `debug.py` module) of Orca provide a means for selectively turning on information to be sent to the console where Orca is running. This information is quite useful in determining what is happening inside Orca as well as what the AT-SPI is sending to Orca.

Let's begin the discussion of the debug utilities with the top question on any script writer's mind: "What do I name my script?" As you may recall, the name of a script is based upon the name of the application as given to us by the AT-SPI. One of the easy ways to determine this is to listen for `window:activate` events that will be issued when an application is started. These events can then be used to determine the name of the application.

Fortunately, the `focus_tracking_presenter` already registers for `window:activate` events, so all you need to do is tell Orca to print these events out when it receives them. The method for doing this involves telling the debug utilities what to do, and this can be done by modifying your `~/.orca/user-settings.py`.

There are two main settings to tell Orca to print out events: an event filter and an event debug level. The event filter is a regular expression that is used to match AT-SPI event types, and the event debug level specifies a threshold for when to actually print information to the console (for more complete detail on these settings, refer to `debug.py`). These settings can be modified by adding the following lines to your `~/.orca/user-settings.py`:

```
debug.setEventDebugFilter(re.compile('window:activate'))
debug.setEventDebugLevel(debug.LEVEL_OFF)
```

Now, when you rerun Orca, it will output information whenever it receives a `window:activate` event from the AT-SPI registry. For example, if you run Star Office, you should see output similar to the following:

```
OBJECT EVENT: window:activate detail=(0,0)
              app='StarOffice' name='StarOffice' role='frame'
              state='ENABLED FOCUSABLE RESIZABLE SENSITIVE SHOWING VISIBLE'
```

The string `app='StarOffice'` indicates the name of the application is 'StarOffice.' As such, if you wanted to write a custom script, you would call it `StarOffice.py`.

The debug module also includes a number of other methods, each of which is described in more detail in `debug.py`. Note that each method includes a debug level threshold. The `debug.py` module has a description of various level settings and what to expect for output.

- `setDebugLevel(newLevel)`: sets the debug level threshold, turning on or off the various debug code built in to the various Orca modules. This is typically called from `~/.orca/user-settings.py`.
- `setEventDebugLevel(newLevel)`: described above; typically called from `~/.orca/user-settings.py`.

- `setEventDebugFilter(regExpression)`: described above; typically called from `~/.orca/user-settings.py`.
- `printException(level)`: if an exception is caught, this can be used to print out detail about it
- `printStack(level)`: prints the current stack; useful for determining when and why a code path is being executed
- `println(level, text)`: prints the given text; useful for general debug output
- `printObjectEvent(level, event)`: prints out the given AT-SPI event
- `printObjectEvent(level, event)`: prints out the given AT-SPI event, using the event debug level as an additional threshold; this is already used by the `focus_tracking_presenter`, so you are unlikely to need it
- `printInputEvent(level, string)`: prints out the given AT-SPI event, using the event debug level as an additional threshold; this is already used by `orca.py` (for keyboard events) and `braille.py` (for braille events), so you are unlikely to need it

NOTE: One debug level of interest is `debug.LEVEL_FINE`. This level will tell you when a script is activated, and can be useful to determine if Orca is actually finding your script! For example, when the script for the `gnome-terminal` is activated by the `focus_tracking_presenter`, you will see the following output:

```
ACTIVE SCRIPT: gnome-terminal (module=orca.scripts.gnome-terminal)
```

Notice that the class of the script instance is included. If you determine that this class is not what you expect when you are developing your custom script, then something went wrong when trying to find or load your custom script. This can often happen because Python performs a lot of late binding and compilation, thus errors are often not encountered until a specific code path is executed at run time. You can tell the `focus_tracking_presenter` to give you more information about any possible failures or exceptions it handles in this area by setting the debug level to `debug.LEVEL_FINEST`.

Speech Synthesis

Orca provides two main modules for speech output: `speech.py` and `speechgenerator.py`. The `speech.py` module provides the main interface to the speech synthesis subsystem. The `speechgenerator.py` module provides a `SpeechGenerator` class that can be used to to actually generate the text to be spoken for various objects. The expected use of the two modules is as follows: a script will instantiate its own instance of a `SpeechGenerator` and will use it to generate text to be spoken. The script will then pass this text to the `speech.py` module to be spoken.

speech.py

For the purposes of script writing, the main entry points of the `speech.py` module are `speak`, `speakUtterances`, and `stop`

See the `speech.py` module for more information.

speechgenerator.py

The primary goal of a `SpeechGenerator` is to create text to be spoken for an accessible object. There are two public entry points into a `SpeechGenerator`:

- `getSpeech(obj, already_focused)`: returns a list (i.e., an array) of strings to be spoken for the given accessible object. The `already_focused` boolean parameter provides a hint to the speech generator about how much text to generate. For example, if a check box that already has focus is to be spoken, usually the reason for this is that the state changed between checked and unchecked. As a result, an appropriate thing to do in this situation is to only speak the new change in state.
- `getSpeechContext(obj, stopAncestor)`: returns a list (i.e., an array) of strings to be spoken that describe the visual context of the given accessible object. This is loosely represented by the hierarchical relationship of the object (i.e., the "Quit" button in the "File" menu in the ...), and the amount of information can be contained by specifying an accessible `stopAncestor` above which we do not want to know anything about. The primary use of this method is to provide the user with feedback regarding the relevant visual context information that changed when the locus of focus changes, but this method is also useful for assisting in "where am I" queries.

NOTE: Orca currently provides some level of support for verbosity via the `VERBOSITY_LEVEL` fields of the `settings.py` module. There are currently two verbosity levels: `VERBOSITY_LEVEL_BRIEF` and `VERBOSITY_LEVEL_VERBOSE`. A `SpeechGenerator` subclass is expected to examine the `speechVerbosityLevel` property of the `settings.py` module and provide the appropriate level of text. The verbosity setting can be obtained via the following call to `settings.getSetting`:

```
# Obtain speech verbosity level.  If it has not been set,
# default to the VERBOSITY_LEVEL_VERBOSE level.
#
verbosity = settings.getSetting(settings.SPEECH_VERBOSITY_LEVEL,
                                settings.VERBOSITY_LEVEL_VERBOSE)
```

One can currently set the verbosity level from their `~/.orca/user-settings.py` module:

```
speechVerbosityLevel = settings.VERBOSITY_LEVEL_BRIEF
```

Braille Output

Like speech, Orca provides two main modules for braille: `braille.py` and `braillegenerator.py`. The `braille.py` module provides the main interface to the braille display. The `braillegenerator.py` module provides a `BrailleGenerator` class that can be used to to actually generate the text to be displayed for various objects. The expected use of the two modules is as follows: a script will instantiate its own instance of a `BrailleGenerator` and will use it to generate text to be display. The script will then pass this text to the `braille.py` module to be displayed.

`braille.py`

TODO: [[[WDW - much writing to be done here, especially regarding how regions will provide automatic support for cursor routing keys.]]]

TODO: [[[WDW - I'm not satisfied with `braille.py`. It will change before Orca 1.0 is released, so be aware of this.]]]

braillegenerator.py

The primary goal of a `BrailleGenerator` is to create text to be displayed for an accessible object. There are two public entry points into a `BrailleGenerator`:

- `getBrailleRegions(obj, groupChildren=True)`: returns a list of two items: the first is an ordered list of braille `Region` instances that represent text to be displayed on the braille display, left-to-right on one line; and the second is an element from the first list that represents which `Region` has "focus" and should be represented by the braille cursor on the display.

  TODO: [[[WDW - describe grouping of children.]]]

- `getBrailleContext(obj)`: returns an ordered list (i.e., an array) of braille `Region` instances that describe the visual context of the given accessible object. This is loosely represented by the hierarchical relationship of the object (i.e., the "Quit" button in the "File" menu in the ...).

Typically, a script will "build up" a single logical line of text for the braille display. The beginning of this line will be the result of the call to `getBrailleContext` and the remainder of the line will be the result of one or more calls to `getBrailleRegions`. Since the logical line will typically be longer than the number of cells on the braille display, the `braille.py` module will scroll to show the braille `Region` with focus. Furthermore, the `braille.py` will also respond to BrlTTY input events to allow the user to use braille display input buttons for scrolling to review the entire line.

NOTE: Orca currently provides some level of support for verbosity via the `VERBOSITY_LEVEL` fields of the `settings.py` module. There are currently two verbosity levels: `VERBOSITY_LEVEL_BRIEF` and `VERBOSITY_LEVEL_VERBOSE`. A `BrailleGenerator` subclass is expected to examine the `brailleVerbosityLevel` property of the `settings.py` module and provide the appropriate level of text. The verbosity setting can be obtained via the following call to `settings.getSetting`:

```
# Obtain braille verbosity level.  If it has not been set,
# default to the VERBOSITY_LEVEL_VERBOSE level.
#
verbosity = settings.getSetting(settings.BRAILLE_VERBOSITY_LEVEL,
                                settings.VERBOSITY_LEVEL_VERBOSE)
```

One can currently set the verbosity level from their `~/.orca/user-settings.py` module:

```
brailleVerbosityLevel = settings.VERBOSITY_LEVEL_BRIEF
```

Orca Testing Plan

Orca Testing Plan

Table of Contents

Foreword

Orca is a flexible, extensible, and powerful assistive technology that provides end-user access to applications and toolkits that support the AT-SPI (e.g., the GNOME desktop). With early input and continued engagement from its end users, Orca has been designed and implemented by the Sun Microsystems, Inc., Accessibility Program Office.

NOTE: Orca is currently a work in progress. As a result, this and other books in the Orca Documentation Series are under continuous modification and are also in various states of completeness.

This book contains the testing plan for Orca.

Chapter 1. Introduction

To be written.

Orca User's Guide

Orca User's Guide

Table of Contents

Foreword

Orca is a flexible, extensible, and powerful assistive technology that provides end-user access to applications and toolkits that support the AT-SPI (e.g., the GNOME desktop). With early input and continued engagement from its end users, Orca has been designed and implemented by the Sun Microsystems, Inc., Accessibility Program Office.

NOTE: Orca is currently a work in progress. As a result, this and other books in the Orca Documentation Series are under continuous modification and are also in various states of completeness.

This book provides a guide for the installation, configuration and use of Orca. It also includes details on how users can customize the keyboard and braille input mappings of Orca.

Chapter 1. Introduction

To be written.